مقدمه ای بر کامپیوترها و برنامه نویسی با استفاده از C++

نویسنده : الکس اف. بیلایف

ایالات متحده – دانشگاه میشیگان – دپارتمان مهندسی هسته ای و دانش های رادیواکتیو

ترجمه شده توسط گروه ترجمه دانشجویان دانشکده مهندسی برق دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

همکاران این پروژه : دانیال ، شیوا ، کیوان ، مهدی

بازنگری و تکمیل ترجمه: کیوان

فصل اول :

این صفحه تعمدا خالی رها شده است!! :دی

فصل دوم

بررسی ساختار ذخیره و بازیابی اطلاعات

۱.۲ بیت، نیبل، بایت، کلمه[۱]

بیت

اجزای کامپیوتر، از مجموعه‌ای از ترانزیستورها تشکیل است که هر کدام از این ترانزیستورها، می‌توانند فقط در دو حالت قرار بگیرند؛ وضعیت با "ولتاژ بالا" و "ولتاژ پایین" . در واقع این حالت ولتاژ بالا (/پایین) را به صورت مجازی، روشن(/خاموش) یا یک(/صفر) می‌نامند. علاوه بر این دو حالت موجود در ترانزیستورها، در سایر قطعات کامپیوترها نیز می‌توان حالتی مشابه را مشاهده کرد. مثلا قسمت‌های مختلف حافظه‌های معناطیسی از جمله دیسک‌سخت، فلاپی‌دیسک و ... می‌توانند، مغناطیسی‌شده، یا مغناطیسی‌نشده باشند. "pit" ها در حافظه‌هایی از جمله CD یا DVD ، سوراخ‌ها در نوارهای کاغذی پانچ. امروز در هر کامپیوتر عادی میلیون ها و شاید هم بیلیون ها بیت در انواع و اقسام فرم های آن وجود دارد. با توجه به اندازه ی تقریبی هر بیت که به طرز قابل توجهی کوچک می باشد[۲]، می توان تعداد بسیار زیادی از آنها را در یک جا قرار داد. همچنین با توجه به اینکه مقدار هر کدام از بیت ها را می توان در زمان بسیار اندکی[۳] تغییر داد، لذا می توان عملیات بسیار زیادی را با استفاده از این خانه های کوچک انجام داد.

نیبل

هر نیبل، مجموعه ای از ۴ بیت است. بدیهی است با توجه به دو حالته بودن هر بیت، هر نیبل می تواند ۱۶ = $۲^۴$ حالت مجزا از هم را داشته باشد.

بایت

هر بایت متشکل از دو نیبل یا ۸ بیت است. هر بایت می تواند ۲۵۶= $۲^۸$ حالت مجزا از هم را داشته باشد. ظاهرا این تعداد برای نگه داری علائم نگارشی کاملا کافی به نظر می رسد. چرا که با در نظرگرفتن تمامی حروف (بزرگ و کوچک)، به همراه اعداد و سایر علائم نگارشی مانند « ! & () : »، تعداد زیادی حالات خالی برای اضافه کردن بسیاری از علائم کاربردی وجود خواهد داشت.

[۱] bit, Nibble, Byte, Word

[۲] تقریبا به اندازه ی یک میکرومتر = $۱۰^{-۶}$ متر

[۳] تقریبا در عرض یک نانو ثانیه = $۱۰^{-۹}$ ثانیه

کلمه

هر کلمه مجموعه ای است از چندین بایت است. معمولا هر کلمه از ۴ بایت تشکیل شده است که با این فرض می تواند $۲^{۳۲}=$ ۴۲۹۴۶۷۲۲۹۶ حالت مختلف را بخود بگیرد. ظاهرا شما باید با دیدن این عدد حسابی شوکه شده باشید! در آینده خواهیم دید که چگونه می توان تمامی اعداد حقیقی را با استفاده از این ابزار برای کامپیوتر مدل کرد.

حال می خواهیم کمی تصور کنیم.

در شکل زیر تصویر سگ یکی از نویسندگان این کتاب را مشاهده می کنید! اولین عکس دارای حجمی به اندازه ی ۱۲ مگابایت است که تقریبا به اندازه ی ۹۶ میلیون بیت دارد! در صورتی شما کمی در عکس زوم کنید، متوجه خواهید شد، کیفیت عکس کمی کم تر شد. در حالت می توانید ببینید که چگونه سبیل های این حیوان زبان بسته(!) ناهموار شدند! اگر کمی بیشتر روی سبیل نامبرده(!)زوم کنیم، خانه های چهار خانه ی عکس کاملا مشخص دیده خواهند شد. هر کدام از این خانه های چهار خانه، از ۲۴ بیت تشکیل شده اند که برای هر رنگ قرمز، آبی، سبز، ۸ خانه در نظر گرفته شده است.

۲.۲ ارقام ۴ بیتی در سیستم مبنای ۲ ، شانزده و ده

حالت های مختلف ۴ بیتی در کنار هم (یک نیبل) را می توان در حالت های گوناگونی نشان داد. برای راحتی کار، معمولا برنامه نویسان این اعداد را با استفاده از نمایش اعداد در مبنای ۱۶ نشان می دهند. در واقع از 1, 2, …, 9, A, B,…, F به جای نمایش 0000, 0001, 0010...1001, 1010, 1011...1111 استفاده می کنند.

نمایش در مبنای ده	نمایش در مبنای شانزده	نمایش بیت های ۴ تایی
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

۳.۲ محاسبات در مبنای دو

در نظر بگیرید ما می خواهیم، محاسباتی را مستقیما بر روی اعداد مبنای دو انجام دهیم. بدیهی است که در این مبنا تنها با اعداد یک و صفر و یک کار داریم. همچنین می دانیم عملگر های جمع و تفریق، دارای خاصیت جابجایی هستند.

عملیات ریاضی در مبنای دو، قابل مقایسه با عملیات ریاضی در مبنای ده است. ظاهرا، استفاده ی انسان از مبنای ده به خاطر آناتومی انسان و ده انگشتی بودن وی است. تمامی اعمال جمع، تفریق، ضرب، تقسیم، همگی قابل مقایسه هستند. در واقع آنچه که مهم است، در نظر گرفتن حداکثر مقدار ۱، برای مبنای دو (به جای رقم ۹ در مبنای ده) ، و انتقال مقدار اضافی به جایگاه عددی بالاتر، بر اساس ضریبی از مبناست. گاهی مانند تفریق، بر عکس این کار هم انجام می شود.

جمع اعداد در مبنای دو به صورت زیر خواهند بود :

0 + ? = ? + 0 = ?
1 + 1 = 10
10 + 1 = 1 + 10 = 11
11 + 1 = 1 + 11 = 100
10 + 10 = 10 + 10 = 100
.
.
1111 + 111 = 111 + 1111 = 10110
.
.
.

حاصلضرب اعداد در مبنای دو نیز به صورت زیر خواهد بود:

0 × ? = ? × 0 = 0
1 × ? = ? × 1 = ?
10 × 10 = 10 × 10 = 100
10 × 11 = 11 × 10 = 110
11 × 11 = 1001
.
.
.
1111 × 111 = 111 × 1111 = 1101001
.
.

۴.۲ تبدیل از مبنای دو به مبنای ده

۱.۲.۴ اعداد طبیعی

معمولا افراد در هنگام برخورد با محاسبات در مبنای دو، دچار ترس و وحشت می شوند! اگر شما هم در چنین وضعیتی هستید می توانید محاسبات خود را به مبنای ده تبدیل کنید.

برای اینکار عدد زیر را در مبنای دو در نظر بگیرید :

$$b_n b_{n-1} b_{n-2} \dots b_2 b_1 b_0,$$

بطوریکه b_i همگی اعداد صفر یا یک هستند.

مقدار عددی در مبنای دو را به صورت زیر محاسبه می کنیم:

$$d_{10} = b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} \dots b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

برای مثال :

$$10110_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2^4 + 2^2 + 2^1 = 16_{10} + 4_{10} + 2_{10} = 22_{10}$$

حاصل جمع $[1111_2(15_{10}) + 111_2(7_{10})]$ است. در حالیکه:

$$1101001_2 = 2^6 + 2^5 + 2^3 + 2^0 = 64_{10} + 32_{10} + 8_{10} + 1_{10} = 105_{10}$$

حاصل ضرب $[1111_2(15_{10}) \times 111_2(7_{10})]$ است.

توجه کنید که زیروند ۲ نشان دهنده ی مبنای دو و زیروند ۱۰ نشان دهنده ی مبنای ده است.

۲.۴.۲ اعداد حقیقی

می خواهیم اعداد حقیقی از مبنای دو به اعداد حقیقی در مبنای ده تبدیل کنیم.

عدد حقیقی زیر را در مبنای دو در نظر بگیرید:

$$b_n b_{n-1} b_{n-2} \dots b_2 b_1 b_0 . b_{-1} b_{-2} \dots b_{-(m-2)} b_{-(m-1)} b_{-m},$$

بطوریکه b_i همگی اعداد صفر یا یک هستند.

مقدار عدد حقیقی در مبنای دو را به صورت زیر محاسبه می کنیم:

$$r_{10} = b_n \times 2^n + b_{n-1} \times 2^{n-1} \dots b_1 \times 2^1 + b_0 \times 2^0 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} \dots + b_{-(m-1)} \times 2^{-(m-1)} + b_{-(m)} \times 2^{-m}$$

۵.۲ محاسبات اعداد صحیح مبنای دو در رایانه

همانطور که ذکر شد، به علت دودویی بودن محاسبات درون رایانه ها، به طور طبیعی محاسباتشان بر اساس مبنای دو انجام می شود. در واقع، این آسان ترین راه برای پیاده سازی ساختاری برای انجام محاسبات درون رایانه است. هر چند که در پیاده سازی چنین ساختاری در واقعیت کار بسیار مشکلی است! به علت مسائل و محدودیت های حافظه ای، رایانه ها، با اعداد به صورت ساختاری با تعداد کاراکتر های محدود برخورد می کند. در ۱۶ بیت، می توان ۶۵۵۳۶ حالت مختلف را در نظر گرفت. امروزه رایانه های ۳۲- بیتی(در واقع هر خانه با ۴۲۹۴۹۶۷۲۹۶ حالت مختلف) کم کم جای خود را به رایانه های ۶۴- بیتی(تقریبا ۱۸×۱۰^{۱۸} حالت) می دهند. این بدین معنی است که سخت افزار های جدید، باید ساختار مطابق با خانه های ۶۴- بیتی داشته باشند.

در اینجا رایانه های ۳۲- بیتی را در نظر بگیرید. در مباحث ریاضی هر مبنا، در صورتی که تعداد ارقام بیش از مقدار ساختار(در اینجا ۳۲- بیت) ، اعداد اضافی، حذف می شوند. درواقع این ارقام اضافی، وارد خانه ای می شوند به نام "سطل آشغال بیت ها"[۴] و در این خانه نابود می شوند.

[4] Bit Bucket

1.5.2 ساختار محاسبات ریاضی" مکمل ۲"[5]

در اینجا بحث ما روی اعداد و بدون علامت[6] (+) علامت دار[7] (+ یا –) است. زمانی که برای عددی، علامتی در نظر می گیریم، در باید برای علامت آن فضایی را اختصاص دهیم. در طول مدت چندین دهه ی قبل که رایانه ها اختراع شدند، روش های گوناگونی برای نحوه ی اختصاص داده مقداری از حافظه برای مشخص کردن منفی بودن اعداد معرفی شده اند. در این میان روش "مکمل ۲" شاید مهمترین روش بود که ارائه شد و از آن استفاده شد. دلیل برتری این روش نسبت به روش های دیگر، آسان بودن جمع دو عدد مثبت، منفی و یا ترکیبی از اعداد مثبت/منفی بود.

فرض کنیم عدد $\bar{i}$ مکمل عدد i باشد. ارقام $\bar{i}$ اینگونه تعریف می شوند که به ازای هر رقم i ، مکمل آن را قرار می دهیم. (مثلا ۱ مکمل صفر است و صفر مکمل ۱ است.)

برای اعداد شامل ۳۲– بیت به ازای هر عدد خواهیم داشت :

$$\bar{i} + i = 11111111111111111111111111111111$$

در واقع می توان نوشته ی بالا را اینگونه تفسیر کرد :

هر عددی + مکمل عدد $= 11111111111111111111111111111111$

به همراه این فرض که :

$$\begin{array}{rl} & 11111111111111111111111111111111 \\ + & 00000000000000000000000000000001 \\ \hline = & 00000000000000000000000000000000 \end{array}$$

چرا که گفتیم اعداد خارج از محدوده ی ۳۲ بیت، دور انداخته می شوند!

لذا با اعمال ساده داریم:

$$i + \bar{i} + 1 = 0 \quad \rightarrow \quad -i = \bar{i} + 1$$

در واقع ما توانستیم منفی عددی را بر اساس مکمل آن تفریف کنیم. پس درواقع دو نکته را باید به خاطر داشت :

- هر گاه با i – مواجه شدیم، می توانیم $\bar{i} + 1$ را به جای آن جایگزین کنیم. اینگونه می توانیم تمای علامت های منفی را به علامت جمع تبدیل کنیم.
- اعدادی آخرین رقم آنها از سمت چپ ۱ بود، اعدادی منفی هستند! در این رابطه، در ادامه بیشتر توضیح خواهیم داد.(البته بهتر است، قبلا خودتان در این رابطه، مثال هایی را امتحان کنید. مثلا چندین عدد در مبنای دو در نظر بگیرید و منفی کنید.)

[5] 2's Complement
[6] Unsigned
[7] Signed

با مثال هایی می توان نشان داد که روش مکمل ۲، چقدر آسان و ساده است.

برای مثال عبارت a-b را می توان اینگونه نوشته : $a+\bar{b}+1$

به عنوان مثال دیگر؛ اعداد ۴ بیتی $3_{10} = 0011_2$ و $4_{10} = 0100_2$ را در نظر بگیرید.

$$3_{10} + 4_{10} = 0011_2 + 0100_2 = 0111_2 = 7_{10}$$
$$3_{10} - 4_{10} = 0011_2 - 0100_2 = 0011_2 + 1100_2 = 1111_2 = -0001_2 = -1_{10}$$
$$-3_{10} - 4_{10} = -0011_2 - 0100_2 = 1101_2 + 1100_2 = 1001_2 = -0111_2 = -7_{10}$$

کار ساده ای بود !

۶.۲ اعداد ۳۲- بیتی در مبنای دو، شانزده، علامت دار و بدون علامت

کامپیوتر های ۳۲- بیتی می توانند ۴۲۹۴۶۷۲۲۹۶ = $۲^{۳۲}$ حالت مختلف را در خود بگنجانند. همانطور که قبلا نیز گفته شد، می توان قرار داد گرد که اعداد "علامت دار" یا "بدون علامت" باشند. در صورتی که علامت به عددی اضافه شد، کران بالا وپایین، آن خانه تغییر خواهد کرد. بدین معنی که با اختصاص دادن خانه ای به عنوان علامت، اعداد منفی از سمت منفی بازه ی عدد اضافه می شوند و از مقدار کران بالا کم می شود. عدد $2147483647 = 2^{31} - 1 = 01111111111111111111111111111111_2$ را در نظر بگیرید. همانطور که گفته شد، در اعداد علامت دار، اعدادی که رقم اول آنها از سمت چپ صفر باشد، مثبت، هستند. لذا عدد بالا بزرگترین عدد مثبت علامت داری است که می توان در اعداد ۳۲-بیتی تعریف کرد. عدد $-2147483648 = -2^{31}$ کوچکترین عددی است که می توان در میان اعداد ۳۲- بیتی، مشخص کرد.

به جدول زیر نگاه کنید. این جدول توزیع اعداد ۳۲- بیتی را برای اعداد علامت دار و بدون علامت، نشان می دهد. همانطور که می بینید که نشان دادن اعداد مبنای دو، در همان مبنا، چندان آسان نیست!! لذا مرسوم است از مبنای ۱۶ برای این کار استفاده شود. چرا که هر ۴ رقم از اعداد مبنای دو، به عنوان یک رقم در مبنای ۱۶ نشان داده می شود.

مبنای دو	مبنای شانزده	عدد صحیح بدون علامت	عدد صحیح با علامت
00000000000000000000000000000000	00000000	0	0
00000000000000000000000000000001	00000001	1	1
00000000000000000000000000000010	00000002	2	2
00000000000000000000000000000011	00000003	3	3
00000000000000000000000000000100	00000004	4	4
.	.	.	.
.	.	.	.
.	.	.	.
00000000000000000000000000001111	0000000F	15	15
00000000000000000000000000010000	00000010	16	16
00000000000000000000000000010001	00000011	17	17
.	.	.	.
.	.	.	.
.	.	.	.
01111111111111111111111111111110	7FFFFFFE	2147483646	2147483646
01111111111111111111111111111111	7FFFFFFF	2147483647	2147483647
10000000000000000000000000000000	80000000	2147483648	-2147483648
10000000000000000000000000000001	80000001	2147483649	-2147483647
.	.	.	.
.	.	.	.
.	.	.	.
11111111111111111111111111111100	FFFFFFFC	4294967292	-4
11111111111111111111111111111101	FFFFFFFD	4294967293	-3
11111111111111111111111111111110	FFFFFFFE	4294967294	-2
11111111111111111111111111111111	FFFFFFFF	4294967295	-1

۷.۲ مسائل فصل

۱- جمع اعداد ۳ بیتی؛ (عدد مبنای ۸)+ (عدد مبنای ۸) = (عدد مبنای ۸)

توجه! "عدد مبنای ۸" عددی است که می توان آن را تنها با ۳ بیت نشان داد.

رایانه ی مجازی را در نظر بگیرید که تنها توانایی انجام عملیات محاسبات ۳- بیتی را داراست. در واقع هر رقمی بعد از ۳ بیت، بعد از ریخته شدن به سطل آشغال بیت ها نابود خواهد شد. حال می خواهیم با این فرض جدول زیر را کامل کنیم. هدف این است که در هر خانه، اعداد متناظر با آن سطر و ستون با همدیگر جمع شود. برای نمونه، حاصل سه خانه، در جدول قرار داده شده اند.

سعی کنید اعداد را بدون تبدیل به مبنای ده، با همدیگر جمع کنید.

+	000	001	010	011	100	101	110	111
000								
001								
010								
011								
100								
101						010		
110								
111					011			110

۲- ضرب اعداد ۳ بیتی؛ (عدد مبنای ۸) × (عدد مبنای ۸) = (عدد مبنای ۸)

این تمرین مشابه نمونه ی قبلی است. با این تفاوت که در اینجا قصد ضرب اعداد را داریم.

×	000	001	010	011	100	101	110	111
000								
001								
010								
011								
100								
101						001		
110								
111					100			001

۳- تمرینی برای اعداد مبنای ۱۶

حال در این تمرین فرض کنید، رایانه ی فرضی شما، ۴ بیت را برای محاسبات خود در اختیار دارد. بنابرین نتیجه هر گونه عملیات ریاضی دو عدد چهار بیتی در این رایانه، یک عدد چهاربیتی خواهد بود.

جدول زیر تمامی اعداد ممکن ۴ بیتی را در مبنای دو، مبنای ۱۶ و با فرض علامت دار بودن و بدون علامت بودن نشان می دهد.

مبنای دو	مبنای شانزده	عدد صحیح بدون علامت	عدد صحیح علامت دار
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	8	-8
1001	9	9	-7
1010	a یا A	10	-6
1011	b یا B	11	-5
1100	c یا C	12	-4
1101	d یا D	13	-3
1110	e یا E	14	-2
1111	f یا F	15	-1

حال شما باید حاصل عملیات ریاضی ۴-بیتی زیر را در مبنای ۲ انجام دهید. سعی کنید با انجام این تمرینات و مقایسه ی نتایج آنها با محاسبات مبنای ده، محاسبات مبنای دو و اعداد منفی در این مبنا را بهتر درک کنید.

الف) حاصل جمع اعداد ۲ و ۳ ؟

ب) حاصل جمع اعداد ۲- و ۳- ؟

ج) حاصل جمع اعداد ۸- و ۸- ؟

د) حاصل ضرب اعداد ۲ و ۳ ؟

ر) حاصل ضرب اعداد ۲- و ۳- ؟

ز) حاصل ضرب اعداد ۸- و ۸- ؟

۴- جمع اعداد مبنای ۱۶

با فرض کردن همان رایانه ی ۴- بیتی، جدول زیر را برای جمع اعداد ۴ بیتی، در مبنای ۱۶ تکمیل کنید.

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0															
1	1	2														
2	2	3	4													
3	3	4		6												
4	4	5			8											
5	5	6				a										
6	6	7					c									
7	7	8						e								
8	8	9							0							
9	9	a								2						
A	a	b									4					
B	b	c										6				
C	c	d											8			
D	d	e												a		
E	e	f													c	
F	f	0														e

۵- ضرب اعداد مبنای ۱۶

×	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0															
1	0	1														
2	0	2	4													
3	0	3		9												
4	0	4			0											
5	0	5				9										
6	0	6					4									
7	0	7						1								
8	0	8							0							
9	0	9								1						
A	0	a									4					
B	0	b										9				
C	0	c											0			
D	0	d												9		
E	0	e													4	
F	0	f														1

۶- تمرینی برای اعداد علامت دار و بدون علامت پنج بیتی

صحیح علامت دار	صحیح بدون علامت	مبنای ۱۶	مبنای دو
0	0	00	00000
1	1	01	00001
2	2	02	00010
3	3	03	00011
4	4	04	00100
5	5	05	00101
6	6	06	00110
7	7	07	00111
	8	08	01000
	9	09	01001
	10	0A	01010
	11	0B	01011
	12	0C	01100
	13	0D	01101
	14	0E	01110
	15	0F	01111
	16	10	10000
	17	11	10001
		12	10010
		13	10011
			10100
			10101
			10110
			10111
			11000
			11001
			11010
			11011
			11100
			11101
			11110
			11111

۷- کار بیشتر روی روش مکمل ۲

حال فرض کنید که رایانه ی شما تنها توانایی استفاده از خانه های ۵ بیتی را دارد! بنابرین حاصل اعمال ریاضی دو عدد ۵ بیتی، یک عدد ۵ بیتی خواهد بود و ارقام خارج از حوزه ی ۵ بیت نابود خواهند شد.

حال اعمال ریاضی زیر را در مبنای دو انجام دهید و با محاسبات در مبنای ده مقایسه کنید. توجه کنید که برای اعداد منفی باید از روش مکمل ۲ استفاده کنید.

الف) حاصل جمع ۳ و ۴ ؟

ب) حاصل جمع ۱۶ و ۱۷ ؟

پ) حاصل جمع ۲- و ۳- ؟

ت) حاصل جمع ۱۶- و ۱۶- ؟

ج) حاصل ضرب ۳ و ۴ ؟

چ) حاصل ضرب ۲- و ۳- ؟

ح) حاصل ضرب ۱۶- و۱۶- ؟

۸- کاربیشتر برای اعداد ۶-بیتی

در جدول زیر ستون اول مربوط به اعداد در مبنای ۶ است. حال شما باید سایر ستون ها را به ترتیبی که در زیر گفته می شود، پر کنید:

۱) در ستون دوم، مکمل اعداد ستون اول را قرار دهید.

۲) در ستون سوم، مکمل ۲ اعداد در ستون اول را بنویسید.

۳) در ستون چهارم، مکمل ۲ اعداد ستون سوم را بنویسید.

۴) در ستون پنجم ، با علامت تیک یا ضربدر مشخص کنید آیا عدد ستون اول(علامت دار)، مثبت است یا منفی؟

۵) در ستون ششم، در صورتی که عدد ستون اول، با فرض بدون علامت بودن، فرد است، علام بگذارید.

۶) در ستون هفتم، در صورتی که عدد ستون اول، با فرض بدون علامت بودن، قابل تقسیم بر ۲ است.

۷) در ستون هفتم، در صورتی که عدد ستون اول، با فرض بدون علامت بودن، قابل تقسیم بر ۴ است.

ستون اول	ستون دوم	ستون سوم	ستون چهارم	ستون پنجم	ستون ششم	ستون هفتم	ستون هشتم
000010							
111001							
110110							
000001							
001101							
110010							
010010							
110110							
101111							
101010							

۹- تمرینی برای تبدیل مبنا

۱) ۵ الگوی ۳۲-بیتی را به مبنای ۱۶ تبدیل کنید.

۲) اعدادی را که می توان آنها را بر ۲ تقسیم کرد را مشخص کنید. (/۲)

۳) اعدادی را که می توان آنها را بر ۴ تقسیم کرد را مشخص کنید. (/۴)

۴) اعداد را با در نظر گرفتن علامت دار بودن، از لحاظ مثبت یا منفی بودن مشخص کنید. (+/-)

۵) عددی که با در نظر گرفتن مکمل ۲، بزرگترین مقدار را دارد، مشخص کنید. (>)

۶) عددی که با در نظر گرفتن مکمل ۲، کوچکترین مقدار را دارد(منفی و بزرگترین مقدار قدر مطلق)، مشخص کنید. (<)

الگوی ۳۲- بیتی	مبنای ۱۶	/۲	/۴	+/-	>	<
1111 0001 0010 0110 1110 0101 0100 0110						
1000 0010 0000 1100 1000 0010 1011 1011						
1011 0001 1000 1011 0100 1111 0000 0111						
0010 0001 1101 1001 0100 1101 0111 1110						
0010 1000 1010 0001 0010 1010 0101 0111						
1110 1100 1000 1000 0011 0101 1110 1101						
1111 0111 0000 1100 0011 1010 0010 1101						
0110 0111 0100 0000 0101 1000 1011 0000						
0011 1110 0000 0011 1100 0110 0000 1101						
0000 1110 1101 1001 1111 1011 0101 1000						
0110 1111 0001 1010 0010 1111 0110 1000						
1010 1000 0000 1110 1111 1010 0010 0011						
0000 0010 0011 0010 0100 1100 0001 0101						
0011 1100 1010 0111 1011 1011 0111 0000						
1100 0011 0100 1011 0110 1011 1111 0011						
1101 0001 1011 1110 1011 1010 0000 0001						
1101 0011 1110 1011 0100 0101 0100 1111						
1111 0011 0001 0111 1110 1110 0011 0000						
0100 1101 1110 1110 0000 0011 1011 1011						
0111 1011 1010 1111 0100 0010 0010 0101						
1101 1001 1101 1111 1101 1101 1010 1101						
1000 1101 1010 0101 0110 1010 1111 0110						
0100 0100 0110 1111 0001 0010 0111 0001						
0011 1110 1010 1100 1100 0111 0010 0111						
1110 1111 0111 1100 1001 0101 0001 1101						
1001 1000 1010 0101 1000 0010 1011 1010						
1011 1111 1000 0100 0011 0101 0011 0111						
1001 1010 0000 0011 1111 1010 0110 1111	9A03FA6F					

فصل سوم

الگوریتم ها[8] و شبه کد ها[9]

در این فصل به توصیف الگوریتم ها و نحوه ی پیاده سازی آنها خواهیم پرداخت .

همچنین در پایان فصل کمی در رابطه ساختمان کامپیوتر ، و نحوه ی پیاده سازی دستورات در هنگام اجرای یک الگوریتم ، توضیح داده خواهد شد.

۱.۱ الگوریتم چیست ؟

الگوریتم مجموعه ای از دستورات کاملا مشخص و با ترتیب مناسب است که برای رسیدن به هدفی خاص ، باید طبق آن ترتیب اجرا شوند.

هر الگوریتم دارای یک نقطه ی شروع[10] و یک نقطه ی پایان[11] است . نقطه شروع می تواند همراه با دریافت چندین ورودی [12] و نقطه خروجی می تواند همراه با چاپ چندین خروجی[13] باشد.

یک الگوریتم از دستورات و مراحل مجزا تشکیل می شود.

یک دستور از عمیات هایی تشکیل شده است که به خوبی تعریف شده اند و عموما بر اساس ورودی ها (آنچه عملیات دریافت می کند) کار می کنند و معمولا خروجی هایی (آنچه عملیات تولید می کند) را تولید می کنند.

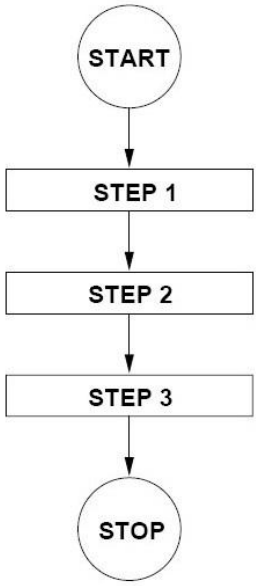
هر الگوریتم به خوبی تعریف شده است و به ازای یک ورودی معتبر خروجی اش قابل پیش بینی خواهد بود.

هر الگوریتم به صوورت یک دیاگرام جریان [14] نشان داده می شود. در هر دفعه که یک مرحله اجرا می شود، اجرا به مرحله ی بعد منتقل می شود.

الگوریتم باید تضمین شده باشد که هر گاه به ازای یک ورودی معتبر اجرا شود، از طریقی عاقلانه هم خاتمه می یابد.

[8] algorithm
[9] Pseudocode
[10] Start Point
[11] Stop Point
[12] Input
[13] Output
[14] Flow Diagram

۲.۳ نمایش الگوریتم با فلوچارت



نمایش شکل بالا ، ۳ مرحله از اجرای یک عملیات را نشان می دهد. همانطور که در شکل نشان داده شده است ، مرحله ی شروع و پایان با بیضی یا دایره نشان داده می شوند. این نوع از نمایش نحوه ی اجرای یک الگوریتم را ، فلوچارت [۱۵] می نامیم که شبیه نمودار مداری درمبحث الکترونیک است.

۳.۳ شبه برنامه یا سودوکد (Pseudocode)

همانطور که اشاره شد، یک الگوریتم، مجموعه ای از دستورات تعریف شده است . سودوکد یک نمونه ی دیگر از بیان مراحل اجرای یک الگوریتم است. در واقع در سودوکد ، هدف تنها بیان با ترتیب دستورات بدون وارد شدن به جزئیات برنامه نویسی آن است. منظور از عدم توجه به جزئیات برنامه نویسی این است که برنامه نویس، بدون توجه به اینکه پیاده سازی برنامه، به چه زبان برنامه نویسی و در چه محیطی انجام خواهد گرفت، دستورات را در ساده ترین شکل آنها خواهد نوشت.

برای مثال پیاده سازی سودوکد فلوچارت بالا به صورت زیر خواهد بود :

START ⇒ Step 1 ⇒ Step 2 ⇒ Step 3 ⇒ STOP

یک حالت قراردادی از سودو کد ، نوشتن دستورات به ترتیب و از بالا به پایین است:

START
Step 1
Step 2
Step 3
STOP

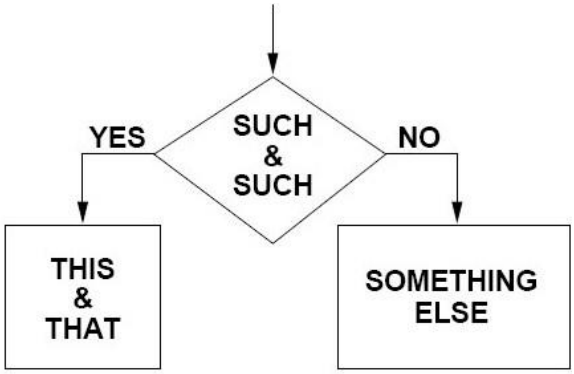
[15] flowchart or schematic diagram

۴.۳ ساختار های تصمیم گیری یا شرطی

یک ساختار شرطی ، قالبی است که بر اساس شرط مورد نظر و جواب آن ، جریان و روند اجرای دستورات به شاخه ی متفاوتی منتقل می شود. ساختار شرطی در قالب سودو کد به صورت زیر بیان می شود:

IF *such-and-such* ***THEN DO***
this-and-that
OTHERWISE DO
something-else

این ساختار در قالب فلوچارت به صورت زیر می تواند باشد:



توجه کنید که در اینجا ، یک شکل جدید معرفی کرده ایم. در واقع برای نشان دادن شاخه شاخه شدن(یا همان ساختار شرطی) ، شکل لوزی را به کار می بریم. معمولا در یک شکل ساختار شاخه ای شدن یا شرطی معمولا یک جریان ورودی داریم و دو جریان خروجی .

۵.۳ ساختار حلقه ای[۱۶] و ساختار پرشی[۱۷]

ساختار پرش یا حلقه ، یک توانایی برای انتقال روند اجرا به سطری مشخص در برنامه است. برای مثال سودو کد زیر را در نظر بگیرید:

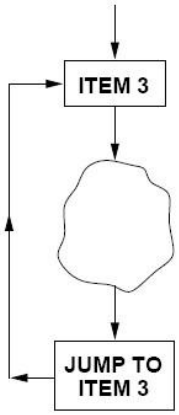
ITEM 3

.
.
a set of other steps
.
.

JUMP TO: ITEM 3

[16] Looping Instruction
[17] Jump Instruction

در واقع دستورات بین ITEM 3 و JUMP TO: ITEM 3 یک ساختار حلقه ای نامیده می شود. نمایش این سودو کد در قالب فلو چارت به صورت زیر است :



۶.۳ به ترتیب نویسی دستورات[18] ، ساختار شرطی و ساختار تکرار

سه واحد ترتیب گذاری دستورات، ساختار شرطی و ساختار حلقه ها (تکرار) تقریبا تمام چیزی هستند که برای ساختار یک برنامه لازم اند. برای پیاده سازی این سه ساختار ،شیوه های گوناگونی وجود دارند ، مثلا به زبان C++ یا MATLAB . اما به یاد داشته باشید که تمام این شیوه به نتیجه ی یکسانی ختم خواهند شد. **در واقع این سه ساختار ستون های اساسی برنامه نویسی اند.**

۷.۳ جمع بندی

سودوکد : زبان مرتب سازی الگوریتم به شیوه ی سودوکد ، چندان قانونمند نیست. سعی کنید با قراردادهای بین خودتان ، مانند برجسته نویسی ، فاصله گذاری، پررنگ نویسی، ایتالیک و با دیگر روش های تاکید بر واژه ، نوشته ی خود را خواناتر کنید.

فلوچارت : این مورد امکانپذیر است تا الگوریتم را با استفاده از فلوچارت، به صورت نموداری درآورد که در واقع همان جریان منطقی عملیات را نشان می دهد. در رابطه با این شیوه تعدادی از ساختار های استاندارد (نه به صورت کاملا جدی ثبت شده) وجود دارد که بسیار شبیه به دیاگرام های مداری در علم الکترونیک است. در واقع این یک تمرین خیلی خوب است که با تعریف کردن هر متغیر و معین کردن نقطه ی شروع و پایان ، برنامه ای خوش تعریف را تمرین کنید. سودوکد و فلوچارت ابزارهای حتمی و واجبی هستند که می توانید در نوشتن یک الگوریتم از آنها با کمترین انرژی، استفاده کنید.

[18] Sequencing

۸.۳ چند مثال

۱.۸.۳ الگوریتم صبحانه !!!

۱ –شروع الگوریتم صبحانه

۲ –اگر امروز، یکشنبه نیست، غذای خسته کننده ی خشکیده ی میوه ها را بخورید! برای جذاب تر کردن غذا، یک نصفه ی موز هم اضافه کنید!! و گرنه ،

a) در یک ظرف دو عدد تخم مرغ (البته بدون پوست !) را هم بزنید .

b) یک فنجان شیر ، ترجیحا کم چرب، را به تخم مرغ ها اضافه و هم بزنید.

c) یک قاشق سوپ خوری روغن زیتون، سه قاشق شیره ی افرا و یک قاشق چای خوری وانیل به مخلوط شیر و تخم مرغ اضافه کنید.

d) یک قاشق بزرگ بردارید.

e) مخلوط را با یک چرخش بزرگ، به هم بزنید.

f) اگر در مخلوط شما هنوز تکه های بزرگی هستند که هنوز کامل مخلوط نشده اند، بروید به مرحله ی 2.(e) در غیر اینصورت بروید به مرحله ی 2.(g). (نکته ی فنی: سعی کنید هنگام مخلوط کردن، خیلی آرام اینکار رو بکنید.در ضمن کمی تکه های مخلوط نشده برای حالت غذای شما خوب است!)

g) یک سیب بزرگ (پوست کنده نشده) را در آن خرد کنید و (اختیاری) نصف فنجان از توت های ایرلندی در آن بریزید.(خوب شسته شده باشند)

h) مقداری روغن را در ظرف سرخ کن بریزید و دمای آن را تا $350^{\circ}F$ بالا ببرید.

i) خمیر را درون ظرف بریزید. دایره هایی به ضخامت ۴–۵ سانتی متر ایجاد کنید. سپس دما را به $300^{\circ}F$ کاهش دهید.مخلوط را دقیقا به مدت سه دقیقه با درب بسته و سپس بهم مدت سه دقیقه با درب باز سرخ کنید.

j) پنکیک را از ظرف سرخ کن بیرون آورید و بر روی یک دستمال سفره ی تمیز قرار دهید تا مقدار اضافه ی روغن را جذب کند.

k) پینکیک را همراه با شیره ی افرا و توت فرنگی های قطعه قطعه شده ، بخورید .

l) من شما را به مبارزه می طلبم تا اینکه بتوانید بیشتر از سه تا از این قطعات را بخورید! (کاری که هنوز انجام نشده!)

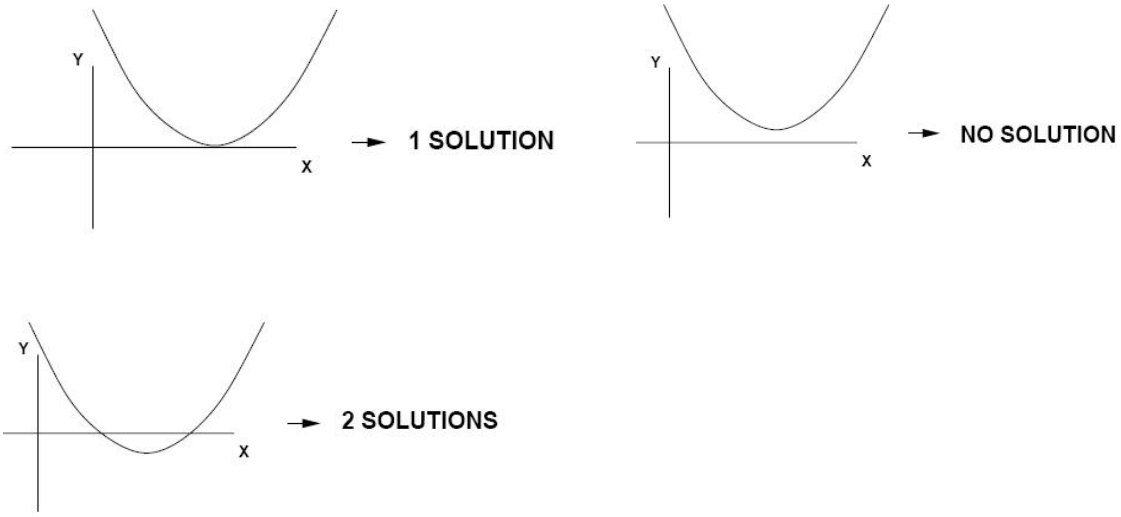
۳ –ظروف خود را پاک کنید.

۴ –پایان الگوریتم صبحانه .

این الگوریتم تقریبا ۸ قطعه پنکیک تهیه می کند که برای ۴ الی ۶ نفر کافیست!

۲.۸.۳ حل معادله ی درجه ی دوم $Ax^2 + Bx + C = 0$ زمانی که A,B,C ضرایبی ثابت دلخواه هستند.

تعریف مسئله: A,B,C اعداد ثابتی هستند که می توانند مقادیر حقیقی ثابتی را دریافت کنند و ما سعی داریم مقدار x را طوری پیدا کنیم که مقدار عبارت جبری $Ax^2 + Bx + C$ برابر صفر باشد. اگر ما این معادله را رسم کنیم ، شکل حاصل یک سهمی مطابق شکل های زیر خواهد بود که بسته به نحوه ی تقاطع این منحنی با محور x ها ، این معادله می تواند ۰ یا ۱ یا ۲ جواب داشته باشد.



ابتدا سعی می کنیم مسئله را از راه حل ریاضی به طور کامل حل کنیم . این مسئله یعنی فهم آنچه از ریاضی برای حل یک مسئله استفاده می کنید، برای نوشتن یک الگوریتم موفق بسیار مفید و واجب است.

$$
\begin{aligned}
Ax^2 + Bx + C &= 0 \\
A\left(x^2 + \frac{B}{A}x + \frac{C}{A}\right) &= 0 \quad \text{N.B.} \quad A \neq 0 \\
x^2 + \frac{B}{A}x + \frac{C}{A} &= 0 \\
x^2 + \frac{B}{A}x + \frac{B^2}{4A^2} - \frac{B^2}{4A^2} + \frac{C}{A} &= 0
\end{aligned}
$$

$$
\begin{aligned}
x^2 + \frac{B}{A}x + \frac{B^2}{4A^2} &= \frac{B^2}{4A^2} - \frac{C}{A} \\
x^2 + \frac{B}{A}x + \frac{B^2}{4A^2} &= \frac{B^2 - 4AC}{4A^2} \\
\left(x + \frac{B}{2A}\right)^2 &= \frac{B^2 - 4AC}{(2A)^2} \\
x + \frac{B}{2A} &= \pm\frac{\sqrt{B^2 - 4AC}}{2A} \quad \text{N.B.} \quad B^2 - 4AC \geq 0 \\
x &= -\frac{B}{2A} \pm \frac{\sqrt{B^2 - 4AC}}{2A} \\
x &= \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \quad A \neq 0, B^2 - 4AC \geq 0
\end{aligned}
$$

بنابرین با در نظر داشتن شروط $(A \neq 0,\ B^2 - 4AC > 0)$ دو جواب خواهیم داشت :

$$x = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \quad \text{and} \quad x = \frac{-B - \sqrt{B^2 - 4AC}}{2A}$$

با توجه به آخرین نتیجه ، در صورتی که $A \neq 0,\ B^2 - 4AC = 0$ تنها یک جواب خواهیم داشت :

$$x = \frac{-B}{2A}$$

ام در صورتی که $B^2 - 4AC < 0$ چطور ؟ *در اینصورت هیچ جواب حقیقی نخواهیم داشت .*

با توجه به قسمت های قبلی اگر A=0 باشد چطور ؟ دوباره محاسبه کنیم!

$Bx + C = 0$ (معادله ی $y = Bx + C$ ، معادله ی یک خط است)

$$x = \frac{-C}{B} \quad \text{N.B.} \quad B \neq 0$$

و این نقطه در واقع همان نقطه ای است که این خط محور x ها را قطع خواهد کرد.

اگر $A = 0 \text{ and } B = 0$ چطور ؟ در اینصورت اگر C=0 باشد ، مساله بی نهایت جواب خواهد داشت ، وگرنه ، هیچ جوابی نخواهد نداشت.

حال ، می توانیم سودو کد را برای الگوریتم حل این معادله ی درجه ی دوم ارائه کنیم .

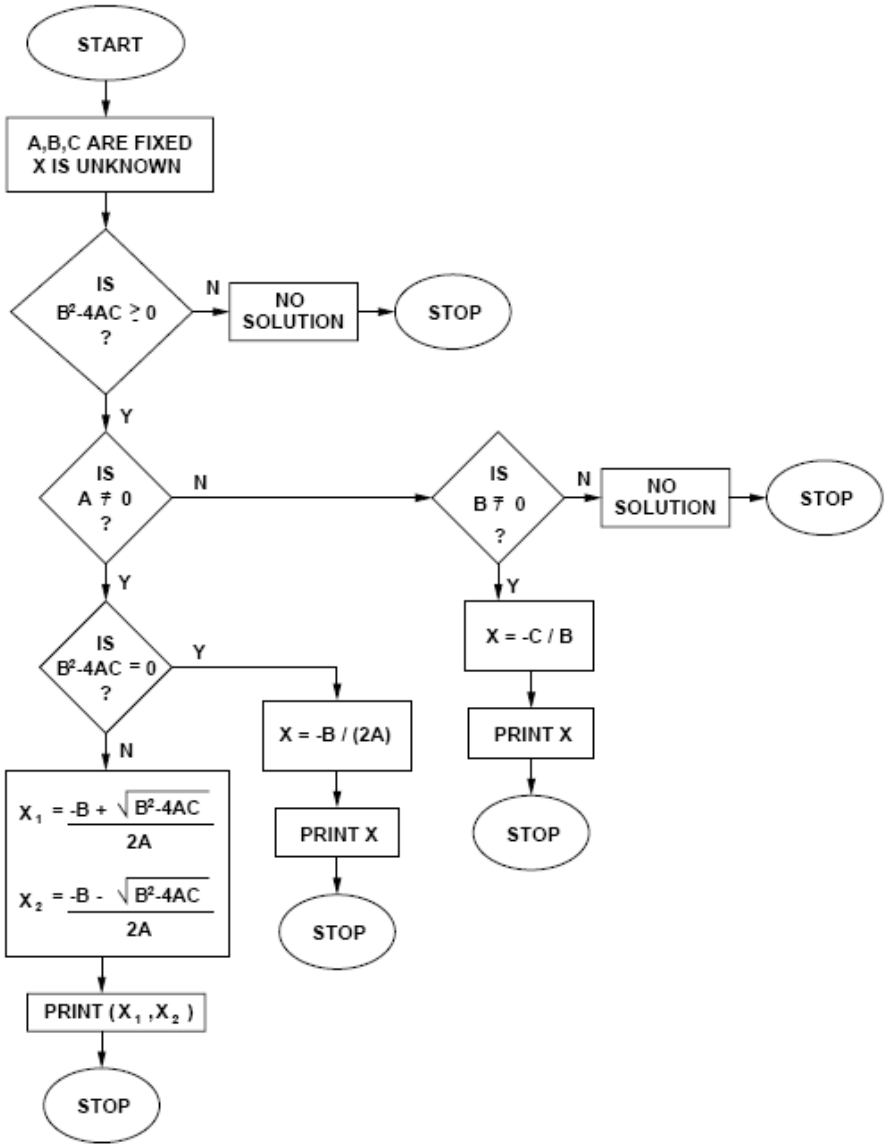
```
START A,B,C are inputs, known to be real, fixedconstan ts and x is the output,
expected
         to be real but unknown at the start
IF (B2 - 4AC < 0)
         PRINT (No solution because B2 - 4AC < 0)
         STOP
ELSE (i.e. B2 - 4AC ≥ 0)
        IF (A _= 0)
                IF (B2 - 4AC = 0)
                            x = -B/(2A)
                           PRINT (One solution: x, because B2 - 4AC = 0)
                          STOP
                ELSE (i.e. B2 - 4AC > 0)
                            x1 = (-B + √B2 - 4AC)/(2A)
```

```
                    x2 = (−B − √B2 − 4AC)/(2A)
                    PRINT (Two solutions are: x1 and x2)
                    STOP
        ELSE (i.e. A = 0)
                IF (B _= 0)
                    x = −C/B
                    PRINT (Only one solution x, because A = 0)
                    STOP
                ELSE (i.e. B = 0)
                    PRINT(No solution because A = 0 and B = 0)
                    STOP
END OF ALGORITHM
```

توجه کنید به... !

- IF ها می توانند به صورت تودر تو هم استفاده شوند.
- دندانه دار نوشته شدن الگوریتم.
- استفاده ی چند باره از نقطه ی پایان
- عدم استفاده ی از تمام قسمت های سودوکد در هر اجرا.
- استفاده از تمام قسمت های الگوریتم به ازای وارد کردن مقادیر گوناگون و کافی به ازای A, B, C
- الگوریتم برای تمامی مقادیر A, B, C جواب خواهد داد.
- می توان روند اجرای دستورات را در قالب فلوچارت نشان داد.

در زیر فلوچارت حل معادله ی درجه ی دوم نشان داده شده است :



۳.۸.۳ ساختار تکرار: یک حلقه ی محاسبه ی جمع

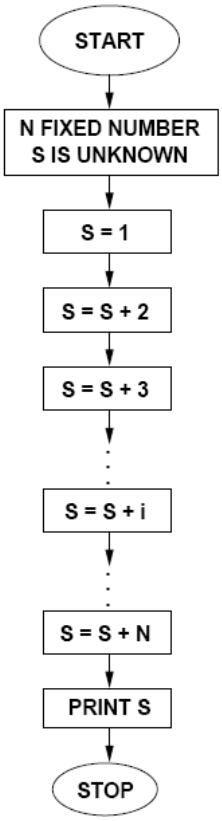
قبل از اینکه به مطالعه این قسمت بپردازید، اگرتابه حال هیچ تجربه ای در رابطه با نوشتن کد S = S + 1 نداشته اید، احتمالا تصور خواهید کرد این که این مورد از لحاظ ریاضی کاملا اشتباه است! کافیست S را از دو طرف حذف کنیم تا به نتیجه ی ۰=۱ برسیم! اگر اینگونه است به قسمت بعد ، "خلاصه ای در مورد ساختار محاسبه ای کامپیوتر " رفته و آنجا را مطالعه کنید و برگردید. اگرچه در نگاه اول یک عبارت ریاضی و یک عبارت کد ، ممکن است ، یکسان به نظر برسند، اما این دو کاملا متفاوت از هم اند.

مسئله : محاسبه ی مجموع اعداد مثبت N۰۰۰ 1, 2, 3 تا N .

یک راه گنگ ! :

START N is an integer (fixed) and S is an integer (unknown)
S = 1
S = S + 2
S = S + 3
.
.
.
S = S + i (for the *i*'th statement)
.
.
.
S = S + N
PRINT(S)
END OF ALGORITHM

این شکل فلوچارت این راه حل گنگ است :



در واقع مسئله در این است که معلوم نیست چه تعداد از دستورات بدنه ی برنامه باید نوشته شوند. N عبارت ، که در صورتی که عددی بزرگ باشد ، نگارش این برنامه بسیار سخت می شود !!

یک راه حل بهتر :

می توانیم از یک حلقه ، استفاده کنیم:

START N is an integer (fixed), S is an integer that will contain the sum . At the start, S is initialized to 1, that is, its starting value will be 1. Another integer, i, is employed to contain the value that is to be added to S. It is initialized to 1.

Start of counting loop:

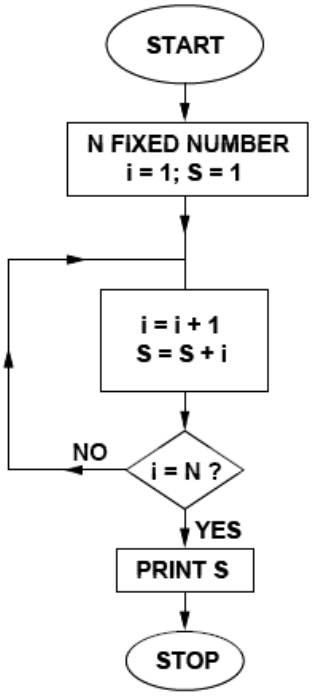
- i = i + 1 (get the new value for i)
- S = S + i (add it to S)
- **IF** i = N jump to **End of counting loop:**
- Jump to the first statement in this loop

End of counting loop:
PRINT(S)

END OF ALGORITHM

در رابطه با کد قبل توجه کنید که:

- عبارات کمتری برای حل مسئله ی یکسان.
- تعمیم کد به ازای مقادیر بزرگ برای N.
- حلقه ها نیز می توانند تو در تو بکار برده شوند.
- می توان همین الگوریتم را به صورت زیر نشان داد.



یک راه حل بهتر !

این راه یک راه ریاضی برای حل این مساله است که فرمول آن توسط گاوس[19]، یک دانش آموز دبیرستانی! کشف شده است.

$$S = \frac{N(N+1)}{2}$$

اگر N زوج باشد:

$$\begin{aligned} S &= [1,2,3,...,(N-2),(N-1),N] \\ &= [1,2,3,...,(N/2)] + [(N/2+1),(N/2+2),(N/2+2),...,N] \\ &= [1,2,3,...,(N/2)] + [N,(N-1),(N-2),...,(N/2+1)] \\ &= [(N+1),(N+1),(N+1),...,(N+1)] \quad (N/2 \ \text{terms}) \\ &= \frac{N}{2}(N+1) \end{aligned}$$

[19] Karl Friedrich Gauss (1777–1855 AD)

اثبات این فرمول به ازای اعداد فرد به عهده ی خودتان!

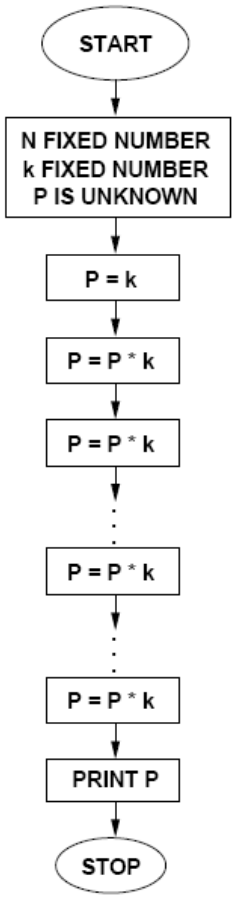
4.8.3 ساختار حلقه: حلقه ی تولید

مساله: محاسبه ی $P = k^N$

یک راه بسیار بسیار بی خود ؟!

START k is an integer or a real (fixed), N is an integer (fixed) and P is an integer or a real number (unknown)
P = k
P = P × k
P = P × k
.
.
.
P = P × k (for the i'th statement)
.
.
.
P = P × k (for the N'th statement)
PRINT(P)
END OF ALGORITHM

این هم فلوچارت این راه حل :



N عبارت ، یعنی تکرار بسیاری از موارد تکراری.

اما یک راه حل بهتر :

START k is an integer or a real (fixed), N is an integer (fixed), $P = 1$ is a number that will contain the product (initialized to 1) and i is an integer (initialized to 0) used as an "index" or "counter" to indicate how many times we have passed through the loop.

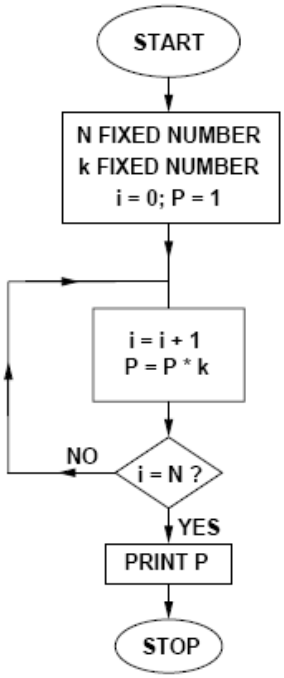
Start of loop:

- `i = i + 1` ("increment" the counter)
- `P = P` × `k` ("accumulate" the sum)
- **IF** `i = N` jump to **End of loop:**
- jump to **Start of loop:**

End of loop:
PRINT(P)

END OF ALGORITHM

این هم فلوچارت الگوریتم بالا :



همانطور که می بینید از تعداد جملات کمتری برای توصیف الگوریتم یکسان استفاده شده است.

اما یک راه خیلی **خیلی** بهتر! الکاشی:

- غیاث الدین جمشید مسعود
- متولد در کاشان (ایران) و متوفی در سمرقند(تاجیکستان)
- مدعی کشف کسر اعشاری
- محاسبه ی عدد π تا ۱۶ رقم اعشار
- تئوری اعداد و محاسبات
- نخستین کاشف بسط دو جمله ای
- مخترع دستگاه تخمین مسیر حرکت اقمار

START k is an integer or a real (fixed), N is an integer (fixed), $P = 1$ is a number that will contain the product (initialized to 1)

Start of loop:

IF (N is even)
    N = N/2
    k = k × k

ELSE (*i.e.* N is odd)
    N = N - 1
    P = P × k

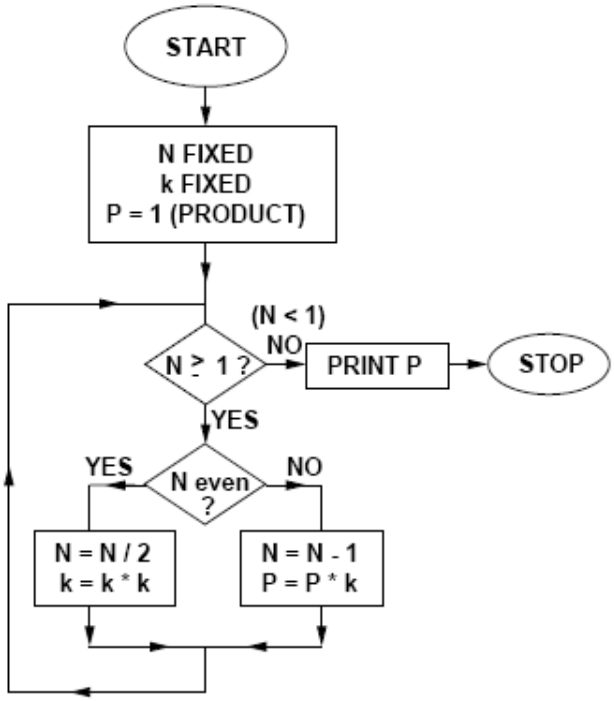
IF (N < 1) jump to End of loop:

ELSE (*i.e.* N ≥ 1) jump to Start of loop:

End of loop:
    PRINT(P)

END OF ALGORITHM

این هم فلوچارت الگوریتم الکاشی:



توجه کنید که مقادیر N و k در هر بار تکرار حلقه تغییر می کنند.

۹.۳ خلاصه ای در مورد ساختار محاسبه ای کامپیوتر

۱.۹.۳ عبارت **S = S + 1** به چه معنی است ؟

اگر شما نگاهی محاسبه گرانه و ریاضی وار داشته باشید ، عبارت قبل هیچ معنایی نخواهد داشت ! در واقع معادل با گفتن عبارت محال ۰=۱ است ! اما اگر مشغول یادگیری یک زبان برنامه نویسی هستید، باید قبل از این مطلب، کمی بیشتر با ساختار و معماری کامپیوتر آشنا شوید تا بدانید در درون کامپیوتر چه می گذرد.

حافظه [۲۰] قسمتی فیزیکی از رایانه است که واحد های حافظه ای[۲۱] در آن ذخیره می شوند.

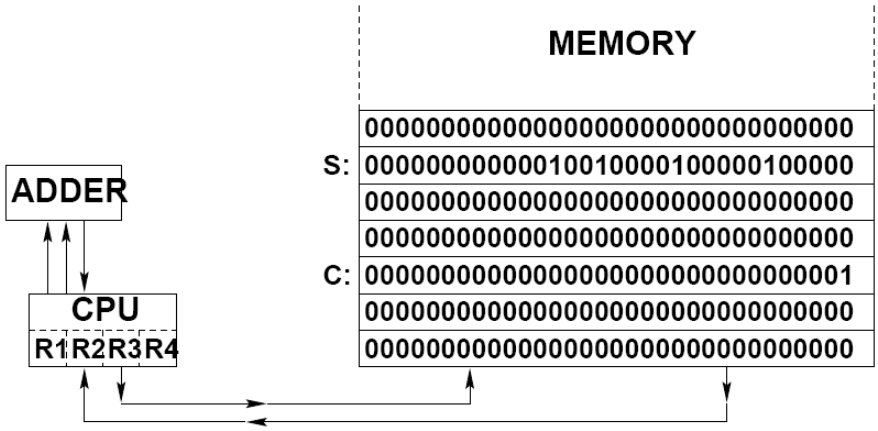
متغیر[۲۲] یک نماد برای تعریف تعداد مشخصی از واحد های حافظه ای (بیت ها) است که برنامه نویس می تواند آن را به دلخواه تعریف یا مقدار آن را تغییر دهد. برای مثال S (در این مثال) به آدرسی از مجموعه ای از واحد های حافظه ای تعلق دارد که قرار است مقدار این متغیر در آن خانه ها ذخیره شود.

ثابت[۲۳] یک نماد برای تعریف تعداد مشخصی از واحد های حافظه ای که برنامه نویس می تواند آنها را تعریف کند و از آن استفاده کند ولی حق تغییر آن را ندارد.

پردازشگر[۲۴] یا همان واحد پردازش مرکزی در رایانه ، در واقع "مدیر" عملیات اجرایی در رایانه ، مراحل مشخص شده را برای هر برنامه انجام می دهد و برای هر قسمت تعیین می کنید که چه چیز را و در چه هنگام باید انجام دهد.

رجیستر[۲۵] مکان بخصوصی در CPU که الگو های حافظه در آن قرار می گیرند .

جمع کننده [۲۶] یک واحد عملیاتی در کامپیوتر (که معمولا در سمت راست تراشه CPU قرار گرفته) که عملیات جمع(/تفریق) را روی بیت های ۳۲- بیتی انجام می دهد.



[20] Memory
[21] Bit
[22] Variable
[23] Constant
[24] Central Processing Unit
[25] Register
[26] Adder

در یک برنامه ی کامپیوتر عبارت S = S + 1 به صورت دستورات زیر ترجمه می یابد:

FETCH S,R1: واحد های حافظه ای را از آدرسی که S به آن اشاره می کند را به اولین خانه ی اول رجیستر CPU (R1) کپی کن.

FETCH C,R2: واحد های حافظه ای را از خانه ای که C به آنها اشاره می کند به دوم خانه ی رجیستر CPU (R2) کپی کن. ممکن است که قبلا مقدار ثابت (00000000000000000000000000000001) برای C تعیین شده باشد .

ADD R1,R2,R3: مقادیر ثابت ذخیره شده در خانه های R1 و R2 را به Adder کپی کن و بعد از انجام عملیات ریاضی ، نتیجه را به خانه ی سوم CPU (R3) کپی کن.

STORE R3,S: مقادیر ذخیره شده در خانه ی سوم حافظه ی سوم CPU را به آدرسی از واحد های حافظه که S به آنها اشاره می کند ، کپی می کند .

پس همانطور که دیدید، در اینجا عملیات افزایش ۱ واحد به مقدار یک متغیر را در چهار مرحله از کار CPU اجرا شد. در واقع عملیاتی که در بالا انجام شد دقیقا همان دستوراتی از زبان برنامه نویسی اسمبلی هستند که باید برای تفسیر S=S+1 انجام بگیرند. بنابرین S = S + 1 صرفا یک میانبر برای نشان دادن مجموعه دستوراتی که باید توسط کامپیوتر انجام گیرند، است.

۱۰.۳ مسائل

۱. سه گانگی در طراحی یک الگوریتم:

ویژگی ها و تواناییهای بسیار مهم برای ساختن یک الگوریتم کدامند؟

- توانایی معرفی متغیر ها
- توانایی مقدار دهی به متغیر ها
- توانایی انتقال خط فرمان از یک عملگر به عملگر دیگر به شیوه ای توصیه شده.
- راهکار ساختن تصمیماتی "درست" یا "اشتباه"
- تعریف متغیر هایی از نوع Integer
- توانایی انجام عملیات ریاضی (برای مثال +، − ، × یا ...)
- توانایی برگشت به دستورالعمل قبلی
- یک کامپایلر[۲۷] (یا مترجم) که بتواند یک متن به زبان انسانی را به زبان ماشین (مانند باینری) تبدیل کند.

۲. یک الگوریتم چیست ؟

در تقریبا ۳۵ کلمه و یا کمتر ، پاسخ دهید که "الگوریتم چیست؟".

۳. یک الگوریتم چیست ؟

سه ستون از برنامه نویسی عبارتند از : ۱)به ترتیب گذاری دستورات ۲) ساختار های شرطی ۳) ساختار حلقه ای

در ۲۵ واژه یا کمتر توضیح دهید که عبارت قبل چه مفهمومی دارد ؟

و :

به ترتیب گذاری دستورات یعنی چه؟ (۲۵ واژه یا کمتر)

ساختار های شرطی یعنی چه ؟(۲۵ واژه یا کمتر)

ساختار های حلقه ای یعنی چه ؟(۲۵ واژه یا کمتر)

۴. غذا هایتان را بر اساس الگوریتم بخورید !

در همین فصل الگوریتمی رو در قالب سودوکد مطالعه کردیم که مراحل تهیه ی غذا را به ترتیب و همراه با جزئیات نحوه ی تهیه ، بیان کرده بود. حال شما باید الگوریتم دیگری را بنویسید که خودتان دوست دارید! (در قالب سودوکد) این الگوریتم باید یک نقطه ی

[27] Compiler

شروع و یک نقطه ی پایان و بیش از پنج گام، بین دو نقطه ی شروع و پایان، حداقل دارای یک ساختار تصمیم گیری (یا تقسیم شاخه ها) و حداقل شامل یک ساختار گردشی باشد.

۵. در دهه ی هفتاد اگر شما مد روز نبودید ، .... بودید !

اینجا یک الگوریتم را در قالب سودوکد ملاحظه می کنید .

```
START
OUTPUT "Enter a positive integer"
INPUT N
i = 1
WHILE (i × i < N)
        i = i + 1
END WHILE
IF (i × i = N)
        OUTPUT "YES"
ELSE
        OUTPUT "NO"
END IF
STOP
```

به ازای کدام مورد یا مورد ها می تواند در خروجی "YES" چاپ شود؟

۲۸ (a)

36 (b)

54 (c)

۸۱(d)

جواب می تواند بیش از یک گزینه باشد .

الگوریتم چه نتیجه ای از داده های ورودی را محاسبه می کند ؟

۶. محاسبه مجموع مربع کامل ها

الگوریتمی را یا در قالب سودو کد و یا در قالب فلوچارت توصیف کنید که با در نظر گرفتن N که مقداریست که از ورودی دریافت می شود ، مقدار S را از فرمول :

$$S = 1^2 + 2^2 + 3^2 + \cdots + N^2$$

که مجموع اعداد مربع کامل از یک تا N^2 است ، را محاسبه کند. شما باید الگوریتم خود را در قالبی از ساختار گردشی بیان کنید. (در واقع از فرمول $S = N(N + 1)(2N + 1)/6$ که همین مجموع را حساب می کند ، نباید استفاده کنید.) دقت کنید که تمام متغیر هایی که استفاده می کنید را اول تعریف کنید .

۷. مثلث ها رو دوست دارید ؟!

عدد N زمانی عدد مثلثی خوانده می شود که شما بتوانید به وسیله ی N تعداد علامت ❋ ، یک مثلث متساوی الاضلاع بسازید . برای مثال در زیر تعدادی از اعداد مثلثی را به همراه شکل آنها مشاهده می کنید.

```
  *          *          *          *          *
 * *        * *        * *        * *        * *
 3         * * *      * * *      * * *      * * *
            6        * * * *    * * * *    * * * *
                      10       * * * * *  * * * * *
                                 15       * * * * * *
                                            21
```

در اینجا شما الگوریتمی را می بینید که سعی می کند تشخیص دهد اعداد ورودی مثلثی هستند یا نه .

اما صبر کنید !! این برنامه یک باگ دارد !!! ... و شما باید این باگ پیدا کنید .

```
START
OUTPUT "Enter a positive integer"
INPUT N
i = 1
S = 0
WHILE (S < N)
        i = i + 1
        S = S + i
END WHILE
IF (S = N)
        OUTPUT "Triangular"
ELSE
        OUTPUT "Not triangular"
END IF
STOP
```

۸. این مثلث را هم با گاوس حل کنید!

در این فصل با فرمول گاس آشنا شدیم که برای محاسبه ی عدد مثلثی n ام بکار می رفت .

$$1 + 2 + 3 + ... + N = \frac{N(N+1)}{2}$$

در اینجا با استفاده از این فرمول ، الگوریتم دیگری نوشته شده است تا بتواند اعداد مثلثی را از اعداد غیر مثلثی شناسایی کند .

آیا فکر می کنید که این الگوریتم کاملا درست است ؟! .... اگر نه ، اشکالش را بر طرف کنید .

START
OUTPUT "Enter a positive integer"
INPUT N
$i = 1$
WHILE $[(i \times (i+1))/2 < N]$
    $i = i + 1$
END WHILE
IF $[(i \times (i+1))/2 = N]$
    OUTPUT "Not triangular"
ELSE
    OUTPUT "Triangular"
END IF
STOP

۹. ضریب نوسان کننده

یک الگوریتم را در قالب سودوکد بنویسید که که عدد مثبت N را از ورودی دریافت کند و یک عدد ۱ (یک) را در خروجی چاپ کند . اگر عدد زوج باشد ، - ۱ (منفی یک) و اگر فرد باشد ، + ۱ (مثبت یک) چاپ کند. (راهنمایی: اگر عدد ۱- به تعداد زوج به خودش ضرب شود، حاصل ۱+، اگر عدد ۱- به تعداد فرد به خودش ضرب شود حاصل ۱- خواهد شد.)

۱۰. محاسبه ی حاصل یک سری با جملات نوسان کننده

یک الگوریتم را در قالب سدوکد و فلوچارت طراحی کنید که حاصل زیر را به دست آورد :

$$S = 1 - x + x^2 - x^3 + x^4 \cdots + (-x)^N$$

در صورتی که مقادیر x و N ورودی الگوریتم و S خروجی برنامه است . شما باید برای محاسبه ی حاصل از ساختار گردشی استفاده کنید.

۱۱. بیایید فاکتوریل زندگی را حساب کنیم !

فاکتوریل را در جاهای زیادی دیده اید... در ریاضی ، در محاسبات مهندسی ، و حتی در زندگی ! فاکتوریل با یگ علامت تعجب(!) نشان داده می شود. بنابرین عبارت "!3" را خواهیم خواند ، "سه فاکتوریل" .

محاسبه ی فاکتوریل بسیار آسان است :

1! = 1
2! = 2 × 1
3! = 3 × 2 × 1
4! = 4 × 3 × 2 × 1

و اگر N یک عدد صحیح مثبت باشد:

$$N! = N \times (N-1) \times (N-2) \cdots 2 \times 1.$$

توجه کنید که با توجه به همین تعریف داریم :

$$N! = N \times (N-1)!$$

شما باید یک الگوریتم را در قالب فلوچارت یا سودوکد بنویسید که مقدار فاکتوریل عدد دلخواه N (ورودی) را که بزرگتر از یک است را محاسبه کند. شما باید الگوریتم را به گونه ای توضیح دهید که گویا قرار است آن را بر روی یک کامپیوتر برنامه نویسی کنید، یعنی شما باید مجموعه دستورالعمل ها را به گونه ای آماده کنید که حتی یک ماشین هم آن را بفهمد. الگوریتم باید شامل دستورالعمل چگونگی شروع و پایان هم باشد. می توانید در نظر بگیرید که کامپیوتری که قرار است الگوریتم روی آن کارگذاشته شود می تواند متغیر ها و حلقه ها را ذخیره کند، و شاخه شدن و اعمال ریاضی ساده را انجام دهد.

۱۲. تباهی و زوال!

شما باید الگوریتیمی را در قالب سودوکد یا فلوچارت طراحی کنید که مقدار تقریبی تابع نمایی e^{-x} را محاسبه کند. می توان حاصل این مقدار را از سری زیر به دست آورد . بدیهی است که هر چه کسر های بیشتری از سری محاسبه شود، حاصل دقیقتر خواهد بود:

$$S = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

در صورتی که x و N اعدادی هستند که از ورودی دریافت می شوند و S خروجی الگوریتم خواهد بود . همچنین به استفاده از فاکتوریل در بالا توجه کنید .

راهنمایی: پر بازده ترین راه حل برای حل این مساله ، به این صورت است که حاصل را به صورت زیر بازنویسی کنید :

$$S = s_0 + s_1 + s_3 + s_4 + \ldots$$

در صورتی که $s_0 = 1$ و :

$$s_n = \left(\frac{-x}{n}\right) s_{n-1}$$

۱۳. سینوس شما چند می شود ؟!

در این الگوریتم شما باید الگوریتم برنامه ای رو بنویسید که بتواند مقدار سینوس عدد دلخواهی را حساب کند .

حاصل زیر تقریب تقریبا خوبی برای محاسبه ی سینوس یک عدد دلخواه دارد، در صورتی که حداقل ۵ عبارت کسری از طرف راست سری زیر محاسبه شود:

$$S = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

مقادیر x و تعداد کسر هایی که در این سری جمع خواهند شد، ورودی های الگوریتم و مقدار S خروجی سری خواهد بود. در سری فوق به استفاده از فاکتوریل توجه کنید.

الگوریتم خود را در قالب سودوکد یا فلوچارت بیان کنید .

۱۱.۳ پروژه ها

توپ متحرک را دنبال کن !

شما در این پروژه باید الگوریتمی را در قالب سودو کد یا فلوچارت بنویسید، بطوریکه؛ در این الگوریتم شروع و پایان و ترتیب اجرای مراحل الگوریتم کاملا مشخص باشد، هدف الگوریتم معلوم باشد. توجه داشته باشید که الگوریتم شما بر روی ماشینی اجرا می شود که توانایی اجرای ساختار های حلقه ای و گردشی و تعریف و مقدار دهی متغیر ها را داراست.

مساله، محاسبه مقدارکل مسافتی است که یک توپ که از ارتفاع h رها می شود و در بار های متوالی به اندازه های کسری از h بالا/پایین می رود. برای مثال در اولین برخورد، مقدار ارتفاعی که توپ بالاتر می رود برابر است با hc که c ضریب ارتجاع و دارای مقداری برابر $0 < c < 1$ است. بنابرین تا ابتدای برخورد ال توپ به زمین ، کل مسافت طی شده $h + 2ch$ شده است. همچنین در دومین برخورد توپ به زمین ، به اندازه ی c^2h بالا می رود و الی آخر.

یک راه ریاضی برای حل این مساله با استفاده از تصاعد حسابی وجود دارد که می توان جواب مساله را از فرمول زیر حساب کرد:

$$d = h\frac{1+c}{1-c}$$

اما شما نباید از این فرمول در الگوریتم خود استفاده کنید. بلکه با استفاده از محاسبه ی حاصل تک تک عبارات از سری مجموع، حاصل را به دست آورید.

اما اینجا مساله ای وجود دارد که نیازمند کمی تامل است. اگرنتیجه را از روی فرمول ریاضی که از بالا به دست آمده حل کنیم، نتیجه مقداری محدود خواهد بود. در صورتی که تعداد واقعی پرش هایی که توپ انجام خواهد داد، مقداریست نامحدود !

هر چند که پس از آنکه ارتفاع پرش توپ مقدار ناچیزی شد، صرف نظر از آنها، مشکل چندانی بر روند محاسبه ی مجموع ارتفاع ها ایجاد نمی کند. اما تا جایی که شما محدوده ای برای الگوریتم خود تعیین نکنید، کامپیوتر، مجموع ارتفاع ها را حتی اگر به مقدار بسیار کوچک باشند، محاسبه خواهد کرد. پس در نتیجه این محاسبه ی رایانه تا بی نهایت ادامه خواهد داشت. پس باید شما یک محدوده را به طور مصنوعی برای محاسبات رایانه تعیین کنید.

فصل چهارم

شروع به برنامه نویسی به زبان C++

۱.۴ نمونه از دستورات ورودی و خروجی: اولین برنامه به زبان C++

شما در اینجا اولین برنامه به زبان C++ را مشاهده می کنید :

```
//File: goBlue.cpp

#include <iostream>

using namespace std;

int main(void)
{
   cout << "Go Blue!\n";
   // Multiple line statement
   cout <<
   "\n\n\t\"Go\t\t\\\n\t\t\t\tBlue!\"\a\a\a\r";
   return(0);
}
```

توجه کنید که همه ی برنامه ی C++ ویژگی هایی مشترکی دارند.

کامنت ها[28] که به دو شیوه نمایش داده می شوند:

- بوسیله ی /* و */ و متن کامنت در بین دو علامت قرار می گیرد ؛ برای مثال :

```
/* This is a comment */
```

- بوسیله ی علامت // که متن کامنت بعد از آن قرار می گیرد؛ برای مثال :

```
// This is a comment
```

این شیوه معمولا زمانی به کار می رود که شما تنها قصد دارید، یک سطر کامنت بنویسید.

در واقع کامنت ها تنها برای افراد انسانی قرار داده می شوند و هیچ تاثیری بر روند پردازش رایانه ندارند. رایانه اهمیت نمی دهد که شما در برنامه ی خود کامنت به کار می برید یا نه، اما انسان ها چرا، بخصوص برای مسئول آزمایشگاه رایانه ی شما، کسی که بر اساس برنامه ی شما به شما نمره و یا حتی پول می دهد!

کامنت ها باید آنچه را که می خواهید اجرا کنید و روشی را که در اجرای آن بکار می برید، توضیح دهند. توضیحات شما در کامنت ها باید به گونه ای کامل باشند، گویا می خواهید برنامه ی خود را برای یک فرد احمق توضیح دهید.

درست زمانی که شما فراموش کرده اید که چرا و چگونه یک کد را نوشته اید، کامنت هایی که در زمان نگارش برنامه نوشته اید، به داد شما خواهند رسید. هر برنامه یا قطعه برنامه ، باید با چندین خط کامنت توضیح در مورد برنامه، همراه با معرفی نویسنده ی برنامه و

[28] Comments

تاریخی که برنامه نگارش یافته است، شروع شود. در واقع هر جا که احساس کردید که اگر یک فرد احمق برنامه ی شما را مطالعه کند، متوجه مفهوم آن نمی شود، باید کامنت قرار دهید . اما مواظب باشید در کامنت گذاری زیاده روی نکنید!

دستورات پیش پردازنده[۲۹] : دستورات پیش پردازنده ، دستوراتی هستند که کامپایلر را در جهت تلفیق و گرد آوری سورس ها، هدایت می کنند. دستورات پیش پردازنده توسط علامت "#" شروع می شوند. برای مثال دستور پیش پردازنده ی **#include** <iostream> ، به کامپایلر دستور می دهد که کد های داخل فایل با استاندارد ANSI [۳۰] را به عنوان یک هدر فایل[۳۱] ،ضمیمه ی کد های برنامه کند. این فایل، خود شامل تعدادی دیگر از دستورات پیش پردازنده به همراه #include می باشد ، که خود آنها نیز سایر کتابخانه ها را فراخوانی می کنند. در واقع این دستورات، انواع قالب های استاندارد در زبان C++ را برای دستورات ورودی و خروجی معرفی می کنند.

فایل های مختلفی به عنوان هدر فایل مورد استفاده قرار می گیرند که در ادامه ی کار به تدریج با آنها آشنا خواهید شد. مانند کتابخانه ی cmath که با دستور پیش پردازنده ی #include <cmath> توانایی استفاده از توابع ریاضی بسیار زیادی را به کاربر می دهد. اما دستورات پیش پردازنده انواع دیگری نیز دارند. مانند "ثابت ها[۳۲]" و "ماکرو ها[۳۳]" که در آینده با آنها آشنا خواهیم شد.

اگر عجله دارید با این فایل ها آشنا شوید، به آدرس زیر سری بزنید: (البته در سیستم های یونیکس!)

/usr/include/g++-3

: using namespace std

وجود این خط به کامپایلر می گوید که ما در برنامه ی خود از استاندارد "namespace" استفاده خواهیم کرد و آن را به صورت زیر بکار می بریم :

```
using namespace std;
```

در حالت کلی می توان گفت استفاده از این عبارت نشان دهنده ی این است که ما قصد داریم از فایل ها و توابع کتابخانه ی استاندارد C++ استفاده کنیم. همچنین این امکان پذیر است که خود شما استاندارد جدیدی را برای خود تعریف کنید. اما حواستان باشد، نام هایی که شما برای توابع و فایل های استاندارد خود تعریف می کنید، با نام های کتابخانه ی استاندارد C++ یا با نام های تعریف شده توسط یکی دیگر، تداخلی نداشته باشد. ما در این دوره تنها قصد کار روی توابع کتابخانه ی استاندارد را داریم. اما اگر شما بخواهید روزی برنامه نویسی را به صورت پیشرفته ادامه دهید، باید بتوانید که برای کار خود کتابخانه تعریف کرده و از namespace خودتان استفاده کنید.

```
int main(void){};
```

[۲۹] Preprocessor directives
[30] American National Standard Institute
[۳۱] Header file
[32] symbolic constants
[33] macros

شروع اجرای هر برنامه ی C++ با اجرای اولین دستورات و ادامه آنها از تابع main ، انجام می شوند. علامت "()" در مقابل نام تابع نشان دهنده ی آن است که main بلوکی از برنامه است که تابع[۳۴] نامیده میشود.

علامت int قبل از نام تابع نشان می دهد که این تابع باید یک مقدار از نوع **عدد صحیح**[۳۵] را به کاربر برگرداند.

بلوک main قسمتی از برنامه است که کامپایلر آن را به عنوان نقطه ی شروع اجرا شناسایی می کند و اجرای برنامه را از آن آغاز می کند. خطوط بین "{ }" را " بدنه ی تابع[۳۶] " می نامیم. این خود تمرین خوبی است که موقعی که در حال نگارش بدنه ی یک تابع هستید، با قرار دادن فاصله های معین، برنامه ی خود را " پله ای[۳۷]" یا دندانه ای ، بنویسید. این فاصله برای شروع هر خط، هر مقداری می تواند باشد ولی فاصله ی توصیه شده، به اندازه ی سه **کاراکتر** فاصله است.

در مثال ما، بدنه ی اصلی ما با عبارت زیر شروع می شود:

```
cout << "Go Blue!\n";
```

عبارت بالا با فراخوانی تابع cout از کتابخانه ی استاندارد C++ دستور می دهد که مجموعه ای از رشته ها (عبارت داخل دو علامت دبل کوتیشن) در خروجی چاپ و سپس مکان نما به آغاز سطر بعد منتقل شود. این تغییر مکان ناشی از وجود عملگر "\n" در بین دو دبل کوتیشن و در انتهای عبارت ماست. این نوع عملگر ها که " عملگر های گریز[۳۸] " نامیده می شوند، درمواقعی بکار می روند که بخواهیم کاراکتری خاص مانند " را چاپ کنیم یا بدون استفاده از کاراکتر اضافه، کاری مانند سر سطر را انجام دهیم.

در اینجا شما لیستی از معروفترین عملگر های گریز را مشاهده می کنید:

\n	شروع سطر جدید و انتقال مکان نما به سطر بعدی (next line)
\t	حرکت افقی، به اندازه ی یک tab (horizontal tab)
\r	بازگرداندن مکان نما به ابتدای سطر جاری (return)
\a	زنگ هشدار (alert)
\\	چاپ یک بک اسلش (\)
\"	چاپ یک دابل کوتیشن (")

یک مثال کلی که کاربرد عملگر های بالا را نشان می دهد:

```
cout <<
"\n\n\t\"Go\t\t\\\n\t\t\t\t Blue!\"\a\a\a\r";
```

- که ابتدا دو سطر پایین می آید.
- سپس به فاصله ی یک tab به جلو حرکت می کند.
- یک علامت دبل کوتیشن به همراه Go چاپ میکند

[34] Function
[35] Integer
[36] Function body
[37] Idention
[38] escape sequence

- سپس به اندازه ی دو tab رو به جلو حرکت می کند
- بعد از چاپ یک علامت بک اسلش \ به سطر بعد حرکت می کند و به اندازه ی فاصله ی چهار tab به سمت جلو حرکت می کند
- یک علامت دبل کوتیشن " چاپ می کند و سه بار بوق می زند.(بسته به سیستم آلارم کامپیوتر شما ، ممکن است این سه بوق پیوسته یا جدا به نظر برسند!)
- به ابتدای سطر باز می گردد.

اینجا شما شبیه آنچه را که قرار است در خروجی چاپ شود را می بینید:

```
"Go          \
      Blue!"
```

توجه کنید که عبارتی که با cout شده است با علامت سمی کولن (;) به پایان رسیده است. سمی کولن در زبان C++ یک علامت خاص است! در واقع این علامت پایان یک دستور را نشان می دهد. بنابرین حتی چند دستور نیز می توانند در یک سطر خاص قرار بگیرند و حتی برعکس این حالت؛ قرار گیری یک دستور در چندین سطر قرار گیرد، نیز، امکانپذیر است.

```
return(0);
```

زمانی که کامپایلر با این خط از برنامه مواجه می شود، دستور return مقداری را به عنوان خروجی تابع (در اینجا تابع main) بر می گرداند و اجرای تابع به پایان می رسد. از آنجایی که نوع داده ی خروجی در ابتدای برنامه از نوع integer معرفی شده بود، مقدار خروجی نیز باید حتما از این نوع باشد. در غیر اینصورت شما باید انتظار خطا[39] از سوی کامپایلر را داشته باشید.

بنابرین با این توضیحات، شما حتی می توانید عدد ۱ را به جای صفر به عنوان خروجی تابع انتخاب کنید.

```
return(1);
```

توجه کنید که شما در برنامه ی خود می توانید دستور return را در چندین جای برنامه ی خود به عنوان نقطه ی ایست برنامه ، استفاده کنید.

۲.۴ کامپایل سورس، اتصال فایل ها، لود برنامه و اجرای آن

تا اینجا ما سورس یک برنامه را نوشته ایم ! حال باید به کاربر بگوییم که با این سورس چه کار کند ! در واقع مسئله این است که کامپیوتر فقط صرفا یک سری اعداد باینری را درک می کند اما برنامه ی نوشته شده توسط ما، یک سری کاراکتر است !

برای کامپایل برنامه، اگر فرض کنیم برنامه را با نام goBlue.cpp ذخیره کرده ایم، دستورات زیر باعث کامپایل یک برنامه بر روی یک سیستم یونیکس خواهند شد: (این دستورات را در محیط فرمان یونیکس وارد کنید)

```
g++ goBlue.cpp
```

در واقع یک *کامپایلر*[40] نرم افزاری است که مجموعه کاراکترهای نوشته شده بر اساس قاعده را به زبانی قابل فهم برای کامپیوتر، تبدیل می کند. کامپیلر ها در طول زمان تغییرات زیادی کرده اند. در کامپیوتر های اولیه (مربوط به دهه ی ۴۰ میلادی و بعد از آن) برنامه ها

[39] Error

خود مستقیما به صورت باینری نوشته می شدند که در واقع کاری بسیار پیشرفته و طاقت فرسا بود. بعد ها، برنامه نویسان متوجه شدند که می توان برنامه را در قالبی قابل درک برای انسان بنویسند و سپس برنامه ای به نام کامپایلر، این کد ها را تبدیل به زبان ماشین کند.

آغاز زبان برنامه نویسی C++ به اواخر دهه ی ۶۰ و اوایل دهه ی ۷۰ و به زبان برنامه ی نویسی C برمی گردد. نیت اصلی برای طراحی زبان C ، طراحی زبانی برای نگارش سیستم عامل[۴۱] ها بود. سیستم عامل نرم افزاری است که به کاربر این اجازه را می دهد تا با ارتباط موثر با داده های موجود در حافظه ی رایانه سایر قسمت ها و عملگر های موجود، بیشترین استفاده را از ابزار موجود ببرد. بعدها در دهه های ۸۰ و ۹۰ ، زبان C++ به عنوان بهبود یافته ی زبان C ، طراحی شد. (در واقع از این منظر که تمام کامپایلرهای C++، زبان C را نیز پشتیبانی می کنند.) تعداد تغییرات ایجاد شده برای بدست آوردن ساختار C++ قابل ملاحظه اند که در ادامه به شرح کامل آنها خواهیم پرداخت.

در طراحی زبان برنامه نویسی C++ تا آنجا که امکان دارد، سعی شده است این زبان مستقل از سخت افزار باشد. یعنی اینکه یک برنامه نوشته شده به زبان C++، بتواند بر روی رایانه ای با مشخصات ساختاری دیگر اجرا شود. زبان اسمبلی ، وابسته به ماشین است. این زبان همواره سخت افزار یا جزئی از آن را مورد خطاب می دهد. بنابراین زبان برنامه نویسی سطح بالای C++ زبانی است که به طور صریح و ضمنی با سخت افزار ارتباط برقرار می کند.

زبان های C و C++ به همراه ویژگی هایی که برای آنها ذکر شد، به عنوان زبان هایی قدرتمند برای طراحی سیستم عامل نیز طراحی شده اند. در واقع این دو، زبان هایی برای اجرای بسیاری از عملیات محاسباتی هستند. شاید C و C++ دو زبان محبوب و پرکاربرد در سراسر دنیا باشند، اما در دنیا، زبان های محبوب دیگری نیز وجود دارند که برای اهداف خاصی تهیه شده اند. مثلا برخی برای اهدافی مانند مسائل محاسباتی طراحی شده اند. اما به نظر می رسد، زبان C++ بتواند به تنهایی تا حدودی ، کار این زبان ها را انجام دهد. به همین خاطر است که می گوییم این زبان، تقریبا یک زبان همه کار و همه فن حریف است!

بعد از اینکه که طراحی یک الگوریتم از لحاظ فکری، سودوکد و فلوچارت، به اتمام رسید، نوبت به نگارش برنامه به زبان C++ (یا زبانی دیگر) می رسد.

در حین اجرای یک برنامه ی C++، چندین مرحله انجام می شود:

- نگارش: در این مرحله، کاربر، کد برنامه را می نویسد. (برای مثال `goBlue.cpp`) این برنامه در یک محیط ویرایشگر نوشته می شود. از جمله ویرایشگر های محبوب در محیط یونیکس، می توان vi، emacs ، pico ، nedit ، kwrite ، Eclips و یا Kdevelop را نام برد. احتمالا ، استاد شما می تواند شما را در رابطه با انتخاب یکی از این ویرایشکر ها و ویژگی های هرکدام از آنها، راهنمایی کند تا اینکه بتوانید تا مدت ها برنامه های مورد نظر خود را، در این محیط بنویسید.

[40] Compiler
[41] Operating System

- **پیش پردازش**[42] **و پیش کامپایل**[43]: دستوراتی مانند `#include` و `#define` در ابتدای برنامه، به نوعی اولین خط مشی را برای کامپایلر نشان می دهند. این کار می تواند شامل اضافه کردن یک " فایل کد" دیگر، به کد های برنامه و یا جایگزینی قسمتی از کد آن باشد.
- **کامپایل**: بعد از مرحله ی پیش پردازنده، کامپایلر، کد برنامه را تبدیل به *"object code"* یا *" binary code"*. می کند. در واقع این مرحله خود شامل چندین مرحله است. اولین مرحله، مرحله تجزیه ی [44] قسمت های مختلف کد است که در این مرحله ، ابتدا کامپایلر برنامه ی شما را از لحاظ درستی قواعد نگارش[45]، بررسی می کند. اگر این مرحله با موفقیت طی شود، مرحله ی بعد، تهیه ی کد اسمبلی از کد C++ شما ، که از لحاظ قواعد نگارش، عاری از عیب و ایراد است. سپس کد اسمبلی تبدیل به فایل باینری یا Object file می شود.
- **اتصال فایل ها**[46]: بعد از اتمام مرحله ی ساخت فایل های باینری، مرحله ی اتصال این فایل ها شروع می شود. در این قسمت تمامی توابعی که در برنامه ی C++ به آنها ارجاع داده شده اند، گرد آوری شوند. اگر این قسمت های ارجاع داده شده، به شکل فایل باینری، همگی به یک مکان مشخص گردآوری شوند، این شیوه را اتصال ایستا یا غیر پویا [47]می نامیم. ولی اگر فقط آدرس مکان این توابع گردآوری شود و در هنگام نیاز، اجرای برنامه با مراجعه به آدرس این فایل ها، از سر گرفته شود. این حالت را نیز، اتصال پویا [48] می نامیم. فایلی که توسط لینکر ایجاد می شود را ماژول لود[49] می نامیم. به طور پیش فرض، سیستم یونیکس، نام این فایل را a.out قرار می دهد که نام می تواند توسط کاربر عوض شود.

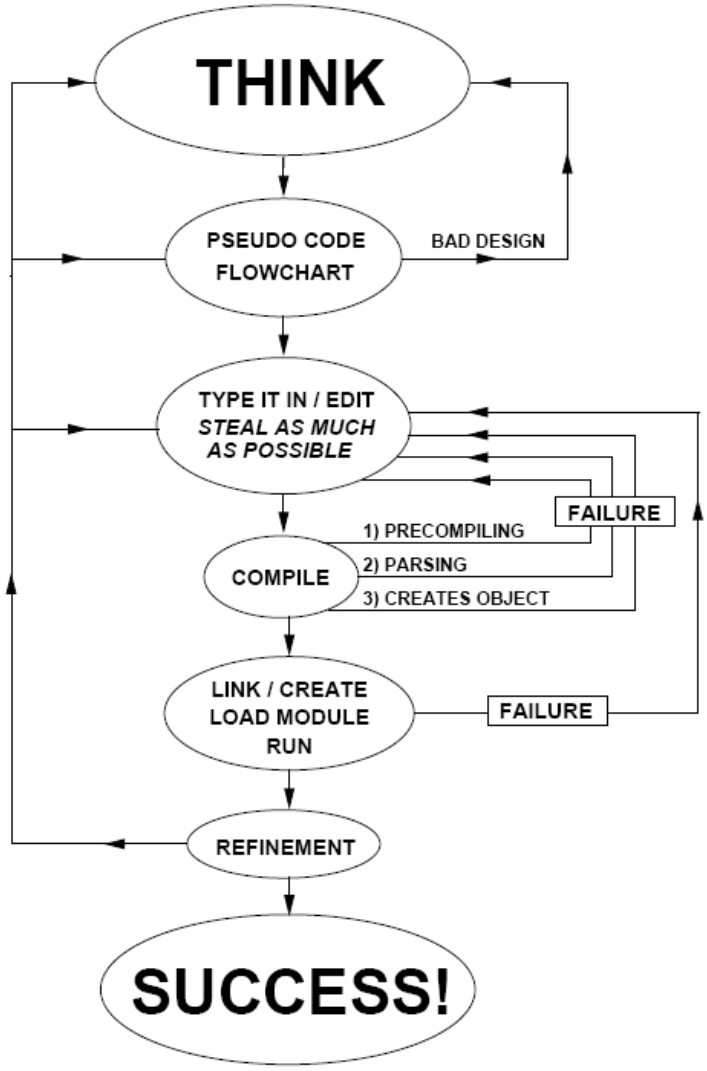
- **بازگذاری**[50]: بارگذارنده یا لودر[51]، بعد از عملیات اتصال فایل ها، ماژول لود را در فضایی قرار می دهد که سیستم عامل انتظار دارد، تمامی فایل های اجرایی را از آن قسمت اجرا کند. در سیستم عامل یونیکس، واحد لودر زمانی شروع به کار می کند که کاربر نام ماژول برنامه ی مورد نظر را تایپ کند.
- **اجرا**[52]: در این مرحله، فایل باینری برنامه، به واحد پردازش مرکزی [53](CPU) منتقل می شود. ماژول لود، دارای اطلاعاتی است که پرداشگر را راهنمایی می کند تا عملیات لازم را برای اجرای، به قسمت های مختلف رایانه محول کند. همچنین ماژول لود دارای اطلاعاتی است یا اینکه آدرس اطلاعاتی را دارد که پردازش برنامه بر اساس آنها انجام می شود. قسمت اجرا، دقیقا همان جایی است که قرار است با مجموعه ای از خطا های منطقی، روبرو شویم. در واقع تمامی خطاهای نگارشی در کد برنامه ، در مرحله ی تجزیه ی برنامه توسط compiler parser مشخص شده است. بنابراین یافتن خطاهای نگارشی، کار چندان سختی نیست. اما خطایابی

42 Preprocess
43 precompile
44 parsing stage
45 Syntax
46 Link
47 Static Linking
48 Dynamic Linking
49 Load Module
50 Load
51 Loader
52 Run
53 central processing unit

برنامه ای که از نظر قواعد نگارشی درست است اما از لحاظ منطقی دارای طراحی درستی نیست، می تواند کاری بسیار دشوار باشد. اینجاست که فلوچارت و سودوکد می توانند به کمک برنامه نویس بیایند.

مراحل ذکر شده برای ویرایش، پیش پردازش، تجزیه ی کد، تولید فایل باینری از کد، اتصال فایل ها، لود کردن و اجرای برنامه، توسط تمامی محیط های توسعه ی کد، صرف نظر از پیچدگی برنامه، همواره اجرا می شوند. حال این برنامه می تواند یک برنامه ی ساده مانند "Go Blue!" باشد یا اینکه یک برنامه ی بسیار پیشرفته مانند مدیریت سیستم تلفن کشور، مدیریت سیستم بانک بین المللی و یا سیستم های امنیت جهانی باشد.

شکل زیر، نه تنها مراحل کامپایل یک برنامه را نشان می دهد، بلکه تمامی پروسه ی نگارش یک برنامه توسط فرد را نشان می دهد. این مراحل نیز دارای حالت کلی هستند و مستقل از زبانی اند که برنامه به آن زبان نگارش می شوند.



۳.۴ معرفی و مقدار دهی متغیر ها

برنامه ی زیر که عملیات مختلف ریاضی را بین دو عدد صحیح انجام می دهد را در نظر بگیرید:

```
//File: integerMath.cpp

#include <iostream>
using namespace std;
int main(void)
{
   int i1 = 5; // Define and initialize
   int i2 = 2; // Define and initialize
   int iResult; // Define only
   iResult = i1 + i2; // sum
   cout << "5 + 2 = " << iResult << "\n";
   iResult = i1 - i2; // difference
   cout << "5 - 2 = " << iResult << "\n";
   iResult = i1 * i2; // product
   cout << "5 * 2 = " << iResult << "\n";
   iResult = i1 / i2; // division
   cout << "5 / 2 = " << iResult << "\n";
   iResult = i1 % i2; // remainder
   cout << "5 mod 2 is " << iResult << "\n";
   return(0);
}
```

در اینجا چندین ویژگی جدید وجود دارند که نیاز به توضیح دارند :

ساختار int i1 = 5; را ، عبارت معرفی[۵۴] می نامیم. نام i1 یک عنوان برای متغیر i1 است ! این ساختار بیان می کند که محتوای i1 یک متغیر از نوع عدد صحیح (به صورت پیش فرض، منظور تمامی اعداد صحیح علامت دار[۵۵] است) می باشد. همچنین در اینجا برای متغیر i1 یک مقدار اولیه (همان مقدار ۵) تعیین شده است. عنوان متغیر، مجموعه ای از کاراکتر ها شامل حروف الفبا، اعداد و کاراکتر های تاکید(_) می تواند باشد. استفاده از علامت تاکید"_" در ابتدای نام متغیر ها، می تواند مشکل سازشود! زیرا که نام اکثر متغیر های کتابخانه ی C++ با _ شروع می شود. نام یک متغیر هرگز با یک عدد شروع نمی شود. بر اساس استاندارد ANSI استفاده از هر تعداد کاراکتر به عنوان نام متغیر مجاز است. استاندارد ANSI C++ بر اساس ANSI C ساخته شده و در استاندارد ANSI C ، تنها ۳۱ کاراکتر اول برای نام متغیر، برای کامپایلر هایی که بر اساس این استاندارد کار می کردند، مهم بود. البته باید گفت که نویسندگان کامپایلر های مختلف C++ ، ممکن است مقادیر دیگری را به عنوان تعداد کاراکتر های با اهمیت تعیین کرده باشند. حال توصیه می شود شما نیز جهت احتیاط هرگز بیش از ۳۱ کاراکتر در نام متغیر خود استفاده نکنید. توجه کنید که کامپایلر C++ ، به بزرگی و کوچکی حروف شما اهمیت می دهد. برای مثال i1 با I1 متفاوت است. اگر چه برنامه ی شما بسیار کوتاه باشد، شما باید متغیر های خود را کاملا واضح تعریف کنید. در عین حال لازم است حتما در برنامه ی خود تعدادی خط کامنت به کار ببرید. همچنین می توانید با ایجاد تغییر در نحوه ی تعریف متغیر ها، آنها را خواناتر کنید. برای مثال با کوهان دار نوشتن[۵۶] نام متغیر؛ مثلا استفاده از numberOfStudents به جای number_of_students متن برنامه تان را خواناتر و واضح تر کنید. توجه کنید که این نوع نامگذاری با حرف کوچک و کلمات جدید در داخل نام متغیر با حرف بزرگ آغاز می شود.

54 declaration statement
55 Signed integer
56 Humpback notation

حال بیایید مرور کنیم !

```
int i2 = 2;
```

در عبارت قبل قسمتی از حافظه را از نوع عدد صحیح، تعیین کردیم و نام i2 را به آن نسبت دادیم و سپس مقدار ۲ را در آن خانه از حافظه، ذخیره کردیم.

```
int iResult;
```

در عبارت قبل ، به متغیری از حافظه که قرار است مقداری صحیح و علامت دار در آن، قرار گیرد نام iResult را نسبت دادیم. اما هیچ مقداری به آن خانه نسبت داده نشده است.

۴.۴ محاسبات اعداد صحیح در C++

به عبارت زیر توجه کنید :

```
iResult = i1 + i2;
```

در این عبارت دو عملگر استفاده شده است : + و = . عملگر = ، عملگر مقداردهی نامیده می شود. ***بنابرین هرگز به آن علامت مساوی نگویید!!*** علامت مقدار دهی ، نتیجه ی یک یا چندین عملیات ریاضی در سمت راست این علامت را در متغیر سمت چپ قرار می دهد. عملگر = ، را یک عملگر باینری می گوییم، زیرا را همواره در حال کار با دو جزء است: جزئی که مقدار، را از آن می گیرد و جزئی که مقدار گرفته شده را در آن قرار می دهد.

عمگر + ، عملگر جمع می نامیم. عملگر جمع نیز خود یک نوع عملگر باینری است. زیرا که با دو جزء، کار می کند.

در اینجا باید گفت که چهار عملگر دیگر نیز وجود دارند:

- عملگر تفریق : –
- عملگر ضرب: *
- عملگر تقسیم: /
- عملگر باقیمانده[۵۷]: %

در رابطه با این عملگر ها باید توجه کرد استفاده از هر کدام از آنها، نیازمند توضیحاتی است.

عملگر تقسیم در تقسیم دو عدد صحیح ، همواره مقداری صحیح بازخواهد گرداند. برای مثال حاصل 5/2 برابر ۲ است نه ۲.۵ . عملگر باقیمانده ی می تواند مقدار باقیمانده ی حاصل را از این تقسیم را به شما برگرداند. بنابرین حاصل ۲ ٪ ۵ برابر یک خواهد بود.

در استفاده از چندین عملگر در کنار هم باید به اولویت عملگر ها[۵۸] توجه کنید. برای مثال * ، / و ٪ اولویت بیشتری نسبت به عملگر ها + و – دارند. عملگرهای / و ٪ دارای اولویت یکسانی هستند. و همچنین عملگرهای + و – دارای اولویت یکسانی اند. برای یک سری از عملگرها با اولویت یکسان، ترتیب اجرای محاسبات از سمت چپ به راست است.

برای مثال :

[57] Modulus
[58] Rules of precedence

```
1 + 10 * 5 - 9 / 3 = 1 + 50 - 3 = 48
```

اگر برنامه نویس بخواهد تغییری در ترتیب اجرای عملگر ها ایجاد کند، می تواند از پرانتزها استفاده کند. زیرا اولویت محاسبات پرانتز ها، از تمام عملگر ها بیشتر است. توجه کنید که ابتدا پرانتز های داخلی تر محاسبه خواهند شد.

برای مثال:

```
(1 + (10 * 5) - (9 / 3)) = 1 + 50 - 3 = 48
```

در عبارت محاسبه ای بالا، پارانتز ها تغییری در اولویت عملگر ها ایجاد نکرده اند، اما نحوه ی محاسبه ی عبارت زیر متفاوت است :

```
((1 + 10) * (5 - 9)) / 3 = (11 * (-4))/3 = -44/3 = -14
```

خلاصه ی اولویت عمللگر ها را در جدول زیر مشاهده می کنید:

اولویت	علامت	عملگر	توضیحات
۰	()	پرانتز ها	پرانتز ها داخلی اولویت محاسبه ای بیشتری دارند.
۱	(مجموعه از عملگر ها) -	قرینه	R→L
۲	* / %	ضرب ، تقسیم ، باقیمانده	L→R
۳	+ -	جمع ، تفریق	L→R

در عبارت زیر :

```
cout << "5 + 2 = " << iResult << "\n";
```

در خروجی رشته ی " 5 + 2 = " را به همراه iResult ، چاپ می کند. سپس \n ، مکان نما را به سطر بعدی منتقل می کند.

باید گفت سبک های مختلفی برای چاپ یک مقدار در خروجی وجود دارد. می توانیم همان متن را توسط عملگر << در چندین سطر بنویسیم.

کد زیر همان مقداری که کد بالا در خروجی نشان می دهد را، چاپ خواهد کرد:

```
cout << "5 + 2 = ";
cout << iResult;
cout << "\n";
```

اینجاست که باید به تفاوت ایندو شیوه ، در پاکیزگی و آراستگی کد، همچنین راحتی کد خوانی توجه کرد.

۵.۴ محاسبات اعداد اعشاری در C++

برنامه زیر را که محاسبات مختلفی را بر روی دو عدد اعشاری انجام می دهد، مشاهده می کنید:

```
//File: floatMath.cpp
#include <iostream>
using namespace std;
int main(void)
{
   cout << "Input the 1st floating point number: ";
   float f1; // Declare the 1st float
   cin >> f1; //and read it in
   cout << "Input the 2nd floating point number: ";
   float f2;
   cin >> f2;
   float fResult = f1 + f2; // sum
   cout << f1 << " + " << f2 << " = " << fResult << "\n";
   fResult = f1 - f2; // difference
   cout << f1 << " - " << f2 << " = " << fResult << "\n";
   fResult = f1 * f2; // product
   cout << f1 << " * " << f2 << " = " << fResult << "\n";
   if (0 == f2)
   { cout << "Division by zero not permitted!\n";
      return(1);
   }
   else
   {
      fResult = f1 / f2; // division
      cout << f1 << " / " << f2 << " = " << fResult << "\n";
      return(0);
   }
}
```

کد بالا ویژگی هایی دارد که نیاز است به آنها اشاره شود:

```
float f1;
```

قسمتی از حافظه را به به نام f1 نسبت می دهد که قرار است در این خانه مقداری اعشاری ذخیره شود. مانند ۰. ۱ ، ۵.۲ ، ۰.۳۳۳.... (معمولا در سیستم های ۳۲ بیتی) . اینکه رایانه این چنین این مقادیر را در خود ذخیره می کند، به آینده موکول می کنیم. اعدد اعشاری خود از دو جنس double و float هستند که هر کدام از اینها، بسته به ساختار سیستم دارای محدوده می باشند. محدوده ی پویا[59] ی تعریف شده برای نوع داده ی foat بین 10^{-38} و 10^{38} و برای نوع داده ی double بین 10^{-308} و 10^{308} است. در رابطه با اعداد اعشاری و نوع داده های double و float ، بحث بسیار است. در واقع این نوع داده از مهمترین ابزار برای محاسبات مهندسی به شمار می رود و ما تا جایی که برایمان لازم باشد، روی آن کار خواهیم کرد.

توجه کنید که در زبان C++ متغیر ها در هر جایی می توانند تعریف شوند. بسیاری از برنامه نویس ها، ترجیح می دهند متغیر را دقیقا قبل از همان جایی که می خواهند استفاده کنند، تعریف کنند.

عبارت مقداری را برای متغیر f1 از ورودی دریافت می کند:

```
cin >> f1;
```

[59] Dynamic Rang

۶.۴ ساختار شرطی if/else if/else

ساختار شرطی if/else if/else یکی از مهم ترین ساختار های کاربردی در زبان C++ ، و در بسیاری از سایر زبان های برنامه نویسی است. در واقع این ساختار یک حالت کلی است که در شرایط نیاز می تواند شکل های دیگری را به خود بگیرد. در حالت تکی، ساختار محدود به فقط یک شرط، if ؛ درحالت دو گانه ، ساختار محدود به دو شرط، if / else ؛ در حالت چند گانه به صورت if / else و یا if / else if /else است.

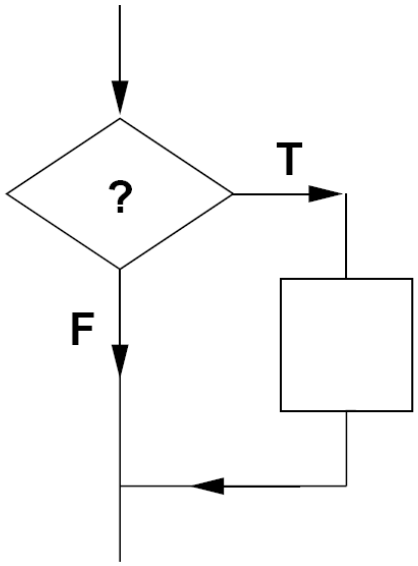
حالت کلی دستور شرطی به شکل زیر است:

```
if (logical expression 1 )
{
      STATEMENT BODY 1
}
else if (logical expression 2 )
{
      STATEMENT BODY 2
}
else if (logical expression 3 )
{
.
.
.
else if (logical expression N + 1 )
{
       STATEMENT BODY N + 1
}
else
{
      STATEMENT BODY N + 2
}
```

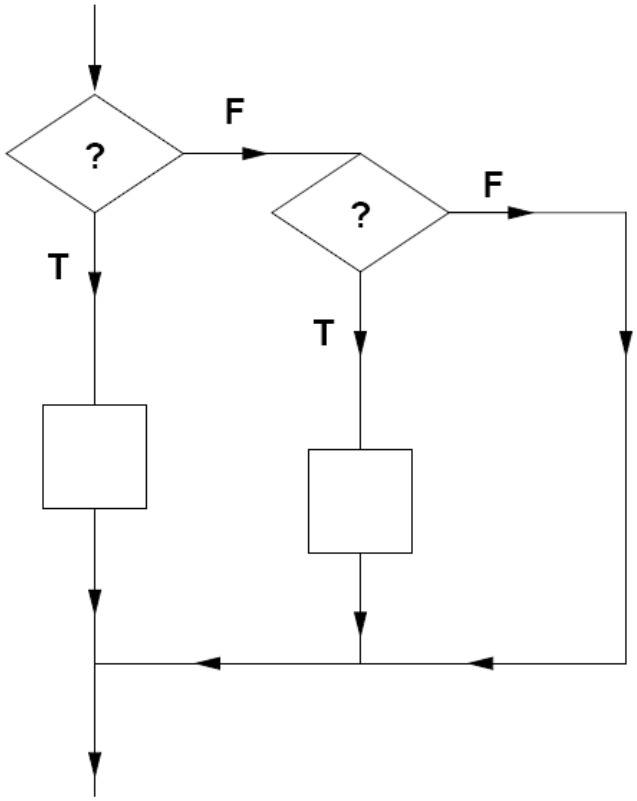
بنابرین راه های مختلفی برای استفاده از ساختار شرطی وجود دارد.

- استفاده از قسمت { STATEMENT BODY } (***logical expression***) else if اختیاری است.
- همچنین استفاده از قسمت { STATEMENT BODY } else اختیاری است .
- در صورتی که قسمت STATEMENT BODY شامل یک دستور باشد، نیازی به گذاشتن علامت های {} در ابتدا و انتهای دستورات نیست. قرار دادن این علامت ها موجب خوش خوانی برنامه است و استفاده از آنها توصیه می شود.
- به دندانه دار نوشتن کد حتما توجه کنید. زیرا که تاثیر به سزایی در خوانا بودن کد دارد.
- توجه کنید که در ساختار if / else if /else حداکثر یک مجموعه STATEMENT BODY اجرا خواهد شد.
- اگر ساختار شرطی به صورت if/else if یا if استفاده شود، ممکن است شرایطی پیش آید که هیچ کدام از دستورات اجرا نشوند.
- چندین ساختار شرطی می توانند به صورت تو در تو استفاده شوند.

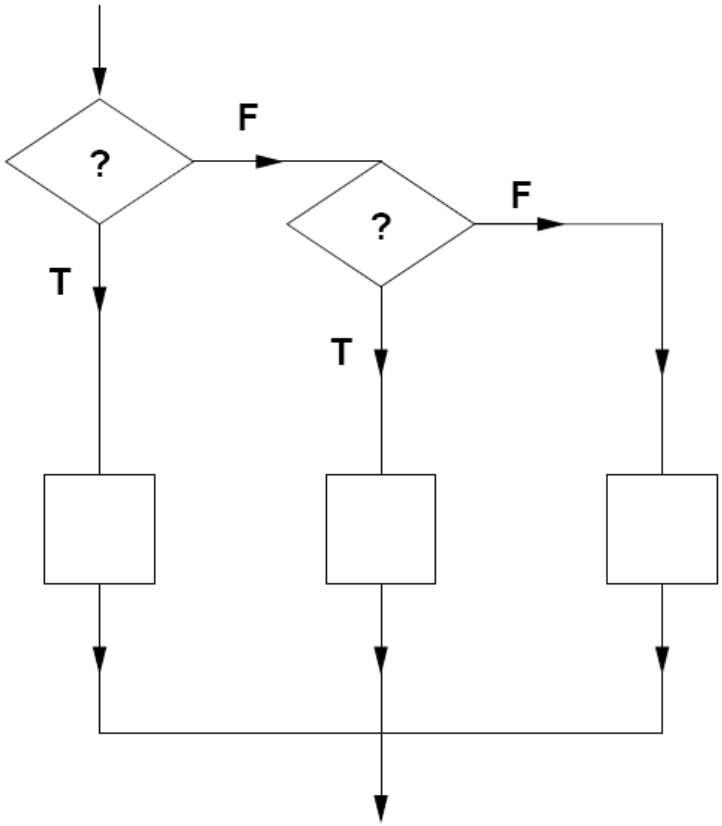
در زیر شکل فلوچارت ساختار if را مشاهده می کنید:



فلوچارت زیر، ساختار if/ else if را نشان می دهد:



و فلوچارت زیر ساختار شرطی if / else if / else را نشان می دهد:



۷.۴ عبارت‌های منطقی

یک عبارت منطقی چه چیزی می تواند باشد ؟؟؟!

یک عبارت منطقی چیزی است که درست یا نادرستی را ارزیابی می‌کند. **نادرستی** دارای ارزش عددی صفر **ودرستی** دارای ارزش عددی غیرصفر می‌باشد.

برای مثال:

```
if (1)
   cout << "This would always print,\n";
else if (0)
    cout << "whereas this would NEVER print\n";
```

دراینجا چند عملگر منطقی به ترتیب اولویت آورده شده اند.

اولویت	علامت	عملگر	توضیحات
۰	()	پرانتز ها	پرانتزهای داخلی اولویت محاسبه ای بیشتری دارند.
۱	! (مجموعه از عملگرها)	قرینه	R→L
۲	<= >= < >	کوچکتر ، کوچکتر مساوی، بزرگتر ، بزرگتر مساوی	L→R
۳	== =!	مساوی با ، مخالف با	L→R

حال بیایید برگردیم به مثال حل معادله درجه دو $Ax^2 + Bx + C = 0$ که آنرا در بخش شبه برنامه و فلوچارت حل کرده بودیم .

دیدیم که در حل این تمرین از دستورات شرطی if به صورت تو در تو استفاده کرده بودیم.

بنابرین ما نیز در اینجا سعی می کنیم بر طبق همان روال، کد این برنامه را با استفاده از دستورات if و else بنویسیم:

```
//File: quadratic.cpp

#include <iostream>
#include <cmath>

using namespace std;

int main(void)
{
   float a,b,c; // Declare the constants...
   cout << "\n"
           "We will solve for x in Ax*x + B*x + C = 0\n"
           "=========================================\n"
           "\n";
   cout << "Input A,B,C: ";
   cin >> a >> b >> c; //...and read them in
   cout << "Calculating roots (x) of the equation\n"
        << a << "*x*x + " << b << "*x + " << c << " = 0...\n";
   if (0 > b*b - 4*a*c) //b*b - 4*a*c is less than 0
   {
       cout << "No solution since b*b-4*a*c < 0\n";
       return(0);
   }
   else // b*b - 4*a*c is >= 0
   {
       if (0 != a)
       {
           cout << "Sol'n 1 = " << (-b + sqrt(b*b - 4*a*c))/(2*a) << ", "
                   "Sol'n 2 = " << (-b - sqrt(b*b - 4*a*c))/(2*a) << "\n";
           return(0);
       }
       else // b*b-4*a*c is >= 0 and a = 0
      {
           if (0 != b) // b*b - 4*a*c is >= 0, a = 0, b != 0
           {
               cout << "One sol'n since a = 0. Sol'n = " << -c/b << "\n";
```

```
                return(0);
            }
            else // b*b - 4*a*c is >= 0, a = 0, b = 0
            {
               cout << "No sol'ns since a and b = 0.\n";
               return(0);
            }
        }
    }
}
```

اگرچه با استفاده از else if ما می‌توانیم برنا مه‌ای بنویسیم که کمتر پیچیده به نظر برسد، اما با این حال، پیچیدگی هنوز هم در برنامه وجود دارد. باید به یاد داشته باشیم تنها زمانی وارد قسمت else if خواهیم شد که شرط عبارت if قبلی، نادرست باشد.

استفاده از else if می تواند به سادگی و خواناتر بودن برنامه کمک زیادی کند.

برای درک این مطلب، به خوانایی برنامه‌ی زیر و برنامه قبلی توجه کنید:

```
//File: quadratic2.cpp

#include <iostream>
#include <cmath>

using namespace std;

int main(void)
{
     float a,b,c; // Declare the constants...

     cout << "\n"
             "We will solve for x in Ax*x + B*x + C = 0\n"
             "========================================\n"
             "\n";
     cout << "Input A,B,C: ";
     cin >> a >> b >> c; //...and read them in

     cout <<
             "Calculating roots (x) of the equation\n"
          << a << "*x*x + " << b << "*x + " << c << " = 0...\n";

     if (0 > b*b - 4*a*c) //b*b - 4*a*c is less than 0
     {
               cout << "No solution since b*b-4*a*c < 0\n";
         return(0);
     }
     else if (0 != a) // b*b - 4*a*c is >= 0, a != 0
     {
               cout <<
                "Sol'n 1 = " << (-b + sqrt(b*b - 4*a*c))/(2*a) << ", "
                "Sol'n 2 = " << (-b - sqrt(b*b - 4*a*c))/(2*a) << "\n";
         return(0);
     }
     else if (0 != b) // b*b - 4*a*c is >= 0, a = 0, b != 0
     {
          cout << "One sol'n since a = 0. Sol'n = " << -c/b << "\n";
          return(0);
     }
     else // b*b - 4*a*c is >= 0, a = 0, b = 0
     {
```

```
            cout << "No sol'ns since a and b = 0.\n";
            return(0);
        }
}
```

۱.۷.۴ عبارت های منطقی با AND و OR

در اینجا تعدادی دیگر از عملگرهای منطقی به همراه ترتیب اولویتشان آورده شده است:

اولویت	علامت	عملگر	توضیحات
۰	()	پرانتز ها	پرانتز ها داخلی اولویت محاسبه ای بیشتری دارند.
۱	(مجموعه از عملگر ها) !	قرینه	R→L
۲	<= >= < >	کوچکتر ، کوچکتر مساوی، بزرگتر ، بزرگتر مساوی	L→R
۳	=! ==	مساوی با ، مخالف با	L→R
۴	AND (منطقی)	&&	L→R
۵(آخرین تقدم)	OR (منطقی)	||	L→R

طرز انجام کار هر یک از عملگرهای AND و OR در جدول زیر دیده می‌شود. تصور کنید هر یک از عبارات منطقی در دو طرف AND و OR می توانند T (True) یا F(False) باشد ، که نتیجه عملگر به این‌دو مقدار درست یا غلط وابسته است.

(T	&&	T)	==	T
(T	&&	F)	==	F
(F	&&	T)	==	F
(F	&&	F)	==	F
(T	||	T)	==	T
(T	||	F)	==	T
(F	||	T)	==	T
(F	||	F)	==	F

اکثر اوقات حفظ کردن چنین جدولی کار دشواری است! جدول زیر از جهات مختلف به ما کمک می کند. اگر با به درست و غلط، به ترتیب، ارزش‌های عددی درست و غلط را نسبت دهیم، خواهیم داشت:

(1	&&	1)	==	1
(1	&&	0)	==	0
(0	&&	1)	==	0
(0	&&	0)	==	0
(1	||	1)	==	1
(1	||	0)	==	1
(0	||	1)	==	1
(0	||	0)	==	0

هنگامی که ما این کار را انجام دهیم، متوجه خواهیم شد که نتیجه (A && B) ازش کمتری نسبت به A یا B دارد. ولی نتیجه (A||B) دارای ارزش بیشتری از هر کدام از A یا B خواهد بود. مطمئن باشد این موضوع پیچیده تر از این نخواهد شد. درواقع این همان حقیقتی است که مدارهای الکتریکی بر آن اساس کار می کنند.

چند مثال:

```
int i = 5;

if (0 < i && 10 > i)
     cout << i << " is a positive, single digit integer\n";

if (0 > i || 10 <= i)
     cout << i << " is either negative, or has more than one digit\n";
```

۲.۷.۴ ترکیب عبارت های منطقی و حسابی

می توان عبارات منطقی و عبارات ریاضی را با هم ترکیب کرد. معمولااین عملیات جایی انجام می شود انتظار می رود یک عبارت منطقی داشته باشیم. برای مثال در قسمت شرط ساختار if / else if / else می توان ترکیبی از عبارات شرطی و عبارات ریاضی را به کار برد که در حقیقت با هم یک عبارت منطقی محسوب می شوند.

زمانی که یک عملگر منطقی و حسابی با هم ترکیب شده و عبارت واحدی را به وجود می آورند اولویت آنها به صورت زیر خواهد بود: بالاترین اولویت در مورد محتوای درونی ترین پرانتز ها است. سپس عملگرهای حسابی مورد پردازش واقع می شوند و در نهایت عبارت های منطقی هستند که روی آنها عمل انجام می شود .در جدول زیر ترتیب اولویت های عبارت های حسابی و منطقی آورده شده است.

اولویت	علامت	عملگر	توضیحات
۰	()	پارانتز ها	پارانتز ها داخلی اولویت محاسبه ای بیشتری دارند. L→R
۱	(مجموعه از عملگر ها) -	قرینه	R→L
۲	* / %	ضرب، تقسیم، باقیمانده	L→R
۳	+ -	جمع ضرب	L→R
۴	< > <= >=	کوچکتر ، کوچکتر مساوی، بزرگتر ، بزرگتر مساوی	L→R
۴	== !=	مساوی با ، مخالف با	L→R
۵	AND (منطقی)	&&	L→R
۶	OR (منطقی)	||	L→R
۷(آخرین تقدم)	=	مقدار دهی	R→L

توصیه ی ما این است که در هر حال چه زمانی که شک دارید و چه زمانی که روی یک عبارت شک ندارید از پرانتز گذاری استفاده کنید. شخصی که برنامه ی شما را می خواند شاید اولویت اجرای عملگر ها را فراموش کند اما وجود پارانتز ها به عنوان اولین ترتیب در اولویت ها ترجمه ، باعث خوانا شدن و قابل فهم شدن برنامه شما می شود.

در حقیقت می توان گفت این یک عرف در میان اکثر برنامه نویسان است که تقریبا در تمام عبارات شرطی، اگربه اندازه ی ذره ای شک داشته باشند، از پرانتز استفاده می کنند. همچنین به طور معمول اطراف همه زیرعبارت های حسابی در عبارت های منطقی-حسابی پرانتز می گذارند.

در اینجا مثالی از ترکیب عبارت های منطقی و حسابی آمده است. از جدول اولویت بالا استفاده استفاده کنید تا ببینید چگونه عبارت زیر ارزیابی میشود:

```
if (i*j == i*i + j*j || i - j <= i*i - j*j && i + j <= i*i + j*j) ...;
```

ترتیب && و || نیز خیلی مهم است. به عنوان مثال با استفاده از ترتیب اولویت های بالا ممکن عبارت منطقی-حسابی بالا به صورت زیر خلاصه شوند: 0 && 1 || 1 در اینصورت شما حتما باشد بدانید که اول باید && را حساب کنید یا || را . اگر به طور اشتباه حاصل || را ابتدا حساب کنید. نتیجه ی عبارت بالا False خواهد بود. در صورتی که ابتدا && حساب می شود و نتیجه مقداری True است.

۸.۴ مسئله ها

۱.یک سؤال مقاله نویسی؟ در کلاس کامپیوتر!

سؤالات زیر را به زبان خودتان پاسخ دهید.

a) الگوریتم چیست؟

b) به جز واحد های start و stop سه الگو را نام ببرید که در نوشتن هر الگوریتمی به آنها نیاز داریم.

c)برای هر یک از این سه الگو یک مثال بیاورید.

d)سه راه که در هر الگوریتم ممکن است توصیف شود را نام ببرید.

e)از هر یک از را ههای بالا یک مثال بیاورید.

۲. محاسبات ابتدایی

برنامه ای بنویسید که به ازای تمام متغیر هایی که در سمت چپ تعیین شده اند، با محاسبه ی مقدار سمت راست آنها، متغیر ها را مقدار دهی نماید. متغیر های i و j از نوع int و x و y از نوع float هستند.

```
i = 1 + 2*3/4+5;
x = 1.0 + 2.0 + 3.0/4.0 + 5.0;
j = (1 + 2)*3/4 + 5;
y = (1.0 + 2.0)*3.0/4.0 + 5;
y = 1 + 2 + 3/4 + 5;
x = 1 - (static_cast<int>(6*5/4.0*3))/2;
```

۳. اساس عملگر های منطقی

عبارت منطقی زیر را بررسی کنید و محاسبه کنید که حاصل هر کدام از آنها کدامیک از مقادیرtrue و یا false می باشند ؟

به یاد داشته باشید که iعددی صحیح است.

```
(1 || 1 && 0 || 0)
(0 > i && 10 < i)
(0 > i || -10 < i)
```

۴. ترکیب عملگر های حسابی و منطقی

عبارت منطقی–حسابی زیر را بررسی کنید و محاسبه کنید که حاصل هر کدام از آنها true و یا false می باشد؟

به یاد داشته باشید که iعددی صحیح است. (راهنمایی: مهم نیست که i چه مقداری داشته باشد، مهم این است که 2*i مقداری زوج و 2*i+1 عددی فرد است.)

```
(2*i + 1)
(4*i%2 && 1)
(2*i + 1 || i/2)
```

۵. ساختار استفاده از دستورات

کدام یک از عبارات زیر صحیح هستند؟

```
int i, j, k = 0;
int i, double j;
int i0 = 4, f = -1, k=0;
int i; float _0j;
int i j k l m;
int The, Wolverines, Are, Rated, Number, 3;
float f, F, _000000, Wolverine;
int i,j ; i == 1 && j || 3%2;
int static_cast<float>(N);
i = -i + -j;
i + j = f;
x = x*x + 3;
y = x^2 + 3;
y = a x + b;
z = x + y123;
z = x + 123y;
for(i = 0,i <= 10,i = i + 1) cout << "OK?\n";
while(thisIsTrue) cout << "OK?\n";
if (((x < 0) && (y > 3) || (z == 2)) cout << "OK?\n";
if (a != b && c > d && e < f && g == h) cout << "OK?\n";
```

۶. خروجی را پیش بینی کنید.

در اینجا قطعاتی از یک برنامه آورده شده است که به درستی نوشته شده است.پیش بینی کنید که اگر این برنامه اجرا شود خروجی آن چه خواهد بود.

- ```
  int N = 0;
        if (N = 0)
        {
            cout << "The conditional expression was TRUE\n";
        }
        else
        {
            cout << "The conditional expression was FALSE\n";
        }
  ```

- ```
  int i = 0;
        if (0 != i)
        cout << "Statement 1\n";
        cout << "Statement 2\n";
        cout << "Statement 3\n";
  ```

- ```
  int j = 0;
        if (0 < j);
        {
           j = j + 1;
        }
        cout << "j = " << j << \n";
  ```
  ```

۷. این سوال واقعا ifی است. واقعاااا ifی!

مثال های خودتان از قوه ی تخیلیتان بیرون بکشید. کد های C++ خودتان را که از نظر دستور زبانی درست نوشته اند برای سوال های زیر بنویسید. جواب سوال اول نوشته شده است. کافی است از آن ایده بگیرید که سوال چه می خواهد. نیازی به تعریف یا توضیح متغیرهایی که استفاده می کنید وجود ندارد.

- ✓ یک مثال برای استفاده از if. اگر شرط TRUE باشد، یک و فقط یک عبارت اجرا می شود:

یک پاسخ صحیح:

```
if (i < N) f = f*I;
```

- ✓ یک مثال از استفاده از if. اگر شرط TRUE باشد، دقیقا دو عبارت اجرا می شوند.
- ✓ یک مثال برای استفاده از if و else if. اگر شرط if درست (TRUE) باشد، دقیقا دو عبارت و اگر شرط else if درست باشد، یک و دقیقا یک عبارت اجرا می شود.
- ✓ یک مثال برای استفاده از if و دو else if.
- ✓ یک مثال برای استفاده از if، دو else if و یک else.

۸.مکان یابی

برنامه ای بنویسید که که از کاربر دو عدد صحیح بگیرد ،سپس در ابتدا عدد اول را به عنوان مختصات x و عدد دوم را به عنوان مختصه ی y دریافت کند و سپس بر اساس جدول زیر نشان دهد نقطه ی (x,y) در چه مکانی قرار گرفته است.

$x > 0$	$y > 0$	upper right quadrant (URQ)
$x > 0$	$y < 0$	lower right quadrant (LRQ)
$x < 0$	$y > 0$	upper left quadrant (ULQ)
$x < 0$	$y < 0$	lower left quadrant (LLQ)
$x = 0$	$y > 0$	upper half (UH)
$x = 0$	$y < 0$	lower half (LH)
$x > 0$	$y = 0$	right half (RH)
$x < 0$	$y = 0$	left half (LH)
$x = 0$	$y = 0$	in the center (C)

می توانیم با استفاده از مخفف ها، آزاد تر عمل کنیم. برای مثال از URQ برای نشان دادن مکان ربع اول (Upper right quadrant) استفاده می کنیم.

در اینجا نمونه ای از آنچه که برنامه باید اجرا کند، را می بینید.

```
% a.out
Enter two int's, representing x and y: 1 1
Upper right quadrant (URQ)
%
```

9.4 پروژه ها

۱.اولین چالش برنامه نویسی شما

شما باید یک برنامه کامل بنویسید که به طور صحیح کامپایل شود. ورودی را قبول کند و خروجی را ارائه دهد و متوقف شود. برنامه شما باید عملیات زیر را انجام دهد.

a) از کاربر بخواهید که دو عدد صحیح مثبت وارد کند.

b) اگر کاربر دو عدد منفی وارد کند برنامه اشکال بگیرد و متوقف شود.

c) اگر دو عدد برابر باشد به کاربر اطلاع دهد در غیر این صورت نشان دهد کدام عدد بزر گتر و کدام یک کوچکتر است.

d) برنامه حاصل تقسیم عدد بزرگتر به کوچکتر را نمایش دهد به جز حالتی که عدد کوچکتر صفر باشد که در این حالت با دادن پیغام به کاربر اطلاع دهد.

e) برنامه، مجموع مربع های دو عدد را نشان دهد.

firstInteger * firstInteger + secondInteger * secondInteger

f) اگر عدد کوچکتر صفر باشد برنامه خاتمه پیدا کند در غیر این صورت برنامه کنترل کند که آیا عدد بزرگتر بر کوچکتر بخش پذیر هست یا نه و به کاربر اطلاع دهد.

g) برنامه خاتمه پیدا کند.

دستور عمل بالا خیلی مهم برای برنامه ی شما می باشد پس آن را به دقت بخوانید!

اینجا چند مثال از استفاده از این برنامه آمده است.برنامه باید دقیقا به صورت مشابه عمل کند.

```
> a.out
Input the 1st integer: -1
Input must be non-negative. Try again!
> a.out
Input the 1st integer: 1
Input the 2nd integer: -1
Input must be non-negative. Try again!
> a.out
Input the 1st integer: 1
Input the 2nd integer: 1
The two numbers are the same!
The ratio is 1, the quadrature sum is 2
1 is a perfect divisor of 1
> a.out
Input the 1st integer: 10
Input the 2nd integer: 0
The bigger number is 10, the smaller number is 0
The smaller number is 0, so no ratio is calculated, the quadrature sum is 100
> a.out
Input the 1st integer: 0
Input the 2nd integer: 10
The bigger number is 10, the smaller number is 0
The smaller number is 0, so no ratio is calculated, the quadrature sum is 100
> a.out
Input the 1st integer: 7
Input the 2nd integer: 4
```

```
The bigger number is 7, the smaller number is 4
The ratio is 1, the quadrature sum is 65
4 is not a perfect divisor of 7
> a.out
Input the 1st integer: 4
Input the 2nd integer: 7
The bigger number is 7, the smaller number is 4
The ratio is 1, the quadrature sum is 65
4 is not a perfect divisor of 7
> a.out
Input the 1st integer: 8
Input the 2nd integer: 2
The bigger number is 8, the smaller number is 2
The ratio is 4, the quadrature sum is 68
2 is a perfect divisor of 8
> a.out
Input the 1st integer: 2
Input the 2nd integer: 8
The bigger number is 8, the smaller number is 2
The ratio is 4, the quadrature sum is 68
2 is a perfect divisor of 8
```

۲. محاسبه ی سال کبیسه

قاعده محاسبه سال کبیسه بصورت زیر است:

سالی که بر ۴ بخش پذیر باشد(مثل ۲۰۰۴ و ۲۰۰۸ و...) سال های کبیسه اند. به جز سال هایی که بر ۱۰۰ بخش پذیرند. (مثل ۲۲۰۰و۲۱۰۰)که این سال ها در هر حال کبیسه نمی باشد.یک استثنا وجود دارد. اگر سالی بر ۴۰۰ بخش پذیر باشد، سال کبیسه است(مثل ۲۰۰۰ و ۲۴۰۰)

حال شما باید برنامه ای بنویسید که یک عدد مثبت بپذیرد و به عنوان یک سال، تست کند که آیا این عدد صفر یا منفی هست یا خیر. که در صورت صفر یا منفی بودن ، صورت بدون انجام عملیاتی از برنامه خارج شود. در غیراین صورت برنامه به کاربر اعلام کند که آیا این سال ورودی کبیسه هست یا خیر.در اینجا مثالی آورده شده است که چگونه این برنامه عمل می کند.

```
unix-prompt> a.out
Input a number for the year (>= 1): -1
Sorry, your year must be greater than 1.
unix-prompt> a.out
Input a number for the year (>= 1): 2001
The year 2001 is not a leap year
unix-prompt> a.out
Input a number for the year (>= 1): 2004
The year 2004 is a leap year
unix-prompt> a.out
Input a number for the year (>= 1): 2100
The year 2100 is not a leap year
unix-prompt> a.out
Input a number for the year (>= 1): 2400
The year 2400 is a leap year
```

۳. چه زمانی در روم...

جدول زیر نشان میدهد چگونه اعداد رومی نشان داده می شوند.

Arabic	Roman	Arabic	Roman	Arabic	Roman	Arabic	Roman
1	I	10	X	100	C	1000	M
2	II	20	XX	200	CC	2000	MM
3	III	30	XXX	300	CCC	3000	MMM
4	IV	40	XL	400	CD		
5	V	50	L	500	D		
6	VI	60	LX	600	DC		
7	VII	70	LXX	700	DCC		
8	VIII	80	LXXX	800	DCCC		
9	IX	90	XC	900	CM		

اعداد رومی به صورت ده دهی تقسیم می شوند. به اینصورت که بالاترین ارزش برای رقمی است که در سمت چپ قرار گرفته است. به ترتیب با کاهش ارش رقم، در سمت راست قرار می گیرند. برای مثال عدد ۱۹۹۹ به صورت رومی به صورت MCMXCIX نشان داده می شود. در اینصورت بزرگترین عددی را که می توان به صورت اعداد رومی نشان داد ۳۹۹۹ است که به صورت MMMCMXCIX نشان داده می شود.

حال شما باشد برنامه ای بنویسید که عدد مثبتی بین 1 و 3999 قبول کند بطوریکه اگر کاربر عددی خارج از این بازه وارد کند برنامه بدون انجام کاری خاتمه پیدا کند.در غیر این صورت برنامه عدد ورودی را به اعداد رومی تبدیل کند.

دراینجا مثالی آورده شده است که نحوه ی عملکرد برنامه را نشان میدهد..

```
unix-prompt> a.out
Input an integer between 1 and 3999 inclusive: 0
Sorry, your integer must be between 1 and 3999 inclusive.
unix-prompt> a.out
Input an integer between 1 and 3999 inclusive: 4000
Sorry, your integer must be between 1 and 3999 inclusive.
unix-prompt> a.out
Input an integer between 1 and 3999 inclusive: 1338
MCCCXXXVIII
unix-prompt> a.out
Input an integer between 1 and 3999 inclusive: 3888
MMMDCCCLXXXVIII
unix-prompt> a.out
Input an integer between 1 and 3999 inclusive: 52
LII
```

۴. یک محاسبه ی ۵ بیتی

برنامه ای بنویسید که ۵ بیت (۰ یا ۱) به ترتیب روبرو دریافت کند: b_0,b_1,b_2,b_3,b_4

وبه صورت $b_4b_3b_2b_1b_0$ نمایش دهد.

حال شما باید برنامه ای بنویسد که :

a) تعیین کند که آیا این عدد در دستگاه اعداد بدون علامت عددی زوج یا فرد است

b) مشخص کند این عدد در دستگاه اعداد علامت دار مثبت است یا منفی.

c) عدد داده شده را در دستگاه اعداد بدون علامت در نظر بگیرد و در مبنای ده نشان دهد.

d) متمم عدد داده شده را نشان دهد.

e)عدد را در دستگاه اعداد اعداد علامت دار را در نظر بگیرد و در مبنای ده نمایش دهد.

در اینجا مثالی از چگونگی عمل کرد این برنامه آمده است.

```
unix-prompt> a.out
Bit b0: 0
Bit b1: 1
Bit b2: 0
Bit b3: 1
Bit b4: 0
5-bit bit pattern is: 0 1 0 1 0
Bit pattern represent an even unsigned integer
Bit pattern represent a positive signed integer
Unsigned integer representation is: 10
Complement 5-bit bit pattern is: 1 0 1 0 1
Signed integer representation is: 10
unix-prompt> a.out
Bit b0: 1
Bit b1: 1
Bit b2: 1
Bit b3: 1
Bit b4: 0
5-bit bit pattern is: 0 1 1 1 1
Bit pattern represent an odd unsigned integer
Bit pattern represent a positive signed integer
Unsigned integer representation is: 15
Complement 5-bit bit pattern is: 1 0 0 0 0
Signed integer representation is: 15
unix-prompt> a.out
Bit b0: 1
Bit b1: 1
Bit b2: 1
Bit b3: 0
Bit b4: 1
5-bit bit pattern is: 1 0 1 1 1
Bit pattern represent an odd unsigned integer
Bit pattern represent a negative signed integer
Unsigned integer representation is: 23
Complement 5-bit bit pattern is: 0 1 0 0 0
Signed integer representation is: -9
unix-prompt> a.out
Bit b0: 1
Bit b1: 0
Bit b2: 0
Bit b3: 0
Bit b4: 1
5-bit bit pattern is: 1 0 0 0 1
Bit pattern represent an odd unsigned integer
Bit pattern represent a negative signed integer
Unsigned integer representation is: 17
Complement 5-bit bit pattern is: 0 1 1 1 0
Signed integer representation is: -15
```

۵. آیا محاسبات دو دویی مبهم اند ؟!

حال برمی گردیم به سیستم حسابی در مبنای دو که در مسئله ی قبل به آن پرداختیم . دستگاهی را تجسم میکنیم که فقط اعداد را در رشته های 5 بیتی می تواند نمایش دهد.در این زمان شما باید برنامه ای بنویسید که اعمال ریاضی را در 5 بیت انجام دهد.به یاد داشته باشید که نباید بیتی بیشتر از 5 تا وجود داشته باشد.نتیجه ی محاسبات اگر در ششمین بیت جا بگیرد باید کنار گذاشته شود.

برنامه ای که شما خواهید نوشت باید بتواند دو عمل زیر را متناسب با انتخاب کاربر انجام دهد.

a)یک عدد ۵ بیتی را در یک عدد ۲ بیتی ضرب کند .به عنوان مثال ۱۱× ۱۰۱۰۱

b)دو عدد ۵ بیتی را با هم جمع کند.به عنوان مثال ۱۰۱۰۱+۱۰۰۱۱

کاربر ، عدد را برای جمع و ضرب به برنامه ارائه میدهد و برنامه نتیجه محاسبات را به صورت 5 ارقام بیتی نشان می دهد.

نحوه استفاده از برنامه :

```
> a.out
What do you want to do?
Type a 0 if you want to add two 5-bit strings
Type a 1 if you want to multiply a 5-bit string by a 2-bit string
(0 or 1): 1
Enter the 5-bit binary number to be multiplied: 1 0 1 0 1
Enter the 2-bit multiplier: 1 1
10101
x 11
------
...is the same as the addition...
10101
+01010
------
11111
>
Here is another example of running the code:
> a.out
What do you want to do?
Type a 0 if you want to add two 5-bit strings
Type a 1 if you want to multiply a 5-bit string by a 2-bit string
(0 or 1): 0
Enter the 1st 5-bit binary number to be added: 1 0 1 0 1
Enter the 2nd 5-bit binary number to be added: 1 0 0 1 1
10101
+10011
------
01000
>
```

۶. بهترین تغییر

برنامه ای بنویسید که یک عدد حقیقی بین ۰.۰۰ و ۱۹.۹۹ را قبول کند. این عدد بقیه یک اسکناس ۲۰ دلاری است که بابت خرید یک جنس توسط مشتری باید به او بر گردانده شود.

شما باید برنامه ای به زبان C++ بنویسید تا اینکه مقدار پول دریافتی را به بهترین نحو، به اسکناس های ۱۰ دلاری، ۵ دلاری، ۱ دلاری، ۲۵ سنتی ، ۱۰ سنتی، ۵ سنتی و ۱ سنتی تقسیم کند.

در اینجا مثالی از چگونگی عملکرد این بر نامه آمده است . برنامه دو بار یک بار با ورودی 19.94 و دیگری با ورودی 16.26 اجرا می شود.به یاد داشته با شید که چگونه از پول های یاد شده استفاده می کند.

```
unix-prompt> a.out
Input a dollar amount between 0.00 and 19.99 inclusive: 19.94
Change of $19.94 is best given as:
One $10 bill
One $5 bill
Four $1 bills
Three quarters
One dime
One nickel
Four pennies
unix-prompt> a.out
Input a dollar amount between 0.00 and 19.99 inclusive: 16.26
Change of $16.26 is best given as:
One $10 bill
One $5 bill
One $1 bill
One quarter
One penny
unix-prompt>
```

۷. تبدیل از مبنای ۱۰ به مبنای ۲

برنامه ای در C++ بنویسید که عدد صحیحی بین ۰ و۱۲۷ قبول کند و آن را به مبنای ۲ ببرد.

شما مجاز به استفاده از loop نیستسد و راه حل شما نباید مانند کد زیر باشد.

```
if (decimal == 0) cout << "0";
else if (decimal == 1) cout << "1";
else if (decimal == 2) cout << "10";
else if (decimal == 3) cout << "11";
.
.
.
else if (decimal == 126) cout << "1111110";
else if (decimal == 127) cout << "1111111";
Here's an example of how it shouldw ork for an input of 126:
unix-prompt> a.out
Input a po Decimal 126 converted to base 2 is 1111110
unix-prompt>
sitive int between 0 and 127 inclusive: 126
```

(راهنمایی: بهتر است متناسب با الگوریتم خود از "\b" (escape sequence) که باعث به عقب برگشتن مکان نما به اندازه ی یک خانه به عقب می شود، استفاده کنید.

8. یک چیز حلقه ای برای فکر کردن

این یک سوال بحث بحثی است نه برنامه ای برای نوشتن زیرا دستور زبان مورد نیاز برای آن را در فصل بعد خواهید خواند .یعنی شما باید در بارهی حلقه ها که موضوع فصل بعد است فکر کنید .چگونه شما می توانید برنامه ای بنویسید که هر عدد صحیحی را به هر مبنای دمخواه بین 0 تا 9 تبدیل کند؟از حلقه ها استفاده کنید.این در حقیقت راحت تر از حل مسئله ی قبل است. در زیر مثالی از چگونگی عملکرد آن برای ورودی 126 آمده است.

```
unix-prompt> a.out
Input the base to convert to between 2 <= b <= 9: 7
Input any positive unsigned int: 126
Decimal 126 converted to base 7 is 240
unix-prompt>
```

فصل پنجم

حلقه ها

حلقه ها سومین و آخرین عنصر برنامه نویسی هستند که ما برای نوشتن هر الگوریتم نیاز داریم. زبان **C++** سه راه آسان را برای انجام این کار فراهم کرده است : حلقه های **while** و **do/while** و **for** .

1.5 حلقهی while

قالب عمومی حلقه while به صورت زیر است:

```
while (logical expression)
{
   STATEMENT BODY 1
}
```

بعضی از ویژگی های حلقه ی **while** :

- اگر **STATEMENT BODY** شامل یک و فقط یک عبارت باشد وجود **{}**های احاطه کننده اختیاری است. (استفاده از آنها می تواند به نظم و ترتیب و وضوح برنامه کمک کند، به هر حال استفاده از آنها شدیدا توصیه میشود.)
- استفاده از تورفتگیها و دندانه دار نوشتن را فراموش نکنید.
- بلافاصله بعد از اینکه شرط همراه با عبارت منطقی نادرست تست شد کنترل پردازش به عبارت بعد از **while** منتقل می شود.
- حلقه های **while** (در حقیقت، هر حلقه ی تکرار) می توانند به صورت تودرتو در داخل یکدیگر قرار گیرند.
- عبارت منطقی می تواند عبارت های منطقی و محاسباتی را برای کمک کردن به فشرده کردن برنامه نویسی با یکدیگر مخلوط کند. (این موضوع برای برنامه نویسان مبتدی توصیه نمی شود و اغلب خواندن برنامه را سخت می کند)
- اگر عبارت منطقی نادرست باشد (حتی دفعه اول) ، **STATEMENT BODY** درون حلقه **while** هرگز اجرا نمی شود.
- اگر شرطی که امتحان میشود درست باشد ولی نه **STATEMENT BODY** و نه عبارت منطقی آنرا تغییر ندهند به گونه ای که نادرست شود[۶۰] ، حلقه برای همیشه اجرا می شود.
- این حالت یک حلقه ی همیشگی[۶۱] **while(1){}** نامیده می شود. حلقه های همیشگی مضر هستند. این نوع حلقه ها کامپیوتر شما را به حالتی دچار میکنند که به طور قراردادی "هنگ کردن"[۶۲] نامیده می شود، مانند اینکه : «اه ، لعنتی، فِک کنم

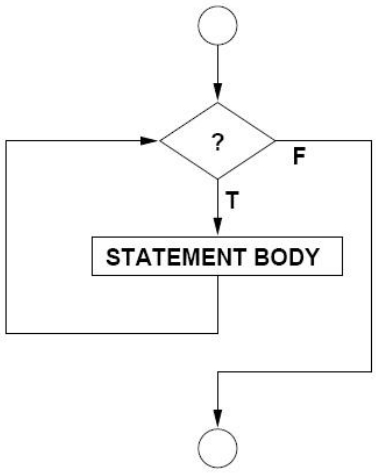
---

[۶۰] – یعنی عبارت منطقی که شرط انجام حلقه است همیشه (و برای هر بار) درست باشد م.

61- forever loop

کامپیوترم هنگ کرده» . این یک تعارف نیست ! کامپیوتر های هنگ کرده با اقدامات سخت گیرانه متفاوتی که شدت آن‌ها در حال افزایش است، از بدبختی خود رها می شوند و در حالتی بسیار بدتر باید خاموش شوند که آخرین راه گریز و واقعا ناپسند است.

تصویر زیر فلوچارت حلقه **while** است :



و اینک مثالی که فرمول مجموع گاوس[۶۳] را می آزماید:

```
//File: gauss.cpp
#include <iostream>

using namespace std;

int main(void)
{  cout << "Input N: ";
   int N;
   cin >> N;

   int i = 0, Sum = 0;
   while (i < N)
   {  i = i + 1;
      Sum = Sum + i;
   }

   cout <<
      "Sum of " << N << " digits is " << Sum
      << "...according to Gauss: " << N*(N+1)/2 << "\n";
   return(0);
}
```

62 - hung
63 - Gauss's summation formula

مثال بعدی خلاصه تر (ولی راهی خطرناک) برای انجام مثال بالا است :

```
//File: gaussAgain.cpp
#include <iostream>

using namespace std;

int main(void)
{  cout << "Input N: ";
   int N;
   cin >> N;

   int i = 0, Sum = 0;
   while (i = i + 1 <= N) Sum = Sum + i; //NO, NO, NO!

   cout <<
      "Sum of " << N << " digits is " << Sum
      << "...according to Gauss: " << N*(N+1)/2 << "\n";
   return(0);
}
```

دلیل خطرناک بودن راه حل بالا این است که اگر شما پراتنز های اطراف **i = i + 1** که قسمتی از شرط است را فراموش کنید یک حلقه ی نامتناهی را ایجاد می کنید! حرف مرا باور نمی کنید ؟ امتحانش کنید! اوم، صبر کنید، این یک حلقه همیشگی است ! خب، امتحانش کنید، بیشتر از چند ثانیه وقت شما را نخواهد گرفت!!!

امتیاز این است : سعی کنید برنامه خودتان را طوری بنویسید که تا حد امکان برای تفسیر کردن راحت باشد. خیلی باهوش نباشید. فضا را فقط به منظور حفظ فضا در عوض قربانی کردن وضوح حفظ نکنید [۶۴]. در زمان های قدیم برنامه نویسی حافظه ها و دیسک ها بسیار گران و نفیس بودند. برنامه نویسان خودشان را مقید می کردندکه برنامه هایشان بسیار خلاصه باشد. خوشبختانه آن روزها مدت مدیدی است که سپری شده اند. هر چند هنوز هم افرادی هستند که اینگونه عمل می کنند. شاید آنها فکر می کنند که یک الکترون به گرانی یک **Higgs' boson** است[۶۵].

[۶۴] – اگر می خواهید به منظور حفظ فضا، وضوح را قربانی کنید ، هرگز این کار را نکنید. م.

[۶۵] – اکترون ها ذاتا آزاد هستد. تعداد بیشماری از آنها در هر چیز وجود دارد ، مثلا در آب. Higgs' boson بسیار گران است. فیزیک دان ها میلیاردها دلار خرج پیدا کردن آنها می کنند. اما هنوز هم پیدا نشده است، ولی تقریبا چرا. اولین فردی که آن را بیابد جایزه نوبل فیزیک را خواهد برد.

۲.۵ حلقه do/while

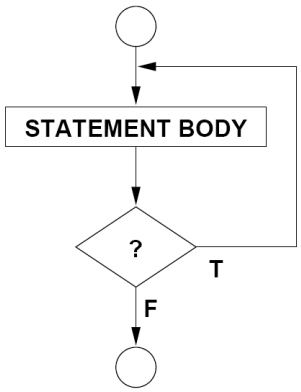
قالب عمومی حلقه **do/while** به صورت زیر است:

```
do
{
   STATEMENT BODY 1
}while (logical expression)
```

برخی ویژگی های حلقه **do/while** :

- بعد از شرط، " ; " را فراموش نکنید.
- اگر **STATEMENT BODY** شامل یک و فقط یک عبارت باشد وجود **{}** های احاطه کننده اختیاری است. (استفاده از آنها می تواند به نظم و ترتیب و وضوح برنامه کمک کند، به هر حال استفاده از آنها شدیدا توصیه می شود.)
- استفاده از تورفتگی ها را فراموش نکنید.
- بلافاصله بعد از اینکه شرط همراه با عبارت منطقی نادرست تست شد کنترل پردازش به عبارت بعد از **while** منتقل می شود.
- حلقه های **do/while** (در حقیقت، هر حلقه‌ی تکرار) می توانند به صورت تودرتو در داخل یکدیگر قرار بگیرند.
- عبارت منطقی می‌تواند عبارت های منطقی و محاسباتی را برای کمک کردن به فشرده کردن برنامه نویسی با یکدیگر مخلوط کند. (این موضوع برای برنامه نویسان مبتدی توصیه نمی شود و اغلب خواندن برنامه را سخت می کند)
- **STATEMENT BODY** حداقل یکبار انجام می شود حتی اگر عبارت منطقی[۶۶] در ابتدا غلط باشد.
- اگر شرطی که امتحان میشود درست باشد ولی نه **STATEMENT BODY** و نه عبارت منطقی آنرا تغییر ندهند به گونه ای که نادرست شود[۶۷] ، حلقه برای هیشه اجرا می شود.

شکل زیر فلوچارت حلقه **do/while** است :



[66] - logical expression

[67] – یعنی عبارت منطقی که شرط انجام حلقه است همیشه (و برای هر بار) درست باشد م.

تنها چیزی که واقعا یک حلقه ی **do/while** را از حلقه **while** متمایز می کند این است که حلقه ی **do/while** شرط را در انتهای **STATEMENT BODY** می آزماید در حالی که حلقه **while** آنرا در ابتدا ارزیابی می کند. حلقه های **while** اغلب زمانی استفاده می شوند که قرار است یک شمارنده نمو داده شود و در بدنه اصلی حلقه (و اغلب با نمو دادن) استفاده شود. حلقه های **do/while** زمانی مفید خواهند بود که شما بخواهید **STATEMENT BODY** را حد اقل یکبار پردازش کنید. به همین دلیل بهتر است برای حلقه هایی که ورودی هایی را از کی‌برد یا یک فایل می پذیرند از حلقه های **do/while** استفاده شود به ویژه زمانی که ورودی به وسیله یک نگهبان[۶۸] خاتمه می یابد.

اکنون مثالی از یک خروجی کنترل شده توسط یک نگهبان برای محاسبه معدل نمرات دانشجو را ارائه می کنیم. نگهبانی برای توقف زمانی است که که نمره کمتر از صفر باشد. حتی بیرحم ترین استاد هم نمره منفی نمی دهد (اگرچه این قانونی است !) بنابراین، این مطلب یک مراقبت خوب برای استفاده از این برنامه است.

```
//File: grades.cpp
#include <iostream>

using namespace std;

int main(void)
{  int nStudents = 0, grade, sumGrades = 0;

   do
   {  cout << "Input a grade: ";
      cin >> grade;

      if (0 > grade) cout << "Grade < 0, end of input.\n";
      else
      {   nStudents = nStudents + 1;
          sumGrades = sumGrades + grade;
      }
   }while (0 <= grade);

   if (0 < nStudents)
      cout << nStudents
      << " grades were read in. "
      << "The class average is: "
      << sumGrades/nStudents
      << "\n";
   return(0);
}
```

[68] - sentinel

دقت کنید که برنامه ی بالا میانگین را تحت عبارت **sumGrades/nStudents** محاسبه می کند : نتیجه ی یک تقسیم در اعداد صحیح. این تقسیم ممکن است کاملا منصفانه برابر میانگین نباشد. مثلا ۷۹/۹ واقعا نباید به صورت ۷۹ گزارش شود.

یک راه برای اصلاح کردن[۶۹] این مسئله این است که تغییرات زیر را اعمال کنیم:

۱- **#include** زیر را در قسمت پیش پردازنده[۷۰] برنامه وارد کنیم:

```
#include <iomanip>
```

۲ -عبارت **cout** را به صورت زیر اصلاح کنیم :

```
cout << nStudents
     << " grades were read in. "
     << "The class average is: "
     << setw(5) //Set field width to 5
     << setprecision(3) //3 digits of precision
     << static_cast<float>(sumGrades)/nStudents
     << "\n";
```

برنامه به صورت کامل در قسمت دروس[۷۱] در وب با نام **grades1.cpp** قرار دارد.

دو ویژگی جدید در عبارت **cout** وجود دارد. یکی معرفی عملگر یکانی **cast** به صورت **static_cast<float>** **(expression)** است که **sumGrades** را موقتا (و ذاتا) به عدد اعشاری[۷۲] تبدیل می کند و محاسبه برایند به گونه ای اتفاق می افتد که گویی در تقسیم فقط از اعداد اعشاری استفاده شده بود. ساختن تبدیلی مشابه برای **nStudents** ضروری نیست. تقسیم یک عدد اعشاری بر یک عدد صحیح یا یک عدد صحیح بر یک عدد اعشاری همیشه به صورت محاسبات اعداد با ممیز شناور[۷۳] نتیجه می دهد به گونه ای که عدد صحیح شرکت کننده به عدد اعشاری "ترقی می یابد".

فرم عمومی عملگر **cast** :

```
static_cast<type_name>(expression) //Don't forget the parentheses!
```

است که برای تغییر **expression** به نوع **type_name** به کار می رود. بنابر این اعداد اعشاری را می توان به اعداد صحیح و بالعکس تبدیل کرد. تغییر بین دیگر انواع متغیر ها نیز امکان پذیر است که بعدا با آنها روبرو می شویم.

---

[69] - fix
[70] - pre-processor
[71] - lectures area
[72] - float
[73] - floating point calculation

ویژگی جدید دیگر، استفاده از **field width** و **precision** برای تغییر دادن خروجی به عدد اعشاری است. تابع **setw(m)** ، **m** فاصله برای چاپ کردن اختصاص می دهد. تابع **setprecision(n)** نیز **n** رقم معنی دار را چاپ می کند. این توابع در کتابخانه **iomanip** [۷۴] تعریف شده اند. بنابراین ما این فایل را اضافه کردیم، در غیر این صورت کامپایلر خطایی را در برنامه پیدا می کرد. دقت کنید که این توابع چگونه به کار گرفته شده اند. فقط باید یک‌بار قرار داده شود. توابع به کار گیرنده‌ی جریان ها [۷۵] آن را به یاد می آورند. در حالی که **setw()** باید هر بار به کار گرفته شود. این یکی دیگر از جریان های کنترل کننده که به طور متداول مورد استفاده[۷۶] قرار می‌گیرد که ثابت[۷۷] نامیده می شود. ما خیلی از آن استفاده نخواهیم کرد.

۳.۵ حلقه for

حلقه **for** شبیه حلقه **while** است به جز این که دستور زبان [۷۸] ، مقدار دهی اولیه را برای شرط ایجاد حلقه وتغییر شمارنده واضح و صریح می کند. این روش عموما زمانی ترجیح دارد که حلقه باید در اسلوبی منظم ، برای یک ضابطه پایان دهی به دقت مشخص شده با فاصله دهی های منظم و یا برای محاسبه راحت تر اندیس شمارنده حلقه، عمل کند.

ساختار عمومی یک حلقه **for** :

```
for (expression 1; logical expression; expression 2)
{
   STATEMENT BODY
}
```

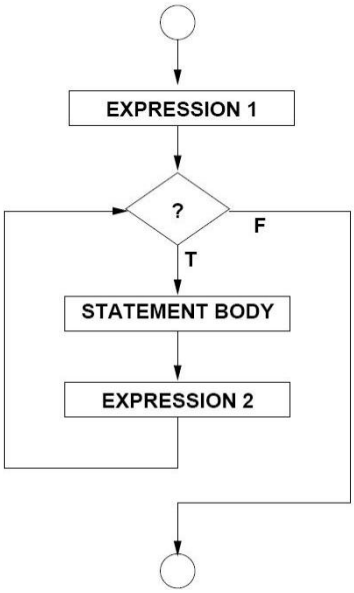
بعضی ویژگی های حلقه **for** :

- اگر **STATEMENT BODY** شامل یک و فقط یک عبارت باشد وجود **{}** های احاطه کننده اختیاری است. (استفاده از آن‌ها می تواند به نظم و ترتیب و وضوح برنامه کمک کند، به هر حال استفاده از آن‌ها شدیدا توصیه می‌شود.)
- استفاده از تو رفتگی‌ها را به خاطر بسپارید.
- بلافاصله بعد از اینکه شرط همراه با عبارتِ منطقیِ نادرست تست شد کنترل پردازش به عبارت بعد از **for** منتقل می‌شود.

[74] - I/O stream manipulator
[75] - stream handlers
[76] - commonly used stream manipulator
[77] - fixed
[78] - syntax

- حلقه های **for** (در حقیقت، هر حلقه‌ی تکرار) می‌توانند به صورت تودرتو در داخل یکدیگر قرار بگیرند.
- عبارت منطقی می‌خواهد از شرطی که به واسطه ی آن اجرای حلقه **for** متوقف می شود جلوگیری کند. این عبارت منطقی می تواند عبارت های منطقی و محاسباتی را برای کمک کردن به فشرده کردن برنامه نویسی با یکدیگر مخلوط کند. (این موضوع برای برنامه نویسان مبتدی توصیه نمی شود و اغلب خواندن برنامه را سخت می‌کند)
- اگر عبارت منطقی در ابتدا نادرست باشد **STATEMENT BODY** اجرا نخواهد شد.
- **expression 1** می خواهد به شمارنده حلقه (یا اندیس) مقدار اولیه بدهد. استفاده از آن اختیاری است. هرچند در این مورد مقدار دهی اولیه هر شمارنده (که باید حتما استفاده شوند) باید پیشاپیش انجام شود.
- استفاده از عبارت منطقی هم اختیاری است. اگر تهیه نشود برنامه همیشه وارد **STATEMENT BODY** می شود. راهی دیگر برای خروج از حلقه باید مهیا شود در غیر این صورت حلقه نامتناهی خواهد بود.
- **expression 2** می خواهد شمارنده را افزایش دهد. عبارت (یا عبارت های)، درون حلقه بعد از **STATEMENT BODY** اجرا می شوند. استفاده از آن نیز اختیاری است. اگر آنها را تعیین نکنیم، افزایش(یا کاهش) هر کدام از متغییرها باید درون **STATEMENT BODY** انجام شود.
- دو **";"** درون تعریف حلقه **for** اختیاری نیستند. حذف یکی یا هر دوی آنها باعث می شود کامپایلر از شما خطا بگیرد.

تصویر زیر فلوچارت حلقه **for** را نشان می دهد :



این هم مثالی که یک شمارش معکوس و یک پرواز موشک را اجرا می کند.

```
//File: blastOff.cpp

#include <iostream>

using namespace std;

int main(void)
{  cout << "Estimate number of counter loops/second: ";
   int loopsPerSecond;
   cin >> loopsPerSecond;

   cout << "How many characters wide is your screen? ";
   int screenWidth;
   cin >> screenWidth;

   cout << "How many characters high is your screen? ";
   int screenHeight;
   cin >> screenHeight;

   for (int i = 10; i >= 0 ; i = i - 1)
   {   // Next line should waste about 1 second
       for (int j = 0; j <= loopsPerSecond; j = j + 1) ;
       cout << i << "\n";
   }

   cout << "\a.\n.\n.\n.\nWe Have ignition!\a\n.\n.\n.\n";

   //"Draw" a rocket
   cout << " /\\\n"
        << " ||\n"
        << " ||\n"
        << " ||\n"
        << " ||\n"
        << "/||\\\n";

   //Start of the vapor trail
   cout << " **\n";
   for (int j = 0; j <= loopsPerSecond/2; j = j + 1) ;
   cout << "*****\n";
   for (int j = 0; j <= loopsPerSecond/5; j = j + 1) ;

   //Vapor trail expands
   for (int i = 6; i <= screenWidth; i = i + 1)
   {  for (int j = 1; j <= i; j = j + 1) cout << "*";
      for (int j = 0; j <= loopsPerSecond/i; j = j + 1) ;
      cout << "\n";
   }

   //Vapor trail at maximum
   for (int i = 1; i <= screenHeight; i = i + 1)
   {  for (int j = 1; j <= screenWidth; j = j + 1) cout << "*";
      for (int j = 0; j <= loopsPerSecond/screenWidth; j = j + 1) ;
      cout << "\n";
   }

   //Vapor trail contracts
   for (int i = screenWidth; i > 0; i = i - 1)
   {  for (int j = 1; j <= i; j = j + 1) cout << "*";
      for (int j = 0; j <= loopsPerSecond/screenWidth; j = j + 1) ;
      cout << "\n";
```

```
    }

    //Clear the screen
    for (int i = 1; i <= screenHeight; i = i + 1)
    {   for (int j = 0; j <= loopsPerSecond/screenWidth; j = j + 1) ;
        cout << "\n";
    }

    return(0);
}
```

۴.۵ مسئله ها:

۱. ملاقاتی دوباره با فلوچارت ها.

برای هر یک از ساختارهای حلقه زنی زیر یک نمایش فلوچارت (با استفاده از خطوط، پیکان ها، مستطیل ها، لوزی ها و کلمات) رسم کنید.

الف) برای حلقه **for** :

```
for (i = 10; i > 0; i = i - 1){Sum = Sum + i;}
```

ب) برای حلقه **do/while** :

```
do{cin >> oddNumber; odd = oddNumber%2}while(!odd);
```

ج) برای حلقه **while** :

```
while (different){z = x; y = x; x = temp; different = x - y;}
```

۲. تست های حلقه

الف) حلقه ی زیر چند بار اجرا می شود؟

```
int i = 0;
do
{ i = i + 1;
} while (i < 0);
```

ب) حلقه ی زیر چند بار اجرا می شود؟

```
int i = 0;
while(i < 0)
{ i = i + 1;
}
```

ج) حلقه ی زیر چند بار اجرا می شود؟

```
int i = 3;
while(i)
{ i = i - 1;
}
```

چ) حلقه ی زیر چند بار اجرا می شود؟

```
int i = 0;
while(i <= 0)
{ i = i - 1;
}
```

ح) حلقه ی زیر چند بار اجرا می شود؟

```
int j = 1;
do
{ j = -2*j + 1;
}while(j != 0);
```

خ) حلقه ی زیر چند بار اجرا می شود؟

```
for(int i = 0; i < 10; i = i + 2)
{ cout << "i = " << i << "\n";
}
```

۳. خروجی را پیش بینی کنید:

تمامی مثال‌های کدهای داده شده در زیر به درستی کامپایل شده اند. خروجی ای را که هر کد در صورتی که شما آنرا کامپایل و اجرا کرده بودید تولید می کرد، پیش بینی کنید

```
(a)
   int N = 0;
   if (N = 0)
      cout << "The conditional expression was TRUE\n";
   else
      cout << "The conditional expression was FALSE\n";

(b)
   int f;
   for(f = 0; f <= 10; f = f + 2)
      cout << f%10;
   cout << "\n";


(c)
   int i = 0, Sum = 0, N = 3;
   do
   { i = i + 1;
      Sum = Sum + i;
   }while(i <= N);
   cout << "Sum = " << Sum << "\n";


(d)
   int i = 0;
   if (0 != i)
      cout << "Statement 1\n";
      cout << "Statement 2\n";
      cout << "Statement 3\n";

(e)
   int j = 0;
   if (0 < j);
   { j = j + 1;
   }
   cout << "j = " << j << "\n";
```

```
(f)
   int i = 2, j = -2;
   while (j <= i)
   { if (j < 0)
          cout << "i is " << i << ".";
          cout << " j is " << j << ".\n";
          i = i - 1;

     j = j + 1;
   }
```

۴. تمارین حلقه

الف) کد ذیل را که حاوی یک حلقه **do/while** است به کدی که شامل یک حلقه **while** است تبدیل کنید به گونه‌ای که اجرای هر دو کد یکسان باشد.

```
int j = 0;
do
{  j = j + 1;
   cout << j << "\n";

}while (j <= 10);
cout << j << "\n";
```

ب) کد ذیل را که حاوی یک حلقه **do/while** است را به کدی که شامل یک حلقه **while** است تبدیل کنید به گونه‌ای که اجرای هر دو کد یکسان باشد.

```
int j = 10;
while (j > 0)
{  j = j - 2;
   cout << j << "\n";
}
cout << j << "\n";
```

ج) کد زیر را ملاحظه کنید:

```
#include <iostream>

using namespace std;

int main(void)
{  int x = 1;
   for (int i = 1; i <= 10 ; i = i + 1)
   {  x = x*2;
      cout << x << " ";
   }
   cout << endl;
   return 0;
}
```

کد بالا زمانی که آنرا اجرا می کنید بر روی صفحه نمایش چه چیزی چاپ می کند؟

برای حلقه **for** در برنامه فوق یک فلوچارت رسم کنید.

د) کد زیر را ملاحظه کنید:

```
#include <iostream>

using namespace std;

int main(void)
{  int i = 5;
   do
   {  i = i - 1;
      cout << i << " ";
   }while(i);
   cout << endl;
   return 0;
}
```

کد بالا زمانی که آنرا اجرا می کنید بر روی صفحه نمایش چه چیزی چاپ می کند؟

برای حلقه **do{}while();** در برنامه فوق یک فلوچارت رسم کنید.

ه) کد زیر را ملاحظه کنید:

```
#include <iostream>

using namespace std;

int main(void)
{  int i = 13;
   while(i%2)
   {  i = i/3;
      cout << i << " ";
   }
   cout << endl;
   return 0;
}
```

کد بالا زمانی که آنرا اجرا می کنید بر روی صفحه نمایش چه چیزی چاپ می کند؟

برای حلقه **do{}while();** در برنامه فوق یک فلوچارت رسم کنید.

ط) کد زیر را ملاحظه کنید:

```
#include <iostream>

using namespace std;

int main(void)
{  for (int i = 1; i <= 2; i = i + 1)
      for (int j = 1; j <= 3; j = j + 1)
         cout << i << "," << j << endl;
   return 0;
}
```

کد بالا زمانی که آنرا اجرا می کنید بر روی صفحه نمایش چه چیزی چاپ می کند؟

برای حلقه ی تو در توی **for** در برنامه فوق یک فلوچارت رسم کنید.

۵. مانند یک کامپیوتر فکر کنید.

کد زیر را در نظر بگیرید.

```
#include <iostream>
#include <cmath>
using namespace std;

int main(void)
{  float x;
   cout << "Input x (must be positive): ";
   cin >> x;

   int i = 0;
   const int NMax = 1000;
   const float SMALL = 0.0001;
   float xPower = 1.0, lastTerm, Sum = 0.0;
   do
   {  i = i + 1;
      xPower = x*xPower;
      lastTerm = xPower/i;
      Sum = Sum + lastTerm;
   }while (i < NMax && lastTerm > SMALL);

   if (i == NMax)
      cout << "Series did not converge\n";
   else
   {
      cout << "i = " << i << "\n"
           << "x = " << x << "\n"
           << "lastTerm = " << lastTerm << "\n"
           << "Sum = " << Sum << "\n";
   }
   return 0;
}
```

الف) چه سری ای به وسیله کد زیر جمع می شود؟ (مابقی را شما تعیین می کنید) **Sum = x +**

ب) اگر کاربر برای **x** مقدار 1.0- را وارد کند، چه چیزی به وسیله ی این کد چاپ می شود؟

پ) اگر کاربر برای **x** مقدار 0.0 را وارد کند، چه چیزی به وسیله ی این کد چاپ می شود؟

ت) اگر کاربر برای **x** مقدار 1.0 را وارد کند، چه چیزی به وسیله ی این کد چاپ می شود؟

ث) اگر کاربر برای **x** مقدار 0.0001 را وارد کند، چه چیزی به وسیله ی این کد چاپ می شود؟

ج) اگر کاربر برای **x** مقدار 0.001 را وارد کند، چه چیزی به وسیله ی این کد چاپ می شود؟

چ) اگر کاربر برای **x** مقدار 0.01 را وارد کند، چه چیزی به وسیله ی این کد چاپ می شود؟

ح) اگر کاربر برای **x** مقدار 0.1 را وارد کند، چه چیزی به وسیله ی این کد چاپ می شود؟ (یک ماشین حساب دستی می تواند کمکتان کند.)

۶ . باز هم مانند یک کامپیوتر فکر کنید.

برنامه **C++** زیر را در نظر بگیرید.

```
#include <iostream>
#include <cmath>
using namespace std;

int main(void)
{  float x;
   cout << "Enter x: ";
   cin >> x;

   int n = 0;
   float lastTerm;
   const float EPSILON = 0.001;
   float sign = 1;
   float denominator = 1;
   float total = 1;
   while(n <= 1 || lastTerm > EPSILON)
   {  n = n + 1;
      if (n/2*2 == n)
      {  denominator = denominator*n*(n - 1);
         lastTerm = pow(x,n)/denominator;
         sign = -sign;
         total = total + sign*lastTerm;
         cout << "Term " << n << " is " << lastTerm << "\n";
      }
   }

   cout << "The total is " << total << "\n";

   return 0;
}
```

الف) اگر شما 0.01 را برای **x** وارد می کردید، چه چیزی چاپ می شد؟

ب) اگر شما 1.0 را برای **x** وارد می کردید، چه چیزی چاپ می شد؟ (یک ماشین حساب دستی می تواند کمک کند.)

پ) شرط دیگری برای عبارت **if** که نتیجه ای مشابه می دهد، کدام است؟

ت) فرمولی که این برنامه اجرا می کند، چیست؟

۷. پیش بینی و تبدیل کنید

الف) قسمت پیش بینی : اگر شما این برنامه را کامپایل و اجرا کرده بودید، خروجی دقیقا چگونه بود؟

```
#include <iostream>

using namespace std;

int main(void)
{  const int maxTerms = 8;
   const int x = 2;

   int sign = 1;
   int term = 1;
   for (int i = 1; i <= maxTerms; i = i + 1)
   {  sign = -sign;
      term = term*x;
      if (0 == i%2 || 2 <= i && 4 >= i)
      {  cout << i << ":";
         if (0 > sign) cout << " - ";
         else cout << " + ";
         cout << term << endl;
      }
   }

   return 0;
}
```

ب) قسمت تبدیل : برنامه بالا را با جایگزینی یک حلقه **do-while** به جای حلقه **for** باز نویسی کنید. راه حل خود خود را بنویسید.

۸. دوباره پیش بینی و تبدیل کنید

الف) قسمت پیش بینی : اگر شما این برنامه را کامپایل و اجرا کرده بودید، خروجی دقیقا چگونه بود؟

```
#include <iostream>

using namespace std;

int main(void)
{  const int maxTerms = 8;
   const int x = 2;

   int term = 1;
   int sum = 1;
   cout << "Sum = 1";
   for (int i = 1; i <= maxTerms; i = i + 1)
   {  term = -term*x;
      if (0 != i%3 && 0 != i%4 )
      {  if (0 > term) cout << " - " << -term;
         else cout << " + " << term;
         sum = sum + term;
      }
   }
   cout << " = " << sum << "\n";

   return 0;
}
```

ب) قسمت تبدیل : برنامه بالا را با جایگزینی یک حلقه **while** به جای حلقه **for** باز نویسی کنید.راه حل خود را بنویسید.

فصل ششم

تجرید اولیه[79] توابع

۱.۶ محرک توابع[80]

بیایید نحوه ی عملکرد الگوریتم **al-kashi** را بررسی کنیم. این الگوریتم مقدار $\mathbf{x^N}$ را محاسبه می کند. **N** می تواند هر عدد صحیح،مثبت، منفی یا عددی غیرصفر باشد. نحوه ی کد نویسی برای این الگوریتم در**C++** را در اینجا آورده ایم.

```
//File : alkashi.cpp
#include<iostream>

Using namespace std;

Int main(void)
{
   cout << "A calculation of X  (float)  to the power of  N  (int)  \n";

   float X;  //  The float to be raised to a power
   int N ; //power to be raised to
   cout << " Input X  (float) and  N (int) :" ;
   cin >> X >> N ;

   float result ;  //The result of the calculation

   {
       //open a new  " scope " or " code " block
       //An implementation of the al-kashi algorithm
       Int  n = N ;  // An int used internally in this code block
       float  x = X ;  // A float used internally in this code block
       if  (0 > N )
       {   //This "trick" allow us to calculate for negative powers!
           n = -N ;
           x = 1/X ;
       }
       result = 1 ; // Initial value
       while (1 <= n )
       {
           if (0 == n % 2 )
           {
               n  =  n/2 ;
               X = x*x ;
           }
           else
           {
  n = n -1 ;
               result = result*x ;
```

[79] Early abstraction
[80] Motivation for functions

```
            }
        }
    }  //  End of the scope block

    cout << X <<  " to the power " << N <<  " = " << result <<  "\n" ;

    return (0) ;
}
```

مثال بالا مفهوم جدیدی به نام "بلوک[۸۱]" ، یا "بلوک کد[۸۲]" را در برنامه نویسی **C++** معرفی می کند.یک "بلوک" مجموعه ای از عبارات است که بوسیله ی دو علامت آکولاد {...} در بر گرفته می شود. در مثال بالا،

```
{ // Open a newa " scope " or  " code " block
    .
    .
    .
} // End of the scope block
```

دو آکولاد نشان داده شده یک بلوک کد را محدود می کنند – این نمونه با الگوریتم **Al-kashi** مرتبط است. متغیرها می توانند داخل یک بلوک کد تعریف شوند. متغیرهایی که داخل یک بلوک تعریف می شوند (مثل **float x** و **int n** دراین مثال) خارج ازبلوک شناخته نمی شوند. اگرشما بخواهید **n** و **x** راخارج از بلوکشان تعریف کنید، خطایی بیانگر اینکه این متغیرها تعریف نشده اند از کامپایلردریافت خواهید کرد. تا رسیدن به بخش "قوانین حوزه ی عملیاتی[۸۳]" به همین میزان اطلاعات درباره ی بلوک های کد بسنده می کنیم. ما قبلا هم بلوکهای کد را درتابع **main** وعبارات **do/while**، **if/else** و **loop** دیده ایم.

اکنون فرض کنید چیزی درباره ی الگوریتم **al-kashi** نمی دانستیم والگوریتم را آسانتر کد نویسی می کردیم:

```
// File: dumbPower.cpp
#include<iostream>

using namespace std

int main(void)
{
   cout << "A calculation of X (float) to the power of N (int) \n";

   floatX ;  //  The float to be raised to a power
   int  N ;   //  Power to be raised to
   cout << " Input  X  (float) and N (int):" ;
   cin >> X >> N ;
   float result ; // The result of the calculation
   {
       // Open a new  " scope " or "code" block
       // An implementation of the dumbPower algorithm
       int n = N ;
       float x = X ;
       if (0>N)
       {
          n = -N ;
          X = 1/X ;
```

[81] Block
[82] Code block
[83] Scope rules

```
        }
        result= 1 ;  // Initial value
        for (int I = 1 ; i<= n ; i = i+1)
        {
           result = result*x ;
        }
   } // End of the scope block
   cout << X << " to the power " << N << "=" << result << "\n" ;
   return(0) ;
}
```

در این دو مثال دیده می شود که کدها خارج از بلوک حوزه ی عملیاتی بسیار تکرار شده اند. کدنویسی تکراری باعث اتلاف وقت شده و روش خوبی نیست. درواقع، تاجایی که به تابع **main** مربوط می شود، به محتویات بلوک کد اهمیتی نمی دهد و تنها می خواهد به جواب برسد. تابع **main** به الگوریتم به عنوان راهی بسیار خلاصه شده برای محاسبه ی توان نگاه می کند و تنها چیزی که به آن احتیاج دارد، دادن متغیرهای **X** و **N** به بلوک حوزه ی عملیاتی [۸۴] می باشد.چیزی که ما واقعاً می خواهیم، روشی است که مستقل از تکنیک های محاسباتی ویژه ای باشد که برای به دست آوردن مقدار **XN** استفاده می شوند. درواقع درکتابخانه ی استاندارد ریاضیات[۸۵] در زبان **c++** تابعی وجود دارد که این کاررا انجام می دهد. این تابع **pow** نامیده می شود.با استفاده از این اطلاعات می توانیم کدهایی به مراتب خلاصه تر بنویسیم:

```
// File : power0.cpp
#include <iostream>
#include <cmath>

Using namespace std ;

Int main(void)
{
   cout << "A calculation of X (float) to the power of N (int)  \n" ;
   float X ;  / /  The float to be raised to a power
   int    N ; / /   Power to be raised to
   cout  <<  "Input  X (flaot) and N (int): " ;
   float result ; // The result of calculation
   result = pow(X,N) ;
   cout << X << " to the power " << N << " = " << result << "\n" ;
   return(0) ;
}
```

به استفاده از **#include <cmath>** که به تفسیر عبارت **result = pow(X,N) ;** کمک می کند، توجه کنید.

این پیاده سازی دو مثال پیشین به شکلی بسیار کم حجم تر و قوی تر بود.این روش به تابع **main** این امکان را می دهد که تنها با ورودی و خروجی پارامترهای گوناگون، **N، X** وجواب نهایی درارتباط باشد.تابع **main** در این مورد بسیار کلی تر است.

بیایید وانمود کنیم که درحال حاضرما دردوران کودکی(!) زبان **c++** قرار داریم و چیزی به نام تابع توان [۸۶] وجودندارد. اگرچه ما به این عاملیت در سایر برنامه های کاربردی خود بطور مکرر احتیاج خواهیم داشت، اما بطور قطع ما الگوریتم **al-kashi** را می شناسیم. از آنجا که می خواهیم کارآمدتر عمل کنیم، الگوریتم **al-kashi** رادرون یک تابع **c++** کدنویسی می کنیم و آن را به صورتی که در ادامه آمده است، به کار میگیریم:

[84] Scope block
[85] Standard math library
[86] Pow function

```
//File: power1.cpp
#include <iostream>
using namespace std ;
float pow(float x,int n)
{
    float result ; // The result of the calculation
    //An implementation of the al-kashi algorithm
    If ( 0 > n )
    {
        n = -n ;
        x = 1/x ;
    }

    result = 1;
    while (1 <= n)
    {
       if (0 == n % 2)
       {
          n = n/2;
          x = x*x ;
       }
       else
       {
          n = n - 1;
          result = result*x;
       }
    }
    return result;
}

Int main(void)
{
    cout << "A calculation of X (float) to the power of N (int) \n" ;
    float X; // The float to be raised to a power
    int   N; // Power to be raised to
    cout << "Input X (float) and N (int): ";
    cin >> X >> N;

    flaot result; // The result of the calculation
    result = pow(X,N) ;
    cout << X << " to the power " << N << " = " << result << "\n";

    return(0);
}
```

حال چند نکته ی مهم را یادآور می شویم. درباره ی توابع چیزهای زیادی برای یادآوری وجود دارد و ما تنها به بررسی سطحی می پردازیم:

- در استفاده از توابع، تعریف توابع و از جمله تابع **main** در یک فایل قرار می گیرند.(البته می توان آنها را در فایل های متفاوتی نیز قرار داد اما باعث پیچیدگی می شود.)

- تعریف توابع در بالای فایل قرار می گیرند، یعنی بعد از دستورالعمل های پیش پردازشگر [87] ر قبل از تعریف تابع **main** (تعریف توابع بعد از تابع **main** باعث پیچیدگی بیشتر می شود.)
- تعریف توابع می بایست پیش از استفاده ی آنها در دیگر توابع از جمله تابع **main** صورت گیرد. (مجددا می توانیم ترتیب دیگری را برای این مورد برگزینیم که ایجاد پیچیدگی بیشتری می کند.)
- توابعی که ما تا اینجا با آنها سروکار داشتیم، تنها می توانند یک چیز را در قالب عبارت **return expression** برگردانند.
- عبارتی که در قسمت [88] **return** نوشته می شود ، می بایست با نوع داده [89] ای که در تعریف تابع برای آن قید شده، مطابقت داشته باشد، همانند آنچه در این مثال داشتیم، ... **float pow()** پارامترهای موجود در تعریف تابع، باید شناسایی شوند ونوعشان مشخص گردد.دراین مثال، (**float x, int n**)
- متغیرهایی که در صدا کردن توابع [90] به کار می روند (در این مثال **X** و **N** در صدا کردن تابع در **main** : **(result= pow(X,N)** قابل تغییر توسط تابع نیستند. (راه دیگری برای صدا کردن توابع به همراه تغییرمتغیرها وجود دارد که در ادامه گفته خواهد شد.)
- زمانیکه تابع به محلی که در آنجا صدا شده باز می گردد، در این مورد **main** ، دیگر اثری از متغیر های موجود در لیست پارامترهای تابع و هر متغیر دیگری که توسط آن تابع شناسایی می شود و نوعش مشخص می گردد، نیست. تنها چیزی که **main** در خود نگه می دارد ارزش و مقدار **return** است. (البته باید متذکر شد که راههایی وجود دارند که جایگاه صداکننده ی تابع [91] می تواند از طریق آنها محلی [92] را که تابع چیزها را در آن به طور موقت ذخیره می کند باز نگری کند، هرچند که این کار پیچیده خواهد بود. به علاوه، این عمل تقریبا هیچگاه انجام نمی گیرد.)
- توابع خود می توانند دیگر تابع ها را نیز صدا کنند.
- با توجه به نکته ی بالا **main** نیز یک تابع دیگر محسوب می شود و می تواند توسط دیگر توابع صدا شود.
- اگر یک تابع در نهایت چیزی را بازنگرداند، نوع آن **void** گفته می شود. مثلا **void returnNothingFunction(void){}**
- اگر یک تابع هیچ پارامتری نداشت، در لیست پارامترهایش کلمه ی **void** را قرار دهید. همچنان که در مثال بالا نمونه اش را مشاهده کردید.
- نهایتا باید این نکته ی مهم را یادآور شد که توابع مبحث بسیار مهمی را به خود اختصاص می دهند. از این رو درک توابع و استفاده صحیح و بهینه از آنها مهم است.در طی این دوره نیز ما با توابع بسیار سروکار خواهیم داشت.

---

87 pre-processor directives
88 Return line
89 Data type
90 Function call
91 Calling routine
92 memory

۲.۶ توابع تعریف شده توسط کاربر[۹۳]

در این بخش به آزمودن الگوهای نوشتاری[۹۴] مربوط به توابع می پردازیم.در **ANSI C++** سه نکته در رابطه با تعریف یک تابع وجود دارد که در اینجا به آنها اشاره می کنیم:

- **نمونه ی اولیه ی تابع** [۹۵]: برای کاربردهای کوچک که کد مبدأ [۹۶]تماما دریک فایل قرار می گیرد، پیش الگو(نمونه ی اولیه)ی تابع غالبا در فضای پیش پردازشگر [۹۷](فضایی که پیش از تعریف توابع قرار دارد _قسمت پایین را ببینید.) قرار می گیرد. در اینگونه موارد چنانچه تابع پیش از صدا شدن تعریف شود، الگوی ابتدایی تابع قابل حذف است (قسمت پایین را ببینید). در موارد استفاده ی بزرگ یا مواردی که بیش از یک کد مبدأ موجود است، پیش الگوی توابع معمولا در یک**"include file"** گنجانده می شوند. مثلا این فایل را **myIncludes.h** نامگذاری می کنیم. این فایل از طریق یک عبارت **include** که در فضای پیش پردازشگر واقع شده قابل دسترسی خواهد بود.

```
#include <myIncludes.h>
```

- **تعریف تابع** : در کاربردهای کوچک، یعنی مواردی که کد مبدأ شامل یک فایل می باشد، تابع پس از نمونه ی اولیه و خارج از تابع **main** تعریف می شود. در کاربردهای بزرگ یا نمونه هایی با بیش از یک کد مبدأ، متداول است که در هر یک از فایل های مبدأ تابع یک بار تعریف شود.
- **صداکردن تابع** : صدا کردن یک تابع می تواند در **main** یا هر تابع دیگری صورت گیرد.در واقع،یک تابع می تواند توسط خودش نیز صدا گردد. این عمل "بازگشت" یا "تناوب"[۹۸] خوانده می شود که در فصول بعدی به آن خواهیم پرداخت.

درک الگوی نوشتاری این سه چیز برای اسفاده ی بهینه از توابع حایز اهمیت است.

استفاده ی کلی از توابع تعریف شده توسط کاربر بصورت زیر است :

```
/* Opening comments */
.
.
.
// Start of pre-processor directives
.
.
.

// function prototype (if required)
(type) myFunction(variable type list separated by , 's);
.
.
.
// end of pre-processor directives
.
```

[93] User-defined functions
[94] syntax
[95] the function prototype
[96] source code
[97] pre-processor area
[98] recursion

```
.
.
/* function definition */
(type) myFunction(variable type list and variables separated by , 's)
{
   .
   .
   .
   /* Function body with declarations and
      Executable statements
   */
   .
   .
   .
}
.
.
.
int main (void)
{
  // Start of main routine
  .
  .
  .
  // function usage
  ...myFunction(list of values separated by , 's)...;
  .
  .
  .
}// End of main routine
```

توضیحات کلی پیرامون توابع:

- پیش الگوی تابع در حکم شناسایی یک تابع است و تعیین می نماید که تابع در نهایت چه چیزی را تحویل می دهد (ارزش و مقدار **return** ،قسمت پایین را ببینید)، لیست پارامترها (داخل پرانتز) را مشخص می کند و شناسه [99](نام) تابع را نیز معلوم می گرداند. اگر کد مبدأ شما به کلی درون یک فایل جای گرفته و نیز اگر کامپایلر برای بار نخست که داخل کدمبدأ به مرور کدها می پردازد، هنگام مواجهه با تابع شما، با تعریف آن روبرو گردد ،نیازی به پیش الگوی تابع نیست. کامپایلر خود الگوی ابتدایی را از روی تعریف تابع تشکیل می دهد! به همین خاطر است که در اکثر موارد وقتی همه چیز در یک فایل مشابه قرار دارد، تعریف توابع در جایگاه مناسب خود مرتب شده و **main** نیز در پایان قرار گرفته، قادریم نمونه ی اولیه ی تابع را حذف کنیم. البته استثنائاتی در این باره وجود دارد که احتمالا در طول این دوره با آن مواجه نخواهیم شد. اگر نخستین مواجهه ی کامپایلر با تابع محل صداشدن آن تابع باشد، کامپایلر خود پیش الگو را تشکیل داده و فرض می کند که نوع **return** صحیح (**int**) بوده و بطور بالقوه لیست آرگومانهای تابع (پارامترهای داخل پرانتز که ورودی های تابع هم تلقی می شوند) را بهم می ریزد. ازاین رو از رویارویی با چنین وضعیتی جدا پرهیز نمایید!
- (**type**) در داخل "پیش الگوی تابع" و "تعریف تابع" کامپایلر را از نوع متغیری که تابع **myFunction** نهایتا تحویل خواهد داد آگاه می سازد. (**type**) می تواند **float**(اعشاری) یا یا انواع دیگر باشد.

```
float myFunction();
```

عبارت بالا بدین معنی است که دستاورد نهایی تابع **myFunction** از نوع اعشاری(**float**) خواهد بود.

[99] identifier

- امکان دارد که (**type**) از نوع **int** یاهریک از انواع دیگر متغیر در **C++** باشد(که در این مبحث هنوز به آن پرداخته نشده است).
- اگر (**type**) مشخص نشده باشد، کامپایلر فرض می کند که متغیر مربوطه از نوع **int** است.
- گزینه ی دیگر برای (**type**) پوچ[100] یا تهی است.پوچ بودن (**type**)بدین معنی است که تابع در نهایت مقداری را تحویل نمی دهد. (در واقع در چنین موردی تابع عملیاتی را بر روی داده ها انجام می دهد ولی عدد یا مقداری تحویل نمی دهد.)
- نوع متغیرها یا (**type**) در "نمونه ی اولیه ی تابع" و "تعریف تابع" باید با یکدیگر مطابقت نماید، در غیر این صورت، کامپایلر، شما را از وجود یک خطای نوشتاری[101] در متن برنامه تان آگاه خواهد ساخت.
- فراموش کردن تحویل دادن یک مقدار، زمانیکه (**type**) نشان می دهد که باید مقداری توسط تابع بازگردانده شود، در یک وضعیت پیش بینی نشده رخ می دهد.
- بازگرداندن یک مقدار زمانیکه (**type**) پوچ (**void**) است،به یک خطای نوشتاری منجر می شود.
- به یاد داشته باشید که

```
int main (void)
{
    .
    .
    .
}
```

یک تعریف تابع محسوب می شود! نوع خروجی آن **int** است. همچنین می توانستیم اینگونه بگوییم :

```
main ()
{
    .
    .
    .
}
```

چرا که نوع مقدار تحویلی توابع **int** فرض شده مگر اینکه نوع دیگری رابرای آن قید کنیم. یک تابع **main** ویژه در **C++** است که سیستم عملیاتی[102] آن را بعنوان محلی برای شروع پردازش می شناسد. تنها **main** چنین ویژگی را داراست. من همیشه نوع تابعی را که تعریف می کنم، مشخص می کنم و شما نیز باید چنین کاری را انجام دهید.

- لیست آرگومان[103] های پیش الگوی تابع، لیستی از انواع متغیرهاست که بوسیله ی کاما از هم تفکیک شده اند.مثل این مورد

```
void myFunction(float, int, float);
```

که نشان می دهد تابع **myFunction** به ترتیب با یک عدد اعشاری، یک عدد صحیح و یک عدد اعشاری صدا و تعریف خواهد شد.در اینجا یک مثال دیگر را مطرح می کنیم:

```
void realQuadraticRoots(float, float, float);
```

---

[100] void
[101] syntax error
[102] operating system
[103] argument list

مشخص نمودن نوع داده برای هر متغیر ناصحیحی لازم است. از این رو، مشخص ساختن نوع یک متغیر صحیح اختیاری است. هرچند پیشنهاد می شود که نوع تمامی متغیرهای موجود در لیست پارامترها معلوم گردد.

- لیست پارامترها نیز می تواند تهی(**void**)باشد.
- لیست پارامترهای داخل تعریف تابع نیز از قوانینی مشابه آنچه در مورد نمونه ی اولیه ی تابع گفته شد پیروی می کند جز اینکه در تعریف تابع متغیرها بایستی نامگذاری شده باشند. مانند:

```
void realQuadraticRoots(float a, float b, float c){/* Function definition */}
```

- صداکردن تابع (درون ساختار تابع **main** یا در متن هر تابع دیگر) همانگونه که در زیر نشان داده شده است:

```
int main (void)
{
   .
   .
   .
   realQuadraticRoots(a, b, c);
   .
   .
   .
}
```

مقادیر متغیرهای موجود در لیست پارامترها را به تابع منتقل می کند.این فرایند "فراخوانی با مقدار "[104]نامیده می شود. طریقه ی دیگری برای این عمل وجود دارد که "فراخوانی با ارجاع"[105]خوانده می شود و بعدا آن را بررسی خواهیم کرد.

- متغیر های مشخص شده در تعریف تابع (ونیزدیگر متغیهای تعریف شده توسط تابع) جزء داخلی تابع محسوب می شوند، حتی اگر نام (شناسه) آن کاملا مشابه نام دیگری در متن برنامه باشد.
- توابع باید تماما در یک بخش مجزا تعریف شوند و نمی توان تعریف یک تابع را در متن تابع دیگری گنجاند.
- زمانی که پیش الگوی تابع خارج از بدنه ی تمام توابع دیگر موجود در برنامه واقع شده باشد، در برابر همه ی موارد صداشدن تابع در فایل مبدا پاسخگو خواهد بود.
- درصورتی که الگوی ابتدایی تابع در بدنه ی یک تابع واقع شده باشد، تنها زمانیکه از درون همان تابع صدا شود پاسخگو خواهد بود.
- تابع به سه شکل نتیجه ی عملیات خود را تحویل می دهد:

۱. با به اجرا درآوردن عملیاتش درون آکولادهای احاطه کننده اش{ (این مورد فقط در مورد توابع پوچ صادق است).

۲. با ختم شدن به عبارت زیر در هر نقطه ای از بدنه ی تابع **return;** } (این مورد فقط در مورد توابع پوچ بکار می رود).

3. منتهی شدن به عبارت زیر در هر جایی از بدنه ی تابع:

```
return (an expression);
```

توابع ناتهی بایستی بدین شکل حاصل کار خود را تحویل دهند. پرانتزهای اطراف **"an expression"** اختیاری هستند. نوع این عبارت (**expression**) باید با نوع متغیری که انتظار می رود تابع تحویل دهد، مطابقت نماید.

[104] call by value
[105] call by reference

- توابع می توانند بطور مستقل توسط برنامه نویسان متفاوتی ساخته شوند. تنها اطلاعاتی که باید بر سر آنها توافق شود نوع مقدار تحویلی، نام تابع و لیست پارامترهاست. درغیراینصورت، هربرنامه نویسی می تواند در صورت رضایت به کار خود بپردازد. در یک کار گروهی تصمیم بر سر پیش الگوها بر مبنای مرحله ی طراحی صورت می گیرد. مثالی در این باره را در ادامه خواهیم دید.

۳.۶ مثال : مسئله ای در خلال کار گروهی

یک تیم متشکل چهار نفر، استاد(رئیس گروه) و سه دانشجوی فارغ التحصیل شده ی او، بیل، جین و یوجین (کارشناس ریاضیات) با یکدیگر مشغول نوشتن یک برنامه هستند.

- استاد به گروه: ما کدی برای حل معادله ی درجه دوم خواهیم نوشت! ما آن را به ثبت خواهیم رساند و من ثروتمند خواهم شد!
- تیم به استاد: بله، درسته!
- استاد به بیل: بیل، می خواهم که تو یک تابع بولین [۱۰۶] به نام **realQuadraticRoots** بنویسی که یک عدد صحیح تحویل دهد و سه عدد اعشاری (**float**) **a** ، **b** و **c** را به ترتیب بپذیرد. اگر هیچ ریشه ی حقیقی وجود نداشت، تابع تو باید مقدار ناصحیح (**false**) را بازگرداند و اگر ریشه ی حقیقی وجود داشت بایستی مقدار صحیح (**true**) را تقویل دهد.
- بیل به استاد: اِ...، استاد، بولین[۱۰۷] چیست؟؟
- استاد به بیل: نوعی از متغیر است که تنها می تواند دو مقدار داشته باشد، درست (**true**) یا نادرست (**false**). این متغیر به شکل زیر شناسایی و مقداردهی می شود:

```
bool truth0rDare = false;
```

بیل به استاد: متوجه شدم، درجریان آن هستم. ضمناً، سهم من کدام است؟

- استاد به جین: جین، می خواهم که تو یک تابع به نام **numberQuadraticRoots** بنویسی که یک عدد صحیح تحویل دهد و سه عدد اعشاری (**float**) **a** ، **b** و **c** را به ترتیب بپذیرد. من تنها زمانی روال تو را صدا خواهم کرد که روند بیل وجود ریشه های حقیقی را نتیجه دهد. عدد صحیحی که روال تابع تو بازمی گرداند باید نشاندهنده ی تعدد ریشه های حقیقی باشد،یعنی عدد "۱" یا "۲".
- جین به استاد: بسیار خوب، واضح است که چه باید بکنم.
- استاد به یوجین: یوجین، بخش ریاضیات بر عهده ی تو باشد! می خواهم که تو...
- یوجین به استاد: احتمالاً از من می خواهی که تابعی مثلاً به نام **twoQuadraticRoots** بنویسم که مقداری را برنگرداند و سه عدد اعشاری (**float**) **a** ، **b** و **c** را به ترتیب بپذیرد و عبارتی را چاپ کند که مقدار دو ریشه را مشخص کند. به این روال زمانی رجوع خواهی کرد که فرایند متعلق به جین وجود دو ریشه ی حقیقی را اطلاع دهد. درسته؟

[106] bool function
[107] bool

- استاد به یوجین : یوجین،حرف مرا قطع نکن!می خواهم که تابعی به نام **twoQuadraticRoots** بنویسی که مقداری را برنگرداند و سه عدد اعشاری (**float**) **a** ، **b** و **c** را به ترتیب بپذیرد و عبارتی را چاپ کند که مقدار دو ریشه را برایم مشخص کند. به این روال زمانی رجوع خواهم کرد که فرایند متعلق به جین وجود دو ریشه ی حقیقی را اطلاع دهد.
- استاد به گروه: کار خود را شروع کنید!
- گروه به استاد: در این کار چه چیزی برای ما وجود دارد؟
- استاد به گروه: اکنون از اعتراض کردن به من دست بکشید! این روال تابع **main** به همراه پیش طرح توابع [۱۰۸] است. شما جاهای خالی آن را پر کنید و بدین ترتیب کد من گردآوری خواهد شد و قطعاً بی کم و کاست است.

در اینجا روال تابع **main** استاد را به همراه "پیش طرح" توابع می بینیم:

```
//File: quadraticRootsStubs.cpp

#include <iostream>
//Do I need anything else, fellow programmers?

using namespace std;

//Function stubs
bool realQuadraticRoots(float a, float b, float c)
{
}
int numberQuadraticRoots(float a, float b, float c)
{
}
void twoQuadraticRoots(float A, float B, float C)
{
}

int main(void)
{
   cout << "a, b, c: ";
   float a, b, c;
   cin >> a >> b >> c;
   if (realQuadraticRoots(a,b,c) && 2 == numberQuadraticRoots(a,b,c))
   twoQuadraticRoots(a, b, c);
   return 0;
}
```

در اینجا حاصل کار تیم را می بینیم:

این کد بیل است. بخاطر دارید که استاد وظیفه ی بیل را بطور کامل مشخص نکرد و این امکان را به او داد که خود به آن بپردازد. بیل می بایست از روی یک زیرروال (زیر برنامه) [۱۰۹] راهی برای پایان دادن به برنامه بیابد از این رو باید به کتابخانه می رفت و کمی تحقیق می کرد. (استاد به ایمیل های دانشجویان فارغ التحصیل شده پاسخ نمی دهد!) بیل به کمک تابع `exit` و دیگر تعاریف ارائه شده در `cstdlib` متوجه شد که چگونه این کار را انجام دهد.

[108] function stubs
[109] subroutine

```
bool realQuadraticRoots(float a, float b, float c)
/*****************************************************************
Purpose: See if there are real roots
Receives: Quadratic constants a,b,c in a*x*x + b*x + c = 0
Returns: true if real roots, false if none
Programmer: Bill Boole (with attitude)
Start Date: 09/17/00
Remark1: Needs <iostream>, <cstdlib>
Remark2: Prof needs to rethink this thing
*****************************************************************/
{
   if (0 > (b*b - 4*a*c))
       return false;
   else if (0 == a && 0 == b)
   {
      cout << "Prof, you numbskull!\n"
    << "You didn't tell me what to do if a and b are both 0!\n"
    << "Go back and redesign the code!\n";
      exit(EXIT_FAILURE);
   }
   else
      return true;
}
```

در اینجا کد جین را با هم می بینیم. با وجود کد بیل کار جین راحت تر شده بود بدین صورت که او می دانست که حداقل یک راه حل وجود دارد.

```
int numberQuadraticRoots(float a,float b,float c)
/*****************************************************************
Purpose: See if there are real roots
Receives: Quadratic constants a,b,c in a*x*x + b*x + c = 0
Returns: 1 if one real root, 2 if two real roots
Programmer: Jane Justintime
Start Date: 09/17/00
Remarks: Enters this routine knowing that there are roots
*****************************************************************/
{
   if (0 == a || (0 == (b*b - 4*a*c)))
       return 1;
   else
       return 2;
}
```

یوجین که تصور می کند یک نابغه و نخبه ی به تمام معنااست، علاقه دارد که کارها را به شیوه ی خود انجام دهد. او می خواهد تمام مقادیری را که تابعش دریافت می کند درون متغیرهای با حروف بزرگ ذخیره کند. یوجین سبک کدنویسی ویژه ی خود را دارد. یوجین همچنین فکر می کند که باهوش تر از کامپایلر است از این رو با معادله ی درجه دو کلنجار رفت تا آن را بصورتی درآورد که فکر می کرد سریعتر عمل کند. (شما می توانید برای خود وارسی کنید و ببینید که آنچه یوجین انجام داده به لحاظ ریاضیات صحیح است.)

```
void twoQuadraticRoots(float A, float B, float C)
/*****************************************************************
Purpose: See if there are real roots
Receives: Quadratic constants a,b,c in a*x*x + b*x + c = 0
Returns: The 2 two real roots of the quadratic equation
Programmer: Eugene Dweebowski (the genius)
Start Date: 09/17/00
Remark1: Needs <cmath>
Remark2: Enters knowing that there are two real roots
Remark3: I dare ya to find something faster and better!
*****************************************************************/
{
     float twoA = 2*A, twoC = 2*C, radical = sqrt(B*B - twoA*twoC) - B;
     cout << "Solution 1: " << radical/twoA << "\n";
     cout << "Solution 2: " << twoC/radical << "\n";
```

برخی توضیحات برای کد استاد صحیح هستند.این کد ناتمام است و زمانیکه تنها یک راه حل وجود دارد، کاری انجام نمی دهد. درواقع کد او به کمی پالایش نیاز دارد تا تکمیل شود و نیز باید تصمیم بگیرد که اگر تمام ثابت [110] ها صفر بودند، چطور واکنش نشان دهد. به عنوان یک ضمیمه برای این داستان، در پروژه های کوچک پیش طرح توابع گزینه ی مناسبی برای نمونه ی اولیه ی توابع هستند. به یاد داشته باشید که پیش طرح کد استاد کامپایل خواهد شد برای اینکه او حداقل بتواند برنامه نویسی خود را چک کند. هرچند برای پروژه های بزرگتر روش پیش الگوها درنظر گرفته می شوند.

۴.۶ فراخوانی با ارزش ، فراخوانی با ارجاع ، پارامترهای ارجاع

در استفاده ی کنونی ما از صداکردن توابع، لیست پارامترها تنها شامل مقادیری است که به تابع منتقل می شوند و در متغیرهای داخلی که تنها برای همان تابع شناخته شده هستند، ذخیره می شوند که این شیوه "فراخوانی با ارزش" خوانده می شود. تابعی که به روش "فراخوانی با ارزش" صدا می گردد، بر روی داده ای که دریافت نموده عمل می کند (بعلاوه ی هرچیز دیگری که به مدد مزایای حوزه قوانین می توانند به تابع منتقل شوند) و در اکثر موارد می تواند به روند صداکننده ی خود یک مقدار را تحویل دهد، که مقدار **return** در غالب یک عبارت **return** به شکل مقابل ظاهر می شود: **return returnValue;**

نیاز به بازگرداندن بیش از یک مقدار ایجاد نارضایتی و محدودیت می کند. راه دیگر برای صداکردن توابع، راهی که توسط آن بتوان بیش از یک چیز را بازگرداند،"فراخوانی با ارجاع" نامیده می شود. این مبحث باید آغاز شود هرچند که ناچاریم مبحث آدرس متغیرها که محل ذخیره شدن آنها را در حافظه(مجازی) کامپیوتر نشان می دهد،را نیز همراه با آن مطرح نماییم.

[110] constant

۱.۴.۶ نشانی یک متغیر

در اینجا به معرفی عملگریگانی که عملگر نشانی [۱۱۱] نامیده می شود و با علامت & مشخص می شود، می پردازیم. برای مشاهده ی کاربرد آن، برنامه ی زیر را در نظر بگیرید:

```
//File: addressLocal.cpp
#include <iostream>
using namespace std;
int main(void)
{
    float x = 1.0f; // Declare and initialize a float
    double y = 3.0; // Declare and initialize a double
    int z = 2; // Declare and initialize an int
    cout << "x's value: " << x << " x's address: " << &x << "\n"
         << "y's value: " << y << " y's address: " << &y << "\n"
         << "z's value: " << z << " z's address: " << &z << "\n";
    return 0;
}
```

چند مورد تازه:

- زمانی که عملگر نشانی **&** به متغیری اعمال می شود، برای مثال **&y** ، باید آن را بدین شکل خواند: " آدرس متغیر دابل[۱۱۲] **y** " .

نتیجه ی کامپایل کردن و به اجرادرآوردن آن بصورت خروجی زیر ظاهر می شود:

```
x's value: 1 x's address: 0xbffff734
y's value: 3 y's address: 0xbffff72c
z's value: 2 z's address: 0xbffff728
```

- زمانیکه دستور **cout** آدرسی را چاپ می کند،خروجی بصورت **0x...** یعنی در مبنای شانزده ارائه می گردد.حروف مقدم **0x** تنها به این منظور ظاهر می شوند که بگویند آنچه در ادامه آمده در مبنای شانزده است.
- به یاد داشته باشید که در "پشته[۱۱۳]" آدرس از یک آدرس بزرگ به آدرس های کوچکتر تبدیل می شود.

بخاطر داشته باشید که سیستم عملیاتی (در این مورد تأسیسات لینوکس[۱۱۴]) اولین متغیری را که با آن روبرو می شود (در این مورد یک عدد اعشاری(**float**) با اندازه ی **32** بیت) در آدرس **0xbffff734** قرار می دهد. (این آدرس هنگامی که به یک عدد صحیح تبدیل شود، چیزی به بزرگی سه بیلیون خواهد بود.) این آدرس در واحد بایت ارائه شده،یک بایت معادل هشت بیت است.یک ماشین **32**بیتی دارای یک فضای آدرس مجازی به اندازه ی 2^{32} یا **4,294,967,296** بایت می باشد که اغلب به اختصار ۴ گیگابایت در نظر گرفته می شود. بالای این آدرس، آغاز شونده در **0xbffff738** در این مورد (این مرز و محدودیت وابسته به ماشین و سیستم عملیاتی است)جایی است که "محیط برنامه" مستقر شده (درادامه به آن خواهیم پرداخت) و بالای آن هنوز محلی برای اسکان سیستم عملیاتی است (شالوده) که نواحی بالاتر فضای آدرس را تا سقف ۴ گیگابایت اشغال می کند.

[111] address operator
[112] double
[113] stack
[114] Linux installation

متغیر بعدی، از نوع دابل است که اندازه ی آن ۶۴ بیت یا ۸ بایت برآورد می شود. به یاد داشته باشید که آدرس آن کوچکتر است (در مقایسه با ۸ بایت). فضای آدرس که برای متغیرهای محلی[۱۱۵] فراهم شده،"چارچوب پشته[۱۱۶]" نامیده می شود و دارای رشدی رو به پایین می باشد.درنظر بگیرید که متغیر بعدی در کجا واقع می شود.

هر تابعی چارچوب پشته ی خود را بمنظور قراردادن متغیرهای محلی خود درون حافظه و رها کردنشان به هنگام خاتمه یافتن عملیات تابع، ایجاد می کند. این مفهوم با "نابودشدن" پشته زمانی که تابع خاتمه می یابد مشترک است. فراخوانی بعدی تابع این فضا را برای چارچوب پشته ی خود استفاده خواهد کرد. برنامه ی قابل اجرا یعنی "پیمانه ی بارشو[۱۱۷]" در انتهای فضای آدرس قرار دارد و از بایت صفر شروع می شود. این ناحیه "بخش متن[۱۱۸]" نامیده می شود. بالای بخش متن متغیرهای سراسری مقداردهی شده و مقداردهی نشده ذخیره شده اند. در بالای این بخش **"heap space"**[۱۱۹] قرار گرفته که در واقع محلی از فضای آدرس است که حافظه ی (حافظه ای که توسط برنامه تخصیص داده می شود) آن جا واقع شده و روند رشدی رو به بالا دارد. اگر برنامه ها و داده های شما به قدری بزرگ هستند که چارچوب پشته و فضای **heap** با یکدیگر برخورد می کنند، بایستی برنامه ی خود را طوری طراحی کنید که فضای کمتری اشغال کند و یا کامپیوتر ۶۴بیتی گرانتری بخرید که دارای فضای ماتریسی[۱۲۱] به اندازه ی $۲^{۶۴}$ بایت باشد.(ظرف چند سال آتی،اکثر کامپیوترها ۶۴بیتی خواهند بود.)

در اینجا مثالی از یک برنامه ی دیگر را می بینیم که ثابت می کند متغیرهای سراسری در محل متفاوتی در حافظه واقع می شوند:

```
//File: addressGlobal.cpp
#include <iostream>
using namespace std;
double y = 3.0; // Declare and initialize a global double
int z = 2; // Declare and initialize a global int
int main(void)
{
    float x = 1.0f; // Declare and initialize a float
    cout << "x's value: " << x << " x's address: " << &x << "\n"
  << "y's value: " << y << " y's address: " << &y << "\n"
  << "z's value: " << z << " z's address: " << &z << "\n";
    return 0;
}
```

کامپایل کردن و اجرا کردن آن خروجی زیر را نتیجه می دهد:

```
x's value: 1 x's address: 0xbffff734
y's value: 3 y's address: 0x8049a70
z's value: 2 z's address: 0x8049a78
```

[115] local variables
[116] stack frame
[117] load module
[118] text segment
[۱۱۹] خانه های حافظه که یک برنامه می تواند جهت عملیات محاسباتی خود آنها را اشغال نموده و سپس خالی کند.
[120] dynamic memory
[121] array space

توجه داشته باشید که چگونه متغیرهای سراسری (در این مورد **y** و **z**) در آغاز یک نشانی پایینتر واقع شده اند**0x8049a70** (چیزی درحدود یک بیلیون). درنظربگیرید که پشته ی سراسری چکونه بزرگ می شود و نیز در خاطر داشته باشید که متغیر دابل ، y ، چگونه هشت بایت را اشغال می کند.

۲.۴.۶ فراخوانی با ارزش دربرابر فراخوانی با ارجاع

برای لحظه ای تصور کنید که **pow()** وجود ندارد وما می خواهیم برنامه ای بنویسیم که مربع و مکعب یک عددصحیح را محاسبه نماید. بعلاوه، درنظرداریم که تابع منحصر به فردی بنویسیم که هر دو مقدار مربع و مکعب را بازگرداند .با تکیه بر آنچه که تاکنون فراگرفته ایم، مأیوس می شویم و پی می بریم که قادر به انجام چنین کاری نیستیم! درعوض بایستی که این عمل را ازطریق دو تابع جداگانه به انجام برسانیم. کد ما چیزی مشابه کد زیر خواهد بود:

```
//File: returnTwice.cpp
#include <iostream>
using namespace std;
int square(int j)
{
   cout << "In function square(), j is located at " << &j << "\n";
   return j * j;
}
int cube(int k)
{
   cout << "In function cube(), k is located at " << &k << "\n";
   return k * k * k;
}

int main(void)
{
   int i = 3, iSquared, iCubed;
   cout << "\n\nIn main():\n"
<< " i is located at " << &i << "\n"
<< "iSquared is located at " << &iSquared << "\n"
<< " iCubed is located at " << &iCubed << "\n\n"
<< "Before the call to square and cube:\n"
<< " i = " << i << "\n"
<< "iSquared = " << iSquared << "\n"
<< " iCubed = " << iCubed << "\n\n";
   iSquared = square(i);
   iCubed = cube(i);
   cout << "\n After the call to square and cube:\n"
<< " i = " << i << "\n"
<< "iSquared = " << iSquared << "\n"
<< " iCubed = " << iCubed << "\n\n";
   return 0;
}
```

درمتن این کد تعداد زیادی عبارات **cout** به چشم می خورد که هدف از آنها مشخص ساختن محل ذخیره شدن متغیرها بوده.پس از کامپایل کردن و اجرا کردن این کد نتیجه بصورت خروجی زیر خواهد بود:

```
In main():
i is located at 0xbffff734
iSquared is located at 0xbffff730
iCubed is located at 0xbffff72c
Before the call to square and cube:
i = 3
iSquared = 134520020
iCubed = 134514715
In function square(), j is located at 0xbffff728
In function cube(), k is located at 0xbffff728
After the call to square and cube:
i = 3
iSquared = 9
iCubed = 27
```

توجه داشته باشید که متغیرهای محلی ( ) **main** یعنی **i**،**iSquared**،**iCubed** در چارچوب پشته ی **main()** مستقر می شوند. همچنین دقت کنید که **iSquared** و **iCubed** توسط **main()** مقداردهی نشده اند بنابراین موقعیت این متغیرها در حافظه شامل بیت های بی معنی و خارج از منطق می باشد که مانند داده های ناخواسته ای هستند که از پردازش و استفاده ی قبلی آن مکانهای حافظه باقی مانده اند. نخستین تابعی که صدا شد **square()** است که یک متغیر محلی به نام **j** دارد و آن را در چارچوب پشته ی خود قرار می دهد که در آدرس **0xbffff728** راه اندازی می شود. **square()** با موفقیت کار خود را انجام می دهد نتیجه ی محاسبات خود را به **main()** بازمی گرداند که بعد از آن تابع **cube()** را صدا می کند. **cube()** یک متغیر محلی به نام **k** دارد که آن را در چارچوب پشته ی خود مستقر می سازد که در آدرس **0xbffff728** راه اندازی می شود، همان آدرسی که توسط **square()** استفاده گردید! زمانی که **square()** به **main()** بازگردانده شد، چارچوب پشته ی آن تخریب شد **cube()** در چنین شرایطی مؤثرترین کار را انجام داد، یعنی چارچوب پشته ی خود را در جایگاه در دسترس بعدی قرار داد، جایگاهی که به تازگی از سوی **square** تخلیه شده بود. در اینجا به معرفی روش "فراخوانی با ارجاع" می پردازیم. این مطلب با ارائه ی یک مثال به بهترین وجه قابل توضیح خواهد بود. در اینجا همان برنامه ی بالا را با تبدیل به یک بار فراخوانی تابع که هر دو مقدار مذکور را با هم تحویل می دهد و علاوه بر این دارای یک مورد تازه نیز می باشد: عوامل ارجاج در لیست پارامترهای تابع.

```
//File: returnTwoThings.cpp
#include <iostream>
using namespace std;
void squareCube(int j, int& j2, int& j3)
{
   cout << "In function squareCube():\n"
 << "j is located at " << &j << "\n"
 << "j2 is located at " << &j2 << "\n"
 << "j3 is located at " << &j3 << "\n";
   j2 = j * j;
   j3 = j2 * j;
   return;
}

int main(void)
{
```

```
   int i = 3, iSquared, iCubed;
   cout << "\n\nIn main():\n"
        << " i is located at " << &i << "\n"
 << "iSquared is located at " << &iSquared << "\n"
 << " iCubed is located at " << &iCubed << "\n\n"
       << "Before the call to squareCube:\n"
 << " i = " << i << "\n"
 << "iSquared = " << iSquared << "\n"
 << " iCubed = " << iCubed << "\n\n";
   squareCube(i, iSquared, iCubed);
   cout << "\nAfter the call to squareCube:\n"
 << " i = " << i << "\n"
 << "iSquared = " << iSquared << "\n"
 << " iCubed = " << iCubed << "\n\n";
   return 0;
}
```

مکانهای حافظه ای متعلق به متغیرهایی که برای **main** محلی محسوب می شوند و نیز این حقیقت که **squareCube()** تصور می کند j2 و j3 در همان مکان ساکنند را بخاطر بسپارید. توجه کنید که در **squareCube()** متغیر محلی **j** در آدرس **0xbffff720** یعنی ۱۲ بایت پایین تر از انتهای چارچوب پشته ی **main()** واقع شده است. اما **j** به عنوان یک عدد صحیح تنها ۴ بایت ظرفیت اشغال می کند. آیا می توانید حدس بزنید که **squareCube()** از خانه های ۸ بایتی حافظه چه استفاده ای می کند؟

در انتها به یاد داشته باشید که عوامل مقداری و پارامترهای ارجاع می توانند به هر ترتیبی در لیست پارامترها با هم درآمیزند. تنها اطمینان حاصل کنید که در تعریف تابع نام تمامی متغیرهای ارجاعی پس از امپرسند (علامت &) ظاهر شده باشند.

تنها مثالی دراین باره که در زندگی حقیقی نیز وجود دارد رابطه ی من با دلال سهام شرکتم است. من روال **main()** هستم و او، آن ولگرد بی کفایت، تابعی به نام **stockBroker()** است. در اینجا تعریف تابع او را می بینید.حال به شما این امکان را می دهم که دریابید چگونه در بورس سهام عمل می کنم.

```
//File: stockBrokerFunction.cpp
void stockBroker(double myFee, double& alexMoney)
{
   double transactionCost = 200.;
   while (0 < alexMoney && 0 < myFee)
   {
      myFee = myFee - transactionCost; //Deduct usual transaction charge
      alexMoney = alexMoney * 0.95; //Make a stupid investment
   }
   return;
}
```

هرچند می خواهم بدانید که من به او دستمزدی دادم، **myFee** همان چیزی است که او باید از آن نگهداری کند. اگرچه او با پول من بازی می کند، **alexMoney**، یک پارامتر ارجاعی است و مقدار چیزی را معلوم می کند که با ناراحتی کاهش سریع آن را اطلاع می دهم. آن پول هنوز پول من است اما من به دلال این توانایی را داده ام که آن را مبادله کند.

۵.۶ قوانین حوزه ی عملیاتی

یکی از قدرتمندترین ویژگیهای C++ پیاده سازی قوانین حوزه ی عملیاتی است. دلیل این امر اینست که اطلاعات (داده ها) بایستی از کد عملیاتی پنهان شوند مگرآنکه این کد واقعاً نیاز داشته باشد درمورد داده ها بداند. به این شکل از خطر تغییر داده ای که نباید توسط برنامه تغییرداده شود، جلوگیری می شود_حتی اگر برنامه تمام متغیرها را با نام یکسان صدا بزند! بدین ترتیب این امکان فراهم می شود که افراد مختلفی بتوانند قسمتهای گوناگون یک برنامه را بر عهده بگیرند و در مدتی که هر برنامه نویس به وظایف خود می پردازد، داده تخریب نمی شود. C++ همچنین مفهوم **namespace** را بمنظور تفکیک چیزها معرفی می کند.در طول این دوره ما همواره از **using namespace std;** استفاده می کنیم.اگرچه امکان استفاده از **namespace** دیگری نیز وجود دارد اما ما تنها از **namespace** استاندارد استفاده خواهیم کرد.

سلسله مراتب های متعددی از حوزه ی عملیاتی وجود دارد:

حوزه ی عمیاتی فایل [122]:شناسه ای که خارج از هر تعریف تابعی شناسانده شده باشد، دارای حوزه ی عمیاتی فایل می باشد. متغیرهای "سراسری"، تعریف توابع و نمونه ی اولیه ی توابع (اگر مورد استفاده ی شما قرار گیرند!) که در خارج از تمامی توابع واقع شده اند دربرابر کل فایل پاسخگو خواهند بود.بسیار عالی است که یک متغیر صحیح را بعنوان مثال بیرون از همه ی توابع یا **main** تعریف کنید. این متغیر در ادامه می تواند در تمامی توابع از جمله **main** مورد استفاده قرار گیرد. اگر فایلهای جداگانه ای بهم پیوسته باشند تا یک برنامه را تشکیل دهند، فایلهای دیگر درمورد شناسه های فایل حوزه ی عملیاتی خارج از فایل خود اطلاعی نخواهند داشت.

حوزه ی عملیاتی تابع : تنها شناسه ای که برای کل تابع تعریف شده برچسب (مطلب) [123] است (شناسه ای که با یک نشانه ی کلن ":" همراه است) مثلاً:

```
ThisIsALabel:
```

انتقال به یک **label** می تواند توسط **goto** یا عبارت **switch** صورت گیردکه هیچکدام در طی این دوره تدریس نخواهند شد.

برچسبها برای کل تابع تعریف شده و خارج از تعریف نشده به حساب می آیند.

حوزه ی عملیاتی بلوک [124] یک "بلوک" هر چیزی است که داخل آکولاد {...} قرار گیرد.برای مثال:

```
{ // This is a block
}

{ // This is another separate block
}

{ // This is a "yet another block"
{ // This is still another block nested

// within "yet another" block
}
}
```

به کاربرد اختیاری فاصله بندی به منظور مشخص نمودن بلوکهای جداازهم در مثال بالا توجه کنید.

[122] file scope
[123] label
[124] block scope

ما مثالهای متعددی از "بلوک" ها را دیده ایم: بلوکهای توابع، بلوکهای **if/else if/else**، بلوکهای **while/do while/for**.
ما همچنین می توانیم بلوک متعلق به خودمان را با احاطه کردن عبارات بوسیله ی آکولادها{...} به منظور تعریف بخشی از کد که حوزه ی جداگانه ی خود را دارد، ایجاد کنیم.

حوزه ی عملیاتی تودرتو[۱۲۵] به شکل خاصی عمل می کند. متغیرهایی که خارج از یک بلوک حوزه ی عملیاتی تعریف شده اند به درون یک بلوک حوزه ی عملیاتی تودرتو انتقال داده خواهند شد مگر آنکه این بلوک تودرتو متغیری با شناسه ی مشابه را تعریف کرده باشد.در این مورد متغیر بلوک خارجی "پنهان" شده تا زمانیکه بلوک درونی خاتمه یابد.مثالهایی از آن را خواهیم دید.

حوزه ی عملیاتی پیش الگوی تابع : یک راه برای تعریف نمونه ی اولیه ی تابع دربرگرفتن شناسه ی متغیرهاست. برای مثال:

```
int myFunction(float a, float b, float c);
```

عبارت بالا نمونه ی اولیه ی یک تابع است که سه شناسه ی **a**، **b** و **c** را نامگذاری می کند.این شناسه ها تنها درون پیش الگوی تابع شناخته شده اند.برنامه نویسان اغلب آن دسته از متغیرها را نامگذاری می کنند که انتظار دارند در یک نمونه ی اولیه ی تابع این مسئله که چه متغیرهایی در چه ترتیبی قرار می گیرند را بخاطر بسپارند.اسامی این متغیرها خارج از عبارت پیش الگوی تابع توسط کامپایلر نادیده گرفته می شوند.

اینها همگی بسیار گیج کننده بنظر می رسند مگر آنکه به حل چند مثال پیرامون آن ها بپردازیم. پس بیایید چند نمونه از آن را باهم ببینیم!

کد زیر را با نام **scope0.cpp** در نظر بگیرید:

```
//File: scope0.cpp
#include <iostream>
using namespace std;
void myFn(float x, float y) // MyFn declares floats x and y
{
   float z; // Internal to myFn
   cout << "myFn: pre-op x = " << x << ", y = " << y << "\n";
   z = x; x = y; y = z; // Variable switch
   cout << "myFn: post-op x = " << x << ", y = " << y << "\n";
}

int main(void)
{
   float x = 1.0, y = 2.0; // Internal to main
   cout << "main: pre-call x = " << x << ", y = " << y << "\n";
   myFn(x, y); // Main gives values to myFn
   cout << "main: post-call x = " << x << ", y = " << y << "\n";
   return 0;
}
```

کامپایل کردن و اجرا نمودن **scope0.cpp** بصورت خروجی زیر نتیجه می شود:

```
main: pre-call x = 1, y = 2
myFn: pre-op x = 1, y = 2
```

[125] nested scope

```
myFn: post-op x = 2, y = 1
main: post-call x = 1, y = 2
```

main دو متغیر **float x** و **y** را معرفی نموده و مقادیر آنها را به **myFn** تحویل می دهد.**myFn** نیز این متغیرها را با نام مشابهی صدا می زند (این نامها همچنین می توانستند متفاوت باشند). وظیفه ی **myFn** تعویض مقادیر این دو متغیر است. این تعویض مقادیر درون حوزه ی **myFn** انجام شده است اما تأثیر این عمل در **main** به چشم نمی خورد. این پیشامد بدین خاطر است که **x** و **y** درون **main** با **x** و **y** درون **myFn** متفاوت است.

کد **scope1.cpp** راکه در ادامه آمده است ملاحظه کنید:

```
//File: scope1.cpp
#include <iostream>
using namespace std;
float x = 1.0, y = 2.0; // Global variables
void myFn(void) // No declarations
{
   float z; // Internal to myFn
   cout << "myFn: pre-op x = " << x << ", y = " << y << "\n";
   z = x; x = y; y = z; // Variable switch
   cout << "myFn: post-op x = " << x << ", y = " << y << "\n";
}

int main(void)
{
   cout << "main: pre-call x = " << x << ", y = " << y << "\n";
   myFn(); // Main just calls myFn
   cout << "main: post-call x = " << x << ", y = " << y << "\n";
   return 0;
}
```

پس از کامپایل و اجرای کد **scope1.cpp** نتیجه بصورت خروجی زیر خواهد بود:

```
main: pre-call x = 1, y = 2
myFn: pre-op x = 1, y = 2
myFn: post-op x = 2, y = 1
main: post-call x = 2, y = 1
```

در این مثال **x** و **y** بطور سراسری تعریف شده اند(خارج از تمام توابع). هم **main** و هم **myFn** می توانند به آنها دسترسی داشته باشند و تغییرشان دهند.وظیفه ی **myFn** تعویض مقادیر این دو متغیر است.در داخل حوزه ی عملیاتی **myFn** این عمل صورت گرفته و این تعویض مقادیر در **main** نیز تأثیر گذاشته است. این رویداد بدین خاطر است که **x** و **y** در **main** همان **x** و **y** در **myFn** است.

کد **scope2.cpp** را که در زیر آمده را ملاحظه نمایید:

```
//File: scope2.cpp
#include <iostream>
using namespace std;
float x = 1.0, y = 2.0; // Global variables
void myFn(float x, float y) // myFn declares x,y
{
   float z; // Internal to myFn
   cout << "myFn: pre-op x = " << x << ", y = " << y << "\n";
   z = x; x = y; y = z; // Variable switch
   cout << "myFn: post-op x = " << x << ", y = " << y << "\n";
```

```
}

int main(void)
{
   cout << "main: pre-call x = " << x << ", y = " << y << "\n";
   myFn(x,y); // Main transfers x and y values
   cout << "main: post-call x = " << x << ", y = " << y << "\n";
   return 0;
}
```

با کامپایل کردن و اجرا کردن آن حاصل بصورت زیر خواهد بود:

```
main: pre-call x = 1, y = 2
myFn: pre-op x = 1, y = 2
myFn: post-op x = 2, y = 1
main: post-call x = 1, y = 2
```

در این مثال **x** و **y** بطور سراسری تعریف شده اند(خارج از تمام توابع). **main** به آنها دسترسی دارد و قادر است تغییرشان بدهد. اما **x yFn** و **y** متعلق به خود را تعریف می کند و از این رو متغیرهای سراسری پنهان گردیده و **myFn** نمی تواند به آنها دسترسی پیداکرده و تغییری رویشان اعمال نماید.

به کد **scope3.cpp** را در ادامه توجه کنید:

```
//File: scope3.cpp
#include <iostream>
using namespace std;
int main(void)
{
   float x = 1.0, y = 2.0, z = 0.0; // Local to main
   cout << " main: pre-call x, y, z = "
        << x << ", " << y << ", " << z << "\n";
   { // Start of block
      float z = 42.0; // Local to {} block
      cout << "block: pre-call x, y, z = "
           << x << ", " << y << ", " << z << "\n";
      z = x; x = y; y = z; // Variable switch
      cout << "block: post-call x, y, z = "
           << x << ", " << y << ", " << z << "\n";
   } // End of block
   cout << " main: post-call x, y, z = "
        << x << ", " << y << ", " << z << "\n";
   return 0;
}
```

پس از کامپایل و اجرای کد مذکور خروجی زیر نتیجه می شود:

```
main: pre-call x, y, z = 1, 2, 0
block: pre-call x, y, z = 1, 2, 42
block: post-call x, y, z = 2, 1, 1
main: post-call x, y, z = 2, 1, 0
```

در این مثال **x**، **y** و **z** برای **main** بصورت محلی تعریف شده اند.درون **main** یک بلوک تودرتو وجود دارد که حوزه ی مخصوص به خود را تعریف می کند.از آنجا که این بلوک **x** و **y** را تعریف نمی کند، این متغیرها وارد بلوک می شوند. اما **z** درون بلوک تودرتو تعریف شده و Z خارجی پنهان شده است.به هرحال زمانی که کنترل به دست **main** داده شد،مقادیر **x** و **y** توسط بلوک تودرتو تغییر یافته است.

به کد **scope4.cpp** در زیر توجه کنید:

```
//File: scope4.cpp
#include <iostream>
using namespace std;
void myFn(void)
{
   float z = 0; // Internal to myFn
   cout << "myFn: pre-op z = " << z << "\n";
   z = 1; // Reset z
   cout << "myFn: post-op z = " << z << "\n";
}
int main(void)
{
   cout << "First call to myFn\n";
   myFn(); // Main calls myFn
   cout << "Second call to myFn\n";
   myFn(); // Main calls myFn again
   return 0;
}
```

پس از کامپایل و اجرای این کد حاصل را بصورت خروجی زیر مشاهده نمایید:

```
First call to myFn
myFn: pre-op z = 0
myFn: post-op z = 1
Second call to myFn
myFn: pre-op z = 0
myFn: post-op z = 1
```

در این مثال **z** برای **myFn** محلی محسوب می شود و **myFn** مقدار اولیه ی صفر را به آن داده است. هربار که **myFn** فراخوانده می شود، **z** مجدداً مقدار صفر را دارا می گردد و عبارات قابل اجرا پردازش می گردند.

حال کد **scope5.cpp** را در نظر بگیرید:

```
//File: scope5.cpp
#include <iostream>
using namespace std;
void myFn(void)
{
   static float z = 0; // Internal to myFn
   cout << "myFn: pre-op z = " << z << "\n";
   z = z + 1; // Add to z
   cout << "myFn: post-op z = " << z << "\n";
}
int main(void)
{
   cout << "First call to myFn\n";
   myFn(); // Main calls myFn
   cout << "Second call to myFn\n";
```

```
   myFn(); // Main calls myFn again
   return 0;
}
```

پس از کامپایل کردن و اجرا نمودن نتیجه را بصورت زیر می بینیم:

```
First call to myFn
myFn: pre-op z = 0
myFn: post-op z = 1
Second call to myFn
myFn: pre-op z = 1
myFn: post-op z = 2
```

در این مثال **z** برای **myFn** محلی تعریف شده و **myFn** مقدار اولیه ی صفر را نیز به **z** داده است اما آن را ثابت[۱۲۶] نگاه می دارد.این ثابت تصریح کننده نشان می دهد که این متغیر باید زمانی که یک روند را ترک می کند مقدار خود را ثابت نگه دارد.در واقع به این شکل مقدار خود را برای ورود بعدی خود به روال حفظ می کند.مقداردهی اولیه(مقدار صفر) تنها برای بار نخست که روال آغاز می شود مؤثر است.

متغیر ایستا (**static**) یک کلاس ذخیره سازی[۱۲۷] نامیده می شودو در واقع یک توصیف کننده برای نوع متغیر به حساب می آید.این بدان معنی است که مقدار آن اندوخته می شود نه اینکه با به پایان رسیدن اجرای تابع ازبین رود.

کد **scope6.cpp** را در ادامه درنظر بگیرید:

```
//File: scope6.cpp
#include <iostream>
using namespace std;
float max(float u, float v) // function max declares u,v
{
   float temp = u;
   u = v;
   v = temp;
   cout << "max: u = " << u << " v = " << v << "\n";
   if (u > v)
   {
      return u;
   }
   else
   {
      return v;
   }
}

int main(void)
{
   float x = 1.0, y = 2.0, maxxy; // Internal to main
   cout << "main: x = " << x << " y = " << y << "\n";
   maxxy = max(x, y);
   cout << "main: max of x and y = " << maxxy << "\n";
   return 0;
}
```

[126] static
[127] storage class

پس از کامپایل و اجرای کد مذکور خروجی زیر را می بینیم:

```
main: x = 1 y = 2
max: u = 2 v = 1
main: max of x and y = 2
```

main دو متغیر اعشاری(**float**) **x** و **y** را تعریف می کند و مقادیر آنها را به **myFn** تحویل می دهد.**myFn** این متغیرها را با نامهای متفاوتی فرامی خواند (همانگونه که در برخی مثالهای قبل دیدیم این نامها می توانند یکسان نیز باشند). وظیفه ی **myFn** این است که مقادیر این دو متغیر را تعویض نموده و تعیین کند که مقدار کدام یک بزرگتر است.درون حوزه ی خود **myFn** این فرایند صورت گرفته اما تعویض مقادیر در **main** تأثیرگذار نبوده. و دلیل این پیشامد این است که **x** و **y** در**main** با **u** و **v** در **myFn** متفاوت است.

۶.۶ مسائل

۱. یک سئوال انشایی دیگر

الف) از قول خود در ۵۰ کلمه یا کمتر، شرح دهید که در هنگام ایجاد یک تابع چه زمانی روش "فراخوانی با ارجاع" بر روش "فراخوانی با ارزش" ترجیح دارد.

ب) سپس مثالی از یک تابع را که به روش ارجاع فراخوانده شده باشد ارائه دهید که در آن تابع قادر است عملی را انجام دهد که از طریق فراخوانی با ارزش ممکن نیست.

ج) برای هریک از موارد زیر یک مثال ارائه دهید:

- اعلان اولیه ی تابع
- فراخوانی تابع

– تعریف تابعی، با یک بلوک عبارت خالی که برای مثال خود می نویسید.

۲. توابع ساده

در روال **main** زیر دقیقاً همانطور که نشان داده شده است تایپ کنید و تعاریف دو تابع **cubeit0** و **cubeit1** را اضافه کنید که هرکدام مکعب آرگومان(شناسه)**x** را به دست دهد.توجه کنید که نمی توانید از تابع **pow()** در کتابخانه ی ریاضیات استفاده کنید.

```
#include <iostream>
//Function cubeit0 goes here
//Function cubeit1 goes here
int main(void)
{
   cout << "Input a number to be cubed: ";
   double x;
   cin >> x;
   cout << cubeit0(x) << endl;
   double result;
   cubeit1(x,result);
   cout << result << endl;
   return 0;
}
```

در اینجا یک مثال را می بینید که طریقه ای را که انتظار می رود این برنامه کار کند، نشان می دهد:

```
% a.out
Input a number to be cubed: 3
27
27
%
```

۳. اشکالات[۱۲۸] را رفع کنید

الف) چرا کد زیر کامپایل نمی شود و یا آنگونه که انتظار می رود کار نمی کند؟ اشکال(ها)آن را برطرف نمایید.

```
#include <iostream>
using namespace std;
void sumInts(int a, int b)
{
   int i;
   i = a + b;
   return i;
}
int main()
{
   int i = 1, j = 2;
   cout << "The sum of ints i = " << i << " and j = " << j << " is: "
        << sumInts(i,j) << "\n";
   return 0;
}
```

[128] bug

ب) چرا کد زیر کامپایل نمی شود و/یا آنگونه که انتظار می رود کار نمی کند؟ اشکال(ها)آن را برطرف نمایید.

```
#include <iostream>
#include <cmath>
using namespace std;
float trigs(float y, &f1, &f2)
{
   f1 = sin(y); f2 = cos(y);
   return 0;
}
int main(void)
{
   float x = 0.0, cosx, sinx;
   int returnValue = trigs(x, sinx, cosx);
   cout << "sin(" << x << ") = " << sinx << ", "
        << "cos(" << x << ") = " << cosx << "\n";
   return 0;
}
```

۴. خروجی را پیش بینی کنید

این کد چه چیزی را چاپ می کند؟

```
#include <iostream>
using namespace std;
void cyclic(int i, int& j, int k, int& l)
{
   int m;
   m = i;
   i = j;
   j = k;
   k = l;
   l = m;
}
int main(void)
{
   int i = 1, j = 2, k = 3, l = 4;
   cyclic(i, j, k, l);
   cout << "i,j,k,l : " << i << j << k << l << "\n";
   return 0;
}
```

۵. خطا(ها)ی طراحی را اصلاح نمایید.

الف) هدف از این کد تعریف تابعی است که ماکسیمم دو عدد اعشاری [۱۲۹]را تحویل دهد و این دو عدد را به صورت نزولی مرتب کند (عدد بزرگتر در رده ی اول، عدد کوچکتر در رده ی دوم). این کد فقط بعضی از مواقع کار می کند. در این کد یک اشتباه طراحی وجود دارد که برخی اوقات به نتایج نادرست می انجامد. فرض کنید که عبارت **scanf** همواره درست اجرا شود. شما نمی توانید این کد را با استفاده از تابعی متعلق به یک کتابخانه که بتواند مقدار ماکسیمم را تعیین کند، تصحیح کنید.

[129] **float**

```
#include <iostream>
using namespace std;
float fMax(float f1, float f2)
{
   if (f2 > f1)
   {
      float temp = f2; // f2 is maximum
      f2 = f1; f1 = temp; // switch them
      return f1;
   }
   else
      return f1; // f1 must be maximum, no need to switch
}

int main(void)
{
   cout << "Input 2 float's: ";
   float a, b;
   cin >> a >> b;
   cout << "Maximum is: " << fMax(a,b) << "\n";
   cout << "Descending order is: " << a << ", " << b << "\n";
   return 0;
}
```

ب) کد زیر بدون خطا کامپایل می شود. چرا این کد آنگونه که انتظار می رود کار نمی کند؟

```
#include <iostream>
using namespace std;
void sort3(int i, int j, int k)
{
   int t;
   if (i > j) {t = i; i = j; j = t;}
   if (j > k) {t = j; j = k; k = t;}
   if (i > j) {t = i; i = j; j = t;}
}
int main(void)
{
   int i = 5, j = 3, k = 1;
   sort3(i,j,k);
   cout << "In ascending order (smallest->largest) "
        << "the three int's are "
        << i << ", " << j << ", " << k << "\n";
   return 0;
}
```

ج) کد زیر بدون خطا کامپایل می شود. چرا این کد آنگونه که انتظار می رود کار نمی کند؟

```
#include <iostream>
using namespace std;
float squareCube(float x, float xx)
{
   xx = x*x;
   return xx*x;
}
int main(void)
{
```

```
   float x = 2.0, xx, xxx;
   xxx = squareCube(x,xx);
   cout << "Square of x: " << xx << ", Cube of x is " << xxx << "\n";
   return 0;
}
```

۶. ریشه های معادله ی درجه دو

تابعی بنویسید که با صداکردن تابع زیر سازگار باشد:

```
bool twoRealRoots = twoRoots(a,b,c,root1,root2);
```

این تابع چنانچه **b2-4ac>0** و **a≠0** باشد، واژه ی **true** را بازمی گرداند. درغیراینصورت واژه ی **false** را برمی گرداند.اگر تابع واژه ی **true** را تحویل داد،آنگاه **root1** و **root2** حاوی ریشه های معادله ی درجه دو **ax + bx + c = 0** خواهند بود که بصورت زیر می باشند:

۷.تابع $\sqrt{}$ خود را طراحی کنید.

الگوریتم زیر می تواند در محاسبه ی $x = \sqrt[2]{a}$ مورداستفاده قرار گیرد:

(الف) با یک حدس اولیه شروع کنید: x = a

(ب) حدس را تصحیح کنید: x'= (x+a/x)/2

(ج) اگر x' و x یکسان هستند، $x = \sqrt[2]{a}$ ،درغیراینصورت x = x' قرار دهید و به مرحله ی ۲ بروید.

تابعی بنویسید که $x = \sqrt[2]{a}$ را با روش بالا محاسبه کند.

۸. مختصات قطبی[۱۳۰]

تابعی بنویسید که با روال **main** زیر همخوانی داشته باشد:

```
#include <iostream>
#include <cmath>
#include <string>
using namespace std;
//Function definition goes here
int main(void)
```

[130] polar coordinates

```
{
   float x, y, r, t;
   cout << "x, y: ";
   cin >> x >> y;
   cout << "x = " << x << " y = " << y << " is in the " << polar(x,y,r,t)
        << ". " << "r = " << r << " t = " << t << " degrees\n";
   return 0;
}
```

تابع **polar()** که در دومین عبارت **cout** در بالا صدا شده،یک رشته [131] را باز می گرداند که نشاندهنده ی موارد زیر است:

- "ربع دایره ی بالا و سمت راست" اگر که x>=0 و y>=0
- " ربع دایره ی بالا و سمت چپ " اگر که x<0 و y>=0
- " ربع دایره ی پایین و سمت چپ" اگر که x<0 و y<0
- " ربع دایره ی پایین و سمت راست " در صورت نبود شرطهای بالا

پس از آنکه تابع نتیجه ی خود را تحویل داد، عدد اعشاری **r(float)** دربرگیرنده ی شعاع است که بصورت زیر محاسبه شده است $r = \sqrt{x^2 + y^2}$

راهنمایی: **r = sqrt(x*x + y*y);** و نیز عدد اعشاری t حاوی زاویه به درجه است که بدین شکل حساب شده است $\theta =$ 180 $\tan^{-1}(y/x)/\pi$. از رابطه ی t = 45*atan2(y,x)/atan(1.0); برای محاسبه ی آن استفاده کنید. در اینجا مثالی از چگونگی کارکرد آن را می بینید:

```
red% a.out
x, y: 1 1 //User types in "1 1" in response to the "x, y: " prompt
x = 1 y = 1 is in the upper right quadrant. r = 1.41421 t = 45 degrees
```

۹. یک عدد صحیح بدون علامت[132] چند بیت دارد؟

الگوریتم زیر می تواند در تعیین تعداد بیتهایی که در ارائه ی یک عددصحیح بدون علامت مصرف می شود، مورداستفاده قرار گیرد.

الف) با یک عدد صحیح بدون علامت که دارای مقدار اولیه ی ۱ است شروع کنید، به طور مداوم آن را در عدد ۲ ضرب کنید تا زمانی که حاصل برابر صفر گردد.

ب) تعداد دفعاتی که شما آن را در ۲ ضرب نمودید مربوط است به تعداد بیت های مصرف شده در ارائه ی آن عدد صحیح بی علامت.

تابعی بنویسید که تعداد بیت های مصرف شده در بیان یک عدد صحیح بدون علامت را تعیین کند و برنامه را روی کامپیوتر اجرا نمایید.

تابع شما باید با روال **main** زیر سازگار باشد.

```
#include <iostream>
using namespace std;
```

[131] string
[132] unsigned int

```
//Function definition goes here
int main(void)
{
   cout << "The unsigned int size is " << intSize()
         << " bits on this computer\n";
   return 0;
}
```

خروجی:

```
Here is how it works:
red% a.out
The unsigned int size is 32 bits on this computer
```

۷.۶ پروژه ها

۱.توابع کتابخانه ی ریاضیات

جدول زیر برخی از توابع متداول در کتابخانه ی ریاضیات C++ را فهرست می کند که می توان با بکارگیری عبارت **#include <cmath>** در فضای پیش پردازشگر[133] در یک فایل C++ به آنها دسترسی یافت.

Math function	Purpose	Example usage
log(x), loge(x), ln(x)	natural logarithm (base e)	`double x = 1.; double y = log(x);`
exp(x), ex	exponential	`double x = 1.; double y = exp(x);`
log10(x)	logarithm (base 10)	`double x = 1.; double y = log10(x);`
sin(x)	sine of x	`double x = 1.; double y = sin(x);`
cos(x)	cosine of x	`double x = 1.; double y = cos(x);`
√x	square root of x	`double x = 1.; double y = sqrt(x);`
|x|	absolute value of x	`double x = 1.; double y = fabs(x);`

به یاد داشته باشید که آرگومان(کمان) توابع مثلثاتی در بالا بایستی بر حسب رادیان باشند.

360° (degrees) = 2π (radians).

یک برنامه ی اصلی بنویسید که این جدول را مشابه آنچه که در صفحه ی بعد نشان داده شده گسترش دهد. به عبارت دیگر، شما بایستی از یک **for-loop** کمک بگیرید که به شکلی اعداد دابل 1.0 ,..., -0.9, -1.0 را تولید نموده،حساب کند و خروجی های **x**، **log(x)**، **exp(x)**، **log10(x)**، **sin(PI*x)**، **cos(PI*x)**، **sqrt(x)** و **fabs(x)** را نتیجه دهد.

چند نکته:

- به آرگومان **sin()** و **cos()** توجه کنید. روشی برای تولید مقدار خوبی برای π در پروژه ی "والدو کجاست؟ " در فصل قبل به شما داده شد.
- شما بایستی از توابع **setw()** و **steprecision()** همانگونه که در فصل ۵ توضیح داده شد استفاده کنید.
- برای ایجاد امکان چاپ دنباله ی تکراری صفرها به یک مورد دیگر نیاز خواهید داشت.پیش از نخستین استفاده ی خود از عبارت **cout** عبارت زیر را قرار دهید

```
cout.setf(ios::fixed);
```

- توجه داشته باشید که خروجی های **-Infinity** و **NaN** زمانیکه آرگومان متعلق به توابع تعریف شده در کتابخانه ی ریاضیات خارج از دامنه ی تعریف شده قرار گیرند، بصورت خودکار توسط دستور **cout** برونداد می شود.

```
 x     log(x)    exp(x)    log10(x) sin(PI*x) cos(PI*x) sqrt(x)  fabs(x)
---- --------- -------- --------- --------- --------- -------- -------
-1.0 -Infinity 0.367879 -Infinity -0.000000 -1.000000      NaN   1.0
-0.9 -Infinity 0.406570 -Infinity -0.309017 -0.951057      NaN   0.9
-0.8 -Infinity 0.449329 -Infinity -0.587785 -0.809017      NaN   0.8
-0.7 -Infinity 0.496585 -Infinity -0.809017 -0.587785      NaN   0.7
-0.6 -Infinity 0.548812 -Infinity -0.951057 -0.309017      NaN   0.6
-0.5 -Infinity 0.606531 -Infinity -1.000000  0.000000      NaN   0.5
-0.4 -Infinity 0.670320 -Infinity -0.951057  0.309017      NaN   0.4
-0.3 -Infinity 0.740818 -Infinity -0.809017  0.587785      NaN   0.3
-0.2 -Infinity 0.818731 -Infinity -0.587785  0.809017      NaN   0.2
-0.1 -Infinity 0.904837 -Infinity -0.309017  0.951057      NaN   0.1
```

[133] Preprocessor

```
0.0 -Infinity 1.000000 -Infinity  0.000000  1.000000 0.000000   0.0
0.1 -2.302585 1.105171 -1.000000  0.309017  0.951057 0.316228   0.1
0.2 -1.609438 1.221403 -0.698970  0.587785  0.809017 0.447214   0.2
0.3 -1.203973 1.349859 -0.522879  0.809017  0.587785 0.547723   0.3
0.4 -0.916291 1.491825 -0.397940  0.951057  0.309017 0.632456   0.4
0.5 -0.693147 1.648721 -0.301030  1.000000  0.000000 0.707107   0.5
0.6 -0.510826 1.822119 -0.221849  0.951057 -0.309017 0.774597   0.6
0.7 -0.356675 2.013753 -0.154902  0.809017 -0.587785 0.836660   0.7
0.8 -0.223144 2.225541 -0.096910  0.587785 -0.809017 0.894427   0.8
0.9 -0.105361 2.459603 -0.045757  0.309017 -0.951057 0.948683   0.9
1.0  0.000000 2.718282  0.000000  0.000000 -1.000000 1.000000   1.0
```

۲. تلاش برای دستیابی به کمال

یک عدد "کامل" عدد صحیح و مثبتی است که برابر مجموع "مقسوم علیه های سره" (تمام مقسوم علیه هایش به جز خود عدد) خود باشد. مجموعه ی مقسوم علیه های عدد **N** شامل خود **N** نمی شود،اما شامل عدد **1** می شود.برای مثال:مقسوم علیه های سره ی شش ، **1،2** و **3** هستند و **6=3+2+1**.بنابراین **6** یک عدد کامل است.در حقیقت **6** نخستین عدد کامل است.دومین عدد کامل **28=1+2+4+7+14** است.از میان اعداد کامل **4** تای نخست برای ریاضیدانان یونان باستان شناخته شده بود.تا به امروز **37** عدد کامل شناخته شده است!درباره ی اعداد کامل اطلاعات زیادی بدست آمده است،اما موارد بیشتری هنوز بصورت یک معما در مورد این اعداد باقی مانده است.بعنوان مثال موارد زیر شناخته شده اند:

آ. بزرگترین و آخرین عدد کامل کشف شده را می توان بصورت

$$2^{3021376} \times (2^{3021377} - 1)$$

نوشت. این شامل **1,819,050** رقم دهدهی است.

ب. تمام اعداد کامل شناخته شده زوج هستند.

ج. تمام اعداد کامل شناخته شده توسط فرمول اقلیدس :

$$N = 2^{k-1}(2k-1)$$

داده می شوند که درآن **k** عددی صحیح و مثبت است و 2^k-1 نیز عددی اول است. این عدد گونه ی ویژه ای از عدد اول است که عدد اول "مرسن[۱۳۴]" نامیده می شود.ایده ی بسیار خوبی است که از این واقعیت در طراحی برنامه ی کامپیوتری خود که در زیر توضیح داده شده، بهره گیرید.

در مورد اعداد کامل چیزهای زیادی دریافته شده است. اما درستی یا نادرستی موارد زیر هنوز معلوم نگردیده است:

- بی شمار عدد کامل وجود دارد.
- همه ی اعداد کامل زوج هستند.
- تمامی اعداد کامل بوسیله ی فرمول اقلیدس ارائه می شوند.

کار شما نوشتن برنامه ایست که موارد زیر را برآورده نماید:

[134] Mersenne

از میان اعداد کامل **5**تای نخست را یافته و مقسوم علیه های سره ی آن را چاپ کند.

برنامه ی شما باید حداقل از دو تابع که به **main** افزوده شوند،تشکیل شده باشد.یکی از آنها بایستی در صورتی که آرگومان صحیح آن عددی کامل باشد واژه ی **true** را بازگرداند و از تابع دوم نیز انتظار می رود که مقسوم علیه های سره ی آن عدد کامل را چاپ کند.اگر مایل باشید می توانید بیش از ۲ تابع تعریف کرده و به برنامه ی خود بیفزایید.

در کمتر از یک دقیقه از زمان واحدپردازش مرکزی[۱۳۵] بر روی یک کامپیوتر عادی و مدرن اجرا را تکمیل کنید.

خروجی هایی مشابه آنچه که در ادامه آمده ایجاد کنید:

```
% a.out
6 is a perfect number
6 = 1 + 2 + 3
28 is a perfect number
28 = 1 + 2 + 4 + 7 + 14
.
.
.
```

۳. مجتمع سازی[۱۳۶] تصادفی

(الف) شما در ادامه به این مورد نیاز پیدا خواهید کرد!

تابعی بنویسید که سه متغیر دابل (**x** ، x_0 و x_1 در زیر) و عدد دابل (**f(x,x0,x1)**درزیر) را بازگرداند برای تابع:

$$f(x, x_0, x_1) = \frac{x \sin(x)}{1+x} \quad \forall \quad x_0 \le x \le x_1$$

اگر **x** خارج از دامنه ی تعریف قرار گیرد،واژه ی **NaN** بایستی بازگردانده شود.

راهنمایی : **NaN** در واقع روشی برای بیان عبارت "**Not a Number**" در مؤسسه ی مهندسین الکتریسیته و الکترونیک[۱۳۷] است.

NaN می تواند زمانی که نتیجه ی محاسبات با استفاده از روشهای ارائه ی **floating-point** تعریف نشده است، تولید گردد. برای مثال کد زیر:

```
double zero = 0;
cout << zero/zero;
```

تولید یک **NaN** می کند.می توانید امتحان کنید!

راهنمایی۱: اینجا یک روال **main** است که می توانید از آن برای آزمودن تابع خود استفاده کنید:

```
#include <iostream>
#include <cmath>
// The function should go here...
int main(void)
```

[135] CPU(central processing unit)
[136] integration
[137] IEEE(Engineers Institute of Electrical and Electronics)

```
{
   cout << "Input x, x0, x1: ";
   double x, x0, x1;
   cin >> x >> x0 >> x1;
   cout << "Evaluating the function at x = " << x
        << " between the limits x0 = " << x0
        << " and " << x1 << "\n";
   cout << "f(" << x << ") = " << myFunction(x, x0, x1) << "\n";
   return 0;
}
```

(ب) روشی تصادفی برای یافتن یک فضا

۱.تابعی بنویسید که موقعیت x_{max} ، را که محل رخ دادن ماکسیمم تابع **f(x)** است را بیابد.این مقدار ماکسیمم بین دو کران x_0 و x_1 واقع می شود که در آن $x_1>x_0$ است. مقداری که برای x_{max} تعیین می کنید، بایستی در خطای مجاز $10^{-12}/|x_1-x_0|$ صدق کند. شما می توانید فرض کنید که تابع تنها دارای یک ماکسیمم محلی در بین دو کران x_0 و x_1 است به صورتی که بین x_0 و x_1، $f(x) \geq 0$ است و تابع ثابت نیست.

راهنمایی صفر: برای کار روی راه حل خود از متغیر دابل استفاده کنید، نه **float**.

راهنمایی ۱: کار آسان شده چرا که تنها یک ماکسیمم محلی بین x_0 و x_1 وجود دارد.این بدان معنی است که شما می توانید با هر **x** ای مانند $x_0 \leq x \leq x_1$ آغاز نمایید و اندکی در یک جهت پیش بروید، مثلا، **x-δ** که در آن **δ** عددی کوچک و مثبت است. اگر تابع در آنجا کوچتر است، یعنی، **f(x-δ)<f(x)** باشد شما دارید آزمایش برای ماکسیمم را در جهت اشتباه انجام می دهید، بنابراین **x+δ** را امتحان کنید.

راهنمایی۲: کار سخت شده است زیرا خطای مجاز بسیار کوچک است. این ممکن نیست که شما دلتا (**δ**) ی اولیه ی خود را با $10^{-12}/|x_1-x_0|$ تطبیق دهید. شما باید با مقداری به مراتب بزرگتر این فرایند را آغاز کنید، مثلا، $.10^{-1}/|x_1-x_0|$ هنگامی که جایگاه ماکسیمم را در این محدوده ی خطای مجاز پیدا کردید، آن را به $10^{-2}/|x_1-x_0|$ کاهش دهید و به همین ترتیب ادامه دهید. این تکنیک پالایش نموی[۱۳۸] معقول و کارآمد است. این تنها یک پیشنهاد است. شما احتمالاً خواهید توانست روشی به مراتب مؤثرتر برای انجام این کار بیابید.

۲. تابع دیگری بنویسید که مساحت زیر **f(x)** که بین دو کران x_0 و x_1 واقع می شود را با استفاده از تکنیکی بنام **نمونه گیری تصادفی**[۱۳۹] تعیین کند.روش نمونه گیری تصادفی در کلاس موردبحث قرار خواهد گرفت.قبل از اینکه بتوانید مساحت زیر تابع را تعیین کنید،نیاز خواهید داشت که جایگاه ماکسیمم تابع را بدانید.

۳. مسئله را با استفاده از تابع زیر تکمیل کنید:

$$f(x) = \frac{x \sin(x)}{1+x}$$

[138] Incremental refinement technique
[139] random sampling

برای محدوده ی $0 \le x \le \pi$ که بیشتر آن در مسئله ی قبل کدگذاری شد، **100,000** نمونه را بکارگیرید تا مساحت را تخمین بزنید.

چگونگی کار بر روی این مسئله:

۱.چند نمونه ی متفاوت که پاسخ آنها را می دانید(بدون دانستن حساب دیفرانسیل و انتگرال!) برای **f(x)** امتحان کنید.موارد زیر می توانند گزینه های مناسبی برای این منظور باشند:

A. $f(x) = x$, for $0 \le x \le 1$.
B. $f(x) = 1 - x$, for $0 \le x \le 1$.
C. $f(x) = x$, for $0 \le x \le 1/2$, $f(x) = 1 - x$, for $1/2 \le x \le 1$.

۲.زمانیکه اطمینان حاصل کردید که الگوریتم شما کار می کند،تابع مقدار دهی شده ی :

$$f(x) = \frac{x \sin(x)}{1 + x}$$

را برای محدوده ی $0 \le x \le \pi$ امتحان کنید.

۳.شبه کد[۱۴۰] ی به تفصیل بنویسید که در طراحی برنامه ی خود از آن استفاده کنید.

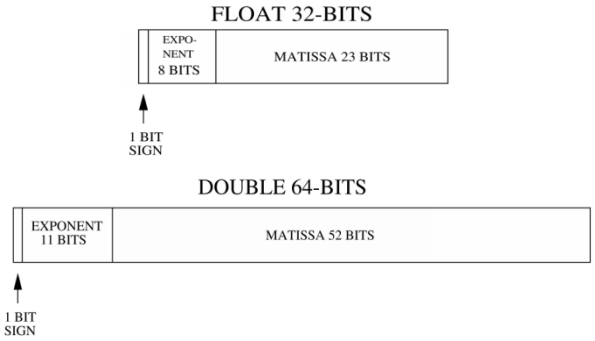
[140] pseudocode

فصل هفتم

بحثی بیشتر در مورد نوع و تجرید داده[۱۴۱]

۱.۷ نمایش اعداد حقیقی با ممیز شناور[۱۴۲]

دیدیم که نوع داده های **int** با ۳۲ بیت فقط دامنه ی محدودی رو دارند و اعداد محدودی را می توانند در خود ذخیره کنند. یک عدد صحیح با علامت دامنه ی اعداد $-2^{31} \leq i \leq (2^{31}-1)$ را در بر می گیرند. در اعداد اعشاری با ممیز شناور هم با محدودیت هایی روبرو هستیم. حتما باید یک بیت را به علامت عدد اختصاص دارد. **23** بیت برای نشان دادن قسمت اعشاری عدد[۱۴۳] بکار می رود و **8** بیت برای نشان دادن توان. در نتیجه **float** دارای محدوده ی تغییرات 10^{-45} تا 10^{+38} می باشد. (تقریبا دارای قدرت تفکیک ۱ قسمت در 10^{7} قسمت.) برای نوع داده های **double 64** بیت استفاده می شود. **15** بیت برای توان استفاده می شودو دارای محدوده ی تغییرات 10^{-4932} تا 10^{+4932} است. (تقریبا دارای قدرت تفکیک **1** در 10^{19}) البته باید توجه کرد که این نمایش در کامپایلر ها و سیستم عامل های مختلف کمی تغییر کند.



برنامه ی زیر را در نظر بگیرید:

```
//File: precisionFloat.cpp
#include <iostream>
using namespace std;
float precision(void)
{
   float one = 1.0f, e = 1.0f, onePlus;
   int counter = 0;
   do
   {
      counter = counter + 1;
      e = e/2.0f;
      onePlus = one + e;
```

[141] Data Abstraction
[142] Floating Point
[143] mantissa

```
   }while(onePlus != one);
   cout<< "Converged after " << counter << " iterations\n";
   return e;
}

int main(void)
{
   cout<< "Float resolution = " << precision() << "\n";
   return 0;
}
```

خروجی نمونه:

```
Converged after 24 iterations
Float resolution = 5.96046e-08
```

در برنامه ی بالا، **float variable = value;** یک متغیر از نوع **float** را تعریف می کند.

```
//File: precisionDouble.cpp
#include <iostream>
using namespace std;
double precision(void)
{
   double one = 1.0, e = 1.0, onePlus;
   int counter = 0;
   do
   {
      counter = counter + 1;
      e = e/2.0;
      onePlus = one + e;
   }while(onePlus != one);
   cout<< "Converged after " << counter << " iterations\n";
   return e;
}

int main(void)
{
   cout<< "Double resolution = " << precision() << "\n";
   return 0;
}
```

خروجی نمونه:

```
Converged after 53 iterations
Double resolution = 1.11022e-16
```

```
//File: precisionLongDouble.cpp
#include <iostream>

using namespace std;

long double precision(void)
{
   long double one = 1.0L, e = 1.0L, onePlus;
   int counter = 0;
```

```
   do
   {
      counter = counter + 1;
      e = e/2.0L;
      onePlus = one + e;
   }while(onePlus != one);
   cout<< "Converged after " << counter << " iterations\n";
   return e;
}

int main(void)
{
   cout<< "Long double resolution = " << precision() << "\n";
   return 0;
}
```

خروجی نمونه:

```
Converged after 64 iterations
Long double resolution = 5.42101e-20
```

برای تست کردن محدوده ی تغییرات float برنامه ی زیر را در نظر بگیرید:

```
//File: rangeFloat.cpp
#include <iostream>
#include <iomanip>

using namespace std;

int main(void)
{
   float x = 1.0f, x0;
   float y = 1.0f, y0;
   int counter = 0;
   do
   {
      counter = counter + 1;
      x0 = x; x = x*2.0f; // Multiply by 2
      y0 = y; y = y/2.0f; // Divide by 2
      cout<<setw(4) << counter << ": "
          <<setw(12) << x << " :: "
          <<setw(12) << y << "\n";
   }while(x0 != x || y0 != y);
   return 0;
}
```

خروجی نمونه:

```
1: 2 :: 0.5
2: 4 :: 0.25
3: 8 :: 0.125
4: 16 :: 0.0625
5: 32 :: 0.03125
.
.
.
125: 4.25353e+37 :: 2.35099e-38
126: 8.50706e+37 :: 1.17549e-38
127: 1.70141e+38 :: 5.87747e-39
128: Inf :: 2.93874e-39
129: Inf :: 1.46937e-39
```

```
130: Inf :: 7.34684e-40
.
.
.
147: Inf :: 5.60519e-45
148: Inf :: 2.8026e-45
149: Inf :: 1.4013e-45
150: Inf :: 0
```

چند نکته ی جدید:

۱. **Inf** ساختار استاندارد **IEEE**[۱۴۴] برای نشان دادن عدد بسیار بزرگ است. **Inf** الگوی بیتی خود را دارد.

۲. دیده می شود که محدوده ی **float** برای اعداد کوچک از لحاظ توان **10** بیشتر است.

```
//File: rangeDouble.cpp
#include <iostream>
#include <iomanip>

using namespace std;

int main(void)
{
   double x = 1.0, x0;
   double y = 1.0, y0;
   int counter = 0;
   do
   {
      counter = counter + 1;
      x0 = x; x = x*2.0; // Multiply by 2
      y0 = y; y = y/2.0; // Divide by 2
      cout<<setw(4) << counter << ": "
          <<setw(12) << x << " :: "
          <<setw(12) << y << "\n";
   }while(x0 != x || y0 != y);
   return 0;
}
```

خروجی نمونه:

```
1: 2 :: 0.5
2: 4 :: 0.25
3: 8 :: 0.125
4: 16 :: 0.0625
5: 32 :: 0.03125
.
.
.
1021: 2.24712e+307 :: 4.45015e-308
1022: 4.49423e+307 :: 2.22507e-308
1023: 8.98847e+307 :: 1.11254e-308
1024: Inf :: 5.56268e-309
1025: Inf :: 2.78134e-309
.
.
.
1073: Inf :: 9.88131e-324
```

[144] Institute of Electrical and Electronics Engineers

```
1074: Inf :: 4.94066e-324
1075: Inf :: 0
```

برای آزمایش محدوده ی **double** برنامه ی زیر را اجرا کنید:

```
//File: rangeLongDouble.cpp
#include <iostream>
#include <iomanip>

using namespace std;

int main(void)
{
   long double x = 1.0L, x0;
   long double y = 1.0L, y0;
   int counter = 0;
   do
   {
      counter = counter + 1;
      x0 = x; x = x*2.0L; // Multiply by 2
      y0 = y; y = y/2.0L; // Divide by 2
      cout<<setw(4) << counter << ": "
          <<setw(12) << x << " :: "
          <<setw(12) << y << "\n";
   }while(x0 != x || y0 != y);
   return 0;
}
```

خروجی نمونه:

```
1: 2 :: 0.5
2: 4 :: 0.25
3: 8 :: 0.125
4: 16 :: 0.0625
5: 32 :: 0.03125
.
.
.
16382: 2.97433e+4931 :: 3.3621e-4932
16383: 5.94866e+4931 :: 1.68105e-4932
16384: Inf :: 8.40526e-4933
16385: Inf :: 4.20263e-4933
.
.
.
16443: Inf :: 1.45808e-4950
16444: Inf :: 7.2904e-4951
16445: Inf :: 3.6452e-4951
16446: Inf :: 0
```

۱.۱.۷ یک مثال جالب از تفاوت بین محاسبات تئوری و محاسبات عددی رایانه ای

در حالت تئوری داریم :

$$\sum_{1}^{N} \frac{1}{N} = 1$$

حال به نتیجه ی برنامه ی زیر که در واقع همان محاسبه ی مجموع بالاست توجه کنید:

```
//File: oneNotOne.cpp
#include <iostream>
#include <iomanip>
using namespace std;

int main(void)
{
   cout<< "N? ";
   double N;
   cin>> N;
   float fraction = 1.0/N;
   cout<< "Will compute " << fraction << " * " << N << " = ";
   float sum = 0;
   for (int i = 1; i <= N; i = i + 1)
      sum = sum + fraction;
   cout<<setprecision(19) << sum << "\n";
   return 0;
}
```

خروجی کد بالا نشان می دهد که حاصل مجموع به ازای ورودی های **N** کوچک، تقریبا یک و به ازای ورودی های **N** بزرگ، حاصل به یک تفاوت خواهد داشت. در واقع دلیل این خطا در محاسبه، محدود بودن خانه های رایانه در نگه داری داده هاست. در واقع گرد شده ی اعداد با هم جمع می شوند. به این خطای ناشی از گرد کردن در محاسبات عددی، خطای برشی گرد کردن [۱۴۵] می گویند. در صورتی که نوع داده ها را به **double** یا نوع داده ی بهتری تبدیل کنیم، این دقت محاسبه بهتر خواهد شد. اما درمان نخواهد شد! چرا که هنوز هم ظرفیت نوع داده ها، محدود هستند. در عوض با استفاده از **double** فضای بیشتری از حافظه اشغال می شود. با بالا رفتن دقت اعداد، سرعت محاسبات نیز کاهش می یابد.

2.7 نوع داده ی کاراکتری

هر کاراکتر در زبان **C++** می تواند توسط نوع داده ی **char** مشخص شوند. در واقع **char** نوع داده ای است که فقط و فقط یک خانه را شامل می شود. اعلان[۱۴۶] یک متغیر کاراکتری نیز مانند اعلان نوع داده های دیگر است:

```
char ch;
```

کد زیر مقدار **'a'** را در خانه ی کاراکتری **ch** قرار می دهد.

```
ch = 'a';
```

توجه کنید که تک کاراکتر ها با علامت تک کوتیشن ' و ' مشخص می شوند. در زیر عمل مقدار دهی و اعلان متغیر با هم انجام می گیرند:

```
char ch = 'a';
```

[145] Round-off error
[146] Declaration

مقدار دهی زیر برای یک کاراکتر اشتباه است و خطای ساختاری[۱۴۷] محسوب می شود.

```
ch = "a";
```

برخی از کاراکتر های خاص بصورت زیر هستند:

- '\n' برای ایجاد خط جدید در نوشته
- '\t' برای ایجاد یک **tab** افقی
- '\0' به عنوان کاراکتر **Null** یا خالی در نظر گرفته می شود. این کاراکتر کاربرد های خاصی در **C++** دارد. در ادامه نمونه هایی از این کاربرد ها را خواهیم دید.

نکته را که باید در مورد خانه های **char** باید متذکر شد، این است که در حقیقت این واحد ها، خانه های **8** بیتی هستند که از $2^8=256$ مقدار مختلف را می توانند در بر بگیرند. لذا می توان مقادیر آنها را با شماره ی اعداد نیز مشخص کرد:

```
char c;
int i;
i = 65;
c = i;
c = 99;
```

برنامه ی زیر را در نظر بگیرید:

```
//File: chars.cpp
#include <iostream>
using namespace std;
int main(void)
{
   for(int i = 0; i < 256 ; i = i + 1)
   {
      char c = i;
      if (i <= 127)
         cout<< "Int " << i << "-->" << c << " in ASCII\n";
      else
         cout<< "Int " << i << "-->" << c << " beyond ASCII\n";
   }
   return 0;
}
```

با اجرای این کد کاراکتر های استاندارد **ASCII**[۱۴۸] که متناظر با اعداد **0** تا **127** هستند و سایر کاراکترها را در خرویجی مشاهده خواهید کرد. برخی از خانه ها، کارهای خاصی را انجام می دهند. برای مثال عدد **7** متناظر با آلارم یا زنگ است! عدد **8**، متناظر با **Back-Space** است. عدد **9** متناظر با **tab** است. عدد **10** متناظر با خط جدید و عدد **13** متناظر با "سر سطر" است.

در استاندارد **UNICODE** کاراکترها را در **2** بایت(**16** بیت) با $2^{16} = 65536$ حالت مختلف کد می کند. این مقدار فضا نه تنها برای ذخیره سازی علایم تمامی زبان های حال حاضر دنیا کافی است بلکه در این استاندارد علایم زبان تصویری مصر باستان نیز کد شده است

[147] Syntax Error
[148] ASCII = American Standard Code for Information Interchange

۱.۲.۷ رشته های کاراکتری

کلاس رشته ها را می توان با اضافه کردن دستور **#include <string>** استفاده کرد.

شما تا اینجا از جنین دستوراتی استفاده کرده اید:

```
cout<< "This is a string!\n";
```

در واقع شما در این دستور نیز بطور ناخود آگاه از رشته های کاراکتری برای نمایش و چاپ مطلبی استفاده کرده اید. حال می خواهیم با کار با این نوع داده ها بیشتر آشنا شویم. برای معرفی و مقدار دهی به یک متغیر رشته ای، این چنین عمل می کنیم:

```
string str = "This is a string!\n";
```

در حقیقت باید گفت رشته ی کاراکتری، آرایه ای از کاراکتر هاست که به دنبال هم قرار گرفته اند. **str[0]** همان **"T"** و **str[1]** همان **"h"** است و... .

برای خواندن رشته های کاراکتری از ورودی، دستورات زیادی وجود دارند. دستور **cin** رشته ی کاراکتری را می خواند تا جایی که به فضای خالی(فاصله) و یا سطر جدید برسد.

به برنامه ی زیر توجه کنید:

```
//File: string1.cpp
#include <iostream>
#include <string>

using namespace std;

int main(void)
{
   string response;
   do
   {
      cin>> response;
      cout<< response + "\n";
   }while ('.' != response[0]); //Note vector notation
   return 0;
}
```

در برنامه ی بالا، شما خواهید دید که تابع cin رشته ها را تا جایی که فاصله ای وجود ندارد، از ورودی گرفته و ادامه ی اطلاعات در حافظه ی رایانه باقی می ماند، تا زمانی که تابع **cin** اطلاعات جدیدی را از ورودی بخواند. این عمل تکراری تا جایی ادامه می یابد که تابع cin، تمامی مقادیر ورودی وارد شده را بخواند. مهم است که بدانید که فاصله در یک رشته ی کاراکتری به معنای علامت پایان یک رشته ی کاراکتری است. گاهی مواقع که شما بخواهید فاصله ها را هم به عنوان ورودی بگیرید، ممکن است این مساله به عنوان عاملی مزاحم باشد. برای رفع این مشکل می توانید از دستور **getline()** از کلاس **<string>** استفاده کنید. برای آشنایی با نحوه ی کار با این تابع به برنامه ی زیر توجه کنید.

```
//File: string2.cpp
#include <iostream>
#include <string>
```

```
using namespace std;

int main(void)
{
   string sentence;
   do
   {
      getline(cin,sentence);
      cout<< sentence + "\n";
   }while ('.' != sentence[0]);
   return 0;
}
```

حال به معرفی برخی از توابع خانواده ی کلاس **string** می پردازیم:

- تابع **length()** طول یک داده ی رشته ای را بر می گرداند.
- تابع **substr()** قسمتی از یک داده ی رشته ای را جدا می کند و آن را به عنوان خروجی بر می گرداند.
- متغیر های رشته ای را می توان توسط عملگر جمع به هم الحاق شوند. در این صورت حداقل یکی از عملوند باید خود متغیر رشته ای باشند. به مثال زیر توجه کنید:

```
//File: string3.cpp
#include <iostream>
#include <string>

using namespace std;

int main(void)
{
   cout<< "Type in a string: ";
   string r1;
   cin>> r1;
   cout<< "Your response was " << r1.length() << " chars long\n";

   cout<< "Type in another string: ";
   string r2;
   cin>> r2;
   cout<< "Your response was " << r2.length() << " chars long\n";

   cout<< "Putting them together: " << r1 + " " + r2 << "\n";

   cout<< "Slicing and splicing them up: "
       <<r1.substr(0,2) + r2.substr(4,7) << "\n";

   return 0;
}
```

3.7 کلاس vector

۱.۳.۷ نوع داده ی vector

تا اینجا تمامی متغیر هایی که از آنها استفاده کرده ایم، به صورت تنها بوده اند. در واقع از نوع داده هایی استفاده کرده ایم که از تمامی فضای آنها، در آن واحد به عنوان یک مقصود واحد استفاده کرده ایم. حال می خواهیم با نوع داده هایی کار کنیم که شامل خانه های

زیاد و گروهی باشد. الزام تعریف این نوع داده، به خاطر مسائلی است که ممکن است با آنها مواجه شویم. برای مثال تعریف نوع داده های زیاد و یا با تعداد نامعلوم، با استفاده از اعلان متغیرها به صورت عادی غیر ممکن است. بنابرین ساختاری ایجاد کرده ایم که بتوانیم مجموعه ای از خانه های پشت سر هم را به هر تعداد که بخواهیم تعریف کنیم. در ساختار های موجود در C++ این طول خانه ها، می تواند دینامیک[۱۴۹] باشد. یعنی طول بتوان طول خانه ها را در طول اجرای برنامه متناسب با نیاز زیاد یا کم کرد. یا اینکه در حالت طول استاتیک[۱۵۰] این طول در ایتدای برنامه بصورت مقداری ثابت تعریف شده و تا آخر نیز ثابت می ماند.

۲.۳.۷ اعلان و نحوه ی استفاده از اعضای کلاس vector

در نظر بگیرید که می خواستم، متغیر های زیادی را با اندیس گذاری در نام آنها تعریف کنم:

```
int gradeForStudent1 = 98;
int gradeForStudent2 = 79;
int gradeForStudent3 = 88;
.
.
.
int gradeForStudent217 = 91;
```

مشخص است که انجام این کار، بیهوده است! چرا که تکرار این همه اعلان کار دشواری است. از طرفی دیگر اگر در جایی تعداد متغیر هایی که با آنها سر و کار داریم، خیلی بیشتر از این تعداد بود یا اینکه تعداد آنها مشخص نبود، چه باید می کردیم؟!

برای اجرای همان کاری که در بالا انجام داده ایم با استفاده از **vector** ها، به این صورت عمل می کنیم:

```
vector<int> gradeForStudent(217);
```

این دستور در واقع، مجموعه ای از **217** خانه ی **int** را به دنبال هم را تعریف می کند. باید متذکر شویم که **vector** برای هر نوع دلخواه از داده ها، بنویسیم. برای دسترسی به مقادیر خانه های هر کدام از خانه های **vector** باید از شمارنده ی اندیس آن که عددی صحیح بزرگتر از صفر است، استفاده کنیم. باید توجه کنیم که این شماره در **C++** به طور قراردادی از صفر شروع خواهد شد. برای مثال:

```
gradeForStudent[0] = 0;
gradeForStudent[1] = 20;
…
gradeForStudent[20] = -10;
```

در صورتی که قاعده ی مشخصی برای وارد کردن ورودی ها به **vector** داشته باشید، می توانید از حلقه های تکرار برای پر کردن مقادیر داخل **vector** ها استفاده کنید:

```
#include <iostream>
#include <vector>
.
.
.
const int NumberOfStudents= 217;
vector<int> gradeForStudent(NumberOfStudents);
for(i = 0; i < gradeForStudent.size(); i = i + 1)
```

[149] Dynamic
[150] Static

```
{
   cout << "Input grade for student number << i << ": ";
   cin >> gradeForStudent[i];
}
```

در اینجا باید به چیزی اشاره کنیم که اصل بحث آن در درس های آینده است. اما چون از آن مجبوریم به مقدار کمی استفاده کنیم، باید آن را معرفی کنیم. در اصل **vector** ها دارای کلاسی[۱۵۱] به این نام در **C++** هستند. یک کلاس دارای متغیر ها و توابعی برای کارهای خاص خود است. در واقع هر کلاس توصیف کننده ی شیئی[۱۵۲] با ویژگی های خاص است که این ویژگی ها توسط متغیرها و توابع آن کلاس تعریف می شوند. برای دسترسی به آن دسته از متغیر ها یا توابع که عضو آن کلاس بوده و بر روی آن شی عمل می کنند، می توانیم از "." استفاده کنیم.[۱۵۳] برای مثال تابع **size()** تابع عضو[۱۵۴] کلاس **vector** است که به ازای هر شی، تعداد اعضای متناظر با آن را در خروجی، نشان خواهد داد:

```
gradeForStudent.size();
```

خروجی ۲۱۷ را به ما نشان خواهد داد.

✓ در تابع بالا، از کلمه ی کلیدی **const** قبل از نوع متغیر استفاده شده است. این به این مفهوم است که متغیر **NumberOfStudents** در طول برنامه دارای مقدار ثابتی خواهد بود و مقدار آن در طول برنامه عوض نخواهد شد. عوض کردن مقدار این متغیر در برنامه، یک خطای ساختاری[۱۵۵] است. استفاده از کلمه ی کلیدی برای جلوگیری از تغییرات سهوی در برنامه است. همچنین استفاده از چنین خدماتی به خواننده ی برنامه برای راحت تر متوجه شدن برنامه کمک می کند.

۳.۳.۷ برخی از قواعد استفاده از خانواده ی vector

- برای استفاده از کلاس **vector** ابتدا باید فایل سرآیند آن را به برنامه ی خود الحاق کنید: **#include <vector>**
- نام گذاری اسم یک **vector** از قوانین نام گذاری نام فایل ها پیروی می کند.
- **vector** ها همانند متغیر های عادی که در فصل های قبل خواندیم می توانند بصورت سراسری[۱۵۶] یا محلی[۱۵۷] تعریف شوند.
- در صورتی که شما از محدوده ی اندیس **vector** ها خارج شدید، یکی از اتفاقات زیر ممکن است بیفتد:
  - برنامه ی شما از کار بیافتد و خارج شود!
  - برنامه، از فضای حافظه های همسایه ی **vector** استفاده کند.

مسلما حالت دوم خطرناک تر است! چون در حالت اول شما متوجه می شوید که چیزی دچار اشکال است ولی در حالت دوم، برنامه با ورودی های اشتباه به کار خود ادامه می دهد.

---

151 Class
152 Object
153 البته در آینده خواهیم دید که بسته به شرایط می توانیم از عملگر های دیگری نیز استفاده کنیم.
154 Member function
155 Syntax error
156 Global variable
157 Local Variable

- **i** امین خانه ی یک **vector** برابر با اندیس **i-1** آن است!
- زمانی که یک **vector** تعریف می شود، قبل از مقداردهی به آن، مقادیر تمامی خانه های آن صفر است.
- زمانی که یک **vector** تعریف می شود، بهترین راه این است که با آن مشابه متغیر های معمولی رفتار کنید.

در زیر **vector1** از قبل تعریف شده است. لذا **vector2** با اندازه ی **vector1** تعریف شده، تمامی مقادیر آن به **vector2** کپی می شود.

```
vector<float> vector2 = vector1;
```

یک مثال؛ محاسبه ی مقدار میانگین

یک کلاس با **10** دانش آموز داریم. می خواهیم میانگین نمره ی امتحانی آنها را حساب کنیم.

```
//File: sumGradesGood.cpp
#include <iostream>
#include <vector> //Need this to use the vector "class"

using namespace std;

int main(void)
{
   const int MaxNoStudents = 10;
   vector <int> grade(MaxNoStudents);

   int sum = 0;
   for (int i = 0; i <grade.size(); i = i + 1)
   {
      cout << "Grade for student #" << i + 1 << ": "; // Note the "+ 1"
      cin >> grade[i];
      sum = sum + grade[i];
   }

   cout << "\nGrade report:\n\n";

   for (int i = 0; i < grade.size(); i = i + 1)
      cout<< "Grade for student #" << i + 1 << ": \t" << grade[i] << "\n";

   cout<< "Class avg is : " <<static_cast<float>(sum)/grade.size() << "\n";
   return 0;
}
```

ورودی نمونه:

```
Grade for student #1: 1
Grade for student #2: 2
Grade for student #3: 3
Grade for student #4: 4
Grade for student #5: 5
Grade for student #6: 6
Grade for student #7: 7
Grade for student #8: 8
Grade for student #9: 9
Grade for student #10: 10
```

خروجی نمونه:

```
Grade report:
Grade for student #1: 1
Grade for student #2: 2
Grade for student #3: 3
Grade for student #4: 4
Grade for student #5: 5
Grade for student #6: 6
Grade for student #7: 7
Grade for student #8: 8
Grade for student #9: 9
Grade for student #10: 10
Class avgis : 5.5
```

مثال؛ یک راه بهتر برای محاسبه ی میانگین

در کد بالا دیدیم که برنامه باید بداند که چه تعدادی از دانش آموزان در کلاس هستند. بنابرین در صورتی که تعداد دانش آموزان تغییر یابد، برنامه باید تغییر یابد.می توانیم از کاربر بخواهیم که تعداد دانش آموزان را در ورودی وارد کند و سپس تعداد خانه های نمره هایمان را بر اساس تعداد دانش آموزان تعریف کنیم:

```
//File: sumGradesBetter.cpp
#include <iostream>
#include <vector> //Need this to use the vector "class"

using namespace std;
int main(void)
{
   cout << "Number of students writing the test: ";
   int nStudents;
   cin >> nStudents;
   vector <int> grade(nStudents);

   int sum = 0;
   for (int i = 0; i < grade.size(); i = i + 1)
   {
      cout << "Grade for student #" << i + 1 << ": "; // Note the "+ 1"
      cin >> grade[i];
      sum = sum + grade[i];
   }

   cout<< "\nGrade report:\n\n";

   for (int i = 0; i <grade.size(); i = i + 1)
      cout<< "Grade for student #" << i + 1 << ": \t" << grade[i] << "\n";;

   cout<< "Class avg is : " <<static_cast<float>(sum)/grade.size() << "\n";
   return 0;
}
```

ورودی نمونه:

```
Number of students writing the test: 3
Grade for student #1: 6
Grade for student #2: 8
Grade for student #3: 9
```

خروجی نمونه:

```
Grade report:
Grade for student #1: 6
Grade for student #2: 8
Grade for student #3: 9
Class avgis : 7.66667
```

مثال؛ یک راه خیلی بهتر برای محاسبه ی میانگین!

در صورتی که بخواهیم تعداد نمره ها را هم از ورودی نگیریم و متناسب با تعداد ورودی ها، تعداد خانه هایمان افزایش یابد، می توانیم از **push_back(value)**، یکی از توابع کلاس **vector**، برای اضافه کردن مقادیر به دنباله ی **vector** مورد نظر استفاده کنیم:

```
//File: sumGradesBest.cpp
#include <iostream>
#include <vector> //Need this to use the vector "class"

using namespace std;

int main(void)
{
   vector<int> grade;
   double sum = 0, score;
   do
   {
      cout << "Input a student's grade: ";
      cin >> score;
      if (0. <= score && 100. >= score)
      {
         sum = sum + score;
         grade.push_back(score);
      }
   }while(0. <= score && 100. >= score);
   cout << "\nGrade report:\n\n";
   for (int i = 0; i < grade.size(); i = i + 1)
      cout << "Grade for student #" << i + 1 << ": \t" << grade[i] << "\n";
   cout<< "Class avg is : " << sum/grade.size() << "\n";
   return 0;
}
```

ورودی نمونه:

```
Input a student's grade: 79
Input a student's grade: 60
Input a student's grade: 99
Input a student's grade: 57
Input a student's grade: -1
Grade report:
Grade for student #1: 79
Grade for student #2: 60
Grade for student #3: 99
Grade for student #4: 57
Class avgis : 73.75
```

یک عضو جدید از خانواده ی **vector**، تابع **push_back(value)** است. این تابع، مقادیر جدید را به انتهای **vetor** اضافه می کند. این تابع خود اندازه ی **vector** را به اندازه ی یک واحد افزایش می دهد. توجه کنید که مقدار **value** باید از جنس **vector** تعریف شده باشد. چرا که قرار است این مقدار در **vector** قرار بگیرد.

یک تابع دیگر از کلاس **vector** که مشابه تابع بالاست، **pop_back()** است که مقدار **vector** را یک واحد کاهش می دهد.

۴.۳.۷ استفاده از **vector** ها در ورودی و خروجی توابع

میتوان از **vector**ها مانند متغیر های معمولی در ورودی و خروجی توابع استفاده کرد. به عناون نمونه به برنامه ی زیر توجه کنید. در زیر انتگرال عددی یک تابع مشخص به ازای بازه های یکسال مشخص بین دو نقطه ی ابتدا و انتهایی آن، محاسبه می شود. در این برنامه این مقدار به سه روش مقدار راست، مقدار چپ و مقدار وسط بازه محاسبه می شود.

```
//File: integrate.cpp
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

double integrateLeft(vector<double> f, vector<double> x)
{
   double sum = 0;
   for (int i = 0; i <f.size() - 1; i = i + 1)
      sum = sum + f[i]*(x[i + 1] - x[i]);
   return sum;
}

double integrateRight(vector<double> f, vector<double> x)
{
   double sum = 0;
   for (int i = 0; i <f.size() - 1; i = i + 1)
      sum = sum + f[i + 1]*(x[i + 1] - x[i]);
   return sum;
}

double integrateMiddle(vector<double> f, vector<double> x)
{
   double sum = 0;
   for (int i = 0; i <f.size() - 1; i = i + 1)
      sum = sum + 0.5*(f[i] + f[i + 1])*(x[i + 1] - x[i]);
   return sum;
}

int main(void)
{
   cout << "Number of points in the integration mesh: ";
   int mesh;
   cin >> mesh;
   vector <double> f(mesh), x(mesh);
   for(int i = 0; i < mesh; i = i + 1)
   {
      x[i] = static_cast<double>(i)/(mesh - 1);
      f[i] = exp(x[i]);
   }
```

```
   double integral = integrateLeft(f, x);
   cout <<"left = \t\t" << integral << " cf " <<exp(1.0) - 1.0
        << " diff = " << integral - (exp(1.0) - 1.0) << "\n";

   integral = integrateRight(f, x);
   cout <<"right = \t" << integral << " cf " <<exp(1.0) - 1.0
        << " diff = " << integral - (exp(1.0) - 1.0) << "\n";

   integral = integrateMiddle(f, x);
   cout <<"midpoint = \t" << integral << " cf " <<exp(1.0) - 1.0
        << " diff = " << integral - (exp(1.0) - 1.0) << "\n";

   return 0;
}
```

.

مسائل

۱. حاصل ضرب داخلی

شما باید در برنامه ی زیر تابع **innerProduct()** را کامل کنید. این تابع باید حاصل ضرب داخلی دو کمیت برداری را به صورت مجموع $x_0y_0 + \cdots + x_{n-1}y_{n-1}$ حساب کند. بطوریکه $x \equiv (x_0, \ldots, x_{n-1})$ و $y \equiv (y_0, \ldots, y_{n-1})$ هستند. پ

```
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

float innerProduct(const vector<float>& a, const vector<float>& b)
{
    //Why did the function use the const qualifiers in the parameter list
    //and pass the vectors by reference?
    //Answer the question and insert your code after this line.
}

int main(void)
{
   intvectorLength = 10;

   vector<float> x (vectorLength);
   cout << " x: ";
   for (int i = 0; i <vectorLength; i = i + 1)
   {
      x[i] = rand()%100;
      cout << "\t" << x[i];
   }
   cout << "\n";

   vector<float> y(vectorLength);
   cout << " y: ";
   for (int i = 0; i < vectorLength; i = i + 1)
   {
      y[i] = rand()%10;
      cout<< "\t" << y[i];
   }
   cout<< "\n";

   cout<< "Inner product is: " <<innerProduct(x,y) << "\n";
```

```
   return 0;
}
```

سوال: چرا در اینجا از توصیف کننده ی **const** برای **vector** های ورودی استفاده کرده ایم و آنها را با آدرس[۱۵۸] به تابع ارجاع کرده ایم؟

۲. مقدار ماکزیمم و مینیموم

در کد زیر باید تابع **maxMin()** را طوری کامل کنید که این تابع حاصل مقدار ماکزیمم و مینیموم **vector** مورد نظر را در خروجی نمایش دهد.

```
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

void minMax(const vector<int>& c, int& min, int& max)
{
   //Why were min and max given as "reference parameters"?
   //Answer the question and insert your code after this line.
}

int main(void)
{
   int vectorLength = 8;

   vector<int> a(vectorLength);
   cout << " a: ";
   for (int i = 0; i <vectorLength; i = i + 1)
   {
      a[i] = rand();
      cout<< "\t" << a[i];
   }
   cout<< "\n";

   int minimum, maximum;
   minMax(a, minimum, maximum);

   cout<< "The minimum array element is: " << minimum << "\n";
   cout<< "The maximum array element is: " << maximum << "\n";
   return 0;
}
```

سوال: چرا در اینجا **vector** های ورودی را با آدرس[۱۵۹] به تابع ارجاع کرده ایم؟

[158] Pass by reference
[159] Pass by reference

۳. فقط مقادیر مثبت

در کد زیر شما باید تابع **nothingNegative()** را کامل کنید که به ازای **vector** که از ورودی دریافت می کند و دارای اعداد صحیح مثبت و منفی است، به خروجی vector با طول کوچکتر یا مساوی ورودی دهد که اعضای آن المان های مثبت **vector** ورودی هستند.

```
#include <iostream>
#include <vector>
#include <cstdlib>

using namespace std;

vector<int> nothingNegative(const vector<int>& a)
{
   //Why can this program, when compiled with just the function stub,
   //go into what looks like an infinite loop?
   //Answer the question and insert your code after this line.
}

int main(void)
{
   int vectorLength = 10;
   vector<int> bothSigns(vectorLength);
   cout << " Input vector: ";
   for (int i = 0; i <vectorLength; i = i + 1)
   {
      bothSigns[i] = rand()%201 - 100;
      cout << "\t" <<bothSigns[i];
   }
   cout << "\n";

   vector<int> noNegs = nothingNegative(bothSigns);

   cout << " Output vector: ";
   for (int i = 0; i <noNegs.size(); i = i + 1)
   {
      cout << "\t" <<noNegs[i];
   }
   cout << "\n";

   return 0;
}
```

سوال: چرا تا زمانی که دستورات داخل تابع **nothingNegative** را کامل نکرده ایم، برنامه مانند یک حلقه بی نهایت اجرا می شود؟!

۴. عملیات روی vector

تابعی بنویسید که متغیر **vector** از **int** را بصورت ارجاع با آدرس[۱۶۰] به عنوان ورودی بگیرد، متغیر **vector** از **int** را به عنوان خروجی بازگرداند. این برنامه در اصل همان برنامه ی بالاست. تنها تغییری که باید ایجاد کنید این است که مقادیر منفی در یک

[160] Pass by Reference

vector خروجی قرار گیرند و مقادیر مثبت فقط **vector** ورودی که بصورت ارجاع با آدرس گرفته شده است، قرار گیرند. توجه کنید که ترتیب اعداد باید به همان صورتی باشد که در ورودی داده ایم. اینجا نمونه ای کار برنامه را می بینید:

ورودی نمونه:

```
0 -6 -8 3 4 -8 0 8 -9 -8 8
```

خروجی نمونه:

- **vector** مورد نظر بعد از اجرای تابع:

```
3 4 8 8
```

- **Vector** خروجی از تابع:

```
-6 -8 -8 -9 -8
```

۵. عملیات **push** و **pop** روی **vector**

در اینجا با نمونه ی چالش بر انگیزتری از مساله ی پیش روبرو هستیم. شما باید تابعی با خروجی **void** بنویسید که دو **vector<int>** را بصورت ارجاع با آدرس بگیرد. اولی دارای اعداد صحیحی هستند که به طور تصادفی تولید شده اند. دومی خالی خواهد بود. تابع شما باید بعد از اتمام، اعداد مثبت را در **vector** اولی و اعداد منفی را در **vector** دومی قرار دهد. توجه کنید که شما باید این کار را بدون استفاده از هیچ **vector** جدیدی انجام دهید!

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
using namespace std;


//Function definition goes here...


int main(void)
{
   srand(time(NULL));
   int size = rand()%20 + 10; //size is a RN between 10 and 29
   vector<int> in(size), out;

   for (int i = 0; i < in.size(); i = i + 1)
      in[i] = rand()%19 - 9; //RN between -9 and +9

   cout << "Original vector is size " << in.size() << endl;
   for (int i = 0; i < in.size(); i = i + 1)
      cout << in[i] << " ";
   cout << "\n\n";

   extractNegs(in, out);

   cout << "Returned vector is size " << out.size() << endl;
   for (int i = 0; i < out.size(); i = i + 1)
      cout << out[i] << " ";
   cout << "\n\n";

   cout<< "Modified vector is size " << in.size() << endl;
```

```
   for (int i = 0; i < in.size(); i = i + 1)
      cout << in[i] << " ";
   cout<< "\n\n";

   return 0;
}
```

۶. مربع کامل

برنامه ی زیر را کامل کنید. تابعی بنویسید که متغیر **vector<int>** را از ورودی بصورت ارجاع دریافت و متغیر **vector<int>** را به عنوان خروجی می دهد. ورودی مجموعه ای از اعداد صحیح مثبت است. خروجی شما باید یک **vector** از اعدادی دریافت شده از ورودی باشد که این اعداد مربع کامل هستند.

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <cmath>
#include <ctime>

using namespace std;

//Function definition goes here...


int main(void)
{
   srand(time(NULL));
   int size = rand()%20 + 10; //size is a RN between 10 and 29
   vector<int> in(size);
   for (int i = 0; i < in.size(); i = i + 1)
   {
      in[i] = rand()%90 + 10;
      cout << in[i] << " ";
   }
   cout << endl;

   vector<int> out = squareIndex(in);

   cout<< "Returned vector is size " << out.size() << endl;

   for (int i = 0; i < out.size(); i = i + 1)
      cout<< out[i] << " ";
   cout << endl;

   return 0;
}
```

۷. عملیات مرتب سازی[۱۶۱]

برنامه ی زیر را کامل کنید. شما باید تابع **sort()** را کامل کنید، بطوریکه خروجی آن **void** است. تابع باید متغیر ورودی از نوع **vector<int>** را از ورودی بگیرد که شامل اعداد صحیح نامرتب است و مرتب شده ی آنها را بر حسب صعودی به نزولی را در خروجی چاپ کند.

```
#include <iostream>
#include <vector>
#include <cstdlib>

using namespace std;

void sort(vector<int>& a){}

int main(void)
{
   cout<< "Input the minimum and maximum random int: ";
   int minRange, maxRange;
   cin >> minRange >> maxRange;

   cout << "Input the length of the vector: ";
   int vectorLength;
   cin >> vectorLength;

   vector<int> unsorted(vectorLength);
   cout << " Vector beforehand: \t";
   for (int i = 0; i < vectorLength; i = i + 1)
   {
      unsorted[i] = minRange + rand()%(maxRange - minRange + 1);
      cout << unsorted[i] << " ";
   }
   cout << "\n";

   sort(unsorted);
   cout << " Vector afterhand: \t";
   for (int i = 0; i < unsorted.size(); i = i + 1)
   {
      cout << unsorted[i] << " ";
   }
   cout << "\n";
   return 0;
}
```

۸. اعداد فرد[۱۶۲]

برنامه ی زیر را طوری کامل کنید. بطوریکه تابع **getPrimes()** که خروجی آن از نوع **vector<int>** است. تابع عدد صحیح **NPrimes** را به عنوان ورودی می پذیرد که در واقع تعداد اعداد فرد متوالی است که با عدد ۲ شروع می شوند. متغیر **vector** خروجی شامل این اعداد فرد به ترتیب است. شما باید اجرای برنامه ی خود را توسط دستوراتی مانند دستور زیر بهینه کنید و اعدادی را فرد نیستند را حذف کنید:

```
if (0 == n%divisor) nIsPrime = false;
```

[161] Sort
[162] Prime Numbers

در اینجا **divisor** یک عدد فرد دیگر خواهد بود!

```
#include <iostream>
#include <vector>
#include <cstdlib>

vector<int> getPrimes(constintNPrimes){}

int main(void)
{
   cout << "Input the number of primes to find: ";
   int NPrimes;
   cin >> NPrimes;

   vector<int> primes = getPrimes(NPrimes); //Get the primes
   cout << "The first " <<primes.size() << " prime numbers: ";

   for (int i = 0; i <primes.size(); i = i + 1)
   {
      cout << primes[i] << " ";
   }
   cout << "\n";
   return 0;
}
```

۹. حذف اعداد تکراری

در برنامه ی زیر تابع **noRepeats()** طوری کامل کنید بطوریکه خروجی آن از نوع **void** باشد. این تابع **vector<int>** **bothSign** را به عنوان متغیر ورودی قبول می کند که در آن اعداد **int** به صورت تصادفی قرارگرفته اند. تابع مورد نظر باید مقادیر تکراری را از این **vector** حذف کند.

```
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

voidnoRepeats(vector<int>& a){}

int main(void)
{
   cout << "Input the minimum and maximum random int: ";
   int minRange, maxRange;
   cin >> minRange >> maxRange;

   cout << "Input the length of the vector: ";
   int vectorLength;
   cin >> vectorLength;

   vector<int> hasRepeats(vectorLength);
   cout << " Vector beforehand: \t";
   for (int i = 0; i < vectorLength; i = i + 1)
   {
      hasRepeats[i] = minRange + rand()%(maxRange - minRange + 1);
      cout << hasRepeats[i] << " ";
   }
   cout << "\n";
```

```
    noRepeats(hasRepeats);

    cout << " Vector afterhand: \t";
    for (int i = 0; i < hasRepeats.size(); i = i + 1)
    {
       cout << hasRepeats[i] << " ";
    }
    cout << "\n";
    return 0;
}
```

فصل هشتم

چند داده ی انتزاعی دیگر؛آرایه ها و استراکچر ها[۱۶۳]

1.8 آرایه ها

آرایه جزئی از مفهوم **vector** (بردار) می باشد. یک آرایه می تواند لیستی از اشیا را به صورت مرتب شده نمایش دهد. برای مثال A_i , i=1,2,3,…,n آرایه یک بعدی گفته می شود که برای نمایش لیستی از اشیا (برای مثال لیستی از نمرات) به کار می رود.دو آرایه یک بعدی (**x[i],f[x]**) می تواند تابع **f(x)** را به ازای مقدار هر **x** روی صفحه ی دوبعدی نمایش دهد و تابع به ازای مقدار هر نقطه ارزش گذاری می شود (**f_i=f(x_i)**) . این ها متداولترین کاربرد آرایه های یک بعدی می باشند.

vector که در بخش قبل با آن آشنا شدیم در واقع یک آرایه یک بعدی می باشد.

آرایه ها ساختارهای سطح پایینی هستند که از زبان **C** به زبان **C++** منتقل شده اند. به بیان ساده آرایه ها مجموعه های منظمی از چیزهایی هستند که شما آنها را می شناسید.به عنوان مثال **int, float, double,…** که با چند قاعده ی خاص به شاخص گذاری، نمایش مناسب و ارتباط با تابع مربوط می شود. **vector** از جهت دیگر یک کلاس استاندارد در **C++** می باشد که دارای توابع قوی مانند **size( ),push_back( ),pop_back( )** ,… می باشد که روی داده ها در **vector** ها عمل می کنند.به همین دلیل ما در این کلاس به دنبال آموزش جزئی آرایه های یک بعدی نیستیم. به جای آن چیز دیگری برای تمرکز روی کلاس های قوی تر **vector** به هنگام کار با موارد تک بعدی انتخاب کنید. به یاد داشته باشید که آرایه های چند بعدی نیز مانند آرایه های یک بعدی هستند.

آرایه های دو بعدی می توانند به عنوان جدولی از مقادیر یا یک سری از سطرها و ستون ها که هر کدام دارای ارزش خاصی هستند تصور شوند. این آرایه ها همچنین برای نمایش صفحه شطرنج، صفحه بازی **Minesweeper** ، زمین فوتبال یا نمایش سطح زمین به کار می رود. ما می توانیم مثال های بسیاری را تصور کنیم، زیرا خلاقیت انسان هیچ محدودیتی ندارد! بعد از این فصل شما قواعد و دستورات کافی برای برنامه شطرنج را خواهید داشت. با کمی آشنایی از مفاهیم فیزیک شما می توانید برنامه ای بنویسید که هوا را پیش بینی کند. بعد از گذراندن فقط**20** فصل شما به این توانایی خواهید رسید.

شما ابزار و قوانین لازم را در اختیار دارید. پس خلاقیت خود را به کار بیاندازید وبه طراحی الگوریتمی که دقیقا از **3** بخش(**3** رکن الگوریتم ها، یعنی: ترتیب، ساختار تصمیم گیری، حلقه) تشکیل شده است تفکر کنید. این کار واقعا ساده و جالب است.

در مدل های انتزاعی بیشتر آرایه های دو بعدی می توانند جدول های شاخص گذاری شده را نمایش دهند. به عنوان مثال A_{ij} در محاسبات عددی در علوم و مهندسی به عنوان آرایه های دو بعدی برای نمایش جداولی از داده ها مانند مثال زیر استفاده می شود.

آرا یه های دو بعدی(برای مثال **x[i][j],f[i][j]**) می تواند یک تابع دو متغیره را نمایش دهد. یعنی صورت مشخص کردن ارزش هر نقطه در صفحه**x-y** و ومقدار تابع در هر نقطه با **f(x,y)** مشخص می شود. با این حال جالب ترین نمونه برای علوم و مهندسی آرایه های دو بعدی مربعی است که ماتریس نامیده می شود. در این فصل ما بحث را فقط روی آرایه های دو بعدی

[163] Structur (یا ساختان داده ها)

محدود می کنیم. چرا که مطلب جدیدی برای ابعاد سه یا بیشتر وجود ندارد که ما در دو بعد ندیده باشیم. قوانینی که برای آرایه های یک بعدی داریم به راحتی برای آرایه های دو بعدی به دست می آید. متأسفانه در**C++** کلاس جدیدی که این موارد را توضیح دهد وجود ندارد.

پس ما باید با آرایه هایی که در **C** به کار می رفت کار کنیم. اگر به زبان برنامه نویسی **Matlab** توجه کنیم، این زبان از آرایه های دوبعدی به عنوان متغیر های بنیادی خود استفاده می کند. این ویژگی قابلیت های بسیاری را به این زبان داده است .

Matlab یک برنامه سطح بالاست (در جهان مهندسی حرفه ای کاربرد دارد). با این حال اگر شما برنامه های خود را در **C++** ارتقا دهید سریع تر (گاهی اوقات خیلی سریع تر) اجرا خواهند شد. پس ما انرژی بیشتری در چگونگی کار با آرایه های دو بعدی در**C++** صرف می کنیم. همچنین انجام این ها تمرین خوبی برای تفکر به الگوریتم ها است که برجسته ترین دلیل تدریس این کلاس می باشد.

آرایه هایی که حد اقل **12** بعد داشته باشند دارای استاندارد **ANSI** می باشند.چه باور کنید چه نه، کاربرد هایی برای آن وجود دارد!

دیگر داستان تعریف کردن کافی است!! بهتر است قبل از به وجود آوردن عبارت ها به دستورالعمل آنها بپردازیم.

1.1.8 شناساندن آرایه های دو بعدی

در این جا مثالی از چگونگی شناساندن و ارزشگذاری آرایه های دو بعدی آورده شده است:

```
const int NROWS = 3;
const int NCOLUMNS = 2;
int a[NROWS][NCOLUMNS];
```

این یک آرایه سه در دو می باشد که برای مجسم ساختن آرایه ای دو بعدی با شماره ردیف ها **0** تا **NROWS-1** از چپ به راست وشماره ستون های **0** تا **NCOLUMNS-1** از بالا به پایین استفاده می شود. این آرایه یک مثال اختیاری است که برای به خاطر سپردن آن کمک می کند.

a[0][0]	**a[0][1]**
a[1][0]	**a[1][1]**
a[2][0]	**a[2][1]**

به عنوان راهی دیگر برای تصور، می توانیم به عنوان یک آرایه یک بعدی به آن نگاه کنیم. البته نشانه گذاری آنها به صورت **[NROWS][NCLUMNS]** به این مطلب اشاره می کند.ما می توانیم آنرا به صورت بردارهای عمودی به طول **NROWS** و بردارهای افقی به طول **NCOLUMNS** تصور کنیم.

2.1.8 مقدار دهی آرایه های دو بعدی

برای مقدار دهی یک آرایه دوبعدی در هنگام تعریف آن ، راه های متنوعی وجود دارد. در اینجا چند مثال آورده شده است:

```
const int NROWS = 3;
const int NCOLUMNS = 2;
int a[NROWS][NCOLUMNS] = {0};
```

عبارت فوق تمام خانه های آرایه را صفر میکند یعنی به صورت:

0	0
0	0
0	0

این مشخصه بسیار مهم است. چرا که با این روش به مقدار زیادی در زمان و تایپ، صرفه جویی می کنید.

برای مقدار دهی اولیه از نشانه گذاری دوتایی نیز استفاده می شود:

```
const int NROWS = 3;
const int NCOLUMNS = 2;
int a[NROWS][NCOLUMNS] = {{1,2}, {3,4}, {5,6}};
```

که نتیجه آن به صورت زیر است:

1	**2**
3	**4**
5	**6**

نشانه گذاری دوتایی راه جالبی است چرا که می تواند محتویات آرایه به شکلی مشخص، قبل از اجرای برنامه، برای شما به نمایش بگذارد. به عنوان مثال:

```
const int NROWS = 3;
const int NCOLUMNS = 2;
int a[NROWS][NCOLUMNS] =
{ {1,2},
  {3,4},
  {5,6}
};
```

استفاده از عبارات تو در تو را به یاد داشته باشید. اگر یک زوج عدد به طور کامل پر نشده باشد، جاهای خالی توسط صفر پر می شود. برای مثال:

```
const int NROWS = 3;
const int NCOLUMNS = 2;
int a[NROWS][NCOLUMNS] = {{1}, {3}, {5}};
```

که نتیجه آن به صورت زیر است:

1	0
3	0
5	0

این مهم است که وقتی یک آرایه را به این صورت مقدار دهی می کنیم تا یک آرایه را با اعداد صحیح تعریف کنیم، سایز ابعاد آن یک عدد ثابت توصیف شود. یعنی یا اینکه مقادیر عددی برای آن وارد شوند. یا اینکه از پارامتر هایی که به صورت **const** تعریف شده اند استفاده شود. در غیر این صورت کامپایلر ایراد می گیرد.

آرایه ها می توانند در هر جای برنامه تعریف شوند. برای مثال:

```
const int nrows = 10;
const int ncolumns = 20;
float f[nrows][ncolumns];
```

برخلاف بردارها، آرایه ها به صورت خودکار توسط صفر مقدار دهی نمی شود. این کار باید انجام شود یا به صورت یک عبارت تعریفی و یا به صورت مستقیم توسط عبارات قابل اجرا به صفر مقدار دهی شود. در غیر این صورت توسط مقدار هایی که از قبل در حافظه موجود بوده است آرایه مقدار دهی می شود.

در اینجا بر نامه کوتاهی آورده شده است که نحوه ی تعریف کردن، مقداردهی و آدرس دهی آرایه ها را نشان می دهد.

```
//File: whereItsAt.cpp
#include <iostream>

int main(void)
{
   const int NROWS = 3;
   const int NCOLUMNS = 2;
   int a[NROWS][NCOLUMNS] = {{1}, {3}, {5}};
   cout << a[0][0] << ":" << a[0][1] << "\n";
   cout << a[1][0] << ":" << a[1][1] << "\n";
   cout << a[2][0] << ":" << a[2][1] << "\n";
   cout << "&a[2][1]: " << &a[2][1] << "\n";
   cout << "&a[2][0]: " << &a[2][0] << "\n";
   cout << "&a[1][1]: " << &a[1][1] << "\n";
   cout << "&a[1][0]: " << &a[1][0] << "\n";
   cout << "&a[0][1]: " << &a[0][1] << "\n";
   cout << "&a[0][0]: " << &a[0][0] << "\n";
   cout << "a: " << a << "\n";
   return 0;
}
```

که دارای خروجی زیر می باشد:

```
1:0
3:0
5:0
&a[2][1]: 0xbffff72c
&a[2][0]: 0xbffff728
&a[1][1]: 0xbffff724
&a[1][0]: 0xbffff720
&a[0][1]: 0xbffff71c
&a[0][0]: 0xbffff718
a: 0xbffff718
```

اما یک سوال! یک آرایه چند بعدی چگونه ذخیره می شود؟ حافظه کامپیوتر به صورت یک آرایه یک بعدی سازمان دهی شده است. از این رو، یک آرایه چند بعدی باید به صورت یک آرایه ی یک بعدی ایجاد شده و سپس ذخیره شود. زبان های مختلف این کار را به روش های متفاوت انجام می دهد. قرار داد **C++** به این صورت است که داده ها را از یک لیست خارجی به یک لیست داخلی ذخیره می کند. که مفهومی مشابه "قرار دادن یک وکتور در داخل وکتور دیگر " را داراست. در مثال آرایه ی دو بعدی ما، این بدین معناست که تمام ستون ها در ردیف اول ذخیره می شوند، سپس ستون های ردیف دوم و....

اینجا یک موضوع اجرایی مهم در زمان نوشتن یک برنامه پیش می آید! به اینصورت که می توان با آگاهی از ترتیب قرار گیری داده ها در آرایه ها، به صورت کار آمدتر و سریع تری از آنها استفاده کرد.

به عنوان مثال به بر نامه ی زیر دقت کنید:

```
//File: fastArray.cpp

int main(void)
{
   const int SIZE = 1000;
   float a[SIZE][SIZE];

   for(int k = 1 ; k <= 100 ; k = k + 1)
      for(int i = 0 ; i <= SIZE - 1 ; i = i + 1)
         for(int j = 0 ; j <= SIZE - 1 ; j = j + 1)
            a[i][j] = 1;

   return 0;
}
```

بر روی رایانه ی من، سرعت پردازش برنامه ی بالا تقریبا دو برابر سرعت اجرای برنامه ی **slowarray.cpp** می باشد که تنها تفاوت آن ها در ترتیب دو حلقه می باشد.

```
//File: slowArray.cpp

int main(void)
{
   const int SIZE = 1000;
   float a[SIZE][SIZE];

   for(int k = 1 ; k <= 100 ; k = k + 1)
      for(int i = 0 ; i <= SIZE - 1 ; i = i + 1)
         for(int j = 0 ; j <= SIZE - 1 ; j = j + 1)
            a[j][i] = 1;

   return 0;
}
```

3.1.8 فرستادن یک آرایه دو بعدی به یک تابع

اینجاست که فرق آرایه های چند بعدی با آرایه یک بعدی مشخص می شود. زمانی که یک آرایه به یک تابع فرستاده می شود که باید اطلاعات کافی داده شود تا مشخص شود چگونه باید روی آرایه عمل کند. در نتیجه ابعاد تمام لیست های بیرونی باید مشخص شود. ببینید این کار چگونه در مثال زیرانجام شده است:

```
//File: 2d.cpp
#include <iostream>
#include <iomanip>
#include <cmath>
#include <vector>

const int SIZE = 24;
const int MAX_PRINT = 24;

void printArray(float a[][SIZE])
{
   if (SIZE <= MAX_PRINT)
   {
      for(int i = 0; i < SIZE; i = i + 1)
      {
          for(int j = 0; j < SIZE; j = j + 1)
          cout << setw(2) << static_cast<int>(a[i][j]) << " ";
         cout << "\n";
      }
   }
   else cout << "Array is too big to print.\n";
   return;
}

void productArray(vector<float>& y, float O[][SIZE], vector<float> x)
{
   for(int i = 0; i < SIZE; i = i + 1)
   for(int j = 0; j < SIZE; j = j + 1)
       y[i] = y[i] + O[i][j] * x[j];
   return;
}

int main(void)
{
   /* Initialize the D array */
   float D[SIZE][SIZE] = {0};
   for(int i = 0; i < SIZE - 1; i = i + 1)
   {
       D[i][i] = -1;
       D[i][i + 1] = 1;
   }
   // Initialize the I array
   float I[SIZE][SIZE] = {0};
   for(int i = 1; i < SIZE; i = i + 1)
       for(int j = 0; j <= i; j = j + 1)
         I[i][j] = 1;

       // Print out D and I arrays
       cout << "The \"D\" array:\n"; printArray(D);
       cout << "The \"I\" array:\n"; printArray(I);

       // Initialize the f and x vectors
       vector<float> f(SIZE), x(SIZE);
       for(int i = 0; i < SIZE; i = i + 1)
       {
           x[i] = -1 + 2.0*i/(SIZE - 1);
           f[i] = 1.0/(1.01 - x[i]);
       }
```

```
        // Operate on f with D and normalize
        vector<float> d(SIZE);
        productArray(d,D,f);
        for(int i = 0; i < SIZE - 1; i = i + 1)
           d[i] = d[i]/(x[i+1] - x[i]);

        // Operate on f with I and normalize
        vector<float> F(SIZE);
        productArray(F,I,f);
        for(int i = 1; i < SIZE; i = i + 1)
           F[i] = F[i]*(x[i] - x[i-1]);

        // Print results
        Cout << " x "
             << " f "
             << " d "
             << " F\n";
        for(int i = 0 ; i <= SIZE - 1 ; i = i + 1)
        cout << setiosflags(ios::fixed)
             << setw(11) << x[i] << " "
             << setw(11) << f[i] << " "
             << setw(11) << d[i] << " "
              << setw(11) << F[i] << "\n";
        return 0;
}
```

نتیجه برنامه بالا به صورت زیر است:

```
The "D" array:
-1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  1  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  1  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  1  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  1  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  1  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  1  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  1  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  1
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

hts

```
The "I" array:
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0
 1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
```

```
     x          f          d          F
-1.000000   0.497512   0.258711   0.000000
-0.913043   0.520009   0.283216   0.088480
-0.826087   0.544637   0.311375   0.135840
-0.739130   0.571713   0.343954   0.185554
-0.652174   0.601622   0.381930   0.237869
-0.565217   0.634833   0.426560   0.293072
-0.478261   0.671925   0.479500   0.351500
-0.391304   0.713621   0.542947   0.413554
-0.304348   0.760834   0.619879   0.479714
-0.217391   0.814736   0.714408   0.550560
-0.130435   0.876859   0.832347   0.626809
-0.043478   0.949237   0.982116   0.709351
 0.043478   1.034638   1.176305   0.799320
 0.130435   1.136925   1.434410   0.898183
 0.217391   1.261657   1.787930   1.007892
 0.304348   1.417129   2.290509   1.131120
 0.391304   1.616304   3.039655   1.271669
 0.478261   1.880621   4.228182   1.435201
 0.565217   2.248289   6.283189   1.630704
 0.652174   2.794654  10.317343   1.873718
 0.739130   3.691814  20.073689   2.194744
 0.826087   5.437352  56.080322   2.667557
 0.913043  10.313904 1031.390381   3.564421
 1.000000 100.000000   0.000000  12.260068
```

این برنامه چه کار می کند؟

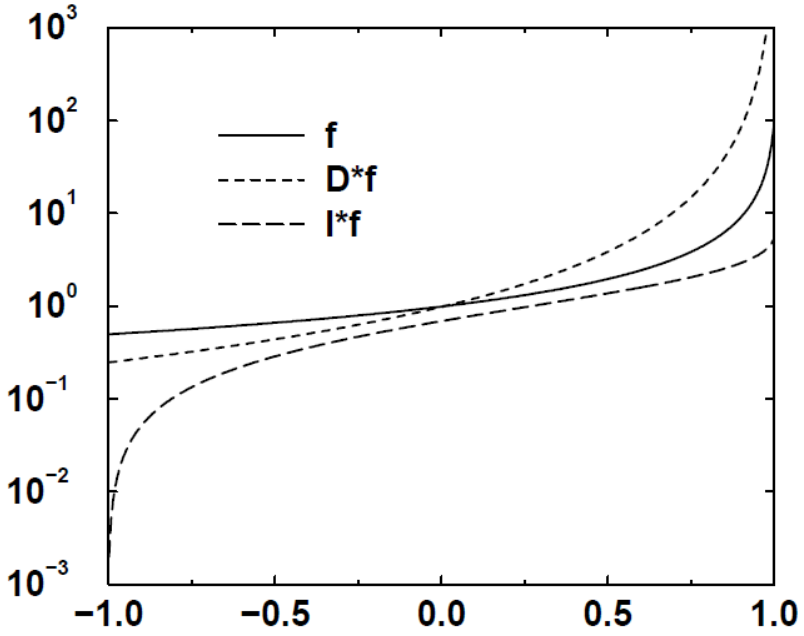
با استفاده از فرمول زیر برای بوجود آوردن یک آرایه دو بعدی با وکتور

$$y_i = A_{i0}x_0 + A_{i1}x_1 + A_{i2}x_2 \cdots$$

hts

می توان دید که آرایه"D"مشتق و آرایه"I"تابع اولیه تابع **f(x)** می باشد. بله،شما می توانید محاسبات خود را توسط یک آرایه، در بازه ای از متغیر **x** که تابع روی آن تغییرات زیادی ندارد، انجام دهید. در واقع این اساس کار اکثر ماشین محاسباتی است که است که برای حل معادلات دیفرانسیل شما به کار گرفته می شوند. (شما توسط اطلاعاتی که در این پاراگراف آمده بود، مورد آزمون قرار نمی گیرید!)

می توان نتیجه ی عبارات بالا را در زیر مشاهده کرد:



۲.۸ آرایه های کارکتر

۱.۲.۸ آرایه های یک بعدی کاراکتری

- قبلا دیده بودیم که رشته ها یک نوع وکتور تغییر شکل یافته است. برای ترکیب رشته ها و کاراکتر ها از آرایه های یک بعدی استفاده می شود . توصیه می کنیم فقط از این روش زمانی که استفاده از رشته ها نا مناسب است، استفاده کنید. چرا که استفاده از این روش نیازمند داشتن شرایطی است. مانند اینکه نیاز حفظ بودن برخی از قوانین هستید و یا ... .

  بنابراین یک رشته کاراکتر می تواند به صورت یک آرایه یک بعدی تک کاراکتری همراه با یک کاراکتر نگهبان در انتهای کاراکتر ها که نشان دهنده ی اتمام رشته هاست، مورد استفاده قرار گیرد. این کاراکتر، کاراکتر **"NULL"** یا همان **'\0'**، است یعنی هستی ای است که برای نسان دادن نیستی به کار می رود!! بنابراین همواره باید به یاد داشته باشیم که در پایان یک

آرایه ی یک بعدی کاراکتر **NULL** قرار می گیرد. کتابخانه ی **String** تمام این کار ها را به صورت خود کار برای شما انجام می دهد ،به همین دلیل بهتر است که در صورت امکان از این کتابخانه استفاده شود.

- در معرفی و مقدار دهی رشته ی زیر:

```
char s[4];
s[0] = 'H';
s[1] = 'i';
s[2] = '\0'; /* null */
```

با وجود اینکه ما **4** بایت را به آن اختصاص دا ده ایم، فقط سه تا از آنها را استفاده کرده ایم زیرا ما در انتهای رشته **'\0'** قرار داده ایم. (که خود مقداری تهی به حساب می آید)

- به یاد داشته باشید که عبارات:

```
s[2] = '\0'; /* null */
s[2] = 0; /* the same thing */
```

کاملا مشابه هم اند.

- رشته ها طول خود را توسط پایان دهنده **'\0'** مشخص می کنند که به عنوان تکمیل کننده رشته عمل می کند.
- آسان ترین راه برای مقدار دهی یک رشته کاراکتر در سطر سطر تعریف آن به صورت زیر است.

```
char s[] = {'H','i','\0'};
```

که راحت تر از روش قبلی می باشد زیرا نیاز ندارد تا شماره ی خانه های آرایه نوشته شوند.

- راه راحت تر هنوز همان:

```
char s[] = "Hi";
```

که استفاده از دابل کوتیشن(") می باشد که با قرار دادن ثابت یا عبارت فراخوانی شده در میان آن تعریف می شود.آرایه بالا دارای طول ۳ می باشد که پایان دهنده**'\0'** به صورت خودکار توسط کامپایلر قرارداده می شود.

- این یک اشتباه متداول است که گذاشتن پایان دهنده **'\0'** فراموش شود. برای مثال ما یک رشته کاراکتر به نام **mistake** به صورت زیر تعریف کرده ایم:

```
char mistake[] = {'b','o','o','b','o','o'};
```

C++ این رشته را به صورت رشته **"booboo.....\0"** در نظر می گیرد که آن را آنقدر ادامه می دهد تا به پایان دهنده **'\0'** برسد یا اینکه فراتر از حجم حافظه اجرا می شود. این اشکالی است که به سادگی می تواند برنامه را دچار اشکال منطقی کند که ناشی از یک اشکال منطقی است که توسط کامپایلر قابل شناسایی نیست.

آیا می توانید حدس بزنید که برنامه زیر چگونه عمل می کند؟

```
//File: charArray.cpp
#include<iostream>
#include<string>

int main(void)
{
    char correct[12] =
        {'I', 'A', 'm', 'C', 'o', 'r', 'r', 'e', 'c', 't', '!', '\0'};
    char moreData[] = "Will this print?";
    char someData[5] = {100, 97, 114, 110, 0};
    char mistake[8] = {'A', 'm', 'i', 's', 't', 'a', 'k', 'e'};
```

```
    cout << mistake << "\n" << correct << "\n";
    return 0;
}
```

۲.۲.۸ آرایه های کاراکتر دو بعدی

می توان آرایه ای به غیر از کاراکتر ایجاد کرد. به عنوان مثال:

```
char checkerboard[8][8];
for (int i = 0; i < 8; i = i + 1)
for(int j = 0; j < 8; j = j + 1)
   checkerboard[i][j] = ' '; //Initialize with spaces
checkerboard[0][0] = 'R'; //Place a red checker on the board
checkerboard[7][0] = 'B'; //Place a black checker on the board
```

۳.۸ استراکچر ها (ساختار ها)

تا کنون ما انواع مختلفی از متغیر ها که از نوع فقط یک نوع داده مانند **bool, char, int, float** یا **double** می باشند را به وجود آورده ایم. این متغیر های اولیه در **C++** برای معرفی داده ای که در یک خانه از حافظه قرار میگیرد استفاده می شوند. ما همچنین داده ها و کتابخانه های پیچیده تری نیز همانند وکتورها و رشته ها داریم. و همچنین آرایه های یک، دو و چند بعدی که از انواع متغیر های اولیه بالا تشکیل شده است که شامل تعداد زیادی داده می باشد که در حالت های مختلف به ترتیب در آرایه ها قرار می گیرند. این پیش تعریف نوع متغیر ها برای با قاعده درآوردن راه حل مسائل کافی نمی باشد. خوشبختانه **C++** این امکان را به برنامه نویس میدهد که ترکیبی از انواع متغیر ها را که به صورت مجموعه ای از متغیر ها درآمده است را بوجود آورد. به این مجموعه از متغیر ها ساختار می گویند.

برای مثال،ما می توانیم یک ساختار ایجاد کنیم که یک فرد را نمایش دهد. اجزای ساختار که می تواند نام فرد باشد، یک عدد صحیح برای سن فرد، یک عدد اعشاری برای قد و وزن فرد و رسته های بیشتر برای آدرس و شغل او و ... را می توانیم تعریف کنیم.

ساختارها یک ترکیب حیاتی از داده ها برای پایگاه های اطلاعاتی می باشند که ارتباط بین اجزای مختلف صنعت نرم افزار را نمایش می دهند. مثال دیگر آن ، کاربرد آن در مسابقات خودرو می باشد. حال ما می خواهیم یک مسابقه ایجاد کنیم که دارای شاخصه های یک بازی باشد، مانند یک سطح صاف که باعث کاهش تماس لاستیک با سطح جاده می شود. مانور دادن در پشت خودرو رقیب و ... . هر مشخصه که قسمتی از مسیر مسابقه را نمایش می دهد می بایست شامل اطلاعات زیادی باشد، ضریب اصطکاک سطح جاده، مشخصات زمانی و مکانی که اتومبیل از آن در هر لحظه عبور می کند (بنابراین سرعت باد نیز باید محاسبه شود). از آن جا که این مثال ها می توانند توسط اطلاعات مجزا در متغیر های متفاوت انجام شوند، چرا آن ها را در متغیر های ترکیبی قرار ندهیم؟ این در حالی است که همه ی آنها به یک چیز واحد مربوط می شوند.

۱.۳.۸ تعریف ساختار

مثالی از تعریف ساختار:

```
// Define a structure called Student
struct Student
{
   string name;
   int section;
   float grade;
};
```

لغت کلیدی **struct** تعریف ساختار را تعریف می کند.کلمه ی **student** را نام ساختار می گویند. مهم است که به یاد داشته باشیم که این مشخصات معرفی یک ساختار است نه تعریف آن. در اینجا ما به کامپایلر اطلاع می دهیم که ما بعدا متغیری از نوع **student** را تعریف خواهیم کرد.

۲.۳.۸ ساختار ها کجا می توانند تعریف شوند؟

ساختار ها در همه جا می توانند تعریف شوند با این حال متداول است که خارج از واحد اصلی برنامه مانند **main** یا هر تابع دیگری تعریف شود که در این حالت تمام قسمت های برنامه به این متغیر ها دسترسی دارند.

۳.۳.۸ اعضای ساختار

متغیر ها به همراه **{}**(آکولاد) اعضای ساختار **student** هستند که این اعضا به صورت ذکر نوع آن که نام آن یک رشته می باشد که برای ذکر نام فرد می باشد،یک عدد صحیح که شامل نام گروه فرد و یک عدد اعشاری برای مشخص کردن درجه او.

۴.۳.۸ تعریف متغیرهای ساختار

نام ساختار به همراه لغت کلیدی **struct** برای تعریف (و به صورت اختیاری به مقدار دهی) نوع متغیر های ساختار به کار می رود. به عنوان مثال اگر ما ساختار **student** بالا را در متن اصلی برنامه یا هر جای دیگرتعریف کردیم و بخواهیم آن را تعریف و مشخص کنیم به صورت زیر عمل می کنیم:

```
struct Student
   Dan = {"Smith, Dan", 210, 79.8},
   Jan = {"Brown, Jan", 201, 89.3};
```

دوساختار **Dan** و**Jan** توسط نام ،گروه و نمره تعریف شدند.

۵.۳.۸ ارزش دهی به مقدار ساختار

هر یک از اعضای ساختار به طور منفرد می توانند توسط "." ارزش دهی شوند. برای مثال اگر ما در قسمت معرفی متغیر آن را مقدار دهی نکرده باشیم، می توانیم هر کدام از پارامتر های ساختار به صورت زیر مقدار دهی کنیم:

```
Dan.name = "Smith, Dan"; Dan.section = 210; Dan.grade = 79.8
Jan.name = "Brown, Jan"; Jan.section = 201; Jan.grade = 89.3
```

۶.۳.۸ مقدار دهی دوباره اعضای ساختار

یک بار که تعریف و مقدار دهی می کنیم، می توانیم به صورت مرتب متغیر ها را به روش های مختلف مقدار دهی، نمایش و کنترل کنیم. به عنوان مثال ما می توانیم به صورت زیر عمل کنیم:

```
Jan.name = "Smith, Jan"; // Jan and Dan get married!
Jan.section = 210; // Jan moves over to Dan's section (bad idea)
```

ما همچنین می توانیم ساختار را به عنوان یک متغیر تنها در نظر بگیریم:

```
Dan = Jan;
```

که اینجا ساختار **"Dan"** ارزش تمام متغیر های **"Jan"** را می گیرد.

۷.۳.۸ فراخوانی ساختار توسط یک تابع بصورت ارسال مقادیر ساختار

از متغیر ساختار ها می توان به صورت **call by value** استفاده کرد. این بدین معناست که مقدار متغیر ساختار ها، عینا در تابع ها کپی می شوند.

در اینجا مثالی آورده شده است که **structure1.cpp** نام دارد:

```
//File: structure1.cpp

#include<iostream>
#include<string>

struct Student{string name; int section; float grade;};

void switchStruct(struct Student a, struct Student b)
{
   // I/O statements left out
   struct Student temp = a; a = b; b = temp; // Switch a and b
   // I/O statements left out
   return;
}
```

```
int main(void)
{
   struct Student
   Dan = {"Smith, Dan", 210, 79.8},
   Jan = {"Brown, Jan", 201, 89.3};
   // I/O statements left out
   switchStruct(Dan,Jan);
   // I/O statements left out
   cout << Dan.name <<endl;
   return 0;
}
```

مشاهده می کنید بعد از چاپ مقادیر در انتهای برنامه، مقدار متغیر هیچ فرقی نکرده است.

۸.۳.۸ فراخوانی تابع به صورت ارجاع ساختار

از راه های متداول دیگر، برای اسال مقادیر به تابع، ارجاع ساختار برای تابع می باشد که برای حفظ حافظه می باشد.در اینجا مثالی مشابه structure1.cpp است که فرق آنها در نحوه ی ارسال ساختار است که به صورت ارجاع ساختار عمل می کند:

```
//File: structure2.cpp

#include<iostream>
#include<string>

struct Student{string name; int section; float grade;};

void switchStruct(struct Student& a, struct Student& b)
{
   // I/O statements left out
   struct Student temp = a; a = b; b = temp; // Switch a and b
   // I/O statements left out
   return;
}

int main(void)
{
   struct Student
   Dan = {"Smith, Dan", 210, 79.8},
   Jan = {"Brown, Jan", 201, 89.3};
   // I/O statements left out
   switchStruct(Dan,Jan);
   // I/O statements left out
   Cout<<Dan.name<<endl;
   return 0;
}
```

همانطور که می بینید در انتهای برنامه بعد از چاپ متغیر مقدار ساختار ها عوض شده اند.

9.3.8 آرایه های ساختار

یکی از استفاده های متداول ساختار ها بوجود آوردن آرایه ای از آنهاست. بیایید یک آرایه از ساختار های **student** تعریف کنیم. فرض کنیم هریک از اجزای آرایه یکی از دانش آموزان کلاس **ENG101** باشند. در اینجا چگونگی انجام آن آمده است. کد آن **structure3.cpp** نام دارد که کلاسی از دانش آموزان موهومی با گروه و نمره بهِ وجود می آورد. سپس آن ها را به ترتیب نام یا گروه و درجه و هرچه شما بخواهید مرتب می کند.

```
//File: structure3.cpp
#include<iostream>
#include<iomanip>
#include<string>
#include<vector>
#include<cstdlib>

// Define a structure called Student
const int maxNameLength = 10;

struct Student
{
   char name[maxNameLength];
   int section;
   float grade;
};

bool switchStruct(struct Student& a, struct Student& b)
{
   struct Student temp = a; a = b; b = temp; // Switch a and b
   return false;
}

void sortStruct(vector<struct Student>& list, bool n, bool s, bool g)
{
   bool sorted;
   if (n)
   {
      do
      {
         sorted = true;
         for (int i = 0; i < list.size() - 1; i = i + 1)
         {
              string string1 = list[i].name;
              string string2 = list[i+1].name;
              if (string1 > string2)
              sorted = switchStruct(list[i],list[i+1]);
         }
      }while (!sorted);
   }
   else if (s)
   {
      do
      {
         sorted = true;
         for (int i = 0; i < list.size() - 1; i = i + 1)
            if (list[i].section > list[i+1].section)
                sorted = switchStruct(list[i],list[i+1]);
      }while (!sorted);
   }
   else if (g)
   {
      do
      {
          sorted = true;
         for (int i = 0; i < list.size() - 1; i = i + 1)
```

```
            if (list[i].grade < list[i+1].grade)
            sorted = switchStruct(list[i],list[i+1]);
        }while (!sorted);
    }
    return;
}


int main(void)
{
    cout << "How many students in the class? ";
    int nStuds;
    cin >> nStuds;
    vector<struct Student> myClass(nStuds);
    for (int i = 0; i < nStuds; i = i + 1)
    {
        myClass[i].section = 200 + rand()%10;
        myClass[i].grade = 0.1*(rand()%1001);
        int nameLength = 2 + rand()%(maxNameLength - 2);
        for (int j = 0; j < nameLength; j = j + 1)
       myClass[i].name[j] = 97 + rand()%26;
        myClass[i].name[nameLength - 1] = '\0';
       myClass[i].name[0] = myClass[i].name[0] - 32;
       cout << setw(maxNameLength) << myClass[i].name << ", "
            << myClass[i].section << ", "
            << myClass[i].grade << "\n";
    }
    bool sortByName = false;
    bool sortBySection = false;
    bool sortByGrade = false;
    cout << "Sort by name (0 (no) or 1 (yes))? ";
    cin >> sortByName;
    if (!sortByName)
    {
        cout << "Sort by section (0 (no) or 1 (yes))? ";
        cin >> sortBySection;
        if (!sortBySection)
        {
           cout << "Sort by grade (0 (no) or 1 (yes))? ";
           cin >> sortByGrade;
        }
     }
     sortStruct(myClass, sortByName, sortBySection, sortByGrade);
     for (int i = 0; i < nStuds; i = i + 1)
         cout << setw(maxNameLength) << myClass[i].name << ", "
              << myClass[i].section << ", "
              << myClass[i].grade << "\n";
     return 0;
}
```

۱۰.۳.۸ مثالی از استفاده ی ساختار ها و آرایه ها: انتقال یونها

برنامه زیر در کلاس توضیح داده خواهد شد.

```
//File: survivor.cpp

#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdlib>

const int NX = 20; // x-grid dimensions
const int NY = 7; // y-grid dimensions
const int STEPS = NY; // # of time steps
const int NCELL = 1000; // # of pairs per cell
const int TOTAL = (NX*NY*NCELL); // total # of pairs

struct N {int nE; int nP;}; // # electrons, # ions
ofstream resultsOnFile; // Open an output file stream

//-------------------------------------------------------------------------
//---------------------- Function definitions -----------------------
//-------------------------------------------------------------------------
// This function initializes the number of e-'s and ions in each cell
void init(struct N grid[][NY])
{
    for (int i = 0; i < NX; i = i + 1)
    {
       for (int j = 0; j < NY; j = j + 1)
       {
          grid[i][j].nE = NCELL;
          grid[i][j].nP = NCELL;
       }
    }
    return;
}

//-------------------------------------------------------------------------
// This function counts the number of e-'s
int countE(struct N grid[][NY])
{
   int count = 0;
   for (int i = 0; i < NX; i = i + 1)
   for (int j = 0; j < NY; j = j + 1)
   count = count + grid[i][j].nE;
   return count;
}

//-------------------------------------------------------------------------
// This function counts the number of positive ions
int countP(struct N grid[][NY])
{
   int count = 0;
   for (int i = 0; i < NX; i = i + 1)
   for (int j = 0; j < NY; j = j + 1)
   count = count + grid[i][j].nP;
   return count;
}

//-------------------------------------------------------------------------
// This function recombines the e-'s and ions
void recombine(struct N grid[][NY], int step)
{
   for (int i = 0; i < NX; i = i + 1) // Loop over all rows
   {
       for (int j = step; j < NY; j = j + 1) // Loop over all columns
```

```
        {
            int nE = grid[i][j].nE;
            for (int k = 1; k <= nE; k = k + 1) // Loop over all e's
            {
               for (int l = 1; l <= grid[i][j].nP; l = l + 1)
               {
                  // Potential annihilation with every ion in the cell
                  if (0 == rand()%2000) // Small chance of annihilation
                  {
                      // If there is an annihilation, reduce numbers by 1
                      grid[i][j].nE = grid[i][j].nE - 1;
                      grid[i][j].nP = grid[i][j].nP - 1;
                      break; // Break out of loop, only one life to give!
                  }
               }
            }
         }
      }
   return;
}


//---------------------------------------------------------------------
/* This function shifts the electrons to the right and counts how
many are collected. Note that, depending on the step, we can start
at the y index where the number of e-'s is non-zero.
*/
int tStep(struct N grid[][NY], int step)
{
    int escaped = 0;
    for (int i = 0; i < NX; i = i + 1)
    {
       escaped = escaped + grid[i][NY-1].nE;
       for (int j = NY - 2; j >= step; j = j - 1)
          grid[i][j+1].nE = grid[i][j].nE;
       grid[i][step].nE = 0;
    }
    return escaped;
}


//---------------------------------------------------------------------
// This function prints out the grid
void fGrid(struct N grid[][NY],int step)
{
   resultsOnFile << "Step: " << step << "\n";
   for (int i = 0; i <= NX - 1; i = i + 1)
   {
       for (int j = 0; j <= NY - 1; j = j + 1)
       resultsOnFile << "|" << setw(5) << grid[i][j].nE
                     << "," << setw(5) << grid[i][j].nP;
       resultsOnFile << "|\n";
   }
   return;
}


//---------------------------------------------------------------------
//---------------------------------------------------------------------
//--------------------------- Main routine ----------------------------
//---------------------------------------------------------------------
int main(void)
{
   struct N grid[NX][NY]; // grid is an N struct
   init(grid); // Initialize the grid
   // Output starting conditions
   cout << "Step " << setw(2) << 0
        << " : nE = " << setw(6) << countE(grid)
```

hts

```
            << " : nP = " << setw(6) << countP(grid)
            << " : nEsc = " << setw(6) << 0 << "\n";
    resultsOnFile.open("N.dat");
    fGrid(grid, 0); // Write the grid to file
    // Loop over time steps, set = to NY grid size
    int tEsc = 0; // Total number of escapees
    for (int step = 0; step < STEPS; step = step + 1)
    {
        recombine(grid, step); // Recombine e- & ions
        int nEsc = tStep(grid, step); // Take a time step
        cout << "Step " << setw(2) << step + 1
             << " : nE = " << setw(6) << countE(grid)
             << " : nP = " << setw(6) << countP(grid)
             << " : nEsc = " << setw(6) << nEsc << "\n";
        fGrid(grid, step + 1); // Write grid, next step
        tEsc = tEsc + nEsc; // Sum total escapes
    }
    resultsOnFile.close(); // Close the file
    cout << "Chance of survival = " << setprecision(3)
         << static_cast<float>(tEsc)/TOTAL << "\n";
    return 0;
}
//---------------------------------------------------------------------
```

فصل نهم

چند نکته ی کاربردی دیگر از زبان C++

۱.۹ تولید اعداد تصادفی

کتابخانه ی استاندارد **cstdlib** ، دارای ابزاری برای تولید اعداد تصادفی است. در واقع اعداد تصادفی تولید شده توسط این تابع اعداد شبه تصادفی[۱۶۴] هستند. تابع **rand()** از این کتابخانه، اعداد تصادفی صحیح بین صفر و عدد ثابت **RAND_MAX** تولید می کند. مقدار ثابت **RAND_MAX** در کتابخانه ی **cstdlib** تعریف شده است.

در زیر برنامه ای برای تولید اعداد تصادفی مشاهده می کنید. حواستان باشد، موقع استفاده از **rand()** حتما کتابخانه ی **cstdlib** را به کد خود اضافه کنید.

```
//File: rand0.cpp
#include <iostream>
#include <iomanip>
#include <cstdlib> // Need this for the rand() function
int main(void)
{
   cout << "RAND_MAX = " << RAND_MAX << "\n\n";
   cout << "The first 10 random numbers are...\n";
   for(int i = 1; i <= 10; i = i + 1)
       cout << "Random number "
            << setw(2)
           << i << " = "
           << rand() // Random int between 0 and RAND_MAX inclusive
           << "\n";
   return 0;
}
```

خروجی برنامه ی فوق به صورت زیر خواهد بود:

```
RAND_MAX = 2147483647

The first 10 random numbers are...
Random number 1 = 1804289383
Random number 2 = 846930886
Random number 3 = 1681692777
Random number 4 = 1714636915
Random number 5 = 1957747793
Random number 6 = 424238335
Random number 7 = 719885386
Random number 8 = 1649760492
Random number 9 = 596516649
Random number 10 = 1189641421
```

[164] Pseudo Random

اگر شما دوباره این برنامه را اجرا کنید، خواهید دید که این اعداد عینا تکرار می شوند! در واقع این مسئله به ساختار تولید اعداد تصادفی این تابع مربوط می شود که در هر دفعه باعث تولید اعداد یکسانی می شود.

در صورتی که شما مایل هستید اعداد کاملا تصادفی با ویژگی مورد نظر خود بدست آوردید، می توانید از تابع **srand()** استفاده کنید.

در زیر نمونه ای از این کار را می بینید :

```
//File: rand1.cpp
#include <iostream>
#include <iomanip>
#include <cstdlib> // Need this for the rand(), srand() functions
int main(void)
{
    cout << "RAND_MAX = " << RAND_MAX << "\n\n";
    cout << "Input any int to seed the random number generator: ";
    int seed;
    cin >> seed;
    srand(seed);
    cout << "The first 10 random numbers are...\n";
    for(int i = 1; i <= 10; i = i + 1)
        cout << "Random number "
             << setw(2)
             << i << " = "
             << rand() // Random int between 0 and RAND_MAX inclusive
             << "\n";
   return 0;
}
```

خروجی نمونه ی کد بالا به صورت زیر خواهد بود:

```
Compiling andexecuting the above code results in:
RAND_MAX = 2147483647

Input any int to seed the random number generator: 123456789
The first 10 random numbers are...
Random number 1 = 1965102536
Random number 2 = 1639725855
Random number 3 = 706684578
Random number 4 = 1926601937
Random number 5 = 71238646
Random number 6 = 1147998030
Random number 7 = 1038816544
Random number 8 = 940714160
Random number 9 = 789063065
Random number 10 = 464968134
```

یکی دیگر از راه ها برای تولید تصادفی تر اعداد این است با استفاده از تابع **time()** است. این تابع عضوی از کتابخانه ی **<ctime>** است که مقدار ثانیه های طی شده از نیمه ی شب اول ژانویه ی **1990** تا کنون را نشان می دهد.

در زیر مثالی از این کار را می بینید :

```
//File: rand2.cpp
#include <iostream>
#include <iomanip>
#include <cstdlib> // Need this for the rand() and srand() functions
#include <ctime> // Need this for the time() functio
int main(void)
{
   cout << "RAND_MAX = " << RAND_MAX << "\n\n";
   srand(time(NULL)); // Seed the random number generator
   cout << "The first 10 random numbers are...\n";
   for(int i = 1; i <= 10; i = i + 1)
       cout << "Random number "
            << setw(2)
            << i << " = "
            << rand() // Random int between 0 and RAND_MAX inclusive
            << "\n";
   return 0;
}
```

این تابع کاربرد های بسیاری در برنامه های مختلف دارد.

برای مثال اگر بخواهید اعداد تصادفی بین یک و سیزده تولید کنید:

```
rndNum = rand() % 13 + 1
```

یکی از کاربرد های تابع **rand()** تولید اعداد تصادفی اعشاری بین صفر و یک است:

```
//File: rand3.cpp
#include <iostream>
#include <iomanip>
#include <cstdlib> // Need this for the rand() and srand() functions
#include <ctime> // Need this for the time() function
int main(void)
{
   cout << "RAND_MAX = " << RAND_MAX << "\n\n";
   srand(time(NULL)); // Seed the random number generator
   int nRandoms = 10;
   cout << "The first " << nRandoms << " random numbers are...\n";
   for(int i = 1; i <= nRandoms; i = i + 1)
   {
       double randomDouble = rand()/static_cast<double>(RAND_MAX);
       cout << "Random number "
            << setw(2)
            << i << " = "
            << randomDouble // Random double between 0 and 1 inclusive
            << "\n";
   }
   return 0;
}
```

به عنوان مثالی از خروجی این برنامه داریم :

```
RAND_MAX = 2147483647

The first 10 random numbers are...
Random number 1 = 0.174225
Random number 2 = 0.130327
.
.
.
```

یک کاربرد تابع تولید کننده ی اعداد تصادفی؛ محاسبه ی عدد π

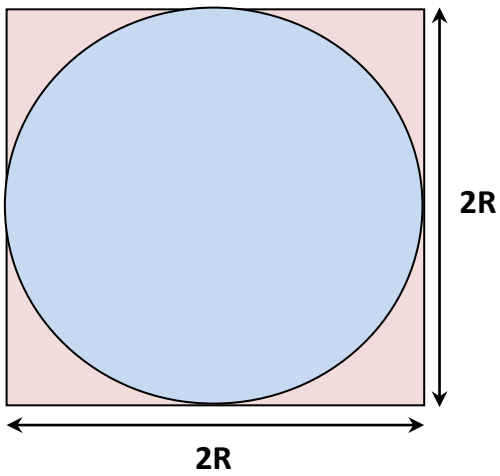
همانطور که می دانید پرتاب یک تیر به سمت نشانه ی دارت، در اعمال و تکرار زیاد، عملی تصادفی محسوب می شود.

حال تصور کنید دایره ای را که در داخل مربعی محاط شده است. به راحتی می توان نشان داد که نسبت مساحت دایره به مربع برابر است با $\pi/4$. در واقع در اینجا می خواهیم روشی را ارائه کنیم که بتوانیم با پرتاپ دارت های متوالی، مقدار عدد پی را با استفاده از نسبت مساحت ها، حساب کنیم!

شروع به پرتاپ دارت ها، به صورت تصادفی به سمت مربع دارت می کنیم!

دارت هایی که به خارج از مربع برخورد می کنند را نادیده می گیریم!

دارت هایی را که به داخل دایره برخورد می کنند را هم می شماریم!



اگر N_1 تعداد برخورد دارت هایی باشد که به دایره برخورد کرده است و N_2 تعداد دارت هایی باشد که به مربع برخورد کرده است، چون احتمال برخورد دارت به مربع در پرتاپ های زیاد برابر با نسبت مساحت های آنهاست، داریم :

$$S_1 = \pi R^2$$

$$S_2 = 4R^2 \quad \Longrightarrow \quad P = \frac{S_1}{S_2} = \frac{\pi}{4} \quad \Longrightarrow \quad \pi = 4P = 4\frac{N_1}{N_2}$$

از این روابط توانستیم تقریبی برای عدد π بدست آوریم!

امن ترین راه(!) برای بدست آوردن این تقریب عبارتست از استفاده از کامپیوتر برای تولید اعداد تصادفی:

```
//File: randomPi.cpp
#include <iostream>
#include <cstdlib> // Need for rand() and srand()
#include <cmath> // Need for pow() and atan()
#include <ctime> // Need for time()
/* Define a conversion function that converts the int returned from
rand() to a double between -1.0 and 1.0 inclusive
*/
#define MYRAND (2.*rand()/RAND_MAX - 1.)
int main(void)
{
   int nDarts = 10000; // Number of darts to throw
   int nInside = 0; // Counter for those that land inside
   srand(time(NULL)); // Seed the random number generator
   for(int i = 1; i <= nDarts; i = i + 1)
   {
      double xDart = MYRAND;
      double yDart = MYRAND;
      if (pow(xDart,2) + pow(yDart,2) <= 1.0) nInside = nInside + 1;
   }
   double pi = 4.0*nInside/nDarts; // Estimate for Pi
   double Pi = 4.0*atan(1.0); // This is how you can get a very
   // accurate estimate of the real Pi!
   cout << "The estimated value of Pi is " << pi << "\n";;
   cout << "The true value of Pi is " << Pi << "\n";;
   return 0;
}
```

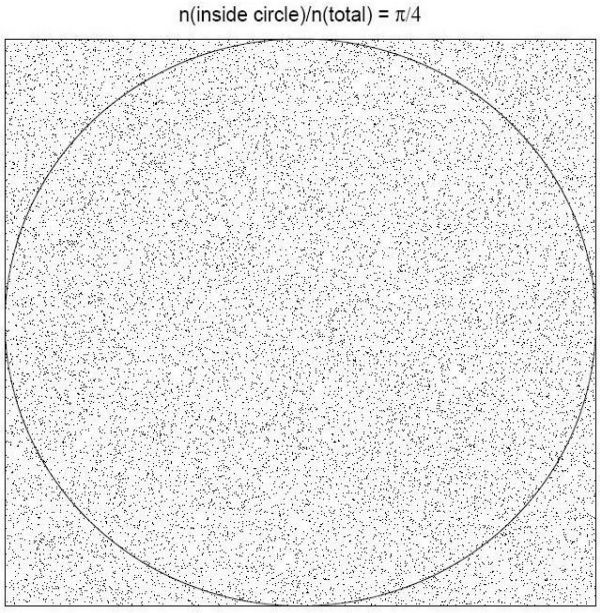
در کد بالا ویژگی هایی وجود دارد که احتمالا شما را گیج خواهد کرد:

- عبارت **#define MYRAND (2.*rand()/RAND_MAX - 1.)** که در قسمت پیش پردازنده قرار دارند، باعث می شوند تا هرجا که عبارت **MYRAND** استفاده شده باشد، به جای آن ، عبارت **(2.*rand()/RAND_MAX - 1.)** جایگزین شود. در واقع این عمل قبل از کامپایل و توسط پیش پردازنده انجام می شود. در واقع با استفاده از دستور **#define** می توانید کار خود را بسیار آسان کنید. البته باید مراقب این دستور باشید، چرا که معمولا کامپایلر های **C++** ، خطاهای دقیقی در رابطه با اشکال برنامه در این قسمت، نمی دهند.
- عبارت **(2.*rand()/RAND_MAX - 1.)** عدد تصادفی حقیقی بین **1** و **-1** تولید می کند. با توجه به نحوه ی پیاده سازی این دستور، می توانید آن را در ساختار های مشابهی بکار ببرید.
- برای پرتاب دارت، شما نیاز به دو عدد تصادفی دارید. یکی مربوط به مختصات محور **x** ها و دیگری مربوط به مختصات مربوط به **y** ها که هرکدام بین ۱ و ۱- قرار دارند. بنابرین گوشه های این مربع عبارتست از:

(x, y) = (-1,-1), (-1, 1), (1, 1), (1,-1).

- تابع **pow()** عضوی از کتابخانه ی ریاضی **<cmath>** می باشد که برای به توان رساندن به کار می رود. برای مثال **pow(x,3)** معادل است با x^3
- عبارت **pow(xDart,2) + pow(yDart,2) <= 1.0** معادل است با $x^2+y^2 \leq 1$ که محاسبه می کند که آیا نقطه در داخل دایره قرار دارد یا خارج از آن است.
- تابع **atan()** عضوی از کتابخانه ی **<cmath>** معادل عبارت **arctan** در ریاضیات است که مقدارا خروجی آن بر اساس رادیان است.

در شکل زیر شما صفحه ی دارتی را می بینید که ۲۰۰۰۰ پرتاب بر روی آن انجام شده است.



۱.۱.۹ یک مثال دیگر؛ انتگرال گیری معین از تابع توسط توالید اعداد تصادفی

در اینجا می خواهیم با استفاده از ایده ی مشابهی، برای انتگرال گیری از یک تابع استاده کنیم. در واقع با توجه به نسبت نقاطی که در زیر تابع قرار دارند که کل نقاط پخش شده، مساحت زیر منحنی را بدست آوریم. به کد زیر توجه کنید:

```
//File: randomIntegrate.cpp
#include <iostream>
#include <iomanip>
#include <cstdlib> // Need for rand() and srand()
#include <cmath> // Need for exp(), the exponential function

// and abs(), the absolute value function
/* Define a conversion function that converts the int returned from
rand() to a double between 0.0 and 1.0 inclusive
*/
#define MYRAND (static_cast<double>(rand())/RAND_MAX)
```

```
int main(void)
{
   int nDarts = 1000000000, under = 0;
   double Ans = 1 - exp(-1);
   for(int i = 1; i <= nDarts; i = i + 1)
   {
      double xDart = MYRAND;
      double yDart = MYRAND;
      if (yDart <= exp(-xDart)) under = under + 1;
      if (0 == i%1000000) // Every million times
      {
          double ans = static_cast<double>(under)/i;
          cout << setiosflags(ios::fixed) << setprecision(8)
               << "#" << setw(10) << i
               << ": I = " << setw(10) << ans
               << ", Delta = " << abs(1 - exp(-1) - ans) << "\n";
       }
   }
   return 0;
}
```

به نکاتی از کد بالا توجه کنید:

- تابع **exp()**[165] عضو کتابخانه ی **<cmath>** است که معادل با تابع e^x است.
- تابع **abs()**[166] عو کتابخانه ی **<cmath>** و معادل با قدر مطلق در ریاضیات است.
- عبارت **setiosflags(ios::fixed)** باعث می شود که تا اعداد اعشاری به همراه صفر هایشان در انتهای عدد چاپ شوند. این کار تنها برای فرمت بندی مناسب اعداد در هنگام چاپ است.
- تابع **setw()** برای مشخص کردن عرض مقدار خروجی در صفحه ی نمایش است. این تابع در کتابخانه ی **<iomanip>** تعریف شده است.
- تابع **setprecision()** برای تنظیم کردن دقت عدد اعشاری چاپ شده در خروجی بکار می رود. این تابع در کتابخانه ی **<iomanip>** قرار دارد.

خروجی برنامه ی بالا مشابه زیر خواهد بود :

```
# 1000000: I = 0.63213700, Delta = 0.00001644
# 2000000: I = 0.63193800, Delta = 0.00018256
# 3000000: I = 0.63197933, Delta = 0.00014123
# 4000000: I = 0.63180525, Delta = 0.00031531
# 5000000: I = 0.63187300, Delta = 0.00024756
.
.
.
```

[165] Exponential
[166] Absolute

۲.۹ یک نمونه مرتب سازی؛ مرتب سازی حبابی

مرتب سازی که در ادامه معرفی خواهیم کرد، به این دلیل حبابی نام گرفته است که، داده های کوچک را را به بالا می برد و داده های بزرگ را به پایین می آورد!

الگوریتم به شکل است که در هر دفعه 2 مقدار کنار سر هم مورد بررسی قرار می گیرند. در صورتی که مقدار بالایی، بزرگتر از مقدار پایینی باشد، جای آندو را عوض می کند. در غیر اینصورت هیچ کاری نمی کند. سپس جفت بعدی در نظر گرفته می شود دوباره همین مقایسه روی آنها انجام می شود. این کار تا انجا روی مجموعه ی داده ها انجام می شود که در آخرین حلقه ی گردش روی داده ها، هیچ تغییر مکانی صورت نگرفته باشد.

برای مثال آرایه ی **{3, 2, 1}** را در نظر بگیرید:

Loop 1: {3, 2, 1} : {(3, 2), 1} ⇒ {2, (3, 1)} : {2, 3, 1} ادامه ,جابجایی انجام گرفت

Loop 2: {2, 3, 1} : {(2, 3), 1} ⇒ {2, (3, 1)} : {2, 1, 3} ادامه ,جابجایی انجام گرفت

Loop 3: {2, 1, 3} : {(2, 1), 3} ⇒ {1, (2, 3)} : {1, 2, 3} ادامه ,جابجایی انجام گرفت

Loop 4: {1, 2, 3} : {(1, 2), 3} ⇒ {1, (2, 3)} : {1, 2, 3} پایان ,بدون جابجایی

در تکه سودوکد زیر قسمتی که این عمل را انجام می دهد را می بینید:

```
bool whileThingsAreChanging = true;
while(whileThingsAreChanging)
{
   whileThingsAreChanging = false; //Assume that the list is sorted
   for(/* appropriate for-loop parameters */) //Careful!
   {
       //Loop through the list in an order fashion in one direction
       //Compare adjacent pairs of things
       //If the pair is out of order, switch them and set
       // whileThingsAreChanging to true
   }
   //When the for loop completes, either nothing has changed and
   //whileThingsAreChanging has remained false, or something changed and
   //whileThingsAreChanging was changed to true
}
//When the execution arrives here, the list is sorted.
```

۳.۹ توابع بازگشتی

گاهی در ریاضیات روش هایی وجود دارند که می توان مسائل را به صورت فشرده تری حل کرد. توابع بازگشتی از جمله ی آن توابعی هستند که می توان با استفاده از آن ها، مسائل بسیار مشکلی را در کمترین میزان کد، حل کرد. میتوان گفت که تابع بازگشتی، تکنیکی برای حل مسئله است.

فاکتوریل اعداد صحیح را در نظر بگیرید:

$$N! = N \times (N-1) \times (N-2) \cdots 3 \times 2 \times 1$$

می توان این فاکتوریل را به صورت فشرده و به صورت مقابل نوشت:

$$\begin{cases} \forall N >= 0 \rightarrow N! = N \times (N-1)! \\ 0! \equiv 1 \end{cases}$$

حال به نحوه ی استفاده از تابع بازگشتی در برنامه ی زیر توجه کنید:

```
//File: recursiveFactorial.cpp
#include <iostream>

int factorial(int n)
{
    if (0 == n)
        return 1;
    else
        return (n * factorial(n - 1));
}

int main(void)
{
    cout << "N? ";
    int N;
    cin >> N;
    cout << N << "! = " << factorial(N) << endl;
    return 0;
}
```

با توجه کردن به صورت کد، می توانید دریابید که این روش چقدر شبیه به فرمول ریاضی بازگشتی است که در بالا نوشته شده است.

همچنین به شرایط **0!** نیز توجه کنید که چگونه پیاده سازی شده است.

در هنگام پیاده سازی تابع بازگشتی می توانید به شرایط زیر توجه کنید:

- توابع بازگشتی در داخل خود ساختاری ساده تر از خود را فراخوانی می کنند. به عنوان نمونه **N!** ساده تر از **(N-1)!** است.
- برای هر تابع بازگشتی، معیاری برای پایان بازگشت ها وجود دارد. در مثلا قبل **0! = 1** باعث پایان بازگشت ها می شد.
- اگر هر تابع، در هر اجرا خود را بیش از یک بار، ساختار هایی از خود را اجرا کند، این کار باعث کند شدن اجرای برنامه می شود و کارایی برنامه را پایین می آورد.[۱۶۷]
- با توجه به مورد قبلی می توان گفت، توابع بازگشتی ابزار خوبی برای مختلل کردن اعمال کامپیوتر می باشد!

[۱۶۷] چنین مواردی در برنامه هایی همچون بازی شطرنج دیده می شود.

برای نشان دادن مثال سوم، مثالی را ارائه می کنیم. دنباله ی فیبوناچی را در نظر بگیرید:

$$\begin{cases} f(N) = f(N-1) + f(N-2) ; & \forall N > 0 \\ f(1) \equiv 1 \\ f(2) \equiv 1 \end{cases}$$

به پیاده سازی دنباله ی فیبوناچی بر اساس تابع بازگشتی، در کد زیر توجه کنید:

```
//File: recursiveFibonacci.cpp
#include <iostream>

int fibonacci(int n)
{
    if (1 == n || 2 == n)
        return 1;
    else
        return (fibonacci(n - 1) + fibonacci(n - 2));
}

int main(void)
{
    cout << "N? ";
    int N;
    cin >> N;
    cout << "Fibonacci(" << N << ") = " << fibonacci(N) << endl;
    return 0;
}
```

مشاهده می کنید، این تابع بازگشتی، دوبار خودش را فراخوانی می کند. همین باعث می شود تا اجرای برنامه به ازای ورودی های بالای 30 ، به طرز چشم گیری کند شود. باور نمی کنید ؟! $f(1000)$ را امتحان کنید!

دیدید که برنامه ی بالا، با اینکه با کمترین تعداد خطوط، نوشته شده بود، ولی کارایی جالبی نداشت. لذا گاهی خوب است که تمیزی و آسانی برنامه را فدای کارایی آن بکنیم.

در زیر پیاده سازی دیگری از دنباله ی فیبوناچی با استفاده از حلقه ها را می بینید:

```
//File: loopFibonacci.cpp
#include <iostream>
int fibonacci(int n)
{
   if (1 == n || 2 == n)
       return 1;
   else
   {
      int fMinus2 = 1;
      int fMinus1 = 1;
      int f;
      for (int i = 3; i <= n; i = i + 1)
      {
          f = fMinus2 + fMinus1;
          fMinus2 = fMinus1;
          fMinus1 = f;
```

```
        }
        return f;
    }
}

int main(void)
{
    cout << "N? ";
    int N;
    cin >> N;
    cout << "Fibonacci(" << N << ") = " << fibonacci(N) << endl;
    return 0;
}
```

کد بالا گرچه، با تعداد خطوط بیشتری پیاده سازی شدهاست، ولی سرعت اجرای آن به مراتب از روش بازگشتی بیشتر است.

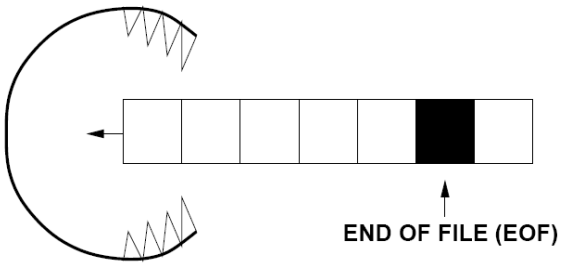
۴.۹ کار با فایل ها در زبان C++

۱.۴.۹ چرا باید با فایل ها کار کنیم؟

تا اینجا در اغلب برنامه هایی که نوشته ایم، معمولا ورودی ها را با استفاده از کیبو رد به برنامه داده ایم. اما همیشه امکان این امر وجود ندارد. مثلا ممکن است داده ها آنقدر زیاد باشند که نتوانیم همه ی آنها را با استفاده از کیبورد به برنامه دهیم. اینجاست که می توانیم ورودی ها را با استفاده از فایل به برنامه دهیم. (البته این تنها یک نمونه از کاربرد های فایل ها بود.)

۲.۴.۹ C++ چگونه به فایل ها نگاه می کند؟

C++ فایل ها را به عناون مجموعه و مجموعه ای از جریان های[۱۶۸] بایت می بیند. پایان هر فایل توسط نشان گذار EOF[۱۶۹] مشخص می شود.



[168] stream
[169] End Of File

زمانی که از طرف برنامه درخواستی مبنی بر دسترسی به فایل به سیستم عامل فرستاده می شود، محتوای فایل به جریانی داده می شود که **C++** تنها به جریان دسترسی دارد. لذا هیچ دسترسی مستقیمی از طرف **C++** به محتوای دیسک سخت وجود ندارد. جریان بیت های فایل توسط **C++** ترجمه می شوند و به برنامه داده می شوند. مستقل بودن **C++** از خواندن مستقیم اطلاعات از روی سخت افزار، باعث مستقل بودن این عمل از سیستم عامل که برنامه بر روی آن اجرا می شود، شده است.

کتابخانه ی استاندارد **iostream**[۱۷۰] و **fstream**[۱۷۱] شامل تمامی دستورات و توابع مورد نیاز برای کار با جریان فایل ها در سیستم عامل را دارا می باشند.

زمانی که کتابخانه ی **<iostream>** به فایل اضافه می شود، می توان با سه نوع از جریان ها استفاده کرد:

۱. **stdin**[۱۷۲] معمولا رشته ای است که توسط کیبورد به سیستم عامل داده می شود و با استفاده از "**cin >>**" می توان به آن دسترسی پیدا کرد.
۲. **stdout**[۱۷۳] رشته ای است که برای چاپ در صفحه بکار می رود و برای چاپ آن از "**cout <<**" استفاده می شود.
۳. **stderr**[۱۷۴] رشته ای ایت که برای چاپ در صفحه بکار می رود و برای چاپ آن از "**cerr <<**" استفاده می شود. بطور جزئی تر، کاربرد این تابع زمانی است که لازم است کاربر از خطا یا اخطاری در زمان اجرای برنامه آگاه شود.

در ادامه نشان خواهیم داد که چگونه می توان با استفاده از **C++**، به ایجاد، نوشتن و خواندن فایل بپردازیم.

۳.۴.۹ ایجاد و نوشتن به فایل ها

۱. **#include <fstream>** را فراموش نکنید !
۲. تعریف اسم شی که باید مشخصات فایلی که در آن نوشته می شود، در آن ذخیره شود، از کلاس **ofstream**، به صورت زیر :

```
ofstream myOutputFile;
```

۳. قبل از شروع به نوشتن به فایل، باید فایل مربوطه توسط متود **open()** باز شود:

```
myOutputFile.open("someData.dat");
```

[170] keyboardand screen-related Input/Output [I/O]
[171] file-related I/O
[172] Standard Input
[173] Standard output
[174] Standard Error

همچنین این متود، فایل **someData.dat** را به عناون فایلی که قرار است، اطلاعاتی در آن ذخیره شود، در شی **myOutputFile** ذخیره می کند.

برای مثال در لینوکس/یونیکس می توانید این چنین وارد کنید :

```
myOutputFile.open("~/Private/someData.dat");
```

یا در ویندوز این چنین فایل را باز کنید :

```
myOutputFile.open("C:\\eng101\\hw\\hw6\\hw6.dat");
```

آیا می توانید دلیل استفاده از دو \ را بگویید؟ (شما قبلا دلیل آن را خوانده اید!)

توجه! متود **open()** تنها برای استفاده و باز کردن فایل هایی است که از قبل وجود داشته اند.

۴. هر بار که فایلی را باز می کنید، کنترل کنید که آیا ب ه طور کاملا درست باز شده است. برای این کار از متود **fail()** استفاده کنید. زمانی که باز کردن فایل با شکست روبرو شده باشد، این متود **true** برمی گرداند.

```
if (myOutputFile.fail())
{
   //File open fail
   return 1; // Return to calling function with abnormal termination code.
}
```

۵. همانند تابع **cout** می توانید به فایل خود اطلاعاتی را ذخیره کنید. برای مثال :

```
int i = 1, j = 2;
myOutputFile << i << ", " << j << endl;
```

۶. زمانیکه کار نوشتن به فایل به اتمام رسید، با استفاده از متد **close()** فایل را ببندید.

```
myOutputFile.close();
```

هدف از بستن فایل این است که اولا منابعی از سیستم عامل که در اختیار برنامه گرفته شده اند، آزاد شوند، و اینکه در صورت **crash** کردن سیستم عامل، نوشتن فایل ها ناقص نماند. (البته نکته ی دوم مسئله ای است که بیشتر به سیستم عامل بستگی دارد.)

در زیر به نمونه ی کاملی از برنامه برای دسترسی و کار فایل ها را می بینید :

```
//File: fileOutput.cpp
#include <iostream>
#include <fstream>
using namespace std;
int main(void)
{
   ofstream myOutputFile; // Create an output file object
```

```
    cout << "What filename do you want to write to: ";
    char fileName[20]; // 1D array of chars
    cin >> fileName;
     myOutputFile.open(fileName); //Connect the stream to the file
    if (myOutputFile.fail())
    {
        // File could not be opened
        cerr << "File called " << fileName << " could not be opened.\n";
        return 1; // Return to O/S with abnormal return code
    }
    else
        cout << "File called " << fileName << " was successfully opened.\n";

    bool keepReading = true; // Keep reading if true
    do
    {
         cout << "Two floats? ";
         float x, y; // Two input floats
         cin >> x >> y;;
        if (cin.eof())
            keepReading = false; // End of file <CNTL>-D detected
        else
            myOutputFile << x << " " << y << endl;
    }while(keepReading);
    cout << "End of input detected\n";
    myOutputFile.close(); //Disconnect the stream from the file
    return 0;
}
```

این کد از متود **eof()**. بررسی می کند آیا جریانی مبنی بر **EOF** دریافت کرده است یا نه ؟

cin.eof() زمانی که **true** برمی گرداند که **EOF** دریافت شده باشد، در غیر اینصورت مقدار **false** را به عنوان خروجی بر می گرداند. در واقع در این برنامه از **eof()** به عنوان نگهبانی برای بررسی اینکه آیا کنترل برنامه به پایان برنامه رسیده است یا نه استفاده می شود. برای ایجاد سیگنال **EOF**، در سیستم عامل لینوکس/یونیکس، از **ctrl+d** و در سیستم عامل ویندوز از **ctrl+z** استفاده می شود.

۴.۴.۹ خواندن از فایل ها در C++، دسترسی ترتیبی به فایل

در بحث خواندن اطلاعات از فایل، با همان **6** حالت قبلی سر و کار داریم، بجز اینکه در هنگام تعریف شی برای فایل، باید از کلاس **ifstream** به جای **ofstream** استفاده کنیم و همچنین از عملگر **>>** برای خواندن، به جای عملگر **<<** برای نوشتن استفاده کنیم.

برنامه ی زیر، برنامه ی کاملی برای نوشتن به فایل است. آن را به دقت اجرا کرده و نتیجه ی آن را مشاهده کنید:

```
//File: fileInput.cpp
#include <iostream>
#include <fstream>
using namespace std;
int main(void)
{
   ifstream myInputFile; // Create an input file object
   cout << "What filename do you want to read from: ";
   char fileName[20]; // 1D array of chars
   cin >> fileName;
   myInputFile.open(fileName); //Connect the stream to the file
```

```
   if (myInputFile.fail())
   {
      // File could not be opened
      cerr << "File called " << fileName << " could not be opened.\n";
      return 1; // Return to O/S with abnormal return code
   }
   else
      cout << "File called " << fileName << " was successfully opened.\n";
   bool keepReading = true; // Keep reading if true
   do{
      float x, y; // Two floats
      myInputFile >> x >> y;;
      if (myInputFile.eof())
          keepReading = false; // End of file detected
      else
          cout << x << " " << y << endl;
   }while(keepReading);
   cout << "End of input detected\n";
   myInputFile.close(); //Disconnect the stream from the file
   return 0;
}
```

باز کردن کردن مطمئن فایل ها برای نوشتن

برای مراقبت کردن از فایل هایی که از قبل وجود داشته اند، زمانی که می خواهیم فایلی را برای نوشتن باز کنیم، باید ابتدا این فایل را ابتدا برای خواندن باز کنید و امتحان کنید که آیا باز کردن فایل با موفقیت انجام می شود یا نه؟ در صورتی که باز کردن این فایل موفقیت آمیز نباشد، نشان دهنده ی آن است که این چنین فایلی وجود ندارد. در صورتیکه فایلی با این اسم وجود داشته باشد، باید به کاربر اعلام کنید تا اسم جدید برای فایل وارد کند.

برنامه ی زیر چنین الگوریتمی را نشان می دهد:

```
//File: fileSafeOutput.cpp
#include <iostream>
#include <fstream>
using namespace std;
int main(void)
{
   ifstream testFile; // Create an output file object
   ofstream myOutputFile; // Create an output file object
   cout << "What filename do you want to write to: ";
   char fileName[20]; // 1D array of chars
   cin >> fileName;
   testFile.open(fileName);
   if (!testFile.fail())
   {
       cout << "File called " << fileName << " already exists.\n";
       cout << "Overwrite (Input 0 for no, any other int to overwrite): ";
      int clobberIt;
      cin >> clobberIt;
      testFile.close();
      if (!clobberIt) return 0;
   }
   myOutputFile.open(fileName); //Connect the stream to the file
   if (myOutputFile.fail())
   {
      // File could not be opened
```

```
        cerr << "File called " << fileName << " could not be opened.\n";
        return 1; // Return to O/S with abnormal return code
    }
    else
        cout << "File called " << fileName << " was successfully opened.\n";
    bool keepReading = true; // Keep reading if true
    do{
       cout << "Two floats? ";
       float x, y; // Two input floats
       cin >> x >> y;;
       if (cin.eof())
           keepReading = false; // End of file <CNTL>-D detected
       else
          myOutputFile << x << " " << y << endl;
    }while(keepReading);
    cout << "End of input detected\n";
    myOutputFile.close(); //Disconnect the stream from the file
    return 0;
}
```

۵.۹ استفاده از ورودی خط فرمان

بیشتر دستورات سیستم عامل لینوکس/یونیکس به زبان **C/C++** نوشته شده اند. شاید تا بحال برای گرفتن لیست فایل ها در یک پوشه از دستور **ls** استفاده کرده باشید. در صورتی که در ورودی مقدار **ls –l** وارد کنید، توضیحات فایل های موجود در پوشه ی جاری را به صورت جامع تر و طویل تر(**long**) نشان خواهد داد. در صورتی که **ls *.cpp** را وارد کنید تمامی فایل های به زبان **c++** را برای شما لیست خواهد کرد....

اینجاست که این سوال پیش می آید برنامه نویس چگونه این مقادیر ورودی را از کاربر می گیرد؟

در زیر نمونه ی کوچکی از این کار وجود دارد:

```
//File: commandLine.cpp
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
   cout << "argc = " << argc << endl;
   cout << "This program's name is: " << argv[0] << endl;
   for (int I = 1; I < argc; I = I + 1)
   {
      cout << "Argument " << I << " is: " << argv[i] << endl;
   }
   return 0;
}
```

به نحوه ی تعریف کردن **main** برای اینکار توجه کنید :

```
int main(int argc, char* argv[])
{
    .
    .
    .
}
```

زمانی که تابع **main** فراخوانی می شود، **argc** شامل تعداد کلمات ورودی است. برای مثال اگر شما برنامه ی خود را به این صورت فراخوانی کنید:

```
a.out word anotherWord
```

مقدار **argc** برابر خواهد بود با **3** . **argv[]** ، شامل آرایه ای از آدرس هاست. آدرس هایی که به اولین کاراکتر از محتویات ورودی اشاره می کند. برای درک درست تر، به برنامه ی زیر توجه کنید:

```
//File: fileOpen.cpp
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char* argv[])
{
   ofstream myOutputFile; // Create an output file object
   char fileName[20];
   if (argc == 1)
   {
       cout << "What filename do you want to write to: ";
       cin >> fileName;
       myOutputFile.open(fileName);
   }
   else if (argc == 2)
   {
       myOutputFile.open(argv[1]);
   }
   else
   {
       cerr << "Usage: a.out [filename]\n";
   }
   if (myOutputFile.fail())
   {
       // File could not be opened
       cerr << "File could not be opened.\n";
       return 1; // Return to O/S with abnormal return code
   }
   myOutputFile.close(); // Disconnect the stream from the file
   return 0;
}
```

درصورتیکه کاربر یک کلمه در ورودی بنویسد، به اسم فایل در نظر گرفته می شود، اگر دو کلمه بنویسد، کلمه ی دوم به عنوان اسم برنامه در نظر گرفته می شود، در غیر اینصورت، برنامه به کاربر پیغام خطا می دهد.

۶.۹ تمرینات

۱.۶.۹ شبه فاکتوریل

برنامه ای بنویسید که شبه فاکتوریل زیر را به صورتی بازگشتی حساب کند.

$$N!! = N \times (N-2) \times (N-4) \cdot \cdot \cdot (2 \text{ or } 1)$$

توجه داشته باشید که تابع بازگشتی شما زمانی که به مقدار ورودی یک یا دو برخورد می کند، باید بایستد.

۲.۶.۹ مرتب سازی یک ساختار

ساختار زیر را در نظر بگیرید :

```
struct Student
{
   int studentNumber;
   float finalGrade;
};
```

بطوریکه **studentNumber** عددی ۹ رقمی است که شماره ی دانشجویی هر فرد است و **finalGrade** مقدار نمره ی هر دانشجو را در امتحان پایانی انجام می دهد.

شما باید تابعی بنویسید که با فراخوانی ارجاعی(**call-by-reference**) آرایه ی پویایی از **struct** هایی ا زنوع بالا **vector<struct Student>** را بر اساس نمره ی هر فرد و به صورت نزولی، مرتب کند .

در پایان:

این کتاب که توسط گروهی از دانشجویان دانشکده مهندسی برق دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران) ترجمه شده به صورت کاملا رایگان در اختیار شما خواننده گرامی قرار گرفته است. هرگونه استفاده مالی از این اثرِ آزاد، برخلاف قوانین نشر محتوای آزاد است.

این پروژه نیز مانند تمامی پروژه های دیگر خالی از اشکال نیست. لطفا در صورتی که اشکال یا اشتباهی در این متن مشاهده کردید آن را به ما یادآوری کنید.

با تشکر

اعضای گروه ترجمه در این پروژه:

۱ – کیوان : nouri.keyvan@gmail.com
۲ – دانیال : d.khashabi@gmail.com
۳ – مهدی : en.yousefi@yahoo.com
۴ – شیوا : shiva.navabi@gmail.com