

بسم الله الرحمن الرحيم

« برنامه سازی در C++ »

استاد: مرتضی عباسزاده

عضو هیات علمی دانشگاه آزاد اسلامی

دانشگاه آزاد اسلامی تبریز

پاییز 1401

صفحه	فهرست مطالب
8	فصل اول: زبانهای برنامه سازی و کلیات C++
8	تعریف الگوریتم
8	خصوصیات الگوریتم
9	تعریف فلوجارت یا روندنما
10	تعریف برنامه
10	خوانایی کد منبع
11	پارادایم (الگو)
11	زبان های برنامه نویسی
12	زبان برنامه نویسی C++
12	شروع به کار با C++
12	نصب IDE C++
13	نوشتن برنامه در C++
13	شکل 1-1: محیط برنامه نویسی Code::Blocks
14	قالب کلی برنامه ها در C++
14	قوانین کلی در C++
14	متغیر
14	ثابت
15	انواع داده ها در C++
15	دستورات پیشوندی داده ها

16	انواع عملگرها در C++
17	برنامه 1: برنامه خوش آمدگویی
17	آشنایی با دستور cout و cin
18	برنامه 2: جمع دو عدد
19	برنامه 3: مجموع و میانگین سه عدد
19	دستور endl
20	فصل دوم: ساختارهای شرطی
20	ساختار if
20	ساختار if else
20	ساختار else if
20	دستور switch
20	عملگر سه گانه
20	دستور break
20	دستور continue
21	برنامه 4: زوج یا فرد بودن عدد
21	ردیابی (Tracing)
22	برنامه 5: قبول یا مردود بودن
23	برنامه 6: معدل مردود، مشروط، عادی، ممتاز، نامعتبر
25	برنامه 7: معادل روزهای هفته
26	فصل سوم: ساختارهای حلقه تکرار

26	ساختار while
26	ساختار for
26	ساختار do...while
26	ساختار goto
27	برنامه 8: مجموع اعداد 1 تا 10
28	برنامه 9: مجموع اعداد فرد 1 تا 20
29	برنامه 10: مجموع اعداد 1 تا N
30	برنامه 11: فاکتوریل عدد N
31	برنامه 12: مجموع اعداد زوج 1 تا N
32	برنامه 13: ارقام عدد N از آخر به اول
33	برنامه 14: مجموع ارقام یک عدد
34	برنامه 15: وارون یک عدد
35	برنامه 16: مجموع ارقام زوج و ضرب ارقام فرد یک عدد
36	برنامه 17: A به توان B با *
37	برنامه 18: A ضربدر B با +
38	برنامه 19: A ضربدر B با + (روش دوم)
39	برنامه 20: مقسوم علیه های یک عدد
40	برنامه 21: کامل بودن یک عدد
41	برنامه 22: اول بودن عدد (روش کند)
42	برنامه 23: اول بودن عدد (روش سریع)

برنامه 24: N جمله از دنباله فیبوناچی

43

برنامه 25: جمله N ام از دنباله فیبوناچی

44

حلقه های تودرتو

45

برنامه 26: جدول ضرب 10×10

46

برنامه 27: برنامه ی ستاره ها

47

برنامه 28: اعداد اول دو رقمی

48

برنامه 29: اعداد کامل کوچکتر از N

49

برنامه 30: مقسوم علیه های اول یک عدد

50

فصل چهارم: آرایه ها

51

آرایه یک بعدی

51

برنامه 31: پیدا کردن ماکزیمم

52

برنامه 32: میانگین نمرات دانشجو

53

برنامه 33: تبدیل عدد به مبنای دودویی

54

برنامه 34: بررسی پالیندروم بودن رشته

55

آرایه دو بعدی

56

برنامه 35: جمع دو ماتریس

57

برنامه 36: ترانهاده ی ماتریس

58

برنامه 37: ضرب دو ماتریس

59

آرایه چند بعدی

60

فصل پنجم: زیربرنامه ها

61

61	اعلان زیربرنامه ها
62	فراخوانی زیربرنامه ها
62	انواع فراخوانی ها (با مقدار، با ارجاع)
63	نوشتن خود زیربرنامه ها
64	برنامه 38: میانگین سه عدد با زیربرنامه ها
65	برنامه 39: زوج یا فرد بودن با زیربرنامه ها
66	برنامه 40: مقسوم علیه های وارون یک عدد با زیربرنامه ها
67	برنامه 41: بزرگترین و کوچکترین عنصر آرایه با زیربرنامه ها
68	توابع بازگشتی (Recursive Functions)
69	برنامه 42: فاکتوریل به روش بازگشتی
70	برنامه 43: جمله ی n ام فیبوناچی به روش بازگشتی
71	فصل ششم: اشاره گر ها
72	برنامه 44: تعویض مقادیر دو متغیر با اشاره گر ها
73	برنامه 45: جستجوی خطی با اشاره گر ها
74	اشاره گر به اشاره گر
74	برنامه 46: دستکاری یک متغیر با اشاره گر به اشاره گر
75	فصل هفتم: فایل ها
75	انواع فایل ها
75	فایل متنی
75	فایل دودویی

75 عملیات روی فایل ها

77 برنامه 47: ایجاد فایل و ذخیره جملات ورودی کاربر در فایل

78 برنامه 48: نمایش محتوای یک فایل در مانیتور

BY: MORTAZA ABBASZADEH

فصل اول: زبانهای برنامه سازی و کلیات C++

الگوریتم (Algorithm): نوعی روش حل مسائل در کامپیوتر می باشد که هر 4 ویژگی زیر را دارد:

1. شروع و پایان آن مشخص باشد،
2. ترتیب اجرای مراحل مشخص باشد.
3. به زبان دقیق و بدون ابهام بیان شود.
4. تمام شرایط حل مساله را در بر بگیرد. (هیچ فرضی در مورد دانش اجرا کننده الگوریتم نباید قائل شد)

پس الگوریتم، مجموعه ای متناهی از دستورالعمل ها است، که به ترتیب خاصی اجرا می شوند و مسئله ای را حل می کنند. به عبارت دیگر یک الگوریتم، روشی گام به گام برای حل مسئله است.

خصوصیات یک الگوریتم:

تمام الگوریتم ها (خوارزمی ها) باید شرایط و معیارهای زیر را دارا باشند:

- ورودی: یک الگوریتم باید هیچ یا حد اقل یک پارامتر را به عنوان ورودی بپذیرد؛
- خروجی: الگوریتم بایستی حداقل یک کمیت به عنوان خروجی (نتیجه عملیات) تولید کند؛
- قطعیت: دستورهای الگوریتم باید با زبانی دقیق، و بی ابهام بیان شوند. هر دستورالعمل نیز باید انجام پذیر باشد. دستورهای نظیر «مقدار ۶ یا ۷ را به X اضافه کنید» یا «حاصل تقسیم پنج بر صفر را محاسبه کنید» مجاز نیستند؛ چرا که در مورد مثال اول، معلوم نیست که بالاخره چه عددی باید انتخاب شود، و در خصوص مثال دوم هم تقسیم بر صفر در ریاضیات تعریف نشده است.
- محدودیت: الگوریتم باید دارای شروع و پایان مشخصی باشد، به نحوی که اگر دستورهای آن را دنبال کنیم، برای تمامی حالت ها، الگوریتم پس از طی مراحل، خاتمه یابد. به علاوه، زمان لازم برای خاتمه الگوریتم هم باید به گونه ای معقول و کوتاه باشد.

به طور کلی جهت ارائه ی یک الگوریتم کامل به ۵ مؤلفه ی اصلی احتیاج داریم که عبارتند از:

- مقادیر معلوم
- خواسته ی مسئله
- عملیات محاسباتی

• دستورهای شرطی

• دستورهای تکرار (حلقه‌ها)

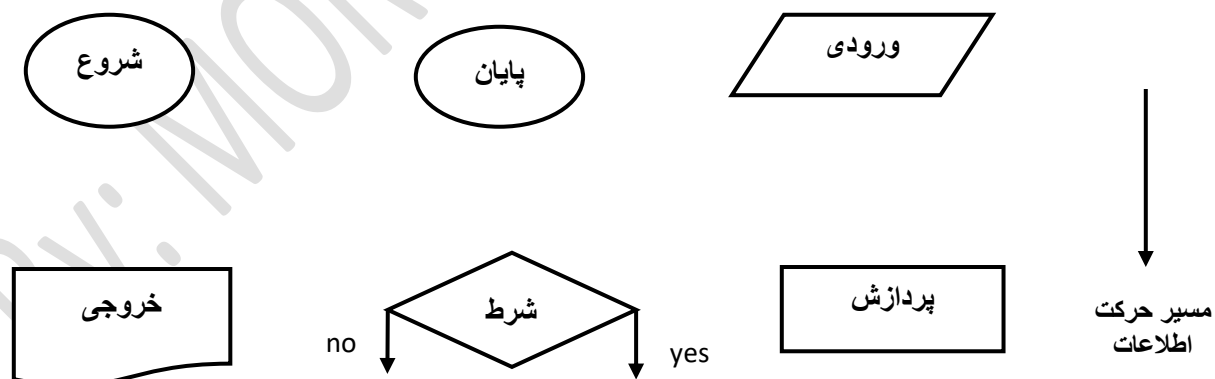
اطلاعات اولیه‌ای که در اختیار ما قرار می‌گیرد و با استفاده از آن‌ها به ارائه‌ی راه حل می‌پردازیم شامل **مقادیر معلوم مسئله** هستند و نتایجی که بر اثر انجام **عملیات محاسباتی** به دست می‌آیند **خواسته‌های مسئله** نامیده می‌شوند.

از آنجایی که هدف اصلی طراحی الگوریتم برای حل یک مسئله، دستیابی به خواسته‌های مسئله می‌باشد بنابراین طی کردن مراحل ۵ گانه‌ی بالا در ارائه‌ی الگوریتم الزامی است.

یک الگوریتم شامل دستورالعمل‌های پشت سر هم است که جهت ارائه‌ی یک خروجی معتبر باید به ترتیب اجرا شوند، از این رو رعایت ترتیب در مولفه‌های اصلی نیز مؤثر است، چرا که اساساً بدون وجود خواسته‌ی مسئله عملیات محاسباتی نیز وجود نخواهد داشت.

با به‌کارگیری دستورهای شرطی می‌توان خروجی و رفتار یک الگوریتم را با توجه به شرایط از پیش تعیین شده‌ی مسئله کنترل کرد، از سوی دیگر استفاده از دستورهای تکرار (حلقه‌ها) به برنامه‌نویس کمک می‌کنند یک دستور تکراری را چندین بار اجرا کند.

فلوچارت یا روندنما (Flowchart): فلوچارت نیز همانند الگوریتم بوده و همان ویژگی‌ها را دارد ولی برای سادگی و درک سریع‌تر انسان، با شکل و تصویر بیان می‌شود. شکل‌های مهم و مورد استفاده در فلوچارت عبارتند از:



برنامه (Program): برنامه نیز در واقع همان الگوریتم و فلوچارت می باشد و همان ویژگیها را دارد، فقط با استفاده از قوانین یکی از زبانهای برنامه نویسی نوشته شده است تا کامپیوتر بتواند آنرا اجرا کند. در یک دسته بندی کلی، زبانهای برنامه نویسی را به 3 سطح تقسیم می کنند که عبارتند از:

- **زبان های برنامه نویسی سطح پایین (Low Level):** زبانهایی هستند که به خود زبان ماشین (یعنی 0 و 1) نزدیکتر باشد. نوشتن کد در این زبانها کمی سخت است ولی در عوض سرعت و دقت اجرای آنها بالاتر است. مانند زبانهای ... Assembly , Machine Code
- **زبان های برنامه نویسی سطح بالا (High Level):** زبانهایی هستند که دستورات آنها شبیه زبان گفتاری و نوشتاری ما انسانها می باشد. نوشتن کد در این زبانها ساده تر می باشد. مانند زبانهای JavaScript , R , Php , Swift , Scala , Go , python , Elm , Ruby , Rust , C++ , Visual Basic .Net , C# , Delphi , Java , ...
- **زبان های برنامه نویسی سطح میانی (Intermediate Level):** زبانهایی هستند که هم برخی از ویژگیهای زبانهای سطح بالا را دارد و هم برخی از قابلیت های زبانهای سطح پایین را دارد. مانند زبان C و ...

خوانایی کد منبع:

در برنامه ریزی رایانه، خوانایی عبارت است از این که خواننده متن کد منبع بتواند هدف، جریان برنامه و عملکرد کد منبع را متوجه شود. این موضوع در مجموعه نیازهای کیفیتی شامل موارد قابلیت انتقال، کاربرد پذیری و نگهداشت پذیری می شود. خوانایی خیلی مهم است به این علت که برنامه نویسان بیشترین وقتشان را به جای این که صرف نوشتن کد جدید بکنند، در زمینه درک و تغییر کد منبع موجود می کنند. کدهای ناخوانا معمولاً به خطا، نارسایی و کدهای تکراری ختم می شوند. یک تحقیق نشان داد که تغییرات جزئی و ساده ای برای خوانایی برنامه می تواند منجر به این شود که کد نهایی نوشته شده کوتاه تر شود و همچنین زمان فهم آن به طور قابل توجهی کاهش یابد.

پیروی از یک سبک برنامه نویسی منسجم معمولاً به خوانایی کمک می کند؛ اگر چه خوانایی چیزی بیش از فقط سبک برنامه نویسی است. عوامل زیادی که معمولاً دخالتی در توانایی رایانه برای ترجمه مناسب و اجرای بهتر برنامه ندارند، به خوانایی کمک می کنند. بعضی از این عوامل عبارتند از:

- بلوک بندی متفاوت (فاصله سفید)
- یادآورها
- تکه تکه کردن نوشته های طولانی
- قواعد نامگذاری برای اشیا (مثل متغیرها، کلاسها، روال ها و غیره)

جنبه‌های دیداری برای این کار (مثل تورفتگی، شکستن خطوط، علامت‌گذاری رنگی و مانند اینها) معمولاً به وسیله ویرایشگر کد منبع فراهم می‌شود، اما جنبه‌های محتوایی بازتابی از استعداد و توانایی‌های برنامه‌نویس است.

زبانهای برنامه‌نویسی دیداری متعددی با این منظور توسعه داده شده‌اند تا بتوانند از روش‌های غیر سنتی برای ارائه کد و تصویر استفاده کنند. محیطهای یکپارچه توسعه نرم‌افزار (IDE) می‌خواهند تا تمام این گونه کمک‌ها را به شکل یکپارچه ارائه کنند. تکنیکهایی برای بازسازی کد می‌تواند خوانایی را افزایش دهد.

پارادایم‌ها (الگو): زبان‌های برنامه‌نویسی گوناگون براساس قابلیت‌های در نظر گرفته شده از شیوه‌ی خط‌های مختلف استفاده می‌کنند. موارد ریزتری مانند چگونگی برخورد با نیازهای پشت پرده‌ی ماشین مانند مدیریت حافظه و مدیریت زباله نیز در زبان‌های مختلف متفاوت است. علاوه بر این‌ها، مفاهیمی متفاوت از (اجرای) یک برنامه تصور شده‌اند که پارادایم یا الگو نام دارند.

زبان‌های برنامه‌نویسی: زبان‌های مختلف برنامه‌نویسی وجود دارند که هر کدام از آن‌ها سبک‌های خاصی را پشتیبانی می‌کنند. (به نام پارادایم‌های برنامه‌نویسی). انتخاب زبان مورد استفاده مورد توجه بسیاری از مباحث از قبیل خط مشی شرکت، مناسب بودن در انجام کار، در دسترس بودن بسته‌های شخص ثالث یا ترجیح شخصی است. زبانها طیف تقریبی را از «سطح پایین» تا «سطح بالا» تشکیل می‌دهند. زبانهای «سطح پایین» به‌طور معمول بیشتر دستگاه محور و سریعتر اجرا می‌شوند، و به زبان ماشین نزدیکتر هستند در حالی که زبانهای «سطح بالا» انتزاعی تر و آسان تر برای استفاده هستند اما سرعت کمتری دارند. معمولاً کدگذاری به زبانهای «سطح بالا» از زبان‌های «سطح پایین» ساده‌تر است.

آلن داونی در کتاب "چگونه به شکل یک استاد رایانه فکر کنیم" می‌نویسد:

جزئیات در زبان‌های برنامه‌نویسی مختلف متفاوت به نظر می‌رسند ولی تعدادی از ساختارهای اساسی در همه زبان‌های برنامه‌نویسی یکسان هستند:

- **ورودی:** داده‌ها را از صفحه کلید، یک فایل یا وسایل دیگر فراهم می‌کند.
- **خروجی:** اطلاعات را روی صفحه تصویر نشان می‌دهد، به یک فایل می‌فرستد یا به دستگاه‌های دیگری انتقال می‌دهد.
- **محاسبات:** اعمال محاسباتی اساسی مثل جمع و ضرب را انجام می‌دهد.
- **حالت‌های شرطی:** شرط‌های مشخصی را کنترل می‌کند و بر اساس آن رشته مناسبی از عبارات را اجرا می‌کند.
- **حلقه:** بعضی اعمال را به شکل تکراری انجام می‌دهد، معمولاً با استفاده از تعدادی از متغیرها این کار انجام می‌شود.

زبان برنامه نویسی C++:

- ✓ C++ یکی از محبوب ترین زبان های برنامه نویسی در دنیا است که برای ایجاد برنامه های کامپیوتری استفاده می شود.
- ✓ C++ یک زبان cross-platform است که می تواند برای ایجاد برنامه های با کارایی بالا مورد استفاده قرار گیرد.
- ✓ C++ توسط Bjarne Stroustrup از زبان C توسعه یافته است.
- ✓ C++ کنترل بالایی بر منابع سیستم و حافظه به برنامه نویسان می دهد.
- ✓ این زبان 3 بار و در سال های 2011، 2014 و 2017 به C++ 11، C++ 14 و C++ 17 به روز شده است.
- ✓ C++ را می توان در سیستم عامل های امروزی، رابط های گرافیکی کاربر و سیستم های جاسازی شده مشاهده کرد.
- ✓ C++ یک زبان برنامه نویسی شی گرا است که ساختار روشنی به برنامه ها می دهد و امکان استفاده مجدد از کد را فراهم می کند و باعث کاهش هزینه های توسعه می شود.
- ✓ C++ قابل حمل است و می تواند برای توسعه برنامه هایی که با چندین سیستم عامل سازگار هستند، مورد استفاده قرار گیرد.
- ✓ C++ سرگرم کننده و آسان برای یادگیری است!
- ✓ از آنجا که C++ به C# و Java نزدیک است، سوچ برنامه نویسی به C++ یا بالعکس را برای برنامه نویسان آسان می کند.

شروع به کار با C++:

- ✓ برای شروع استفاده از C++، به دو مورد نیاز دارید:
 - یک ویرایشگر متن، مانند Notepad، یا محیط خود C++، برای نوشتن کد C++
 - یک کامپایلر مانند GCC برای ترجمه کد C++ به زبانی که کامپیوتر درک می کند (زبان ماشین)
- ✓ ویرایشگرها و کامپایلرهای متن زیادی برای انتخاب وجود دارد. در این کلاس، ما از IDE استفاده خواهیم کرد.

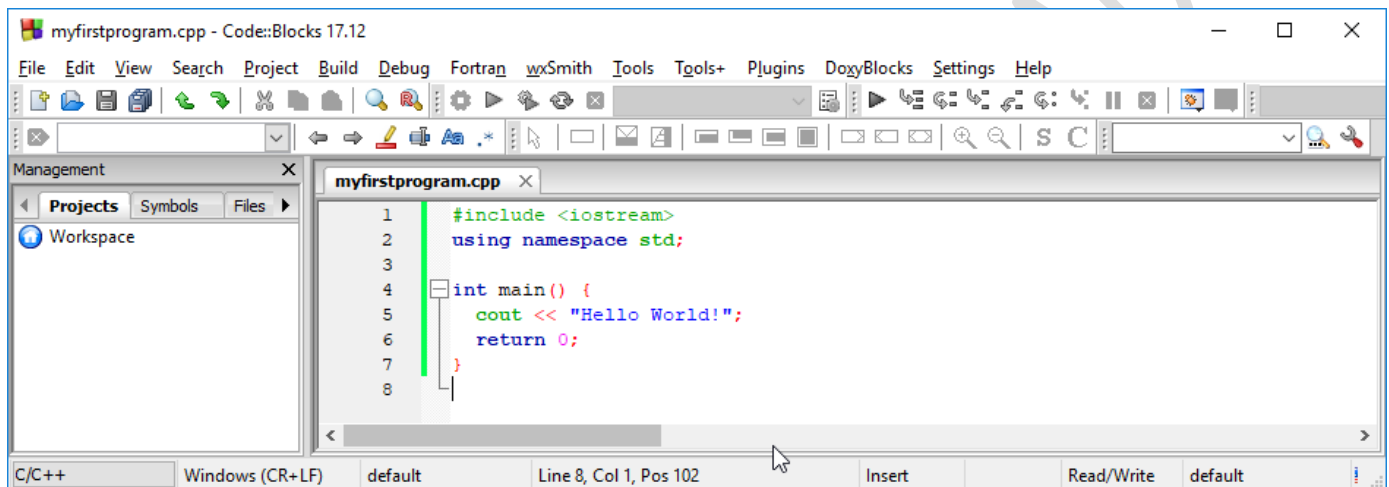
نصب IDE C++:

- ✓ برای نوشتن، ویرایش و کامپایل کد از IDE (Integrated Development Environment) (محیط توسعه یکپارچه) استفاده می شود.
- ✓ IDE های محبوب شامل Code :: Blocks، Eclipse و Visual Studio هستند. اینها همه رایگان هستند و می توان از آنها برای ویرایش، اجرا و اشکال زدایی کد C++ استفاده کرد.
- توجه: IDE های مبتنی بر وب نیز بصورت آنلاین وجود دارند که می توان استفاده کرد، اما عملکرد آنها محدود است.
- ✓ ما در این کلاس از Code :: Blocks استفاده خواهیم کرد، که محیط خوبی برای شروع C++ است.

✓ می توانید آخرین نسخه Codeblocks را در <http://www.codeblocks.org/downloads/26> پیدا کنید.
فایل mingw-setup.exe را بارگیری کنید که ویرایشگر متن را با یک کامپایلر نصب می کند.

نوشتن برنامه در C++:

- ✓ بیاید اولین برنامه C++ خود را ایجاد کنیم.
- ✓ Codeblocks را باز کرده و به File> New> Empty File بروید.
- ✓ کد C++ زیر را بنویسید و فایل را به صورت myfirstprogram.cpp ذخیره کنید (File> Save File as):



شکل 1-1: محیط برنامه نویسی Code::Blocks

✓ سپس، برای اجرای برنامه، به Build> Build and Run بروید. نتیجه (خروجی) چیزی مشابه این خواهد بود:

```
Hello World!
Process returned 0 (0x0) execution time : 0.011 s
Press any key to continue.
```

نکته: چنانچه بخواهیم در محیط Microsoft Visual Studio کدها را اجرا کنیم باید این سطر به بالای کدها اضافه شود:

```
#include "stdafx.h"
```

نکته: وبسایت های زیادی وجود دارند که می توان بصورت آنلاین و حتی با تلفن همراه، کدها را آنجا وارد کرده و نتیجه ی اجرا را مشاهده کرد. در اینصورت، نیازی به نصب هیچ نرم افزاری نیست. مانند:

https://www.onlinegdb.com/online_c++_compiler

قالب کلی (چارچوب، فرمت، اسکلت) برنامه ها در C++:

فایل های سرآیند (با دستور include یا using به ابتدای برنامه الحاق میشوند)
int main()
{
تعریف داده‌های مورد نیاز برنامه (متغیرها، آرایه ها، رشته ها و ...)
کدهای اصلی برنامه (فراخوانی زیربرنامه‌ها در صورت لزوم)
return(0);
}
زیربرنامه‌ها (توابع و تکه کدهای فرعی)

قوانین کلی در C++:

- منظور از فایل‌های سرآیند، برنامه‌های استاندارد و از قبل نوشته شده هستند مانند `iostream`, `cmath`, ...
- انتهای هر دستور نقطه ویرگول `;` گذاشته میشود (semi colon).
- دستور `return(0)` به سیستم عامل اعلام می‌کند که برنامه بصورت عادی، در حال اتمام است.
- حروف بزرگ با حروف کوچک تفاوت دارند. (مثلاً `a` برابر `A` نیست)
- دستورات با حروف کوچک نوشته میشوند. (مانند `if`, `else`, `for`, `while`, `int`, ...)
- برای نوشتن توضیحات (کامنت `comment`) میتوان از `//` قبل از توضیحات، و یا `/*.....*/` استفاده کرد.
- برنامه های زبان C++ در فایل متنی و با پسوند `.cpp` ذخیره میشوند.
- بهتر است هر دستور در یک سطر نوشته شود. ولی میتوان در یک سطر چندین دستور نیز نوشت.
- بهتر است برنامه بصورت خوانا، بلوکی و دندانه دار نوشته شود تا درک و فهم و نیز اشکال‌زدایی آن راحتتر باشد.

متغیر (Variable): یک متغیر در واقع مکانی از حافظه اصلی (Ram) می‌باشد که یک برنامه نویس آنها تعریف میکند تا اعداد و مقادیر مورد نیاز برنامه را برای محاسبه و پردازش داخل آنها قرار دهد. انتخاب نام متغیرها دلخواه است، فقط نباید تکراری و یا از اسامی رزرو شده‌ی خود زبان (مثل `if`, ...) باشد.

ثابت: نوع خاصی از متغیر است که مقدار آن همیشه ثابت بوده و در طول اجرای برنامه، مقدار آن عمدی یا غیرعمدی قابل تغییر نیست. با دستور `const` یا دستور `define` میتوان ثابت تعریف کرد. (مثلاً برای عدد پی: `pi=3.14159265`)

انواع داده ها (Data Type) در C++:

دستور	نوع داده	مصرف حافظه	توضیح و مثال
char	کاراکتری	1 بایت	char A; char ch='a';
int	عدد صحیح	4 بایت (بعضاً 2 بایت)	int B,C,D; int Max=1000;
float	عدد اعشاری	4 بایت	float M=17.25;
double	اعشاری مضاعف	8 بایت	double K=9999999.87654321;
bool	بولی	1 بایت	bool T; فقط میتواند true یا false باشد.
void	تهی	-	در مورد توابعی که مقداری برگشت نمیدهند کاربرد دارد.
string	رشته	به تعداد کاراکترها	string msg1="Hello, welcome";

نکته: عدد اعشاری تا 7 رقم دقت اعشار، و عدد اعشاری مضاعف تا 15 رقم دقت اعشار می تواند داشته باشد.

نکته: در جدول بالا، انواع داده های اصلی که مبنای کار در C++ هستند، بیان شده است. در C++ شیءگرا، برنامه نویس میتواند با استفاده از کلاس ها (class) هر نوع داده ای دلخواه جدیدی را برحسب نیاز تعریف و از آن استفاده کند.

نکته: برای استفاده از نوع داده ای string باید سرآیند آنرا به برنامه خود اضافه کنیم: `#include <string>`

دستورات پیشوندی برای انواع داده ها:

- **long:** محدوده ای عدد را طولانی و بزرگتر می کند. (معمولاً دو برابر)
- **short:** محدوده ای عدد را کوتاه و کوچکتر می کند. (معمولاً نصف)
- **signed:** عدد را با علامت در نظر می گیرد. سمت چپ ترین بیت بعنوان بیت علامت. (پیش فرض است)
- **unsigned:** عدد را بدون علامت (نامنفی) در نظر می گیرد. تمام بیت ها داده هستند. (بیت علامت ندارد)

مثال:

- عدد صحیح طولانی: `long int A=5000000;`
- عدد صحیح کوتاه: `short int B=5;`
- عدد صحیح نامنفی: `unsigned int C=1399;`
- عدد صحیح طولانی نامنفی: `unsigned long int K;`

انواع عملگرها در C++:

+	جمع
-	تفریق
*	ضرب
/	تقسیم (صحیح/صحیح=تقسیم صحیح، حداقل یکی از صورت یا مخرج اعشاری باشد=تقسیم اعشاری)
%	باقیمانده (مخصوص اعداد صحیح); $k=9\%$ آنگاه $k=4$
=	انتساب، قرار دادن، کپی کردن (راست - چپ) مانند $A=20$ (ولی این اشتباهه $20=A$)
==	تساوی (برابر بودن یا نبودن) مثال: $if(A==B)...$ معادل نیست با: $if(A=B)...$
>	بزرگتر
<	کوچکتر
>=	بزرگتر یا مساوی بودن
<=	کوچکتر یا مساوی بودن
!=	نامساوی
++	افزایش یک واحدی. مثال $A++$; یا $++A$; همان $A=A+1$
--	کاهش یک واحدی. مثال $B--$; همان $B=B-1$
+=	عملگر ترکیبی جمع سپس انتساب. مثال $A+=1$; معادل $A=A+1$; یا معادل $++A$
-=	عملگر ترکیبی تفریق سپس انتساب. مثال $A-=5$; معادل $A=A-5$
=	عملگر ترکیبی ضرب سپس انتساب. مثال $A=2$; معادل $A=A*2$
/=	عملگر ترکیبی تقسیم سپس انتساب. مثال $A/=3$; معادل $A=A/3$
%=	عملگر ترکیبی باقیمانده سپس انتساب. مثال $A\%=2$; معادل $A=A\%2$
&&	And منطقی (و) مثال برای تشخیص عدد دو رقمی $if(n>=10 \&\& n<=99)...$
	Or منطقی (یا) مثال برای تشخیص نمره نامعتبر $if(n<0 n>20) ...$
!	Not نقیض مثال $if !(A>B)...$ یعنی اگر A از B بزرگتر نباشد...
...	و سایر عملگرها

برنامه 1: الگوریتم، فلوچارت و برنامه ای بنویسید که پیغام "Hello, welcome" را نشان دهد.

حل: تعداد ورودی: 0 تعداد خروجی: 1 تعداد متغیرهای لازم: 0

<p>برنامه:</p> <pre>#include <iostream> using namespace std; int main() { cout<< "Hello, welcome"; return(0); }</pre> <p>تمرین 1: این برنامه را در کامپیوتر (داخل Code::Blocks و یا Visual Studio) اجرا کرده و نتیجه را ببینید.</p>	<p>الگوریتم:</p> <p>مرحله 1: شروع مرحله 2: پیغام "Hello, welcome" را نشان بده مرحله 3: پایان</p> <hr/> <p>فلوچارت:</p> <pre>graph TD START([START]) --> Process[\"Hello, welcome\"] Process --> END([END])</pre>
---	--

دستور (...<<cout): این دستور یکی از اصلی ترین و پرکاربردترین دستورات زبان C++ برای نشان دادن پیغام و نشان دادن مقادیر متغیرها در خروجی است.

دستور (...>>cin): این دستور یکی از اصلی ترین و پرکاربردترین دستورات زبان C++ برای گرفتن ورودی است.

نکته: سرآیند `iostream` باعث فعال شدن و کارکردن دستورات معمول زبان C++ مانند `cout` و ... می شود.

نکته: استفاده از فضای نام استاندارد `std` قبل از `main` باعث می شود تا دستوراتی مثل `cout` بدون هیچ پیشنیازی کار کنند. اگر در ابتدای برنامه آنرا ننویسیم باید برای هر دستور آنرا بصورت `std::cout<<...` بنویسیم.

برنامه 2: الگوریتم، فلوچارت و برنامه‌ای بنویسید که دو عدد صحیح را از ورودی گرفته و مجموع آنها را در خروجی نشان دهد.

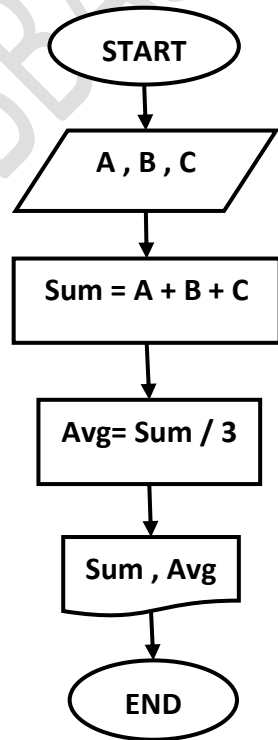
حل: تعداد ورودی: 2 تعداد خروجی: 1 تعداد متغیرهای لازم: 3

<p>برنامه:</p> <pre>#include <iostream> using namespace std; int main() { int A , B , Sum; cin>>A; cin>>B; Sum=A+B; cout<<Sum; return(0); }</pre>	<p>الگوریتم:</p> <p>مرحله 1: شروع مرحله 2: دو عدد صحیح را از ورودی بگیر و داخل A,B قرار بده مرحله 3: مقادیر A,B را جمع زده نتیجه را در Sum قرار بده مرحله 4: مقدار داخل Sum را نشان بده. مرحله 5: پایان</p> <p>فلوچارت:</p> <pre>graph TD Start([START]) --> Input[/A , B/] Input --> Process[Sum = A + B] Process --> Output[Sum] Output --> End([END])</pre>
--	--

تمرین 2: این برنامه را در کامپیوتر (داخل Code::Blocks و یا Visual Studio) اجرا کرده و خروجی بگیرید.

برنامه 3: الگوریتم، فلوچارت و برنامه‌ای بنویسید که سه عدد صحیح را از ورودی گرفته، مجموع و میانگین آنها را در خروجی نشان دهد.

حل: تعداد ورودی: 3 تعداد خروجی: 2 تعداد متغیرهای لازم: 5

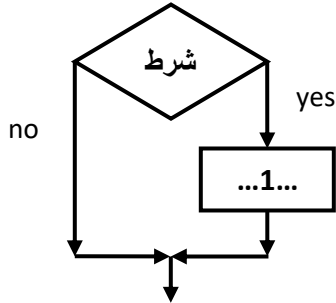
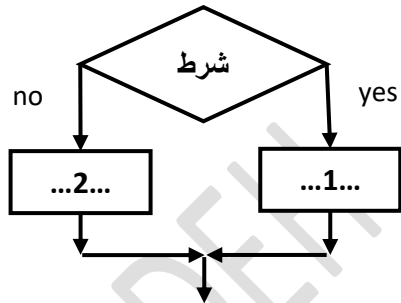
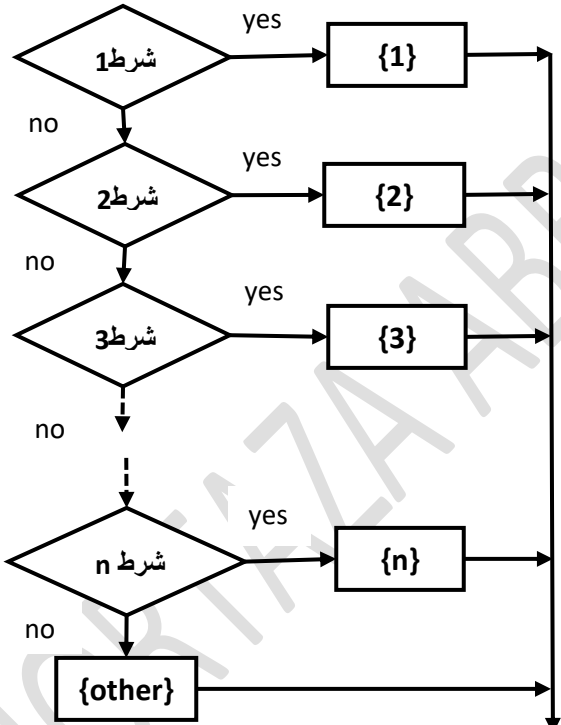
<p>برنامه:</p> <pre>#include <iostream> using namespace std; int main() { int A, B, C, Sum; float Avg; cin>>A>>B>>C; Sum=A+B+C; Avg=Sum/3.0; cout<<"Majmoo: "<<Sum; cout<<endl; cout<<"Meianguin: "<<Avg; return(0); }</pre> <p>تمرین 3: این برنامه را در کامپیوتر (داخل Code::Blocks و یا Visual Studio) اجرا کرده و خروجی بگیرید.</p>	<p>الگوریتم:</p> <p>مرحله 1: شروع مرحله 2: سه عدد صحیح را از ورودی بگیر و داخل A,B,C قرار بده مرحله 3: مقادیر A,B,C را جمع زده نتیجه را در Sum قرار بده مرحله 4: مقدار Sum را بر 3 تقسیم اعشاری کن و نتیجه را در Avg قرار بده مرحله 5: مقدار داخل Sum , Avg را نشان بده. مرحله 6: پایان</p> <p>فلوچارت:</p>  <pre>graph TD Start([START]) --> Input[/A, B, C/] Input --> Sum[Sum = A + B + C] Sum --> Avg[Avg = Sum / 3] Avg --> Output[/Sum, Avg/] Output --> End([END])</pre>
---	--

نکته: میانگین ممکن است عدد اعشاری باشد لذا حداقل یکی از صورت یا مخرج باید اعشاری باشد تا تقسیم اعشاری عمل کند. و یا اینکه میتوان تبدیل نوع انجام داد. یعنی بنویسیم:

`Avg=(float)Sum/3;`

نکته: دستور `endl` یا کاراکتر کنترل `"\n"` باعث رفتن به ابتدای سطر بعدی می‌شود.

فصل دوم: ساختارهای شرطی

<p>if(شرط) {...1...}</p>  <p>اول</p>	<p>if(شرط) {...1...} else {...2...}</p>  <p>دوم</p>	
<p>if (شرط 1) {...1...} else if(شرط 2) {...2...} else if(شرط 3) {...3...} . . . else if(شرط n) {...n...} else {...other...}</p>	 <p>سوم</p>	<p>switch(عبارت) { case حالت اول {1} break; case حالت دوم {2} break; case حالت سوم {3} break; . . . case حالت n {n} break; default: {other} }</p>

نکته: ساختار دوم را میتوان در حالت ساده (عملگر سه گانه) چنین نیز نوشت: $Result = (Condition) ? ...1... : ...2...;$

نکته: همانگونه که در ساختار سوم شرطی دیده می شود از بین چندین حالت فقط یک حالت می تواند اتفاق بیافتد. در این حالت نباید از if های متوالی استفاده کرد، بلکه باید از ساختار else if استفاده کرد. البته این ساختار را میتوان با دستور switch نیز نوشت. دستور switch برای مقادیر اعشاری کاربرد ندارد.

نکته: دستور break باعث خارج شدن از ساختار switch، حلقه و یا بلوک جاری می شود. دستور continue باعث می شود کنترل اجرای برنامه، به ابتدای حلقه برگردد.

برنامه 4: فلوچارت و برنامه‌ای بنویسید که عدد صحیحی مانند A را از ورودی بگیرد، و در خروجی نشان دهد که این عدد زوج است یا فرد. برای دو عدد مانند 18 , 5 ردیابی کنید.

حل: به عددی زوج گفته می‌شود که اگر بر 2 تقسیم کنیم باقیمانده‌ی آن برابر صفر باشد. (Mod)

تعداد ورودی: 1 تعداد خروجی: 1 تعداد حالت ها: 2 تعداد شرط های لازم: 1

برنامه:	فلوچارت:												
<pre>#include <iostream> using namespace std; int main() { int A,B; cin>>A; B = A % 2; if(B==0) cout<<"Zoj Ast"; else cout<<"Fard Ast"; return(0); }</pre>	<pre> graph TD Start([START]) --> A[/A/] A --> B[B = A mod 2] B --> C{B = 0} C -- yes --> D[زوج است] C -- no --> E[فرد است] D --> End([END]) E --> End </pre>												
<p style="text-align: center;">ردیابی:</p> <table border="1" style="width: 100%; border-collapse: collapse; margin-bottom: 10px;"> <thead> <tr> <th>A</th> <th>B</th> <th>خروجی</th> </tr> </thead> <tbody> <tr> <td>18</td> <td>0</td> <td>زوج است</td> </tr> </tbody> </table> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>A</th> <th>B</th> <th>خروجی</th> </tr> </thead> <tbody> <tr> <td>5</td> <td>1</td> <td>فرد است</td> </tr> </tbody> </table>	A	B	خروجی	18	0	زوج است	A	B	خروجی	5	1	فرد است	<p style="text-align: center;">تمرین 4: این برنامه را در کامپیوتر اجرا کرده و خروجی بگیرید.</p>
A	B	خروجی											
18	0	زوج است											
A	B	خروجی											
5	1	فرد است											

نکته: اگر داخل بلوک {...} فقط یک دستور باشد، میتوان خود { } را ننوشت.

ردیابی (Trace): منظور از ردیابی این است که از روی فلوچارت یا برنامه، از اولین مرحله تا آخرین مرحله پیش برویم. این کار معمولاً با استفاده از یک جدول انجام می‌شود که باعث درک بهتر شده و عیب یابی آن را نیز راحتتر می‌کند.

برنامه 5: فلوچارت و برنامه‌ای بنویسید که نمره‌ی دانشجو را از ورودی بگیرد، و در خروجی نشان دهد که آیا قبول است یا مردود؟ برای دو نمره مانند 16.25 , 5 ردیابی کنید.

حل: نمره زیر 10 مردود است. کلاً دو حالت وجود دارد پس یک شرط کافی است.

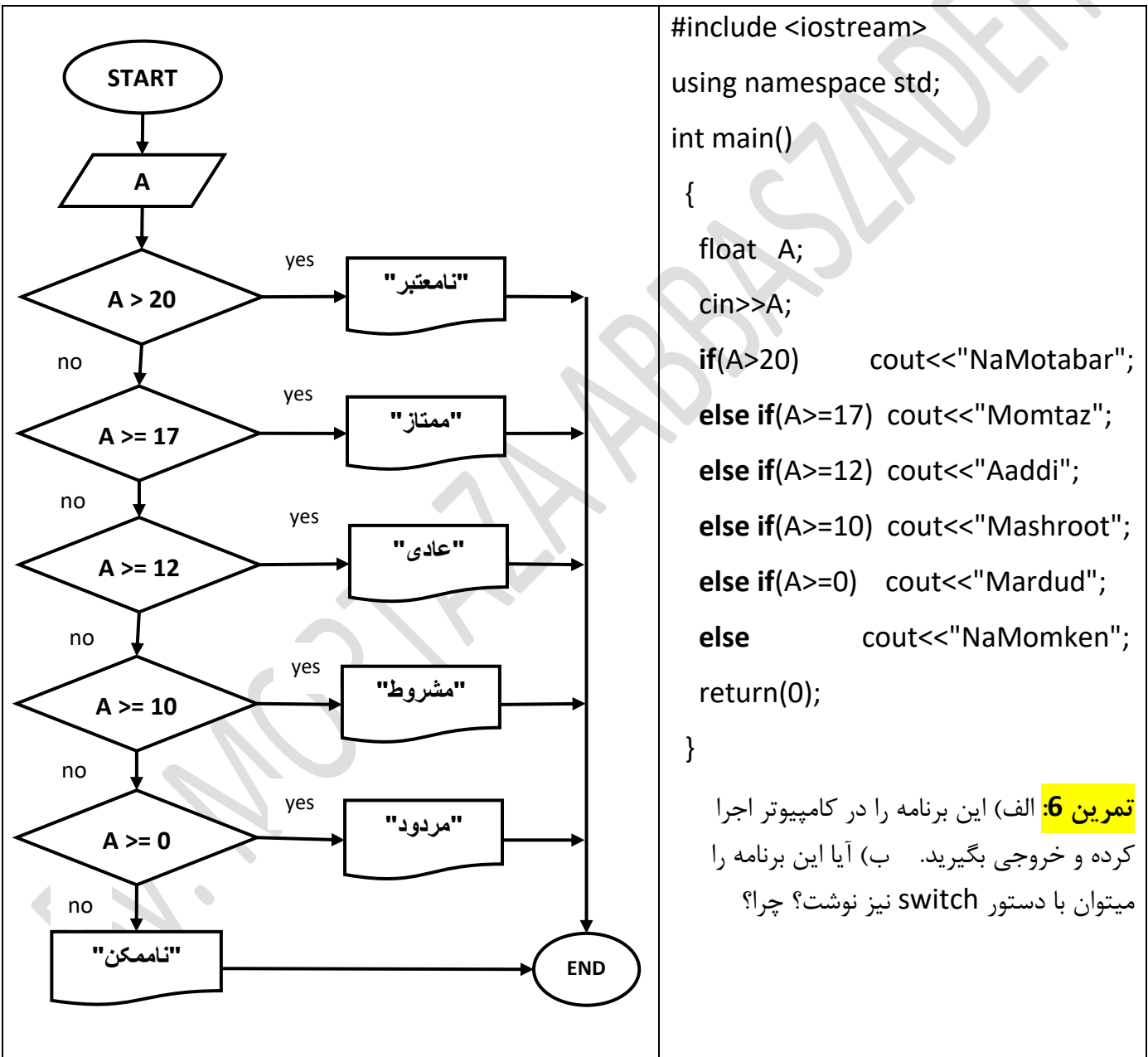
تعداد ورودی: 1 تعداد خروجی: 1

برنامه:	فلوچارت:								
<pre>#include <iostream> using namespace std; int main() { float A; cin>>A; if(A>=10) cout<<"Gabool Ast"; else cout<<"Mardood Ast"; return(0); }</pre>	<pre> graph TD Start([START]) --> Input[/A/] Input --> Decision{A >= 10} Decision -- yes --> YesBox[قبول است] Decision -- no --> NoBox[مردود است] YesBox --> End([END]) NoBox --> End </pre>								
<p style="text-align: center;">ردیابی:</p> <table border="1" style="width: 100%; border-collapse: collapse; margin-bottom: 10px;"> <thead> <tr> <th style="width: 50%;">A</th> <th style="width: 50%;">خروجی</th> </tr> </thead> <tbody> <tr> <td>16.25</td> <td>قبول است</td> </tr> </tbody> </table> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 50%;">A</th> <th style="width: 50%;">خروجی</th> </tr> </thead> <tbody> <tr> <td>5</td> <td>مردود است</td> </tr> </tbody> </table>	A	خروجی	16.25	قبول است	A	خروجی	5	مردود است	<p>تمرین 5: این برنامه را در کامپیوتر اجرا کرده و خروجی بگیرید.</p>
A	خروجی								
16.25	قبول است								
A	خروجی								
5	مردود است								

نکته: وقتی در یک مساله، N حالت وجود دارد که از بین آنها قرار است فقط یک حالت اتفاق بیافتد، آنگاه نوشتن یا رسم کردن (N-1) شرط (لوزی) کافی است.

برنامه 6: فلوچارت و برنامه‌ای بنویسید که معدل دانشجو را از ورودی بگیرد، اگر معدلش بیشتر از 20 بود پیغام "نامعتبر"، اگر بین 17 و 20 بود پیغام "ممتاز"، اگر بین 12 و 17 بود پیغام "عادی"، اگر بین 10 و 12 بود پیغام "مشروط"، اگر بین 0 و 10 بود پیغام "مردود" و اگر زیر 0 بود پیغام "ناممکن" را نشان دهد.

حل: کلاً 6 حالت است، که فقط یک حالت اتفاق می‌افتد. پس 5 شرط کافی است. (تعداد ورودی: 1، تعداد خروجی: 1)



ردیابی:

A	خروجی
25	نامعتبر

A	خروجی
19.25	ممتاز

A	خروجی
15.5	عادی

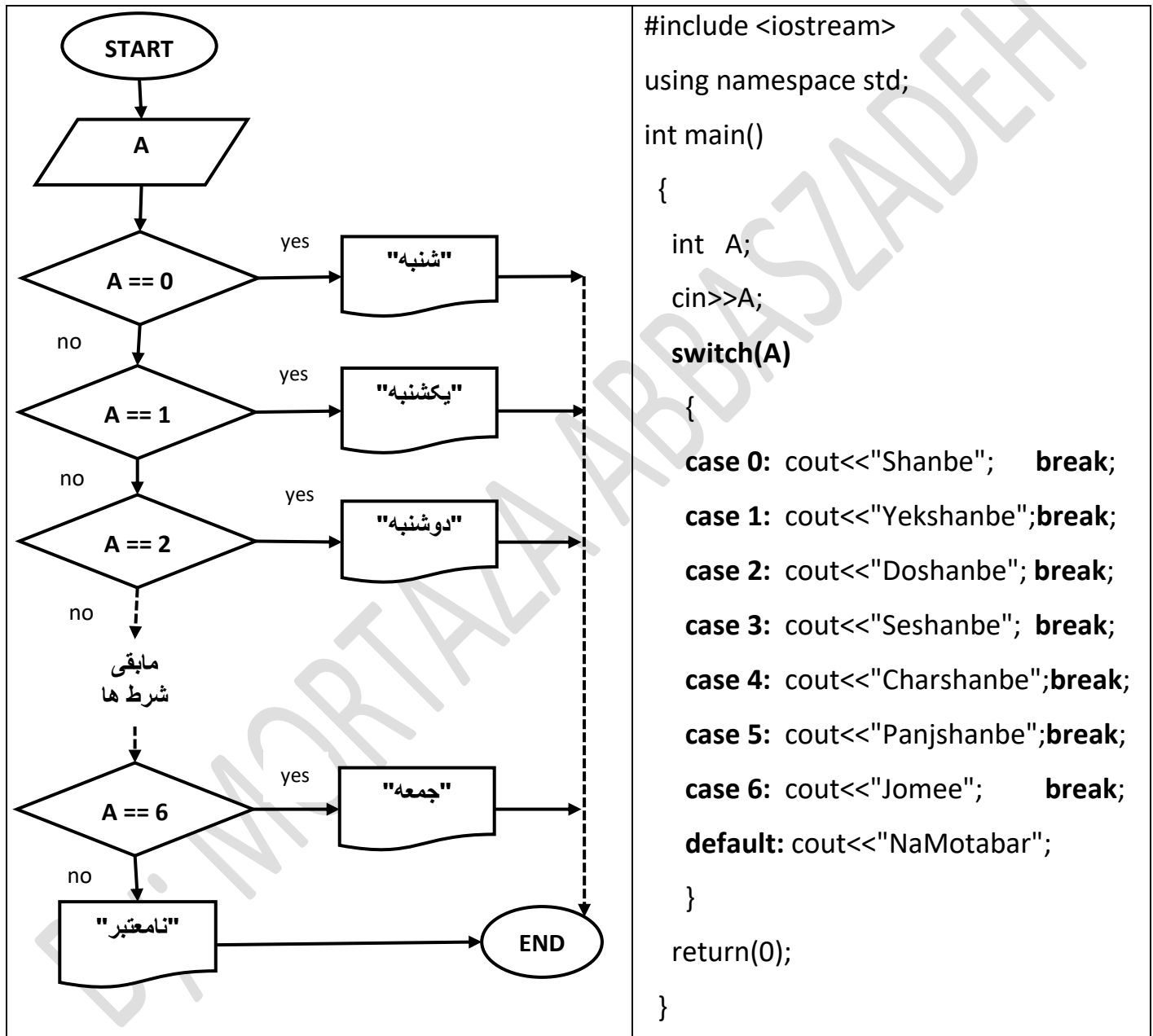
A	خروجی
11	مشروط

A	خروجی
7.75	مردود

A	خروجی
-5	ناممکن

برنامه 7: فلوچارت و برنامه‌ای بنویسید که عدد معادل روزهای هفته را از ورودی بگیرد، و معادل روز آن عدد را نشان دهد. (0 برای شنبه، 1 برای یکشنبه و ...). این برنامه را با دستور **switch** بنویسید.

حل: کلاً 8 حالت است که فقط یک حالت اتفاق می‌افتد. پس 7 شرط کافی است. (تعداد ورودی: 1، تعداد خروجی: 1)



تمرین 7: الف) این برنامه را در کامپیوتر اجرا کرده و خروجی بگیرید. ب) آیا این برنامه را میتوان با **else if** نیز نوشت؟ اگر جواب مثبت است آنرا نوشته و اجرا کنید.

فصل سوم: ساختارهای تکرار (حلقه)

<pre> graph TD Start(()) --> Init[مقداردهی اندیس حلقه] Init --> Cond{شرط حلقه} Cond -- no --> Exit[خارج] Cond -- yes --> Body[بدنه حلقه] Body --> Step[گام حرکت] Step --> Cond </pre>	<p>مقداردهی اندیس حلقه</p> <p>while (شرط حلقه)</p> <pre> { دستورات داخل حلقه (بدنه حلقه) گام حرکت اندیس حلقه } </pre> <p>یا</p> <p>for (گام حرکت ; شرط حلقه ; اندیس حلقه)</p> <pre> { دستورات داخل حلقه (بدنه حلقه) } </pre>
<pre> graph TD Start(()) --> Init[مقداردهی اندیس حلقه] Init --> Body[بدنه حلقه] Body --> Step[گام حرکت] Step --> Cond{شرط حلقه} Cond -- no --> Exit[خارج] Cond -- yes --> Body </pre>	<p>مقداردهی اندیس حلقه</p> <p>do</p> <pre> { دستورات داخل حلقه (بدنه حلقه) گام حرکت اندیس حلقه } while (شرط حلقه); </pre> <p>یا</p> <p>مقداردهی اندیس حلقه</p> <p>برچسب:</p> <pre> دستورات داخل حلقه (بدنه حلقه) گام حرکت اندیس حلقه goto (شرط حلقه) </pre>

نکته: در ساختار اول ابتدا شرط حلقه بررسی می شود سپس وارد بدنه حلقه می شود. ولی در ساختار دوم ابتدا بدنه حلقه اجرا می شود و چنانچه شرط حلقه هنوز برقرار بود، به ابتدای حلقه برمی گردد. در ساختار اول ممکن است بدنه حلقه اصلاً اجرا نشود ولی در ساختار دوم، بدنه حلقه، حداقل یکبار اجرا خواهد شد.

برنامه 8: فلوچارت و برنامه‌ای بنویسید که مجموع اعداد 1 تا 10 را محاسبه کرده و نشان دهد.

حل: یک شمارنده (Counter) مانند i لازم داریم تا آنرا با استفاده از حلقه‌ی تکرار، 10 بار جمع کنیم.

تعداد ورودی: صفر تعداد خروجی: 1 تعداد شمارنده: 1

```
#include <iostream>
using namespace std;
int main()
```

```
{
    int i , Sum=0;
    i=1;
    while ( i <= 10)
    {
        Sum = Sum + i ;
        i++;
    }
```

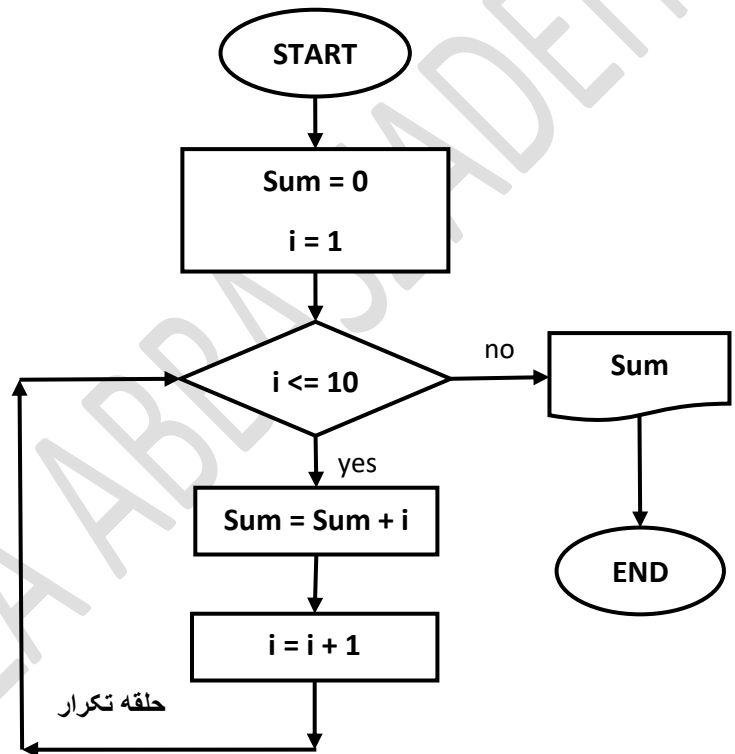
```
    cout<<"Majmoo 1 ta 10: "<<Sum;
    return(0);
}
```

//.....OR.....//

```
#include <iostream>
using namespace std;
int main()
```

```
{
    int i , Sum=0;
    for (i=1 ; i<=10 ; i++)
        Sum+=i;
```

```
    cout<<"Majmoo 1 ta 10: "<<Sum;
    return(0);
}
```



تمرین 8: این برنامه را در کامپیوتر اجرا کرده و خروجی بگیرید.

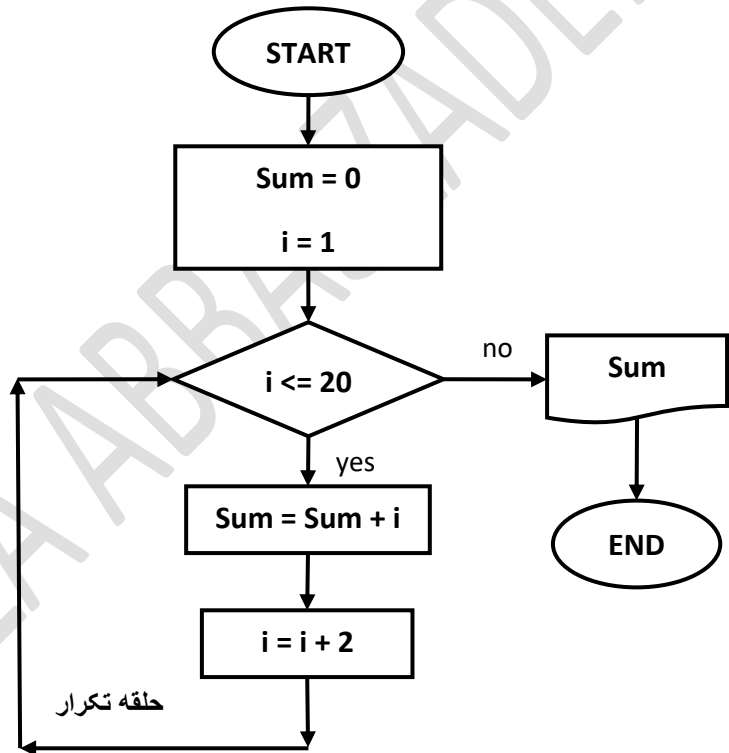
برنامه 9: فلوچارت و برنامه‌ای بنویسید که مجموع اعداد فرد 1 تا 20 را محاسبه کرده و نشان دهد.

حل: شمارنده‌ی i را از 1 شروع کرده و هر بار 2 واحد به آن اضافه می‌کنیم.

تعداد ورودی: صفر تعداد خروجی: 1 شمارنده: 1 (گام حرکت: 2)

```
#include <iostream>
using namespace std;
int main()
{
    int i , Sum=0;
    i=1;
    while (i<=20)
    {
        Sum+=i;
        i+=2;
    }
    cout<<"Majmoo Fard 1 ta 20: "<<Sum;
    return(0);
}

//.....OR.....//
#include <iostream>
using namespace std;
int main()
{
    int i , Sum=0;
    for (i=1 ; i<=20 ; i+=2)
        Sum+=i;
    cout<<"Majmoo Frad 1 ta 20: "<<Sum;
    return(0);
}
```



تمرین 9: این برنامه را در کامپیوتر اجرا کرده و خروجی بگیرید.

برنامه 10: فلوجارت و برنامه‌ای بنویسید که عددی مانند N را گرفته، مجموع اعداد 1 تا N را محاسبه کرده و نشان دهد. در پایان برای عددی مانند $N=5$ را ردیابی کنید.

حل: شمارنده i را از 1 شروع کرده و تا زمانی که از N کوچکتر است، آنرا با Sum جمع می‌زنیم.

تعداد ورودی: 1 تعداد خروجی: 1 شمارنده: 1 (گام حرکت: 1)

```
#include <iostream>
using namespace std;
int main()
{
    int i , N , Sum=0;
    cin>>N;
```

```
    i=1;
    while (i<=N)
    {
        Sum+=i;
        i++;
    }
```

```
    cout<<"Majmoo 1 ta N: "<<Sum;
    return(0);
}
```

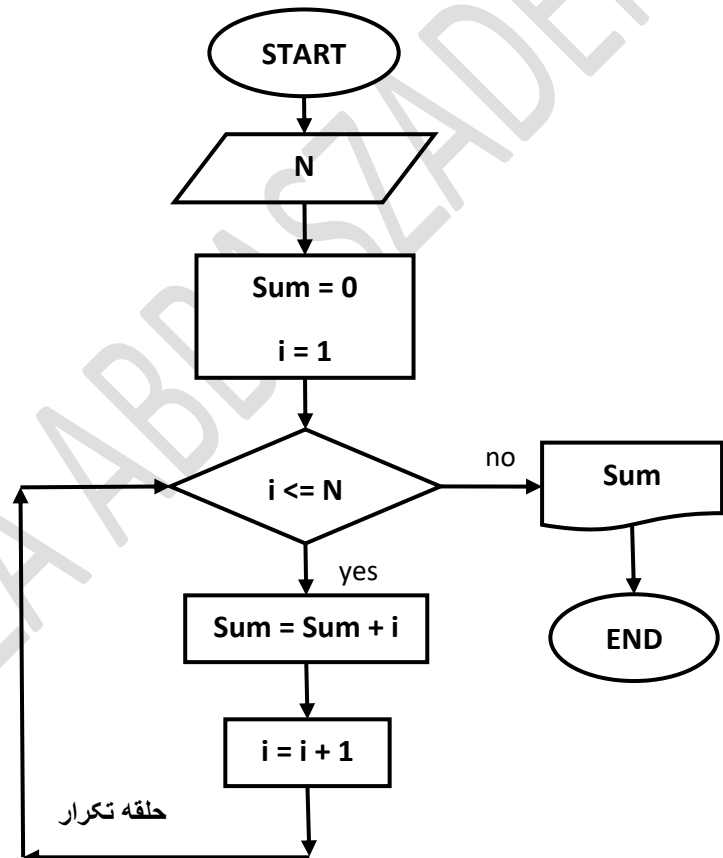
//.....OR.....//

```
#include <iostream>
using namespace std;
int main()
{
```

```
    int i , N , Sum=0;
    cin>>N;
```

```
    for (i=1 ; i<=N ; i++)
        Sum+=i;
```

```
    cout<<"Majmoo 1 ta N: "<<Sum;
    return(0);
}
```



N	i	Sum
5	1	0
	2	1
	3	3
	4	6
	5	10
ردیابی	6	15

تمرین 10: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

برنامه 11: فلوچارت و برنامه‌ای بنویسید که عددی مانند N را گرفته، فاکتوریل آن را محاسبه کرده و نشان دهد. در پایان برای عددی مانند $N=4$ را ردیابی کنید.

حل: شمارنده i را از 1 شروع کرده و تا زمانی که از N کوچکتر است، آنرا به $Fact$ ضرب می‌کنیم.

تعداد ورودی: 1 تعداد خروجی: 1 شمارنده: 1 (گام حرکت: 1)

```
#include <iostream>
using namespace std;
int main()
{
    int i , N;
    long int Fact=1;
    cin>>N;
```

```
    i=1;
    while (i<=N)
    {
        Fact*=i;
        i++;
    }
```

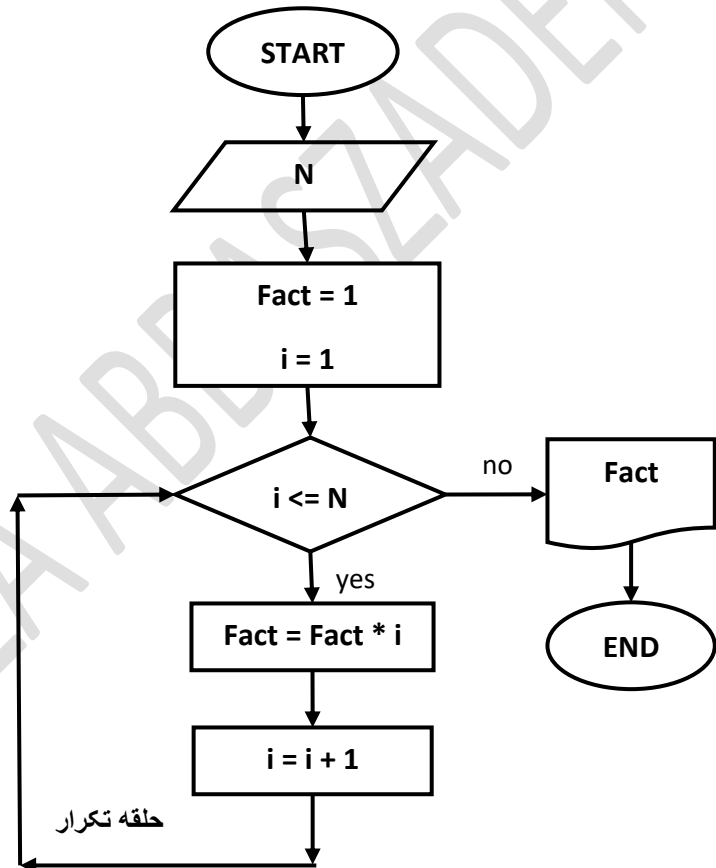
```
    cout<<"Factoriel N: "<<Fact;
    return(0);
}
```

//.....OR.....//

```
#include <iostream>
using namespace std;
int main()
{
    int i , N;    long int Fact=1;
    cin>>N;
```

```
    for (i=1 ; i<=N ; i++)
        Fact*=i;
```

```
    cout<<"Factoriel N: "<<Fact;
    return(0);
}
```



N	i	Fact
4	1	1
	2	1
	3	2
	4	6
ردیابی	5	24

تمرین 11: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

برنامه 12: فلوچارت و برنامه‌ای بنویسید که عددی مانند N را گرفته، مجموع اعداد زوج 1 تا N را محاسبه کرده و نشان دهد. در پایان برای عددی مانند $N=9$ را ردیابی کنید.

حل: شمارنده i را از 2 شروع کرده و تا زمانی که از N کوچکتر است، آنرا با Sum جمع می‌زنیم.

تعداد ورودی: 1 تعداد خروجی: 1 شمارنده: 1 (گام حرکت: 2)

```
#include <iostream>
using namespace std;
int main()
{
    int i , N , Sum=0;
    cin>>N;
```

```
    i=2;
    while (i<=N)
    {
        Sum+=i;
        i+=2;
    }
```

```
    cout<<"Majmoo Zoj 1 ta N: "<<Sum;
    return(0);
}
```

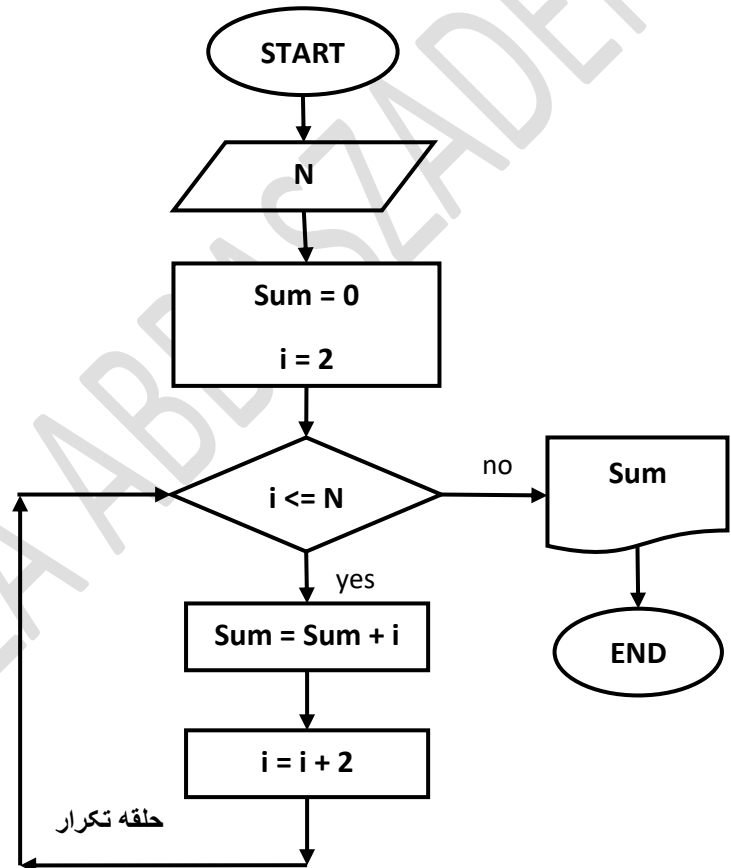
//.....OR.....//

```
#include <iostream>
using namespace std;
int main()
{
```

```
    int N , i , Sum=0;
    cin>>N;
```

```
    for (i=2 ; i<=N ; i+=2)
        Sum+=i;
```

```
    cout<<"Majmoo Zoj 1 ta N: "<<Sum;
    return(0);
}
```



N	i	Sum
9	2	0
	4	2
	6	6
	8	12
	10	20

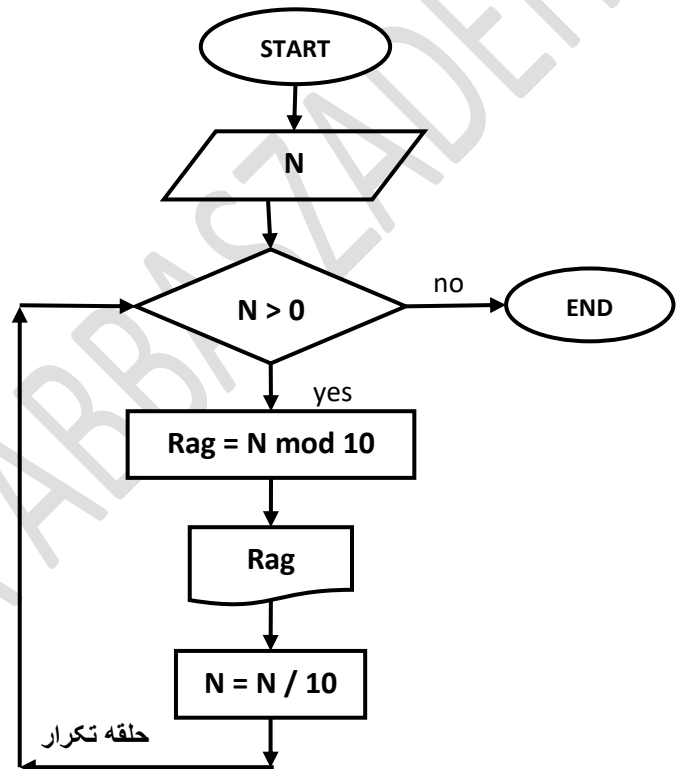
تمرین 12: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

برنامه 13: فلوچارت و برنامه‌ای بنویسید که عددی مانند N را گرفته، رقم‌های آنرا از آخر به اول نشان دهد. در پایان برای اعدادی مانند 1399، 206 ردیابی کنید.

حل: هر عددی را تقسیم بر 10 کنیم، باقیمانده همان رقم یکان خواهد بود. (تقسیمات متوالی بر 10)

تعداد ورودی: 1 تعداد خروجی: به تعداد رقم‌ها

```
#include <iostream>
using namespace std;
int main()
{
    long int N;
    short int Rag;
    cin>>N;
    while (N > 0)
    {
        Rag = N % 10;
        cout<<Rag<<endl;
        N /= 10;
    }
    return(0);
}
```



ردیابی:

N	Rag	خروجی
1399	9	9
139	9	9
13	3	3
1	1	1
0		

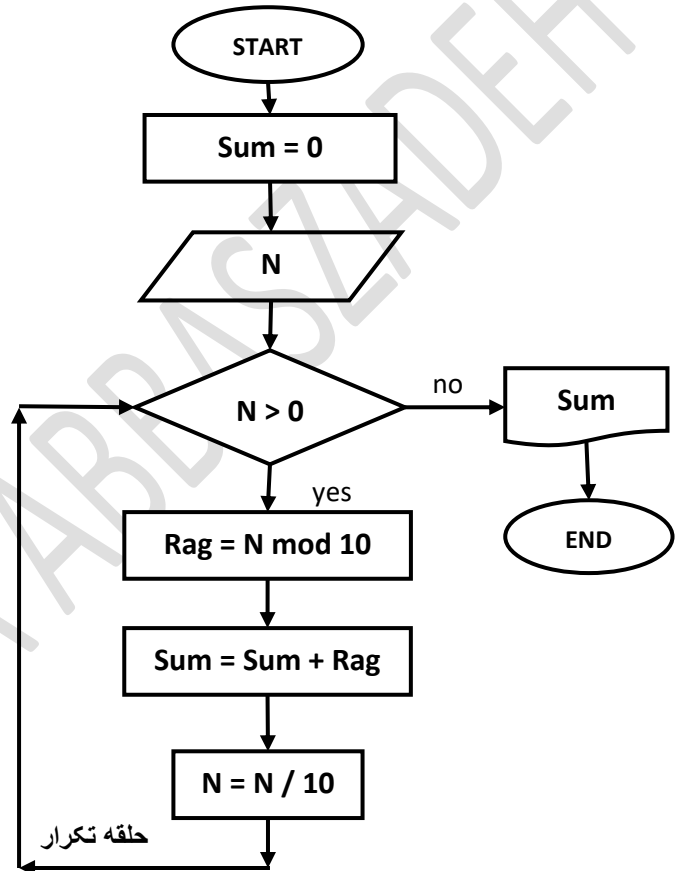
N(عدد)	Rag(رقم)	خروجی
206	6	6
20	0	0
2	2	2
0		

تمرین 13: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

برنامه 14: فلوچارت و برنامه‌ای بنویسید که عددی مانند N را گرفته، مجموع رقم‌های آنرا محاسبه و نشان دهد. در پایان برای اعدادی مانند 1399، 206 ردیابی کنید.

حل: با تقسیمات متوالی بر 10، ارقام را پیدا کرده و با Sum جمع می‌زنیم. تعداد ورودی: 1 تعداد خروجی: 1

```
#include <iostream>
using namespace std;
int main()
{
    long int N, Sum=0;
    short int Rag;
    cin>>N;
    while (N > 0)
    {
        Rag = N % 10;
        Sum += Rag;
        N /= 10;
    }
    cout<<"Majmoo Argaam: "<<Sum;
    return(0);
}
```



ردیابی:

N	Rag	Sum
1399	9	0
139	9	9
13	3	18
1	1	21
0		22

N(عدد)	Rag(رقم)	Sum
206	6	0
20	0	6
2	2	6
0		8

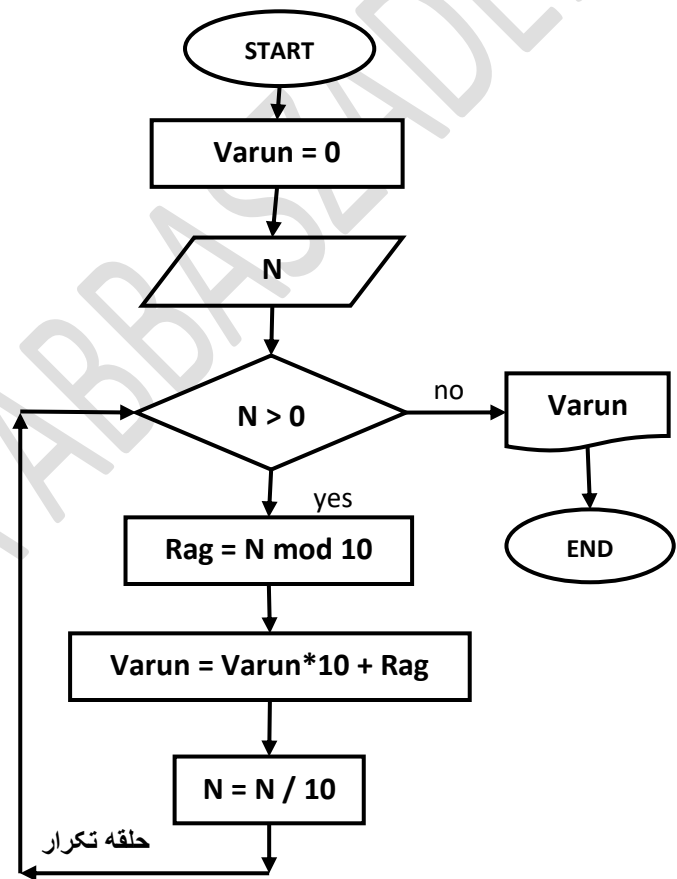
تمرین 14: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

برنامه 15: فلوچارت و برنامه‌ای بنویسید که عددی مانند N را گرفته، وارون (برعکس) آنرا محاسبه و نشان دهد. در پایان برای اعدادی مانند 1399، 206 ردیابی کنید.

حل: در هر مرحله، رقم جدید را پیدا کرده و سپس $\text{Varun} = (\text{Varun} * 10) + \text{Rag}$

تعداد ورودی: 1 تعداد خروجی: 1

```
#include <iostream>
using namespace std;
int main()
{
    long int N, Varun=0;
    short int Rag;
    cin>>N;
    while (N > 0)
    {
        Rag = N % 10;
        Varun = Varun*10 + Rag;
        N /= 10;
    }
    cout<<"Varoon e Adad: "<<Varun;
    return(0);
}
```



ردیابی:

N	Rag	Var
1399	9	0
139	9	9
13	3	99
1	1	993
0		9931

N(عدد)	Rag(رقم)	Var(وارون)
206	6	0
20	0	6
2	2	60
0		602

تمرین 15: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

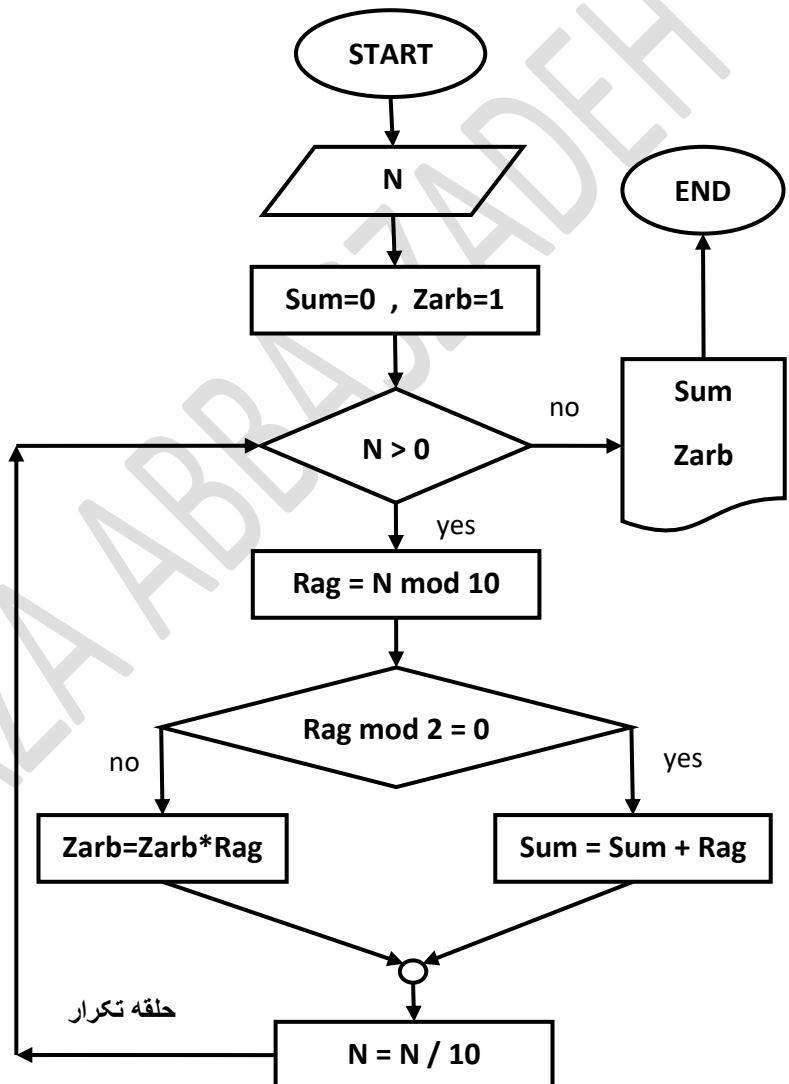
برنامه 16: فلوچارت و برنامه‌ای بنویسید که عددی مانند N را گرفته، و در خروجی، مجموع ارقام زوج و حاصلضرب ارقام فرد آن عدد را محاسبه و نشان دهد. در پایان برای عددی مانند 380452 ردیابی کنید.

حل: در هر مرحله، رقم جدید را پیدا کرده و بررسی می‌کنیم که زوج است یا فرد. (تعداد ورودی: 1 تعداد خروجی: 2)

```
#include <iostream>
using namespace std;
int main()
{
    long int N , Sum=0 , Zarb=1;
    short int Rag;
    cin>>N;
    while (N > 0)
    {
        Rag = N % 10;
        if(Rag % 2 == 0)
            Sum = Sum + Rag;
        else
            Zarb = Zarb * Rag;
        N /= 10;
    }
    cout<<"Majmoo Argam Zoj: "<<Sum;
    cout<<endl;
    cout<<"Zarb Argam Fard: "<<Zarb;
    return(0);
}
```

ردیابی:

N	Rag	Sum	Zarb
380452	2	0	1
38045	5	2	
3804	4		5
380	0	6	
38	8	6	
3	3	14	
0			15

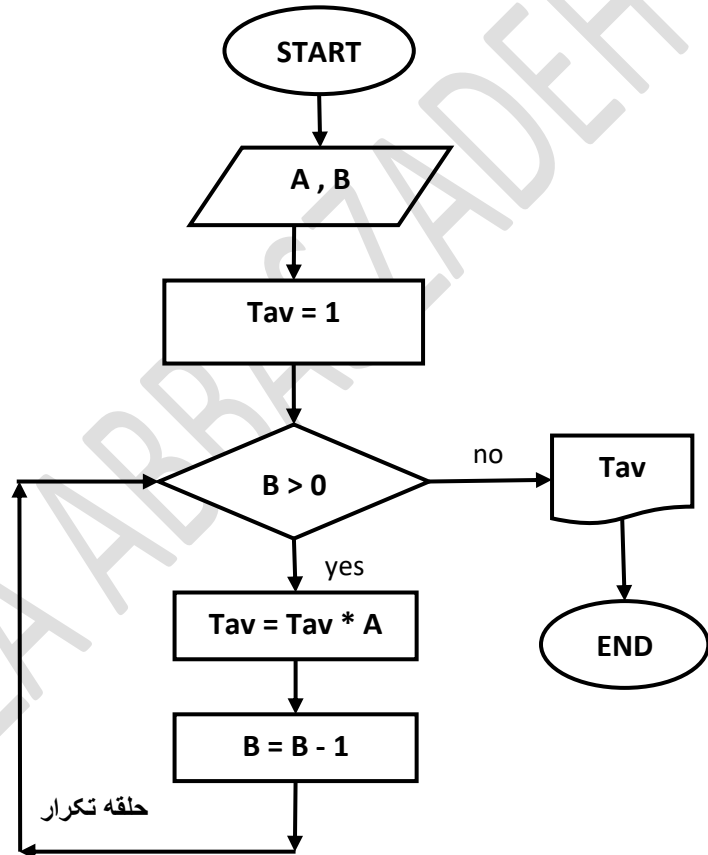


تمرین 16: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

برنامه 17: فلوجارت و برنامه‌ای بنویسید که دو عدد مانند A , B را گرفته، نتیجه A به توان B را به روش ضرب کردن محاسبه کرده و نشان دهد. در پایان برای اعدادی مانند B=3 , A=2 ردیابی کنید.

حل: باید B بار عدد A را در خودش ضرب کنیم. تعداد ورودی: 2 تعداد خروجی: 1

```
#include <iostream>
using namespace std;
int main()
{
    int A , B;
    long int Tav=1;
    cout<<"Lotfan 2 Adad vared konid: ";
    cin>>A>>B;
    while (B > 0)
    {
        Tav *= A;
        B --;
    }
    cout<<"Hasel Tavan ba *: "<<Tav;
    return(0);
}
```



ردیابی:

A	B	Tav
2	3	1
	2	2
	1	4
	0	8

تمرین 17: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

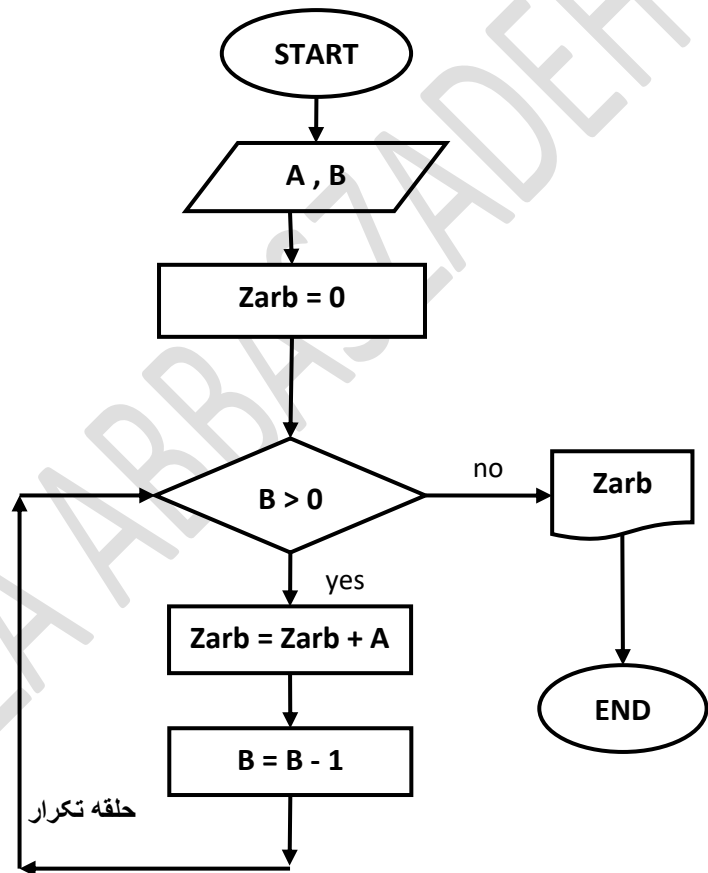
تمرین 18: فلوچارت و برنامه‌ای بنویسید که دو عدد مانند A , B را گرفته، نتیجه A ضربدر B را به روش جمع کردن محاسبه کرده و نشان دهد. در پایان برای اعدادی مانند B=3 , A=5 ردیابی کنید.

حل: باید B بار عدد A را با خودش جمع کنیم. تعداد ورودی: 2 تعداد خروجی: 1

```
#include <iostream>
using namespace std;
int main()
{
    int A , B;
    long int Zarb=0;
    cout<<"Lotfan 2 Adad vared konid: ";
    cin>>A>>B;
    while (B > 0)
    {
        Zarb += A;
        B --;
    }
    cout<<"Hasel Zarb ba +: "<<Zarb;
    return(0);
}
```

ردیابی:

A	B	Zarb
5	3	0
	2	5
	1	10
	0	15

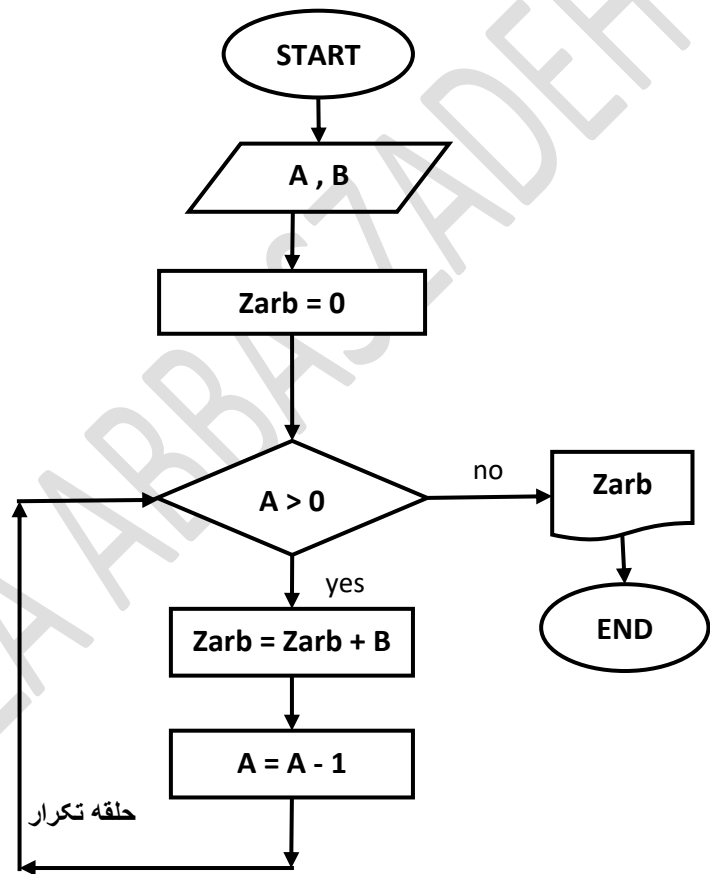


تمرین 18: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

برنامه 19: فلوجارت و برنامه‌ای بنویسید که دو عدد مانند A , B را گرفته، نتیجه A ضربدر B را به روش جمع کردن محاسبه کرده و نشان دهد. در پایان برای اعدادی مانند B=3 , A=5 ردیابی کنید.

حل: باید A بار عدد B را با خودش جمع کنیم. تعداد ورودی: 2 تعداد خروجی: 1

```
#include <iostream>
using namespace std;
int main()
{
    int A , B;
    long int Zarb=0;
    cout<<"Lotfan 2 Adad vared konid: ";
    cin>>A>>B;
    while (A > 0)
    {
        Zarb += B;
        A --;
    }
    cout<<"Hasel Zarb ba +: "<<Zarb;
    return(0);
}
```



ردیابی:

A	B	Zarb
5	3	0
4		3
3		6
2		9
1		12
0		15

تمرین 19: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

برنامه 20: فلوجارت و برنامه‌ای بنویسید که عددی مانند N را گرفته، و در خروجی تمام مقسوم علیه های آنرا نشان دهد. در پایان برای عددی مانند 6 ردیابی کنید.

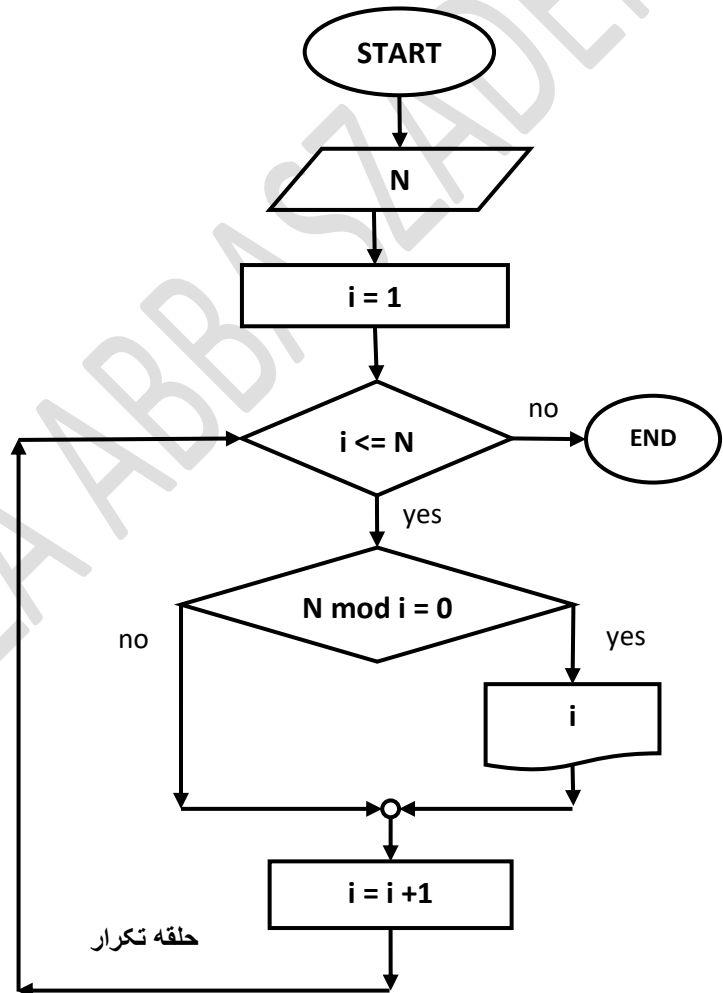
حل: شمارنده‌ی i از عدد 1 شروع می‌شود و تا خود N پیش می‌رود. اگر عدد N به i بخشپذیر باشد، i چاپ می‌شود.

تعداد ورودی: 1
تعداد خروجی: به تعداد مقسوم علیه ها

```
#include <iostream>
using namespace std;
int main()
{
    long int N, i=1;
    cin>>N;
    cout<<"Magsom Aleiyyha: ";
    while (i <= N)
    {
        if(N % i == 0)
            cout<<i<<" ";
        i++;
    }
    return(0);
}
```

ردیابی:

N	i	خروجی
6	1	1
	2	2
	3	3
	4	
	5	
	6	6
	7	



تمرین 20: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

تمرین 21: فلوچارت و برنامه‌ای بنویسید که عددی مانند N را گرفته، و نشان دهد که کامل (تام) است یا نه. در پایان برای اعدادی مانند 8، 6 ردیابی کنید. (به عددی کامل گفته می‌شود که مجموع مقسوم‌علیه‌های کوچکتر از خودش، برابر خود عدد باشد. مانند 6، 28، ...)

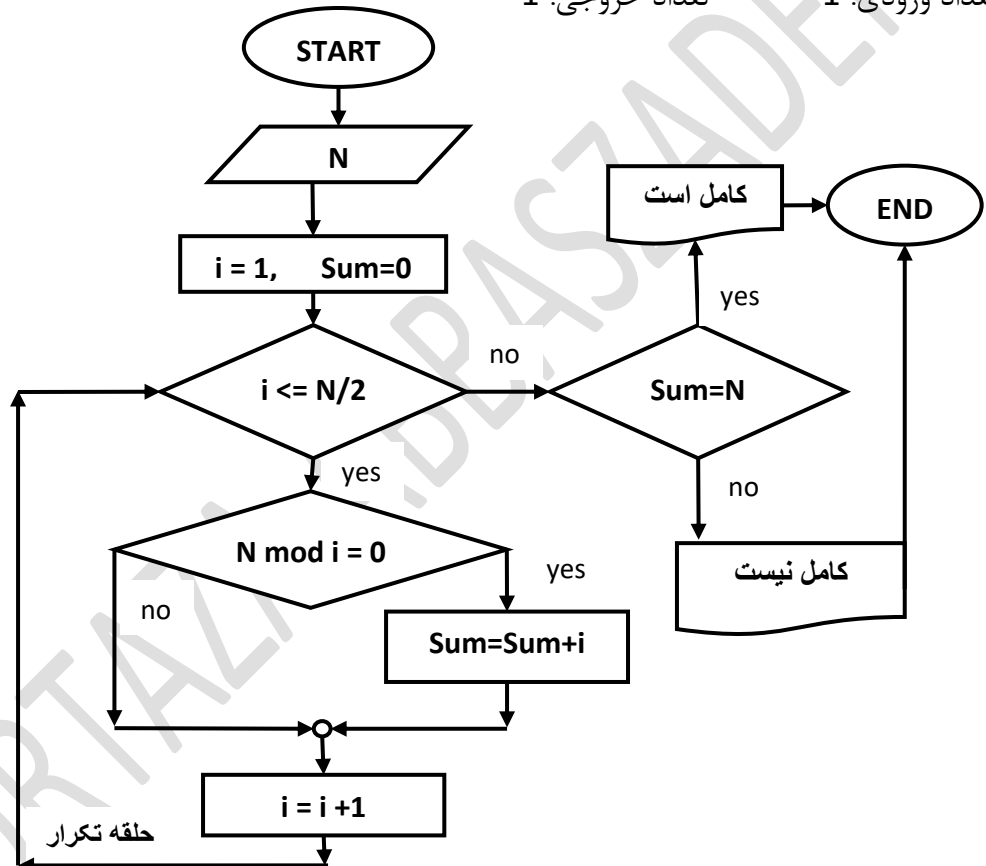
حل: مقسوم علیه‌ها را تا نصف عدد، پیدا کرده باهم جمع می‌زنیم. سپس بررسی می‌کنیم با خود عدد برابر است یا نیست.

تعداد ورودی: 1 تعداد خروجی: 1

ردیابی:

N	i	Sum	خروجی
6	1	0	
	2	1	
	3	3	
	4	6	کامل است

N	i	Sum	خروجی
8	1	0	
	2	1	
	3	3	
	4		
	5	7	کامل نیست



```

#include <iostream>
using namespace std;
int main() {
    int N, i, Sum=0;
    cin >> N;
    for (i=1; i <= N/2; i++)
        if(N % i == 0) Sum += i;
    if (N == Sum)    cout<<"Kamel Ast.";
    else            cout<<"Kamel Nist.";
    return(0);
}
    
```

تمرین 21: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

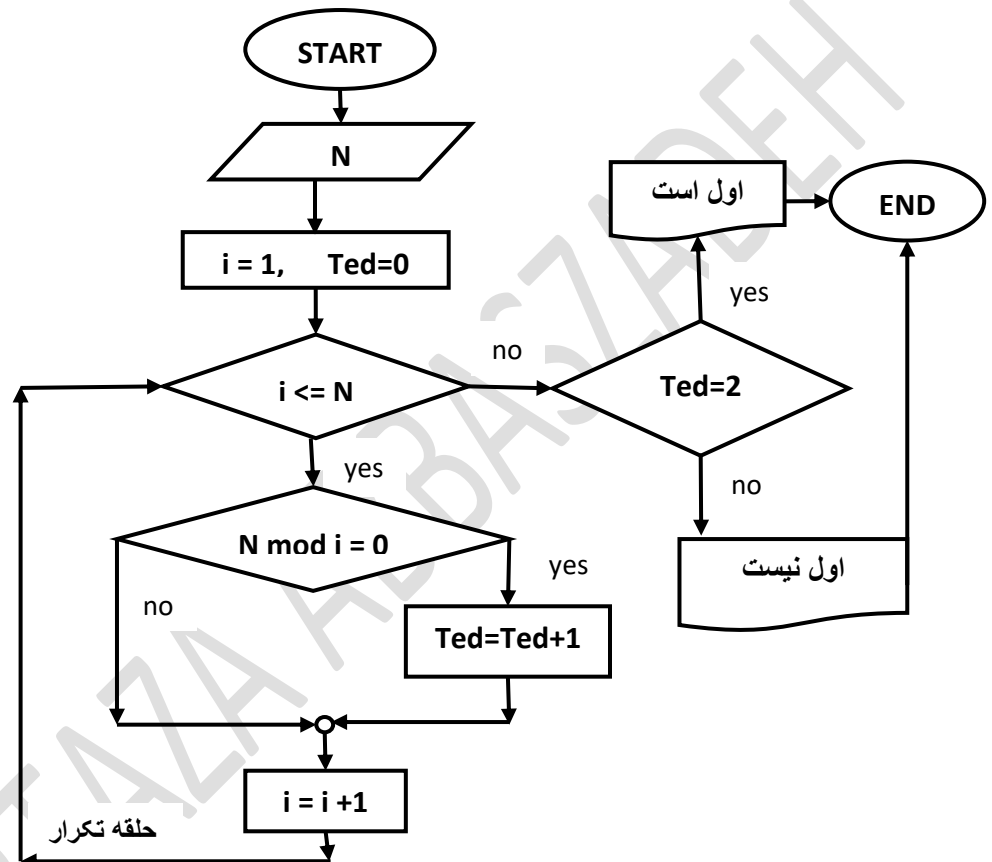
برنامه 22: به عددی که فقط دو مقسوم علیه داشته باشد اول گفته می شود. با این روش، فلوچارت و برنامه ای بنویسید که عددی مانند N را گرفته، و نشان دهد اول است یا نه. برای اعدادی مانند 5، 6 ردیابی کنید.

حل: تعداد مقسوم علیه ها را پیدا می کنیم، سپس بررسی می کنیم برابر 2 است یا نه. (روش کند) (ورودی: 1 خروجی: 1)

ردیابی:

N	i	Ted	خروجی
4	1	0	
	2	1	
	3	2	
	4		
	5	3	اول نیست

N	i	Ted	خروجی
5	1	0	
	2	1	
	3		
	4		
	5		
	6	2	اول است



```
#include <iostream>
using namespace std;
int main() {
```

```
    int N, i, Ted=0;
    cin >> N;
    for (i = 1; i <= N; i++)
        if (N % i == 0) Ted++;
    if (Ted == 2)      cout << "Avval Ast.";
    else               cout << "Avval Nist.";
    return(0);
}
```

نکته: این روش بسیار کند است. مثلاً برای اینکه بفهمد عدد 1000 اول است یا نه، هزار بار حلقه را تکرار کرده و تمام مقسوم علیه های آنرا پیدا می کند. (در حالیکه واقعاً لازم نیست!) **تمرین 22:** برنامه در کامپیوتر برای اعداد مختلف، اجرا شود.

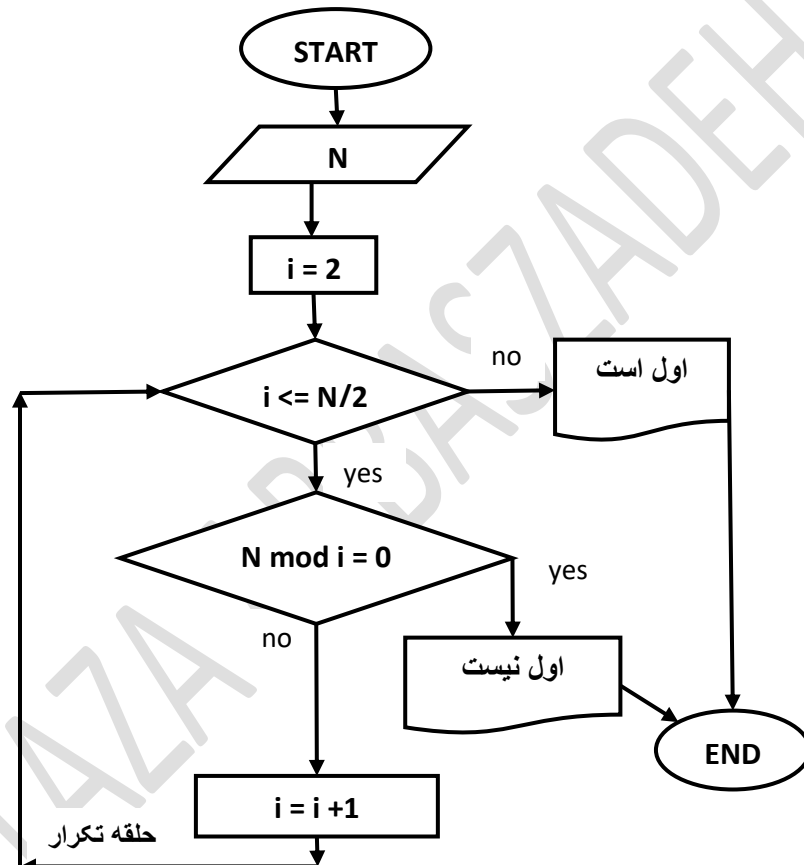
تمرین 23: اگر عددی به غیر از 1 و خودش به چیزی بخش پذیر باشد، اول نیست. با این روش، فلوچارت و برنامه‌ای بنویسید که عددی را گرفته، و نشان دهد اول است یا نه. در پایان برای اعدادی مثل 15، 13 ردیابی کنید.

حل: شمارنده از 2 شروع شده و تا نصف عدد، نباید به چیزی بخش پذیر باشد. (روش سریع) ورودی: 1 خروجی: 1

ردیابی:

N	i	خروجی
13	2	
	3	
	4	
	5	
	6	
	7	اول است

N	i	خروجی
15	2	
	3	اول نیست



```

#include <iostream>
using namespace std;
int main() {
    int N , i;
    cin >> N;
    for (i = 2 ; i <= N/2 ; i++)
        if(N % i == 0) { cout<<"Avval Nist."; return(0); }
    cout<<"Avval Ast.";
    return(0);
}
    
```

تمرین 23: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

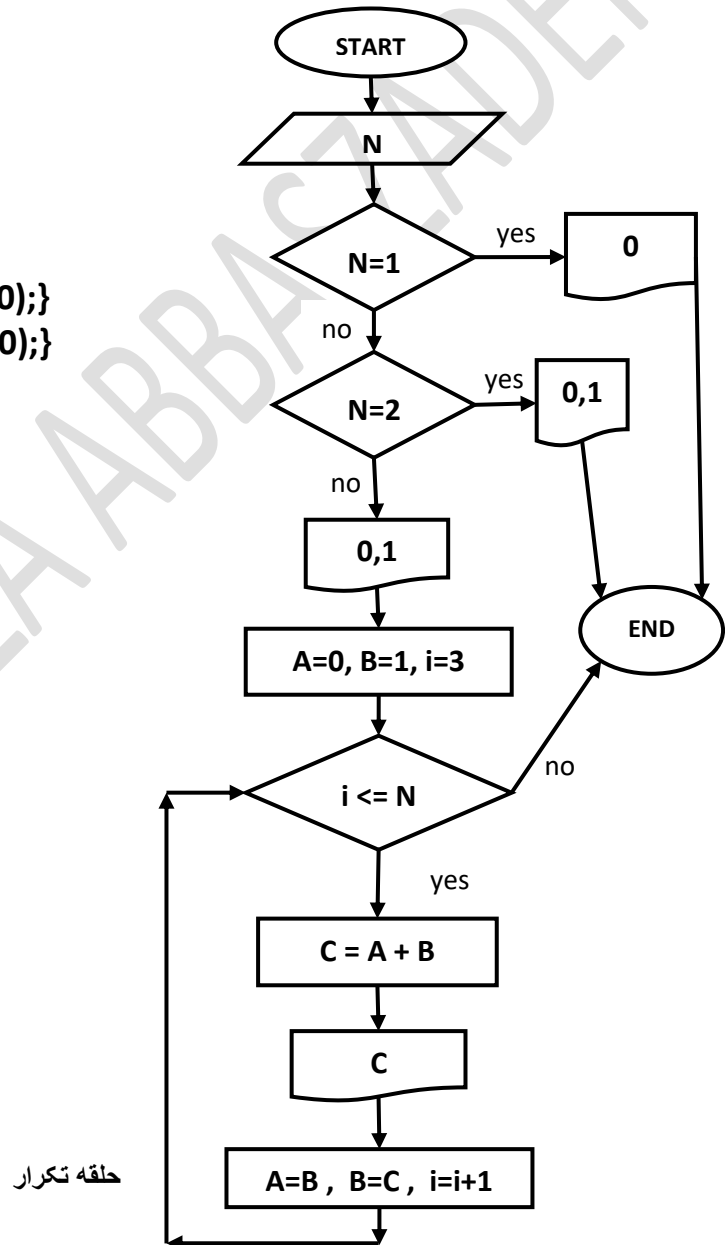
برنامه 24: دنباله فیبوناچی (مساله رشد جمعیت خرگوش ها) اینگونه است: 0,1,1,2,3,5,8,13,21,34,55,89,144,...
جمله اول 0، جمله دوم 1، سپس هر جمله برابر است با جمع دو جمله قبل از خودش. با این روش، فلوچارت و برنامه ای بنویسید که N جمله فیبوناچی را نشان دهد. در پایان برای N=8 ردیابی کنید.

حل: اولین جمله که پیدا می کنیم، جمله سوم است پس شمارنده از 3 شروع می شود. تعداد ورودی: 1 تعداد خروجی: N

```
#include <iostream>
using namespace std;
int main()
{
    int N , i;
    long int A=0,B=1,C;
    cin >> N;
    if(N==1) {cout<<"0"; return(0);}
    else if(N==2) {cout<<"0,1"; return(0);}
    cout<<"0,1";
    for (i = 3 ; i <= N ; i++)
    {
        C=A+B;
        cout<<" , "<<C;
        A=B;
        B=C;
    }
    return(0);
}
```

ردیابی:

N	i	A	B	C	خروجی
8	3	0	1	1	0,1
	4	1	1	2	,1
	5	1	2	3	,2
	6	2	3	5	,3
	7	3	5	8	,5
	8	5	8	13	,8
	9	8	13		,13



تمرین 24: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

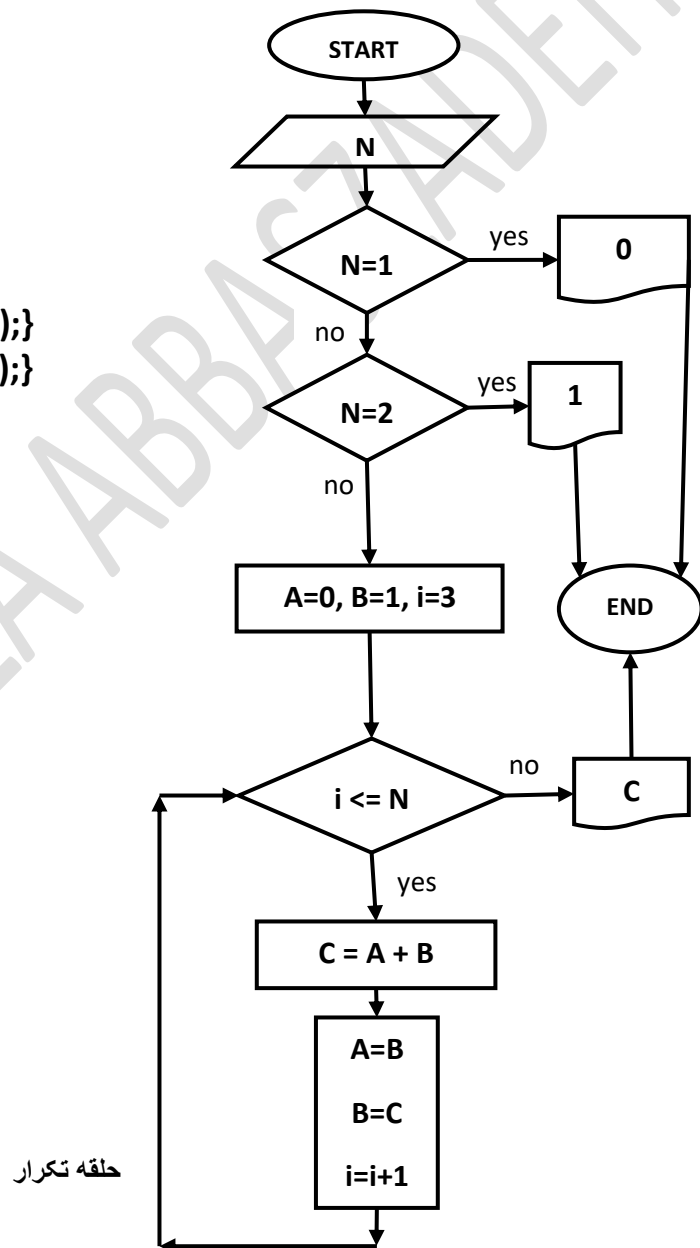
برنامه 25: دنباله فیبوناچی (مساله رشد جمعیت خرگوش‌ها) اینگونه است: 0,1,1,2,3,5,8,13,21,34,55,89,144,... جمله اول 0، جمله دوم 1، سپس هر جمله برابر است با جمع دو جمله قبل از خودش. با این روش، فلوچارت و برنامه‌ای بنویسید که فقط جمله‌ی N ام فیبوناچی را نشان دهد. در پایان برای N=8 ردیابی کنید.

حل: چاپ خروجی، خارج از حلقه خواهد بود. تعداد ورودی: 1 تعداد خروجی: 1

```
#include <iostream>
using namespace std;
int main()
{
    int i , N;
    long int A,B,C;
    cin >> N;
    if(N==1) {cout<<"0"; return(0);}
    else if(N==2) {cout<<"1"; return(0);}
    for (A=0,B=1,i=3 ; i <= N ; i++)
    {
        C=A+B;
        A=B;
        B=C;
    }
    cout<<C;
    return(0);
}
```

ردیابی:

N	i	A	B	C	خروجی
8	3	0	1	1	
	4	1	1	2	
	5	1	2	3	
	6	2	3	5	
	7	3	5	8	
	8	5	8	13	
	9	8	13		13



تمرین 25: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

حلقه‌های تودرتو (لانه‌ای Nested):

اگر در داخل یک حلقه، دوباره حلقه‌ی دیگری وجود داشته باشد، به آن حلقه‌ی تودرتو گفته می‌شود. برای حل بعضی از برنامه‌ها لازم است تا از حلقه‌های تودرتو استفاده شود.

برای مثال:

```
for(x=1 ; x<=10 ; x++)
```

```
{
```

```
//...1
```

```
for(y=1 ; y<=20 ; y++)
```

```
{
```

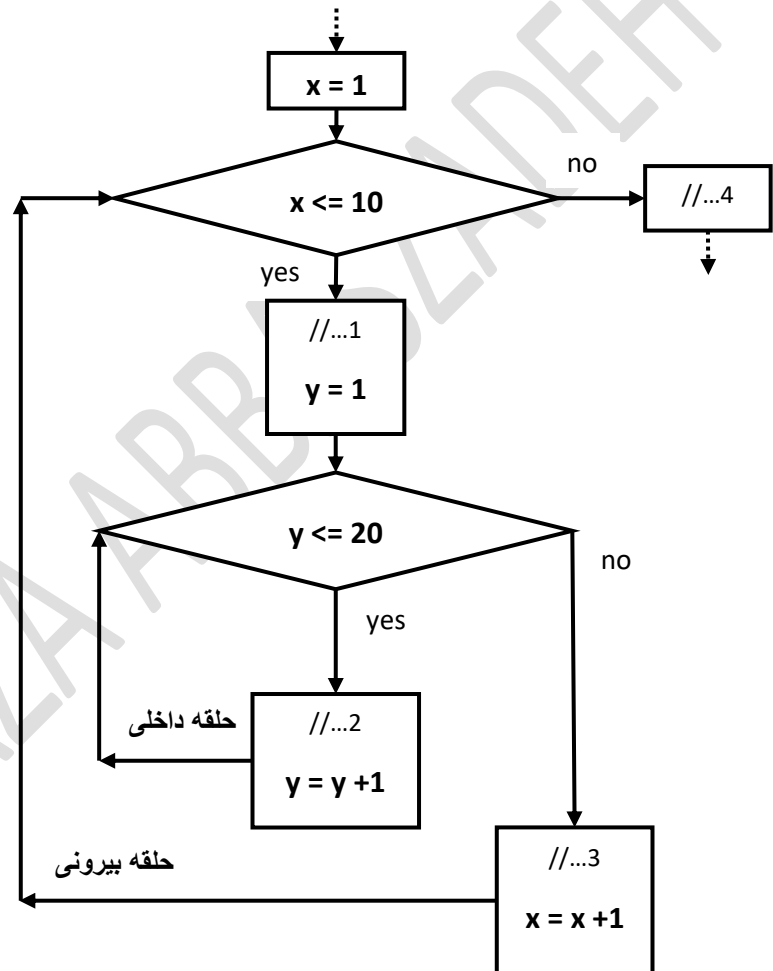
```
//...2
```

```
}
```

```
//...3
```

```
}
```

```
//...4
```



دقت: در این برنامه، حلقه اول 10 بار و حلقه دوم 20 بار تکرار می‌شود. یعنی دستوراتی مثل //...1 یا //...3 فقط 10 بار و آن هم در داخل حلقه بیرونی (حلقه اول) تکرار می‌شوند، ولی دستوراتی مثل //...2 چون در داخل حلقه تودرتو هستند لذا 200 بار (10*20) اجرا خواهند شد. باید دقت داشت که به ازای هر بار x++، ده بار y++ اجرا خواهد شد.

برنامه 26: برنامه‌ای بنویسید که خروجی آن همانند جدول ضرب 10×10 و به صورت زیر باشد.

خروجی:

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
...									
...									
...									
10	20	30	40	50	60	70	80	90	100

حل: از حلقه‌های تودرتو استفاده می‌شود. (تعداد ورودی: 0 تعداد خروجی: 100)

```
#include <iostream>
using namespace std;
int main()
{
    int x,y;
    for (x =1 ; x <= 10 ; x++)
    {
        for (y =1 ; y <= 10 ; y++)
            cout<<x*y<<"\t";
        cout<<endl;
    }
    return(0);
}
```

نکته: کاراکتر کنترل `"\t"` همان معادل کلید Tab صفحه‌کلید است که باعث مرتب دیده شدن خروجی می‌شود.

تمرین 26: این برنامه را در کامپیوتر اجرا کرده و برای اعداد مختلف، خروجی بگیرید.

برنامه 27: برنامه‌ای بنویسید که با گرفتن تعداد سطرها از ورودی، خروجی آن بصورت زیر باشد (پایین مثلثی):

```
*
* *
* * *
* * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

تعداد خروجی: به تعداد مجموع 1 تا ورودی)

حل: از حلقه‌های تودرتو استفاده می‌شود. (تعداد ورودی: 1

```
#include <iostream>
using namespace std;
int main()
{
    int N,i,j;
    cout<<"Lotfan Tedad Satr: ";
    cin>>N;
    for (i=1 ; i <= N ; i++)
    {
        for (j =1 ; j <= i ; j++)
            cout<<"* ";
        cout<<endl;
    }
    return(0);
}
```

تمرین 27: الف) این برنامه را در کامپیوتر اجرا کرده، خروجی بگیرید. ب) برنامه‌ای مانند برنامه‌ی فوق بنویسید که خروجی آن بالا مثلثی باشد.

برنامه 28: فلوچارت و برنامه‌ای بنویسید که اعداد اول دو رقمی را نشان دهد.

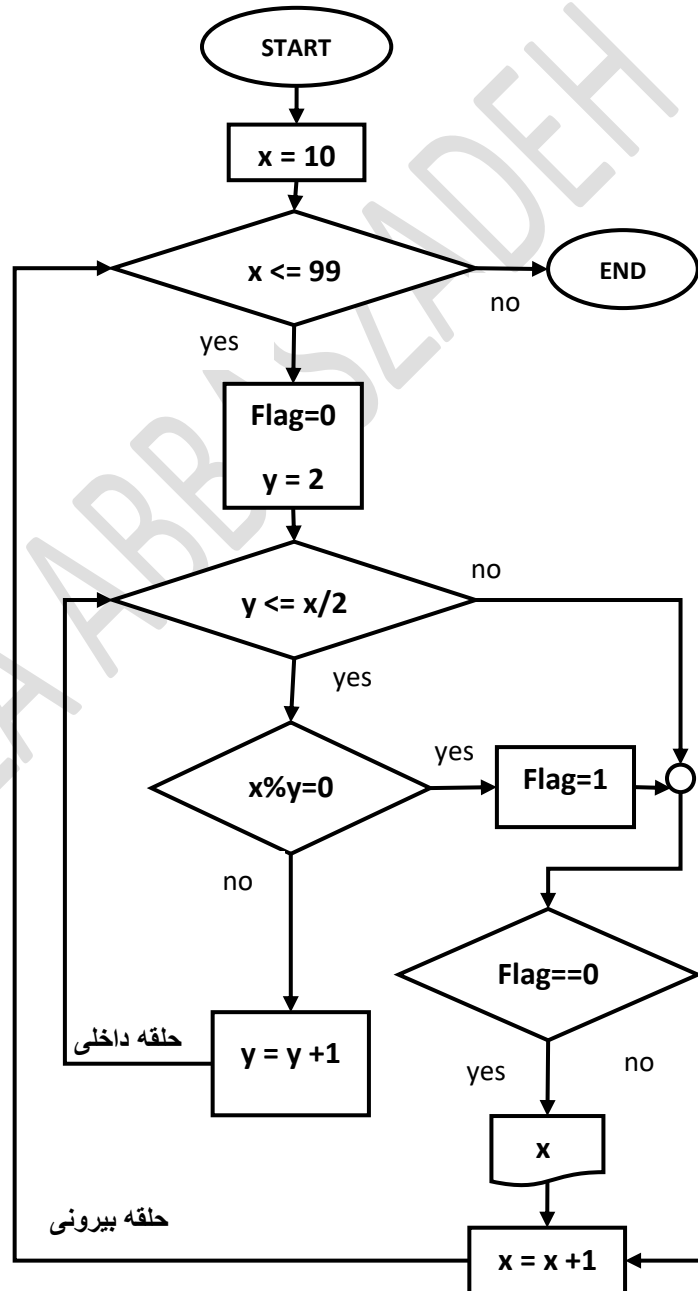
حل: حلقه بیرونی: از 10 تا 99

حلقه درونی: اول بودن

```
#include <iostream>
using namespace std;
int main()
{
    int x,y,Flag;
    cout<<"Adad e Avval 2-Ragami: ";
    for (x =10 ; x <= 99 ; x++)
    {
        Flag=0;
        for (y =2 ; y <= x/2 ; y++)
            if(x%y==0)
                {Flag=1; break;}
        if(Flag==0) cout<<x<<" , ";
    }
    return(0);
}
```

خروجی:

11, 13, 17, 19, 23, ..., 97



تمرین 28: الف) این برنامه را در کامپیوتر اجرا کرده، تمام خروجی‌ها را بنویسید. ب) این برنامه را برای اعداد 3-رقمی نوشته، در کامپیوتر اجرا کرده، تمام خروجی‌ها را بنویسید.

برنامه 29: فلوچارت و برنامه‌ای بنویسید که اعداد کامل کوچکتر از N را نشان دهد. (N را کاربر وارد می‌کند)

حل: حلقه بیرونی: از 1 تا N حلقه درونی: کامل بودن

```
#include <iostream>
using namespace std;
int main()
{
    int x,y,N,Sum;
    cin>>N;
    cout<<"Adad e Kamel 1-N: ";
    for (x =1 ; x <= N ; x++)
    {
        Sum=0;
        for (y =1 ; y <= x/2 ; y++)
            if(x%y==0)
                Sum+=y;
        if(Sum==x) cout<<x<<" ";
    }
    return(0);
}
```

تمرین 29: الف) فلوچارت این برنامه را رسم کنید. ب) این برنامه را در کامپیوتر اجرا کنید و به ازای $N=50000$ خروجی برنامه را بنویسید.

برنامه 30: فلوچارت و برنامه‌ای بنویسید که مقسوم‌علیه‌های اول عددی مانند N را نشان دهد.

حل: حلقه بیرونی: پیدا کردن مقسوم‌علیه
حلقه درونی: بررسی اول بودن آن مقسوم‌علیه

```
#include <iostream>
using namespace std;
int main()
{
    int x,y,N,Flag;
    cin>>N;
    cout<<"Magsum Aleiy ha ye Avval e N: ";
    for (x =2 ; x <= N ; x++)
    {
        if(N%x==0)
        {
            Flag=0;
            for (y =2 ; y <= x/2 ; y++)
                if(x%y==0)
                    {Flag=1; break;}
            if(Flag==0) cout<<x<<" ";
        }
    }
    return(0);
}
```

تمرین 30: الف) فلوچارت این برنامه را رسم کنید. ب) این برنامه را در کامپیوتر اجرا کنید و برای اعداد مختلف، خروجی برنامه را بنویسید.

فصل چهارم: آرایه‌ها

آرایه (Array): آرایه عبارتست از یک اسم برای مجموعه‌ای از چندین خانه‌ی هم‌نوع و پشت سر هم در حافظه.

اگر در برنامه‌ای به چندین متغیر هم‌نوع نیاز باشد بهتر است بجای اینکه آنها را تک تک تعریف کنیم، از آرایه‌ها استفاده کنیم. البته آرایه‌ها بر حسب نیاز می‌توانند یک بعدی، دو بعدی، ... ، n بعدی تعریف و استفاده شوند. تعداد خانه‌های آرایه باید از قبل مشخص باشد. (تخصیص حافظه ایستا یا static)

اندیس آرایه (Index): برای اینکه بتوان به خانه‌های آرایه دسترسی پیدا کرد، خانه‌های آن را شماره گذاری کرده‌اند، که به این شماره‌ها، اندیس آرایه گفته می‌شود. باید دقت داشت که در زبان C++ اندیس ها از عدد 0 شروع می‌شوند (نه 1).

نام (اسم) آرایه: نام آرایه در واقع اشاره‌گر به اولین خانه‌ی آرایه (اندیس 0) می‌باشد.

نحوه‌ی تعریف آرایه:

نوع خانه‌های آرایه	نام آرایه	؛[بعد آخر] ... [بعد دوم] [بعد اول]
--------------------	-----------	------------------------------------

مثال 1: آرایه یک بعدی بنام A با 3_خانه از نوع عدد صحیح:

`int A[3];`

ورودی	خروجی	20
2	1	0

`A[0]=20;` `cin>>A[2];` `cout<<A[1];`

مثال 2: مقداردهی اولیه به خانه‌های آرایه‌ی یک بعدی: (آرایه B با 5_خانه‌ی اعشاری)

`float B[]={1.5 , 2 , 5.2 , 3.3 , 19.75};`

1.5	2.0	5.2	3.3	19.75
0	1	2	3	4

رشته (string): در واقع همان آرایه‌ای از کاراکترها می‌باشد. مثال:

`string name="Mohammad";`

`char name[]= {'M' , 'o' , 'h' , 'a' , 'm' , 'm' , 'a' , 'd'};`

برنامه 31: برنامه‌ای بنویسید که یک آرایه 12 خانه‌ای از اعداد دلخواه را تعریف کرده و آنها را نشان دهد. سپس ماکزیمم آنها را پیدا کرده و نشان دهد.

حل: از یک آرایه‌ای تک بعدی 12 خانه‌ای و از نوع صحیح استفاده می‌کنیم. ابتدا فرض می‌کنیم اولین خانه ماکزیمم است. سپس آنرا با استفاده از یک حلقه، با سایر خانه‌ها مقایسه کرده و ماکزیمم واقعی را پیدا می‌کنیم.

0	1	2	3	4	5	6	7	8	9	10	11

```
#include <iostream>
using namespace std;
int main()
{
    int Ar[ ]={1,2,3,4,5,6,777,8,9,100,11,12} , Max;
    cout<<"Array: ";

    for (int i =0 ; i < 12 ; i++)    // حلقه‌ی نشان دادن مقادیر آرایه
        cout<<Ar[i]<<" , ";

    Max=Ar[0];
    for (int i =1 ; i < 12 ; i++)    // حلقه‌ی پیدا کردن ماکزیمم در آرایه
        if(Ar[i]>Max) Max=Ar[i];

    cout<<"Maximum: "<<Max;
    return(0);
}
```

تمرین 31: این برنامه را در کامپیوتر اجرا کرده، خروجی برنامه را بنویسید.

برنامه 32: برنامه‌ای بنویسید که 7 نمره‌ی یک دانشجو را از ورودی گرفته و در آرایه‌ای قرار دهد. سپس معدل دانشجو را حساب کرده و نشان دهد.

حل: از یک آرایه‌ای تک بعدی با 7 خانه‌ی اعشاری استفاده می‌کنیم.

0	1	2	3	4	5	6

```
#include <iostream>
using namespace std;
int main()
{
    float Ar[7], Avg=0;
    int i;
    cout<<"Lotfan 7 Nomre vared konid: ";

    for (i =0 ; i < 7 ; i++)           // حلقه‌ی گرفتن مقادیر آرایه
        cin>>Ar[i];

    cout<<"Nomarat: ";
    for (i =0 ; i < 7 ; i++)           // حلقه‌ی نشان دادن مقادیر آرایه
        cout<<Ar[i]<<" ";

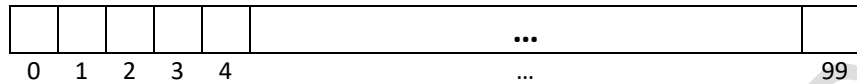
    for (i =0 ; i < 7 ; i++)           // حلقه‌ی جمع زدن نمرات داخل آرایه
        Avg += Ar[i];

    Avg=Avg/7;                         // محاسبه‌ی معدل
    cout<<"Moaddel: "<<Avg;
    return(0);
}
```

تمرین 32: این برنامه را در کامپیوتر برای نمرات مختلف اجرا کرده، خروجی برنامه را بنویسید.

برنامه 33: برنامه‌ای بنویسید که عددی را گرفته (در مبنای دهدهی) و آن را در مبنای دودویی نشان دهد.

حل: برای این کار باید عدد را بصورت متوالی بر 2 تقسیم کنیم و باقی‌مانده‌ها را از آخر به اول بنویسیم. پس داخل یک حلقه، تقسیمات متوالی بر 2 کرده و باقیمانده‌ها را به ترتیب در خانه‌های آرایه قرار می‌دهیم. پس از اتمام تقسیمات، باید مقادیر آرایه را از آخر به اول نمایش دهیم.



```
#include <iostream>
using namespace std;
int main()
{
    short int Bin[100], i, N;
    cout<<"Lotfan Adad ra Vared Konid: ";
    cin>>N;

    for (i = 0 ; N > 0 ; i++)          // حلقه‌ی تقسیمات متوالی بر 2
    {
        Bin[i] = N%2;
        N /= 2;
    }

    cout<<" Binary: ";
    for (--i ; i >= 0 ; i--)          // نمایش خانه‌های آرایه از آخر به اول
        cout<< Bin[i];

    return(0);
}
```

تمرین 33: الف) فلوچارت این برنامه را رسم کنید. ب) این برنامه را در کامپیوتر اجرا کرده، خروجی برنامه را برای ورودی‌های مختلف، بنویسید.

برنامه 34: برنامه‌ای بنویسید که رشته‌ای را گرفته تشخیص دهد که آیا پالیندروم است یا نه؟ این برنامه همیشه در حال اجرا بوده و با زدن عدد 0 خاتمه یابد.

حل: به رشته‌ای پالیندروم (Palindrom) گفته می‌شود که از چپ و راست به یک شکل خوانده می‌شود. مثلاً رشته‌های زیر پالیندروم هستند: ... , toot , 123454321 , damad

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int i, j;
    bool ispalindrom;
    string str;
    while (1) // حلقه‌ی بی‌نهایت
    {
        cout << endl << "Please enter a STRING or <0> to Exit: ";
        cin >> str;
        if (str == "0") return(0); // خروج از حلقه‌ی بی‌نهایت
        ispalindrom = true;
        for (i = 0, j = str.length() - 1; i < j; i++, j--)
            if (str[i] != str[j])
                { cout << " No Palindrom! "; ispalindrom = false; break; }

        if(ispalindrom) cout << " Is Palindrom ";
    }
    return(0);
}
```

نکته: برای محاسبه طول یک رشته می‌توان از متد `length()` استفاده کرد.

تمرین 34: این برنامه را در کامپیوتر اجرا کرده، خروجی برنامه را برای ورودی‌های مختلف، بررسی کنید.

آرایه‌های دو بعدی:

به آرایه‌ای گفته می‌شود که هم سطر دارد و هم ستون. مانند صفحه کاغذ، عکس، مانیتور، گوشی، دیوار، ماتریس و ...

مثال 1:

```
int A[2][3];
```

0	20		ورودی
1		خروجی	
	0	1	2

```
A[0][0]=20;      cin>>A[0][2];      cout<<A[1][1];      ...
```

مثال 2:

```
double B[4][2]={ {1.5 , 2} , {5.2 , 3.3} , {19.75} , {0,0} };
```

```
for(i=0 ; i<4 ; i++)
```

```
{
```

```
for(j=0 ; j<2 ; j++)
```

```
cout<<B[ i ][ j ]<<"\t";    // نمایش محتوای آرایه دو بعدی به شکل مناسب
```

```
cout<<endl;
```

```
}
```

	0	1
0	1.5	2.0
1	5.2	3.3
2	19.75	نامعلوم
3	0.0	0.0

نکته: اگر به یک متغیر یا آرایه مقداردهی اولیه نشود آنگاه دو حالت داریم:

الف: سراسری (global): چنانچه بالای main() تعریف شود به صورت خودکار مقدار اولیه آنها 0 می‌شود.

ب: محلی (local): چنانچه در داخل main() یا یک زیربرنامه تعریف شود مقدار اولیه آنها نامعلوم (شائسی) می‌شود.

توجه: از متغیرهای محلی فقط در آن محل (فقط داخل آن زیربرنامه) می‌توان استفاده کرد، ولی از متغیرهای سراسری می‌توان در همه جای برنامه و بصورت مشترک در داخل تمام زیربرنامه‌های یک برنامه استفاده کرد.

برنامه 35: برنامه‌ای بنویسید که مقادیر دو ماتریس 3×4 را از کاربر بگیرد، سپس مجموع این دو ماتریس را محاسبه کرده و نتیجه را به شکل مناسب در خروجی نشان دهد.

```
#include <iostream>
using namespace std;
int main()
{
    int A[3][4] , B[3][4] , Sum[3][4] , i , j;
    cout<<"Array A[3][4] ra Vared konid:" <<endl;
    for (i =0 ; i < 3 ; i++)
        for (j =0 ; j < 4 ; j++)
            cin>>A[i][j]; // حلقه‌ی گرفتن مقادیر آرایه اول
    cout<<"Array B[3][4] ra Vared konid:" <<endl;
    for (i =0 ; i < 3 ; i++)
        for (j =0 ; j < 4 ; j++)
            cin>>B[i][j]; // حلقه‌ی گرفتن مقادیر آرایه دوم
    for (i =0 ; i < 3 ; i++)
        for (j =0 ; j < 4 ; j++)
            Sum[i][j] = A[i][j] + B[i][j]; // حلقه‌ی محاسبه مجموع دو آرایه
    cout<<"Array Sum[3][4]:"<<endl;
    for (i =0 ; i < 3 ; i++)
    {
        for (j =0 ; j < 4 ; j++)
            cout<<Sum[i][j]<<"\t"; // حلقه‌ی نمایش مجموع دو آرایه
        cout<<endl;
    }
    return(0);
}
```

تمرین 35: این برنامه را در کامپیوتر اجرا کرده و برای ورودی‌های مختلف، خروجی را بررسی کنید.

برنامه 36: برنامه‌ای بنویسید که مقادیر یک ماتریس 2×3 را از ورودی بگیرد و در خروجی ترانپاده‌ی (Transpose) آنرا به شکل مناسب نشان دهد.

حل: منظور از ترانپاده این است که جای سطر و ستون را عوض کنیم. برای این کار، مقادیر آن ماتریس را در یک آرایه‌ی دوبعدی قرار داده سپس با حلقه‌های تودرتو، جای سطر و ستون را عوض می‌کنیم.

```
#include <iostream>
using namespace std;
int main()
{
    int A[2][3], T[3][2], i, j;
    cout<<"Array A[2][3] ra Vared konid:" <<endl;
    for (i = 0 ; i < 2 ; i++)
        for (j = 0 ; j < 3 ; j++)
            cin>>A[i][j]; // حلقه‌ی گرفتن مقادیر آرایه اصلی

    for (i = 0 ; i < 3 ; i++)
        for (j = 0 ; j < 2 ; j++)
            T[i][j] = A[j][i]; // حلقه‌ی محاسبه ترانپاده

    cout<<"Transpose Array[3][2]:"<<endl;
    for (i = 0 ; i < 3 ; i++)
    {
        for (j = 0 ; j < 2 ; j++)
            cout<<T[i][j]<<"\t"; // حلقه‌ی نمایش ترانپاده

        cout<<endl;
    }
    return(0);
}
```

تمرین 36: این برنامه را در کامپیوتر اجرا کرده و برای ورودی‌های مختلف، خروجی را بررسی کنید.

برنامه 37 (تمرین 37): برنامه‌ای بنویسید و اجرا کنید که یک ماتریس 3×4 و یک ماتریس 4×5 با مقادیر دلخواه تعریف کند. این برنامه، حاصلضرب این دو ماتریس را محاسبه کرده و به شکل مناسب نشان دهد.

BY: MORTAZA ABBASZADEH

آرایه‌های چند بعدی:

آرایه تک بعدی <<< ساخته شده است از <<< در کنار هم قرار گرفتن چند متغیر یا خانه‌ی ممنوع و هم‌اندازه
آرایه دو بعدی <<< ساخته شده است از <<< در کنار هم قرار گرفتن چند آرایه یک بعدی ممنوع و هم‌اندازه
آرایه سه بعدی <<< ساخته شده است از <<< در کنار هم قرار گرفتن چند آرایه دو بعدی ممنوع و هم‌اندازه
آرایه چهار بعدی <<< ساخته شده است از <<< در کنار هم قرار گرفتن چند آرایه سه بعدی ممنوع و هم‌اندازه
آرایه پنج بعدی <<< ساخته شده است از <<< در کنار هم قرار گرفتن چند آرایه چهار بعدی ممنوع و هم‌اندازه

•
•
•

آرایه N بعدی <<< ساخته شده است از <<< در کنار هم قرار گرفتن چند آرایه N-1 بعدی ممنوع و هم‌اندازه

مثال ساده برای تجسم کردن: قوطی‌های کبریت داخل بسته، جعبه، ...

فصل 5: زیربرنامه‌ها (توابع Function)

تعداد سطرهای یک برنامه هرچقدر بیشتر و بیشتر شود (برنامه بزرگ)، دیگر درک و فهم آن و ردیابی و اشکال‌زدایی آن راحت نخواهد بود. چنین برنامه‌هایی به سختی توسعه یافته و بروزرسانی می‌شوند. بهتر است که یک برنامه‌ی بزرگ، به چندین زیربرنامه‌ی کوچک و قابل فهم تقسیم شود. استفاده از زیربرنامه‌ها مزایای زیادی دارد که برخی از آنها عبارتند از:

- ✓ کار تیمی چند برنامه‌نویس بر روی یک پروژه و تهیه‌ی آن در حداقل زمان،
- ✓ درک و فهم برنامه و الگوریتم آن راحت‌تر،
- ✓ اشکال‌یابی و اشکال‌زدایی ساده‌تر،
- ✓ هزینه‌ی کمتر در بحث نگهداری برنامه،
- ✓ قابلیت توسعه‌ی بهتر برنامه و بروزرسانی ساده‌تر،
- ✓ حجم کدنویسی کمتر می‌شود.

در هر برنامه‌ای که بخواهیم از زیربرنامه‌ها (توابع فرعی) استفاده کنیم، باید 3-مرحله زیر را انجام دهیم:

مرحله 1: اعلان زیربرنامه (Declaration): این بخش باید بالای main نوشته شود تا برنامه بتواند زیربرنامه‌ها را بشناسد. در این بخش، نوع خروجی زیربرنامه، اسم زیربرنامه، و پارامتر (Parameter)‌های ورودی زیربرنامه، مشخص و اعلان می‌شود:

نوع خروجی برگشتی زیربرنامه	اسم زیربرنامه	؛ (پارامترهای ورودی زیربرنامه)
----------------------------	---------------	--------------------------------

نکته: اگر زیربرنامه‌ای، خروجی برگشتی نداشته باشد باید void بنویسیم، و اگر چیزی بنویسیم معمولاً معادل int در نظر گرفته می‌شود. ولی اگر زیربرنامه‌ای پارامتر ورودی نداشته باشد، میتوان خالی گذاشت یا void نوشت. در بحث زیربرنامه‌ها، حداکثر تعداد برگشتی برابر یک مقدار است. ولی تعداد پارامترهای ورودی به دلخواه برنامه‌نویس است. دستور برگشت دادن خروجی، return(...) است، که در داخل پرانتز بجای ... ، مقدار برگشتی نوشته می‌شود. البته میتوان آنرا خالی نیز استفاده کرد return; که در اینصورت فقط از آن زیربرنامه خارج می‌شود.

با توجه به قالب فوق، می‌توان 4-نوع زیربرنامه تعریف کرد:

(الف) زیربرنامه‌ای که پارامتر ورودی دارد و به فراخواننده مقداری برگشت می‌دهد. مثال: float Average(int,int);

(ب) زیربرنامه‌ای که پارامتر ورودی دارد ولی به فراخواننده مقداری برگشت نمی‌دهد. مثال: void Sum(int,int,int);

(ج) زیربرنامه‌ای که پارامتر ورودی ندارد ولی به فراخواننده مقداری برگشت می‌دهد. مثال: double Random();

(د) زیربرنامه‌ای که پارامتر ورودی ندارد و به فراخواننده مقداری برگشت نمی‌دهد. مثال: void ClearScreen();

مرحله 2: فراخوانی زیربرنامه (صدازدن یا Call): پس از اعلان یک زیربرنامه (مرحله 1)، میتوان یک یا چندین بار آن زیربرنامه را جهت اجرا شدن، فراخوانی کرد. برای این کار نوشتن اسم زیربرنامه و ارسال آرگومانها به عنوان پارامترهای ورودی زیربرنامه الزامی است. البته باید ترتیب، تعداد و نوع پارامترها رعایت شود. اگر آن زیربرنامه، مقدار برگشتی داشته باشد باید در محل فراخوانی، متغیری برای قرار دادن خروجی برگشتی داخل آن تدارک دیده و یا مستقیماً آن را چاپ کرد. مثال:

```
int Sum(int,int);           // مرحله 1: اعلان زیربرنامه
int main()
{
...
A = Sum(10 , 20);          // مرحله 2: فراخوانی و ارسال آرگومانها
...
return (0);
}
...
```

دقت: این برنامه هنوز ناقص است چون مرحله ی 3 آن هنوز نوشته نشده است.

نکته: به اعداد و مقادیر ارسالی در مرحله ی فراخوانی (مرحله 2) آرگومان (Argument) گفته می شود.

انواع فراخوانی:

1- فراخوانی با مقدار (call by value)

2- فراخوانی با ارجاع (call by reference)

در فراخوانی بامقدار، خود متغیرهای اصلی به زیربرنامه ارسال نمی شود بلکه کپی آنها به زیربرنامه ارسال می شود. یعنی زیربرنامه هر تغییری در پارامترهای دریافتی خود داشته باشد، مقدار متغیرهای اصلی تغییر نمی یابد. ولی در فراخوانی باارجاع، آدرس متغیرها با استفاده از عملگر &، به عنوان آرگومان به زیربرنامه ارسال می شود، یعنی زیربرنامه می تواند با مراجعه به آن آدرس، به محتوای متغیرهای اصلی دسترسی داشته و یا آنها را دستکاری کند.

نکته: در هر بار عمل فراخوانی، و برگشتن از زیربرنامه ی فراخوانی شده، وقفه (تاخیر) ایجاد می شود و سرعت اجرای کل برنامه کاهش می یابد. برای حل این مشکل می توان از توابع inline یا ماکروها (macro) استفاده کرد.

مرحله 3: نوشتن محتوای خود زیربرنامه (Function Definition): پس از اعلان زیربرنامه در مرحله 1، و نیز فراخوانی آن در مرحله 2، باید محتوای خود زیربرنامه را نوشت. معمولاً خود زیربرنامه در پایین و پس از اتمام main نوشته می‌شود. اگر یک زیربرنامه پارامتر ورودی داشته باشد باید تعداد، ترتیب و نوع آنها مطابق اعلان آن زیربرنامه در مرحله 1 بوده و برای هر پارامتر یک اسم نیز تعیین کرد تا مقادیر آرگومانهای ارسالی حین فراخوانی، در داخل این پارامترها کپی شوند. معمولاً مقادیر داخل زیربرنامه، کپی مقادیر ارسالی اصلی می‌باشند (فراخوانی با مقدار).

```
int Sum(int,int);           // مرحله 1: اعلان زیربرنامه

int main()
{
    int M , A=10 , B=5;
    M = Sum(A ,B);          // مرحله 2: فراخوانی زیربرنامه (با مقدار)
    cout<<"Majmoo: "<<M;
    return(0);
}

int Sum(int x, int y)       // مرحله 3: خود زیربرنامه
{
    return(x+y);
}
```

نکته: در واقع مقدار پارامتر x از روی آرگومان A، و مقدار پارامتر y از روی آرگومان B کپی می‌شود.

نکته: می‌توان اعلان یک زیربرنامه (مرحله 1) را با خود زیربرنامه (مرحله 3) ترکیب کرده و باهم نوشت. در اینصورت باید بالای main نوشته شوند تا زیربرنامه‌ی اصلی main بتواند آنرا بشناسد. مثال:

```
int Sum(int x , int y) { return(x+y); } // مرحله 1 و 3: اعلان زیربرنامه و خود آن

int main()
{ int  A=10 , B=5;
  cout<<"Majmoo: "<< Sum(A ,B);          // مرحله 2: فراخوانی زیربرنامه (با مقدار)
  return(0);
}
```

نکته: تمام زیربرنامه‌ها در داخل main() اجرا می‌شوند. یعنی همیشه اولین و آخرین سطر اجرا شده در تمام برنامه‌ها، مربوط به main() است.

برنامه 38: برنامه‌ای بنویسید که زیربرنامه‌ی `main()` سه عدد صحیح را از ورودی گرفته و آنها را به زیربرنامه‌ی فرعی `Avg()` بفرستد. زیربرنامه‌ی `Avg()` میانگین آن سه عدد را پیدا کرده و برگشت دهد. میانگین را `main()` نشان خواهد داد.

```
#include <iostream>
using namespace std;
float Avg(int,int,int);           // مرحله 1: اعلان زیربرنامه

int main()
{
    int A,B,C;
    cout<<"Lotfan 3 Adad vared konid: ";
    cin>>A>>B>>C;
    cout<<"Meiangingin: "<< Avg(A ,B,C);    // مرحله 2: فراخوانی زیربرنامه (با مقدار)
    return(0);
}

float Avg(int x, int y, int z )    // مرحله 3: خود زیربرنامه
{
    return( (x+y+z)/3.0 );
}
```

نکته: پارامترهای `x,y,z` کپی آرگومانهای `A,B,C` هستند.

نکته: اگر در مخرج بجای `3.0`، فقط `3` نوشته می‌شد، آنگاه تقسیم بصورت صحیح عمل می‌کرد نه اعشاری.

تمرین 38: این برنامه را در کامپیوتر اجرا کرده و برای ورودی‌های مختلف، خروجی را بررسی کنید.

برنامه 39: برنامه‌ای بنویسید که تابع اصلی `main()` عدد صحیح (و شاید طولانی) را از ورودی گرفته و به تابع فرعی `zofjard()` ارسال کند. این تابع فرعی بررسی می‌کند اگر عدد زوج بود، `true` و اگر فرد بود، `false` برگشت می‌دهد. سپس `main()` یکی از توابع `zofj()` یا `fard()` را فراخوانی و آن توابع فرعی نیز پیغام مناسب مبنی بر زوج یا فرد بودن، نشان می‌دهند.

```
#include <iostream>
using namespace std;
bool zofjard(long int);
void zofj(); // مرحله 1: اعلان زیربرنامه‌ها
void fard();
int main()
{
    long int A;
    cin>>A;
    if(zofjard(A) == true) zofj(); // مرحله 2: فراخوانی زیربرنامه‌ها (با مقدار)
    else fard();
    return(0);
}
bool zofjard(long int A) // مرحله 3: خود زیربرنامه‌ها
{
    if(A%2==0) return true;
    else return false;
}
void zofj()
{ cout<<"Zoj Ast. "; }
void fard()
{ cout<<"Fard Ast. "; }
```

نکته: دستور شرطی را می‌توان اینچنین نیز نوشت: `if(zofjard(A))...`

نکته: در زبان C++ عدد 1 معادل `true` و عدد 0 معادل `false` است.

نکته: متغیرهای `A` در `main()` و `zofjard()` هرچند همنام هستند ولی هر کدام محلی بوده و مستقل از هم هستند.

تمرین 39: این برنامه را در کامپیوتر اجرا کرده و برای ورودی‌های مختلف، خروجی را بررسی کنید.

برنامه 40: برنامه‌ای بنویسید که زیربرنامه اصلی `main()` یک عدد صحیح نامنفی (و شاید طولانی) را از ورودی گرفته و به زیربرنامه فرعی `Varun()` ارسال کند. این زیربرنامه، وارون آن عدد را پیدا کرده، نشان داده و به زیربرنامه `Magsum()` ارسال کند. این زیربرنامه هم مقسوم‌علیه‌های وارون را پیدا کرده و خودش نشان دهد.

```
#include <iostream>
using namespace std;
void Varun(unsigned long int);
void Magsum(unsigned long int); // مرحله 1: اعلان زیربرنامه‌ها

int main()
{
    unsigned long int A;
    cout<<"Lotfan Adadi NaManfi vared konid: ";
    cin>>A;
    Varun(A); // مرحله 2: فراخوانی زیربرنامه (با مقدار)
    return(0);
}

void Varun(unsigned long int N) // مرحله 3: خود زیربرنامه‌ها
{
    unsigned long int Var;
    for(Var=0 ; N>0 ; N/=10) Var = Var*10 + N%10;
    cout<<"Varun e Adad:"<<Var<<endl;
    Magsum(Var); // فراخوانی زیربرنامه در داخل یک زیربرنامه‌ی دیگر
}

void Magsum(unsigned long int var)
{
    unsigned long int i;
    cout<<"Magsum Aleihaye Varun:";
    for(i=1 ; i<=var ; i++) if( var % i == 0 ) cout<<i<<" , ";
}
```

تمرین 40: این برنامه را در کامپیوتر اجرا کرده و برای ورودی‌های مختلف، خروجی بگیرید.

برنامه 41: برنامه‌ای بنویسید که زیربرنامه اصلی `main()` مقادیر یک آرایه‌ای 5-خانه‌ای را از ورودی گرفته و آن آرایه را به زیربرنامه فرعی `FindMaxMin()` ارسال کند. این زیربرنامه، بزرگترین و کوچکترین عنصر آرایه را پیدا کرده، نشان دهد.

حل: از فراخوانی با ارجاع استفاده کرده و نام آرایه را که اشاره‌گر به اولین خانه‌ی آرایه می‌باشد، به زیربرنامه ارسال می‌کنیم.

```
#include <iostream>
using namespace std;
void FindMaxMin( int[ ] ); // مرحله 1: اعلان زیربرنامه

int main()
{   int Ar[5] , i;
    cout<<"Lotfan 5 adad vared konid: ";
    for(i=0 ; i<5 ; i++) cin>>Ar[i];
    FindMaxMin(Ar); // مرحله 2: فراخوانی زیربرنامه (با ارجاع)
    return(0);
}

void FindMaxMin(int *B) // مرحله 3: خود زیربرنامه
{   int Max , Min , i;
    Max = Min = B[0];
    for(i=1 ; i<5 ; i++)
        { if(B[i]>Max) Max = B[i];
          else if(B[i]<Min) Min = B[i]; }
    cout<<" Maximum: "<<Max<<endl;
    cout<<" Minimum: "<<Min<<endl;
}
```

نکته: نام آرایه بعنوان آرگومان به زیربرنامه ارسال می‌شود. نام آرایه در واقع آدرس اولین خانه از آرایه می‌باشد. لذا از فراخوانی با ارجاع استفاده می‌شود. یعنی زیربرنامه فرعی نیز با استفاده از اشاره‌گر B به آرایه اصلی در داخل `main()` دسترسی دارد.

تمرین 41: این برنامه را در کامپیوتر اجرا کرده و برای ورودی‌های مختلف، خروجی بگیرید.

توابع بازگشتی (Recursive Function):

به آن دسته از برنامه‌ها یا زیربرنامه‌هایی که در داخل خود، خودشان را فراخوانی می‌کنند، بازگشتی گفته می‌شود.

برنامه‌هایی که دارای حلقه هستند را می‌توان به دو روش حل کرد:

1. روش غیربازگشتی: برای ایجاد حلقه، از این دستورات استفاده می‌شود: `while` , `for` , `do...while` , `goto`

2. روش بازگشتی: یک زیربرنامه، با یک فرمول خاص، خودش را فراخوانی کرده و باعث ایجاد حلقه می‌شود.

روش بازگشتی، نوعی روش حل برنامه‌های کامپیوتری می‌باشد که یک مساله‌ی بزرگ را رفته رفته به زیرمساله‌های کوچک و قابل حل تقسیم کرده و هنگام بازگشت، آن مساله بزرگ را کم کم حل می‌کند. البته فقط آندسته از برنامه‌ها را می‌توان بازگشتی نوشت که دو شرط زیر را داشته باشند:

1. دارای حلقه‌ی تکرار باشد،

2. نتیجه هر مرحله را بتوان از روی مراحل قبلی محاسبه کرد.

تمام برنامه‌های بازگشتی شامل دو بخش زیر هستند:

1. بخش پایان بازگشتی: در این بخش معمولاً شرطی گذاشته می‌شود تا روال فراخوانی بازگشتی پایان یابد. یعنی اگر این بخش نباشد، حلقه‌ی بی‌نهایت ایجاد می‌شود.

2. بخش فرمول بازگشتی: در این بخش معمولاً با یک فرمول یا روش خاص، آن زیربرنامه، خودش را فراخوانی می‌کند و در هر بار فراخوانی سعی می‌کند قسمتی از مساله را حل نماید. این قسمت باعث ایجاد حلقه می‌شود.

مزایای روش بازگشتی: الف) حجم کد نویسی (تعداد سطرهایی که برنامه‌نویس می‌نویسد) خیلی کم است. ب) برخی از مسائل پیچیده، اولین بار با روش بازگشتی حل شده‌اند.

معایب روش بازگشتی: الف) فهمیدن، ردیابی کردن و پیدا کردن فرمول بازگشتی برخی مسائل، سخت است. ب) سرعت اجرای برنامه‌ها در حالت بازگشتی خیلی کم است. (به دلیل فراخوانی‌های زیاد) ج) مصرف حافظه‌ی برنامه‌های بازگشتی خیلی زیاد است. (به دلیل `push()` و `pop()` کردن‌های زیاد در حافظه)

نکته: در سیستم‌هایی که محدودیت حافظه دارند، یا سرعت پردازنده‌ی آنها پایین است، نباید از روش بازگشتی استفاده کرد.

برنامه 42: برنامه‌ای بنویسید که زیربرنامه‌ی اصلی `main()` عددی نامنفی و کوتاه را از ورودی گرفته و آنرا به دو زیربرنامه‌ی فرعی بفرستد. زیربرنامه‌ی اول به روش غیربازگشتی، و زیربرنامه‌ی دوم به روش بازگشتی، فاکتوریل آن عدد را پیدا کرده و جهت نمایش به `main()` برگشت دهند.

```
#include <iostream>
using namespace std;
unsigned long int Fact1(unsigned short int);           // مرحله 1: اعلان زیربرنامه‌ها
unsigned long int Fact2(unsigned short int);
int main()
{
    unsigned short int A;
    cout<<"Lotfan Adadi NaManfi vared konid: ";
    cin>>A;
    cout<<"Factoriel in Not-Recursive: "<<Fact1(A);      // مرحله 2: فراخوانی زیربرنامه‌ها (با مقدار)
    cout<<endl<<"Factoriel in Recursive: "<<Fact2(A);
    return(0);
}
unsigned long int Fact1(unsigned short int N)           // مرحله 3: خود زیربرنامه‌ها
{
    unsigned long int F=1;
    do{ F*=N; N--; }while(N>1);
    return(F);
}
unsigned long int Fact2(unsigned short int N)
{
    if(N==0) return(1);
    else return( N * Fact2(N-1) );
}
```

نکته: زیربرنامه‌ی `Fact2()`، در داخل خود، مجدد `Fact2()` را فراخوانی کرده است. پس بازگشتی بوده و حلقه‌ی تکرار دارد.

تمرین 42: این برنامه را در کامپیوتر اجرا کرده و برای اعدادی مثل 4, 5, 6, خروجی بگیرید.

برنامه 43: برنامه‌ای بنویسید که با استفاده از یک زیربرنامه، جمله‌ی N ام دنباله‌ی فیبوناچی را به روش بازگشتی محاسبه کرده و نشان دهد.

```
#include <iostream>
using namespace std;
unsigned long int Fib(unsigned short int); // مرحله 1: اعلان زیربرنامه
int main()
{
    unsigned short int A;
    cout<<"Lotfan Adadi NaManfi vared konid: ";
    cin>>A;
    cout<<"Jomle N om Fibonachi (Bazgashti): "<<Fib(A); // مرحله 2: فراخوانی زیربرنامه (با مقدار)
    return(0);
}
unsigned long int Fib(unsigned short int N) // مرحله 3: خود زیربرنامه
{
    if(N<=2) return(N-1);
    else return( Fib(N-1) + Fib(N-2) );
}
```

نکته: زیربرنامه‌ی $Fib()$ ، در داخل خودش، دو بار $Fib()$ را فراخوانی کرده است. پس بازگشتی بوده و حلقه‌ی تکرار دارد.

تمرین 43: این برنامه را در کامپیوتر اجرا کرده و برای ورودی‌های مختلف مانند $N=6$ ، $N=10$ ، $N=20$ ، خروجی بگیرید.

فصل ششم: اشاره گرها (Pointer)

اگر متغیری مانند A داشته باشیم که در داخل آن آدرس متغیر دیگری مانند B باشد، آنگاه A را اشاره گر به B می نامیم. در تعریف اشاره گر و دسترسی به محتوای جاییکه اشاره می کند، از عملگر * استفاده می شود. برای آدرس دهی نیز از عملگر & استفاده می شود. مثال:

```
int B=20;
```

```
int *A;           // تعریف اشاره گری به اسم A، که قرار است به یک عدد صحیح اشاره کند:
```

```
...
```

```
A = &B;          // آدرس متغیر B را در داخل اشاره گر A قرار بده (یعنی A به B اشاره بکند):
```

```
B++; یا
```

```
*A++;           // به محتوای جاییکه A به آن اشاره می کند (B)، یک واحد اضافه کن:
```

نکته:

- ✓ متغیر از هر نوعی که باشد، اشاره گرش نیز باید از همان نوع تعریف شود.
- ✓ هر اشاره گر، از هر نوعی که باشد فقط 4 بایت فضا مصرف می کند، چون داخل آن فقط آدرس ذخیره می شود. آدرس های حافظه معمولاً در قالب شانزده-شانزدهی هستند (Hexadecimal)، که هنگام نمایش یک آدرس، ابتدای آنها با 0x شروع، و با حرف H پایان می یابد. مانند: 0x FFFF 101E H
- ✓ در داخل حافظه هر اشاره گر با استفاده از عملگرهایی مثل ++ یا --، می تواند به اندازه ی نوع خود، به خانه های جلوتر یا عقب تر حافظه حرکت کرده و اشاره بکند. (مثلاً با اجرای A++ در مثال بالا، اشاره گر A، به 4 بایت بعد از متغیر B، در حافظه، اشاره خواهد کرد).
- ✓ اشاره گرها، قدرت و انعطاف زیاد و گاهی خطرناکی به برنامه نویسان می دهد. برنامه نویس می تواند با تعریف یک اشاره گر، به هر نقطه از حافظه دسترسی داشته باشد. شاید به خاطر همین موضوع باشد که در زبانهایی مثل Python, Java و ...، دیگر خبری از اشاره گرها نیست! البته در C++ شیء گرا (Object Oriented C++) می توان با استفاده از کلاس ها (class)، مجوز دسترسی به داده های برنامه را تعریف و محدود کرد.
- ✓ اشاره گرها در نوشتن برنامه های سیستمی مانند تولید یک زبان برنامه نویسی، سیستم عامل، هک کردن و ... کاربرد دارند.

برنامه 44: برنامه‌ای بنویسید که با استفاده از فقط دو اشاره‌گر، محتوای دو متغیر A , B را عوض کند.

```
#include <iostream>
using namespace std;
int main()
{
    int A=100 , B=200;
    int *X,*Y;
    X = &A;
    Y = &B;
    cout<<" A (befor)= "<< *X << endl;
    cout<<" B (befor)= "<< *Y << endl;
    *X = *X + *Y;
    *Y = *X - *Y;
    *X = *X - *Y;
    cout<<" A (after)= "<< *X << endl;
    cout<<" B (after)= "<< *Y << endl;
    return(0);
}
```

تمرین 44: این برنامه را در کامپیوتر اجرا کرده و خروجی را بررسی کنید.

برنامه 45: برنامه‌ای بنویسید که زیربرنامه اصلی `main()` یک آرایه‌ی 20 خانه‌ای با مقادیر دلخواه تعریف کند. سپس عددی را از کاربر گرفته و با استفاده از یک زیربرنامه، آن عدد را داخل آرایه جستجو کند. اگر پیدا کرد، اندیس خانه‌ی مورد نظر را برگشت دهد، و اگر پیدا نکرد، عدد -1 را برگرداند.

حل: از فراخوانی با ارجاع استفاده کرده و نام آرایه را که اشاره‌گر به اولین خانه‌ی آرایه می‌باشد، به زیربرنامه ارسال می‌کنیم.

```
#include <iostream>
using namespace std;
int LinearSearch( int , int[ ] ); // مرحله 1: اعلان زیربرنامه

int main()
{
    int N , Result , Ar[20]={9,8,7,6,5,4,3,2,1,0,10,20,30,40,50,60,70,80,90,555};
    cout<<"Lotfan adadi vared konid: ";
    cin>> N;
    Result = LinearSearch ( N , Ar); // مرحله 2: فراخوانی زیربرنامه (با ارجاع)
    if(Result == -1) cout<<"Not found in Array! ";
    else cout<<"Found from index [ "<<Result<<" ] in Array. ";
    return(0);
}

int LinearSearch( int N, int *B ) // مرحله 3: خود زیربرنامه
{
    int i;
    for(i=0 ; i<20 ; i++)
        if ( *(B+i) == N) return(i);
    return(-1);
}
```

نکته: نام آرایه `(Ar)`، اشاره‌گر به اولین خانه‌ی آرایه می‌باشد. اشاره‌گر `B` در داخل زیربرنامه نیز، به ابتدای آرایه اشاره می‌کند. می‌توان بجای `*(B+i)`، اینچنین نیز نوشت: `B[i]`

تمرین 45: این برنامه را در کامپیوتر اجرا کرده و برای ورودی‌های مختلف، خروجی بگیرید.

اشاره گر به اشاره گر: اگر اشاره گری مانند A داشته باشیم که به B اشاره کند و خود B نیز اشاره گری باشد که به متغیری مانند C اشاره بکند، آنگاه A را اشاره گر به اشاره گر می گوییم. در تعریف آنها از ** استفاده می شود. مثال:

```
int C = 20;

int *B;           // تعریف اشاره گری به اسم B، که قرار است به یک عدد صحیح اشاره کند:

int **A;          // تعریف اشاره گر به اشاره گری به اسم A، که قرار است به یک اشاره گر به عدد صحیح اشاره کند:

...

B = &C;           // آدرس متغیر C را در داخل متغیر B قرار بده (یعنی B به C اشاره بکند):

A = &B;           // آدرس اشاره گر B را در داخل اشاره گر A قرار بده (یعنی A به B اشاره بکند):

C++;              یا

*B++;             یا // به محتوای جاییکه B به آن اشاره می کند (C)، یک واحد اضافه کن:

**A++;            // به محتوای جاییکه A به آن اشاره گر به اشاره گر است (C)، یک واحد اضافه کن:
```

برنامه 46: برنامه ای بنویسید که با استفاده از مفهوم اشاره گر به اشاره گر، داده ای را دستکاری و نشان دهد.

```
#include <iostream>
using namespace std;
int main()
{ long A=1000000, *X = &A, **Y = &X;
  cout<<" A (befor)= "<< **Y << endl;
  A ++;
  (*X) ++;
  (**Y) ++;
  cout<<" A (after)= "<< **Y << endl;
  return(0);
}
```

تمرین 46: این برنامه را در کامپیوتر اجرا کرده و خروجی را بررسی کنید.

فصل هفتم: فایل ها (File)

همانگونه که می دانیم، تمام برنامه ها، در حافظه ی اصلی (Ram) اجرا می شوند. یعنی با قطع جریان برق و یا خاموش شدن کامپیوتر، تمام داده های برنامه از بین می رود. در بعضی از برنامه ها لازم است تا یکسری نتایج یا داده های آن برنامه، در داخل یک رسانه ی ذخیره سازی دائمی مثل دیسک سخت (هارد)، فلش و ... ذخیره شوند تا بتوان بعداً نیز از آنها استفاده کرد. در اینصورت باید اطلاعات موجود در حافظه ی اصلی را به مثلاً هارد انتقال داد، که به این عمل ذخیره سازی در فایل گفته می شود. یعنی، با خاموش شدن کامپیوتر، اطلاعات داخل فایل از بین نخواهد رفت، و هر وقت که لازم شد، برنامه می تواند به آن فایل دسترسی داشته و داده های موردنظر خود را دوباره از آن فایل به حافظه ی اصلی بارگذاری کرده و از آن داده ها استفاده کند.

انواع فایل:

1. **فایل متنی:** به فایلی گفته می شود که داده ها در داخل آن بصورت کاراکتری و با کد اسکی ذخیره می شود تا بعداً نیز بتوان آن فایل را برای ویرایش، باز کرد و استفاده کرد. محتوای فایل متنی، برای کاربر، قابل مفهوم است.
2. **فایل دودویی:** به فایلی گفته می شود که داده ها در داخل آن بصورت عددی و با کد ماشین ذخیره می شود. یعنی اگر کاربر بخواهد داخل چنین فایلهایی را با یک ویرایشگر باز کند و نگاه کند، یکسری علائم نامفهوم دیده خواهد شد. البته سرعت فایل های دودویی بیشتر از فایل متنی است، همچنین حجم فایلهای دودویی کمتر از فایل های متنی معادل آن است.

عملیات روی فایل ها: در فایل ها عملیات زیادی می توان انجام داد که برخی از آنها عبارتند از:

- ایجاد فایل (Create): فایلی با نام و ویژگی های مشخص شده، در یک مسیر مشخص یا جاری ایجاد می شود.
- بازکردن فایل (Open): فایلی را برای استفاده کردن، باز می کنیم.
- حذف فایل (Delete): فایل موردنظر با داده های داخل آن از بین می رود.
- جستجو در فایل (Search): داده های داخل فایل، مورد پیمایش و جستجو قرار می گیرد.
- ذخیره در فایل (Save): یکسری داده ها را داخل فایل ذخیره می کنیم.
- بارگذاری از فایل (Load): از داخل فایل یکسری داده ها را با حافظه ی اصلی می آوریم.
- ویرایش در فایل (Edit): یکسری داده های داخل فایل را تغییر می دهیم.
- اضافه به انتهای فایل (Append): اطلاعات قبلی فایل دست نخورده مانده و داده های جدید به هر بار فقط به انتهای فایل اضافه می شود. (EOF)
- بستن فایل (Close): هر فایلی که باز شده است، نهایتاً باید بسته شود تا اولاً عمل تخلیه ی بافر هنگام ذخیره سازی بصورت کامل انجام شود و دوماً سایر برنامه ها بتوانند از آن فایل آزاد شده، استفاده کنند.

هنگام استفاده از فایل‌ها در زبان C++ باید سرآیند `#include <fstream>` نیز با ابتدای برنامه اضافه شود. ضمناً یکی از سه حالت زیر در استفاده از فایل‌ها هنگام ایجاد فایل، خواندن از فایل و نوشتن (ذخیره) در فایل، باید مشخص گردد:

- `ofstream`: ایجاد فایل و ذخیره کردن در فایل (فقط نوشتنی)
- `ifstream`: خواندن از فایل (فقط خواندنی)
- `fstream`: ترکیب هر دو: ایجاد، خواندن و نوشتن در فایل.

نکته: هنگام باز کردن یک فایل، باید یکی از 3 حالت بالا را مشخص کنیم.

نکته: از عملگر درج (`<<`) می‌توان جهت نوشتن در فایل استفاده کرد.

نکته: برای استفاده از هر فایل، باید یک اشاره‌گر به فایل داشته باشیم. (مانند `myfile` در برنامه‌ی بعدی)

نکته: اگر برای ایجاد فایل از `ofstream` استفاده شود و فایل از قبل وجود داشته باشد، محتوای فایل قبلی پاک می‌شود.

نکته: در داخل فایلی که از نوع `ifstream` باز شده باشد، نمی‌توان چیزی نوشت.

نکته: از داخل فایلی که از نوع `ofstream` باز شده باشد، نمی‌توان چیزی خواند.

نکته: در زبان C++ کاراکتر `\`، جزء کاراکترهای کنترلی محسوب می‌شود. لذا اگر بخواهیم از خود آن در مسیر فایل استفاده کنیم باید بجای یک `\`، دو `\\` نوشته شود.

نکته: اگر هنگام باز کردن یک فایل، مسیر فایل مشخص نگردد، آنگاه فایل موردنظر را در مسیر جاری (همان مسیری که برنامه هم اکنون در آنجا اجرا می‌شود) باز خواهد کرد.

برنامه 47: برنامه‌ای بنویسید که فایلی با نام `test.txt` در مسیر `D:\` ایجاد کرده و جملاتی که کاربر وارد می‌کند را داخل آن ذخیره کند. برنامه زمانی پایان می‌یابد که کاربر `Enter` خالی وارد کند.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
    string s;
    ofstream myfile("d:\\test.txt");           // ایجاد فایل و اشاره کردن به فایل
    do{
        getline(cin , s);                       // گرفتن یک سطر از ورودی
        myfile << s;                             // ذخیره‌ی محتوای رشته در فایل
        myfile << '\n';                         // رفتن به ابتدای سطر بعدی در فایل
    } while(s.empty() == false);               // تا زمانی که رشته خالی نباشد
    myfile.close();                             // بستن فایل
    cout<<" File Saved Successfully. "<< endl;
    return(0);
}
```

نکته: با دستور `getline()` می‌توان کل یک سطر (شامل کلید فاصله `Space`، کلید جهش `Tab`) را از ورودی گرفت. ولی با دستور `cin>>` فقط یک کلمه یا یک عدد را می‌توان از ورودی گرفت. و بقیه کلمات یا اعداد در بافر صفحه کلید می‌ماند.

تمرین 47: این برنامه را در کامپیوتر اجرا کرده و بررسی کنید که آیا واقعاً در کامپیوتر شما، چنین فایلی ایجاد شد؟ آیا ورودی‌هایی که وارد کرده‌اید در داخل فایل وجود دارد؟

برنامه 48: برنامه‌ای بنویسید که فایلی با نام **test.txt** را که از قبل در مسیر **D:** وجود دارد را باز کرده و محتوای آن فایل را سطر به سطر در مانیتور نشان دهد.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
    string S;
    ifstream myfile("d:\\test.txt");           // دسترسی به فایل و اشاره کردن به فایل
    if(!myfile)
        {cout<<" File Not Found. "<< endl;
        return(0);}
    while(getline(myfile , S))                  // بارگذاری یک سطر از فایل تا زمانیکه سطری وجود داشته باشد
        cout << S << endl;
    myfile.close();                             // بستن فایل
    return(0);
}
```

تمرین 48: این برنامه را در کامپیوتر اجرا کرده و خروجی بگیرید. اگر آن فایل در **D:\test.txt** نباشد، چه اتفاقی می‌افتد؟