جزوه درسی اصول برنامه نویسی به زبان C++

رقیه عسگری

پاییز ۹۵ (بازبینی شده در زمستان ۹۶)

پیش‌گفتار

در دنیای امروز، برنامه‌نویسی برای کامپیوتر بیشتر از آن که یک تخصص باشد، به یک مهارت تبدیل شده است و به نظر می‌رسد که در آینده‌ای نه چندان دور تسلط بر یک زبان برنامه‌نویسی همانند زبان‌های گفتاری الزامی شود. بنابراین گنجاندن واحد درسی برنامه‌نویسی کامپیوتر در بین دروس ارائه شده برای دانشجویان رشته‌های فنی مهندسی قطعا بی‌فایده نخواهد بود. همانطور که آموختن هر علم جدید نیازمند ابزاری مناسب است، یادگیری برنامه‌نویسی نیز معمولا با زبان C++ آغاز می‌شود. این زبان که توسعه یافته زبان برنامه‌نویسی C است، از قواعد مستحکم و در عین حال ساده‌ای برخوردار است و به کمک آن می‌توان برنامه‌هایی ساخت یافته تولید کرد. در کنار برنامه‌نویسی ساخت یافته، روش دیگری از برنامه‌نویسی تحت عنوان برنامه‌نویسی شیءگرا قرار دارد که در آن عناصر اصلی تشکیل دهنده برنامه اشیاء هستند. شیءگرایی می‌تواند منجر به تولید برنامه‌هایی شود که از قابلیت خوانایی و تغییرپذیری بالایی برخوردار باشند. زبان برنامه‌نویسی C++ علاوه بر پشتیبانی از شیوه برنامه‌نویسی ساخت یافته، یک زبان شیء گرا نیز است و این بدان معناست که قابلیت‌های لازم را برای نوشتن چنین برنامه‌هایی را دارد.

جزوه‌ای که در پیش روی شماست به دو بخش مختلف تقسیم‌بندی شده است. در بخش اول اصول کلی برنامه‌نویسی ساخت یافته در زبان C++ شرح داده می‌شود و در بخش دوم مفاهیم پایه شیءگرایی معرفی شده و با بخش‌های مختلف برنامه‌های ساخت یافته ادغام می‌شود.

بخش اول:

اصول برنامه‌نویسی ساخت یافته

مقدمه

شاید به نظر همه کامپیوتر یک موجود بسیار باهوش باشد که قادر به انجام پیچیده‌ترین اعمال است. اما در واقعیت این گونه نیست. این ماشین دست‌ساز بشر که دنیا را تکان داده و بدون آن قطعا زندگی بسیار سخت خواهد بود (اگر نگوییم ناممکن خواهد بود)، فقط یک مشت قطعه الکترونیکی و سیم است که جز محاسبات اولیه و ساده، قادر به انجام کار دیگری نیست. چیزی که به این موجود بی‌جان روح می‌بخشد و آن را تا این اندازه باهوش جلوه می‌دهد، برنامه است. برنامه یک بازی کامپیوتری، یک امکان برای خلق و ویرایش تصویر، یک محاسبه‌گر پرقدرت و دقیق داده‌های هواشناسی و به طور خلاصه هر آن چیزی است که امروزه از ماشینی به نام کامپیوتر انتظار داریم که برایمان انجام دهد. تولید چنین نرم‌افزارهایی نیاز به مهارت برنامه‌نویسی بالا و وجود زبان‌هایی قدرتمند دارد که طیف وسیعی از آن‌ها را در دنیای امروزه می‌توان مشاهده کرد. زبان برنامه‌نویسی C++ یکی از اولین زبان‌هایی است که برای برنامه‌نویسی استفاده می‌شد و همچنان نیز مورد توجه بسیاری از برنامه‌نویسان قرار دارد. دو شیوه برنامه‌نویسی ساخت یافته و شیءگرا از جمله روشهای معمول برنامه‌نویسی هستند که امکانات آن‌ها در زبان C++ فراهم شده است.

در این فصل ابتدا مروری بر انواع کلی زبان‌های برنامه‌نویسی خواهیم داشت. سپس زبان C++ به عنوان یک زبان برنامه‌نویسی سطح بالا معرفی شده و مفاهیم برنامه‌نویسی ساخت یافته در آن شرح داده می‌شود.

۱ـ۱ مروری بر انواع زبان‌های برنامه‌نویسی

به زبان ساده یک برنامه کامپیوتری لیستی از دستورات است که در اختیار کامپیوتر قرار می‌گیرد تا آن‌ها را یکی پس از دیگری انجام دهد. به عنوان مثال اعمال زیر بخشی از دستوراتی است که می‌تواند به کامپیوتر داده شود:

۱. دو مقداری را که از طریق صفحه کلید وارد می‌شود دریافت کن و آن را به عنوان اعداد صحیح در نظر بگیر.
۲. یکی از مقادیر را در متغیر X و دیگری را در متغیر Y قرار بده.
۳. X و Y را با هم جمع کن و نتیجه را در متغیر Z قرار بده.
۴. متغیر Z را در صفحه نمایش نشان بده.

مسلم است که دستورات فوق برای ما که انسان هستیم قابل فهم است و نمی‌توان از یک مشت قطعه الکترونیکی انتظار داشت که آن‌ها را بفهمند و انجام دهند. پس سوالی که در اینجا مطرح می‌شود آن است که این قطعات چه زبانی را می‌فهمند؟ پاسخ، *زبان ماشین*[۱] است. زبان ماشین تنها زبانی است که کامپیوتر متوجه می‌شود. حروف تشکیل دهنده این زبان دو عدد صفر و یک هستند (برخلاف زبان انگلیسی که از ۲۶ حرف a تا z تشکیل شده است) که به آن‌ها اعداد دودویی[۲] نیز گفته می‌شود. بنابراین هر آنچه که انتظار می‌رود که کامپیوتر برای انسان انجام دهد باید به زبان صفر و یک بیان شود. کمااینکه برنامه‌نویسان اولیه نیز دستورات برنامه‌های خود را به زبان ماشین می‌نوشتند. به عنوان مثال دستور Z = X + Y در زبان ماشین به شکل زیر بیان می‌شد:

---
[۱] Machine language
[۲] Binary

```
10100001 10111100 10010011 00000100
00001000 00000011 00000101 11000000
10010011 00000100 00001000 10100011
11000000 10010100 00000100 00001000
```

واضح است که برنامه‌نویسی به زبان ماشین کاری زمان‌بر، دشوار و بسیار خطاپذیر بود. به همین دلیل برنامه‌نویسان تصمیم گرفتند از نمادهایی شبیه به کلمات زبان انگلیسی برای بیان اعمال مورد نظر خود در برنامه استفاده کنند. بنابراین زبانی نو به نام *زبان اسمبلی*[۳] یا همان زبان نمادها ابداع شد. در این زبان به جای مقادیر دودویی از یک سری نماد استفاده می‌شد که معنی خاصی داشتند. به عنوان مثال دستورات زیر معادل دستورات فوق در زبان اسمبلی بود:

```
load x
add y
store z
```

همان طور که مشاهده می‌کنید، دستورات این زبان برای انسان قابل فهم بودند. اما برای این که این دستورات برای ماشین نیز قابل اجرا باشند، لازم بود که به زبان ماشین ترجمه شوند. عمل ترجمه یا تبدیل دستورات زبان اسمبلی به زبان ماشین توسط برنامه‌ای به نام اسمبلر انجام می‌شد.

هرچند که دستورات زبان اسمبلی قابل فهم‌تر از دستورات زبان ماشین بود، اما برنامه‌نویسان همچنان مجبور بودند از دستورات زیادی برای انجام یک عمل ساده استفاده کنند. بدین ترتیب *زبان‌های سطح بالا*[۴] ظهور یافتند. این زبان‌ها به گونه‌ای بودند که برنامه‌نویس به راحتی می‌توانست در یک جمله، چندین دستور را بیان کند. به عنوان مثال معادل سه دستور فوق در زبان اسمبلی، در زبان C++ به شکل زیر بیان می‌شد:

```
Z = X + Y;
```

با ظهور زبان‌های سطح بالا که از کلماتی مشابه با زبان انگلیسی و نمادهای ریاضی مرسوم برای بیان دستورات استفاده می‌کردند، برنامه‌نویسان توانستند برنامه‌های کوتاه‌تری را در زمان کمتری ایجاد کنند. اما نکته‌ای که وجود داشت این بود که این برنامه‌ها برای سخت‌افزار قابل فهم نبودند. به همین دلیل برنامه‌هایی به نام مترجم یا کامپایلر[۵] ساخته شدند که قادر بودند برنامه‌های نوشته شده به زبان سطح بالا را به زبان ماشین ترجمه کنند. امروزه زبان‌های سطح بالای بسیاری همچون زبان‌های C، C++، Java، Python و ... وجود دارند که همچنان در کاربردهای مختلف مورد استفاده قرار می‌گیرند.

۲ـ۱ زبان C++ و برنامه‌نویسی ساخت یافته در آن

زبان C++ توسعه یافته زبان C است که در اوایل سال ۱۹۸۰ در آزمایشگاه بل تولید شد. هرچند که این زبان برنامه‌نویسی از قابلیت‌های جدیدی نسبت به زبان C برخوردار بود، اما از روش برنامه‌نویسی ساخت‌یافته[۶] نیز که در

---

[۳] Assembly language
[۴] High-level language
[۵] Compiler
[۶] Structured programming

زبان C مرسوم بود، حمایت می‌کرد. قبل از برنامه‌نویسی ساخت‌یافته، برنامه‌ها به شکل غیر ساخت‌یافته تولید می‌شدند. در این برنامه‌ها معمولا از دستوری به نام goto برای کنترل اجرای برنامه استفاده می‌شد (درمورد دستور goto در زبان C++ توضیح داده خواهد شد) که استفاده بیش از حد از آن منجر به کاهش خوانایی و قابلیت خطایابی برنامه‌ها می‌شد. به همین دلیل طراحان زبان‌های برنامه‌نویسی رویکرد دیگری به نام برنامه‌نویسی ساخت‌یافته را پدید آوردند که در آن، برنامه به بخش‌هایی با ورودی و خروجی مشخص تقسیم می‌شد. ایجاد چنین بخش‌هایی در برنامه نیازمند فراهم کردن امکانات لازم برای توالی، انتخاب و تکرار در زبان برنامه‌نویسی است. زبان C++ از جمله زبان‌هایی است که دارای چنین امکاناتی است.

در ادامه برای آشنایی با امکانات مختلف زبان برنامه‌نویسی C++ ابتدا ساختار یک برنامه ساده را بررسی کرده‌ایم و پس از آن به معرفی هریک از امکانات پرداخته‌ایم.

۱ـ۲ـ۱ ساختار یک برنامه ساده در C++

به طور کلی یک برنامه C++ می‌تواند شامل دستورات پیش‌پردازنده، متغیرها، عبارات و جمله‌ها، توضیحات، توابع، ساختارهای انتخاب و ساختارهای تکرار باشد. برای شرح هریک از این مفاهیم مثال‌هایی آورده شده است.

مثال ۱:

برنامه‌ای بنویسید که یک پیغام سلام به کاربر نشان دهد.

```
1 #include <iostream>
2 using namespace std;
3 // Avvalin Barnamye C++
4 int main(){
5     cout << "Salam.\n";
6     return 0;
7 }
```

خروجی برنامه

```
Salam.
```

توضیح برنامه

خط شماره (۱) یک *دستور پیش‌پردازنده* را نشان می‌دهد. (پیش‌پردازنده بخشی از کامپایلر است که وظیفه اجرای این نوع از دستورات را بر عهده دارد). همان طور که مشاهده می‌کنید، این دستور با علامت # آغاز می‌شود و پس از آن کلمه کلیدی include به همراه علامت‌های <> آمده است (در مورد کلمات کلیدی صحبت خواهیم کرد). کلمه iostream که داخل علامت‌های <> قرار دارد، نام یک فایل کتابخانه‌ای است که شامل تعدادی تابع

و شیء از پیش تعریف شده است. (فایل کتابخانه‌ای چیست؟ تابع چیست؟ شیء چیست؟ نگران نباشید در مورد همه این‌ها در ادامه جزوه توضیح داده شده است).

خط شماره (۲) یک جمله C++ است زیرا به علامت ; ختم شده است (پس تا الآن فهمیدیم که هر چیزی که در زبان C++ به علامت ; ختم شود یک جمله محسوب می‌شود). هدف از آوردن این جمله آن است که بتوانیم در بدنه برنامه از اشیای موجود در فایل کتابخانه‌ای iostream استفاده کنیم (تا همین حد در مورد این جمله بدانید کافی است).

خط شماره (۳) یک توضیح در زبان C++ است. همان طور که مشاهده می‌کنید این خط با علامت‌های // آغاز شده است (پس تا الآن فهمیدیم که هر چیزی که در زبان C++ با علائم // آغاز شود، یک توضیح است). توضیحات توسط کامپایلر ترجمه نمی‌شوند و هدف از آوردن آن‌ها، یادآوری یک نکته به خود برنامه‌نویس است (در ادامه مثال‌های بیشتری خواهیم دید).

خط شماره (۴) بیانگر تعریف تابع اصلی برنامه است که شامل دستوراتی است که با اجرای آن‌ها، هدف برنامه محقق می‌شود. در تعریف این تابع ابتدا کلمه کلیدی int به معنای نوع داده صحیح، سپس کلمه main به عنوان نام تابع و پرانتزهای () و علائم {} آورده شده است. کد اصلی برنامه در داخل دو علامت {} نوشته می‌شود (در مورد انواع داده‌ها در زبان C++ و همچنین منظور از تعریف یک تابع توضیح خواهیم داد).

خط شماره (۵) جمله‌ای است که عبارت Salam. را در خروجی استاندارد (همان صفحه نمایش) چاپ می‌کند. کلمه cout در این خط، نام شیئی است که در فایل iostream تعریف شده است و به خروجی استاندارد اشاره می‌کند. به عبارت دیگر با آوردن این کلمه و بعد از آن علامت‌های >> و سپس عبارت Salam. در داخل دو علامت ""، به شیء cout دست یافته و عبارت مورد نظرمان را در خروجی چاپ می‌کنیم. کامپایلر هر چیزی که در داخل علامت‌های "" آورده شود را عینا در صفحه نمایش چاپ می‌کند. ترکیب حروف \n که در انتهای عبارت Salam. آمده است، یک *ترکیب کنترلی* است که خود در خروجی نمایش داده نمی‌شود، بلکه فقط برای تعیین نحوه نمایش عبارت مورد نظر به کار می‌رود. این ترکیب موجب می‌شود که کامپایلر پس از چاپ عبارت Salam. در خروجی، اشاره‌گر را به ابتدای سطر بعدی منتقل کند. به جای این ترکیب می‌توانستیم از کلمه endl نیز استفاده کنیم:

```
cout << "Salam." << endl;
```

endl نام تابعی است که در کتابخانه iostream تعریف شده است و برای انتقال اشاره‌گر به خط بعدی به کار می‌رود.

حال اگر به جای ترکیب \n در خط شماره (۵)، ترکیب حروف \t وجود داشت:

```
cout << "Salam.\t";
```

آنگاه، کامپایلر به جای انتقال اشاره‌گر به ابتدای خط بعدی، آن را به اندازه ۸ تا فضای خالی جلو می‌برد.

خط شماره (۶) که از ترکیب کلمه کلیدی `return` و عدد `0` تشکیل شده است، دستوری است که مقدار بازگشتی تابع `main` را مشخص می‌کند (بازهم نگران نباشید، منظور از مقدار بازگشتی یک تابع را توضیح خواهیم داد).

نکته: در برنامه فوق و تمام برنامه‌هایی که به زبان C++ نوشته می‌شوند، توالی دستورات مهم است. به عبارت دیگر تا زمانی که دستور خط شماره ۱ اجرا نشود، دستور خط شماره ۲ اجرا نخواهد شد.

مثال ۲:

برنامه‌ای بنویسید که در یک خط عبارت `Salam:` و در خط بعدی آن، عبارت `Khoda Hafez!` را چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 // Dovvomin Barnamye C++
4 int main(){
5     cout << "Salam:\n";
6     cout << "Khoda Hafez!";
7     return 0;
8 }
```

خروجی برنامه

```
Salam:
Khoda Hafez!
```

خطوط شماره (۵) و (۶) در برنامه فوق را می‌توان به شکل زیر هم نوشت:

```
cout << "Salam:\n" << "Khoda Hafez!";
```

و البته به شکل زیر:

```
cout << "Salam:" << "\nKhoda Hafez!";
```

تمرین: اگر به جای خط فوق، دستور زیر را در برنامه می‌نوشتیم چه می‌شد؟ (خودتان فکر کنید و جواب دهید)

```
cout << "Salam:\t" << "Khoda Hafez!";
```

مثال ۳:

در برنامه فوق می‌خواهیم به جای یک خط توضیح (comment) چند خط توضیح قبل از تابع `main` بنویسیم.

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

```
3 //Dovvomin Barnamye C++
4 //ke Gharar Ast Dar Safhe Nemayesh Peygham Salam va
5 //KhodaHafez ra Ba Yek Khat Fasele Chap Konad
6 int main(){
7     cout << "Salam:\n";
8     cout << "Khoda Hafez!";
9     return 0;
10 }
```

خطوط (۳) تا (۵) حاوی سه خط توضیح هستند که نوشتن آن‌ها به شکل زیر هم درست بود:

```
/*Dovvomin Barnamye C
ke Gharar Ast Dar Safhe Nemayesh Peygham Salam va
KhodaHafez ra Ba Yek Khat Fasele Chap Konad */
```

یعنی به جای افزودن دو علامت // در ابتدای هر خط، وجود علامت‌های */ در ابتدای خط اول و علائم /* در انتهای خط آخر نشان دهنده چند خط توضیح در زبان C++ هستند.

۲ـ۲ـ۱ عبارات ریاضی در زبان C++

همان طور که قبلا هم گفته شد زبان C++ یک زبان سطح بالا است. به این معنی که دستورات نوشته شده به این زبان برای انسان قابل فهم هستند. پس همان طور که به راحتی می‌توانیم با کمک نمادهای +، ـ، × و ÷ عبارت‌های ریاضی بنویسیم، در زبان C++ نیز می‌توان با استفاده از این نمادها، اعمال ریاضی انجام داد. با این تفاوت که شکل بعضی از آن‌ها متفاوت است. در جدول زیر این نمادها و اعمال مربوطه نشان داده شده است:

جدول ۱: اعمال ریاضی در زبان C++

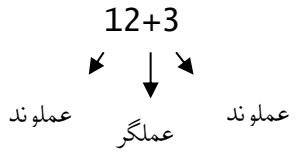
نام عمل	نماد مربوطه در زبان انسان	نماد مربوطه در زبان C++
جمع	+	+
تفریق	-	-
ضرب	×	*
تقسیم	÷	/
باقیمانده	نداریم	%

همان طور که در جدول فوق مشاهده می‌کنید، علاوه بر چهار عمل اصلی، عمل باقیمانده‌گیری نیز در زبان C++ وجود دارد که با نماد % نشان داده می‌شود (این عمل، کاربرد بسیاری دارد که در مثال‌های آینده خواهید دید).

در زبان C++ برای نمادهای √(جذر)، [] (جزء صحیح)، توان و بسیاری از نمادهای دیگر که در ریاضی دیده‌ایم، عملگر مستقیمی وجود ندارد. برای استفاده از این اعمال در برنامه لازم است فایل‌کتابخانه‌ای `math.h` یا `cmath` را با دستور پیش‌پردازنده `#include` به برنامه اضافه کرده و توابعی را که قبلا برای این منظور تعریف شده‌اند، فراخوانی کنیم (نگران نباشید در بخش‌های بعدی جزوه در این مورد مثال‌هایی خواهیم دید).

نکته: نماد % در زبان C++ فقط برای اعداد صحیح قابل استفاده است (پس یعنی اگر برای اعداد اعشاری از آن استفاده شود، کامپایلر C++ اعلام خطا می‌کند).

مسلما هر عبارت ریاضی علاوه بر نمادهای فوق، به یک سری عدد برای انجام عملیات روی آن‌ها نیاز دارد. در زبان C++ به نمادهای مورد استفاده برای اعمال، *عملگر* (Operator) و اعدادی که عمل مورد نظر روی آن‌ها انجام می‌شود، *عملوند* (Operand) گفته می‌شود. در عبارت زیر، عملگر و عملوندها نشان داده شده‌اند:



زبان C++ اجازه نوشتن هر نوع عبارت پیچیده ریاضی را با استفاده از عملگرهای جدول فوق می‌دهد. به عنوان مثال، عبارت زیر یک عبارت معتبر در زبان C++ است:

```
-10.3 / 4 + 6 * 9 % 4 - 7.6
```

تمرین: معادل عبارت $1 + \frac{2\times3+4.4}{5}$ در زبان C++ چیست؟ (خودتان فکر کنید و جواب دهید)

همان طور که در مثال فوق مشاهده می‌کنید، عملوندهای یک عبارت می‌توانند به صورت عدد اعشاری باشند. در زبان C++ نمایش‌های مختلفی برای اعداد اعشاری وجود دارد که برخی از آن‌ها در جدول نیز آمده است:

جدول ۲: نمایش‌های معتبر برای اعداد اعشاری در زبان C/C++

نمایش به کار رفته در زبان C/C++	معادل آن در زبان انسان
`6.`	`6.0`
`.65`	`0.65`
`-32.3e+5`	-32.3×10^5
`10e-6`	10×10^{-6}

۳ـ۲ـ۱ تقدم عملگرها در عبارات ریاضی

ممکن است در ذهن شما این سؤال مطرح شود که اگر در یک عبارت ریاضی چندین عملگر وجود داشته باشد، آنگاه کامپایلر C++ کدام عملگر را ابتدا اجرا خواهد کرد؟ طراحان زبان C++، قوانین خاصی را برای این منظور وضع کرده‌اند که برای عملگرهایی که تاکنون با آن آشنا شده‌ایم به شکل جدول ۳ است.

جدول ۳: قوانین تقدم عملگرهای ریاضی در زبان C/C++

نام عملگر	علامت مربوطه در زبان C++	الویت بین خودشان
پرانتز	()	از چپ به راست
ضرب، تقسیم و باقیمانده	*، / و %	از چپ به راست
جمع و تفریق	+ و -	از چپ به راست

همان طور که در جدول فوق مشاهده می‌کنید، الویت عملگر پرانتز از همه عملگرها بیشتر است. پس از آن، الویت عملگرهای به کار رفته برای ضرب، تقسیم و باقیمانده با یکدیگر برابر و بیشتر از جمع و تفریق است. الویت عملگرهای جمع و تفریق نیز با یکدیگر برابر است. منظور از ستون سوم جدول فوق آن است که اگر در یک عبارت ریاضی چندین عملگر با الویت یکسان وجود داشته باشد، بین آن‌ها الویت با کدام است. در عبارت زیر، ترتیب اجرای عملگرها طبق جدول فوق شماره‌گذاری شده است:

```
-10.3 / 4 + 6 * 9 % 4 - 7.6 = -8.175
```

1 2 3 4 5

نکته: اگر هر دو عملوند به کار رفته برای یک عملگر در یک عبارت ریاضی عدد صحیح باشند، آنگاه کامپایلر آن عبارت را به صورت عدد صحیح ارزیابی می‌کند. اما کافی است که یکی از عملوندها از نوع عدد اعشاری باشد، آنگاه نتیجه عبارت به صورت اعشاری محاسبه می‌شود. به عنوان مثال در عبارت فوق، بخشی که با عدد 1 نشان داده شده است (-10.3 / 4) به صورت اعشاری، بخش 2 به صورت صحیح، بخش 3 به صورت صحیح، بخش 4 به صورت اعشاری و در نهایت بخش 5 نیز به صورت اعشاری ارزیابی شده و در نتیجه حاصل کل عبارت به صورت یک عدد اعشاری خواهد بود.

تمرین: طبق قوانین تقدم عملگرها، ترتیب انجام عملیات در عبارت زیر چیست و حاصل آن چه خواهد بود؟ (خودتان فکر کنید و جواب دهید)

```
(4 - 1.5) * 6 + 11 % 4
```

تمرین: در صورتی که عبارت فوق را به شکل زیر تغییر دهیم، حاصل چه خواهد بود؟

```
4 - 1.5 * 6 + 11 % 4
```

تمرین: اگر عبارت فوق را به صورت زیر بنویسیم با خطای کامپایلری مواجه می‌شویم علت چیست؟ (با توجه به نکات ذکر شده پاسخ دهید)

```
4 - 1.5 * (6 + 11) % 4
```

مثال ۴:

برنامه‌ای بنویسید که میانگین سه عدد ۲، ۳ و ۵ را محاسبه و در صفحه نمایش چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 // Sevvomin Barnamye C++
4 int main(){
5     cout << "Miangin se adad "
6     << 2 << "," << 3 << "," << 5
7     << " = "
8     << (2 + 3 + 5)/3.
9     << endl;
10    return 0;
11 }
```

خروجی برنامه

```
Miangin se adad 2,3,5 = 3.3333
```

توضیح برنامه

در برنامه فوق دستور چاپ خروجی در پنج خط بیان شده است. در این دستور علاوه بر نوشتن مقادیر ثابت و اعداد، یک عبارت ریاضی نیز آورده شده است. این عبارت هنگام اجرای برنامه ارزیابی شده و نتیجه آن در خروجی استاندارد چاپ می‌شود.

در خط شماره (۸)، طبق قوانین تقدم عملگرها، ابتدا عبارت داخل پرانتز ارزیابی شده و سپس حاصل آن بر عدد اعشاری `3.` تقسیم می‌شود. با دقت در این عبارت، به راحتی می‌توان به این نکته پی برد که گذاشتن پرانتز در این عبارت اجتناب‌ناپذیر است. به عبارت دیگر، با اعمال قانون تقدم عملگرها حاصل عبارت زیر متفاوت از نتیجه مورد نظر ما خواهد بود:

```
2 + 3 + 5/3
```

طبق جدول ۲ـ۲، عدد `3.` معادل با عدد `3.0` است و ما عمداً حاصل جمع سه عدد 2، 3 و 5 را بر این عدد تقسیم کرده‌ایم. زیرا طبق نکته‌ای که قبلاً ذکر کردیم، برای انجام یک تقسیم اعشاری لازم است حداقل یکی از عملوندهای تقسیم، از نوع اعشاری باشد. بنابراین در صورتی که عبارت به صورت زیر نوشته می‌شد، حاصل تقسیم به صورت عدد صحیح محاسبه شده و نتیجه اشتباه بدست می‌آمد:

```
(2 + 3 + 5)/3
```

تمرین: اگر عبارت فوق را به شکل زیر بنویسیم آیا باز هم نتیجه اعشاری خواهیم داشت؟ (برنامه مربوطه را نوشته و نتیجه را بررسی کنید)

```
(2. + 3 + 5)/3
```

۴ـ۲ـ۱ متغیرها در زبان C++

در برنامه‌ای که برای مثال ۴ نوشته شد، سه عدد ۲، ۳ و ۵ اعداد ثابتی هستند که به طور مستقیم در عبارت ریاضی محاسبه میانگین استفاده شده‌اند. در این مثال می‌توانستیم به جای استفاده از اعداد ثابت، از *متغیرها* استفاده کنیم. متغیرها، اسامی هستند که برای خانه‌های حافظه در نظر گرفته می‌شوند. علت این که آن‌ها را متغیر می‌نامیم آن است که می‌توانند حاوی مقادیر مختلفی باشند. بنابراین استفاده از آن‌ها در یک عبارت محاسباتی، امکان ارزیابی آن عبارت را برای مقادیر مختلف فراهم می‌کند. در ادامه مثالی آورده شده است که معادل با مثال ۴ است، با این تفاوت که به جای استفاده از مقادیر ثابت در عبارت محاسبه میانگین، از متغیرها استفاده شده است. هدف از آوردن این مثال صرفا آشنایی با نحوه تعریف، مقداردهی و استفاده از متغیرها است.

مثال ۵:

برنامه‌ای بنویسید که مقادیر ۲، ۳ و ۵ را در متغیرهایی قرار داده و میانگین آن‌ها را محاسبه و در صفحه نمایش چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 // Chaharomin Barnamye C++
4 int main(){
5     int a, b, c;
6     float d;
7     a = 2;
8     b = 3;
9     c = 5;
10    d = (a + b + c)/3.;
11    cout << "Miangin se adad "
12    << a << "," << b << "," << c
13    << " = "
14    << d
15    << endl;
```

```
16 return 0;
17 }
```

خروجی برنامه

```
Miangin se adad 2,3,5 = 3.3333
```

توضیح برنامه

خطوط شماره (۵) و (۶) جملاتی هستند که متغیرهای a، b، c و d را تعریف می‌کنند. برای تعریف یک متغیر در زبان C++ در ابتدا باید نوع متغیر و سپس با یک فاصله نام آن را نوشت:

نام متغیر نوع متغیر;

در خط شماره (۵) کلمه int (مخفف integer) به معنی نوع *عدد صحیح* و حروف a، b و c اسامی متغیرها هستند که می‌توانند حاوی مقادیر صحیح باشند. از آنجایی که هر سه متغیر از یک نوع هستند، تعریف هر سه در یک جمله بیان شده است. اما این خط را می‌توان به صورت سه جمله مجزا نیز تعریف کرد:

```
int a;
int b;
int c;
```

خط شماره (۶) جمله‌ای است که متغیری با نام d را از نوع عدد اعشاری ممیز شناور یعنی نوع float (مخفف floating point) تعریف می‌کند. این متغیر می‌تواند حاوی مقادیر اعشاری باشد.

خطوط شماره (۷) تا (۹) جملاتی هستند که مقداری را به یک متغیر منتسب می‌کنند به همین دلیل به آن‌ها عبارت انتساب گفته می‌شود. البته مقداردهی متغیرها را می‌توان به هنگام تعریف آن‌ها نیز انجام داد که به آن *مقداردهی اولیه* (initialization) گویند. برای این مثال کافی است که خطوط شماره (۷) تا (۹) را حذف کرده و خط شماره (۵) را به شکل زیر تغییر داد:

```
int a = 2, b = 3, c = 5;
```

خط شماره (۱۰) یک عبارت ریاضی معتبر در زبان C++ است که میانگین سه متغیر a، b و c که در حال حاضر مقادیر ۲، ۳ و ۵ هستند، محاسبه می‌کند. همانند مثال ۴، در این مثال نیز برای انجام یک تقسیم اعشاری، عدد 3. را به عنوان مخرج کسر در نظر می‌گیریم. البته می‌توانیم این عدد را ابتدا در یک متغیر قرار داده و از آن متغیر به عنوان مخرج کسر استفاده کنیم که در این صورت خطوط شماره (۶) و (۱۰) به شکل زیر تغییر می‌یابند و یک عبارت انتساب دیگر برای قرار دادن مقدار 3. در متغیری اعشاری به نام e به برنامه افزوده می‌شود:

```
float d, e;
e = 3.;
d = (a + b + c)/e;
```

خط شماره (۱۲) مقدار موجود در متغیرهای a، b و c را در صفحه نمایش چاپ می‌کند. در حال حاضر این متغیرها حاوی مقادیر تعیین شده در خطوط (۷) تا (۹) هستند (پس اگر در دستور چاپ نام یک متغیر بدون علامت‌های "" درج شود، مقدار موجود در آن در خروجی چاپ خواهد شد). اگر بخواهیم اسامی متغیرها نیز علاوه بر مقدار متغیرها در خروجی نشان داده شود، می‌توان خطوط شماره (۱۱) تا (۱۵) را به شکل زیر تغییر داد:

```
cout << "Miangin se motagayyer "
          << "a = " << a << ","
          << " b = " << b << ","
          << " c = " << c
          << " mosavi ast ba "
          << d
          << endl;
```

که در این حالت، خروجی برنامه به شکل زیر خواهد بود:

```
Miangin se motagayyer a = 2, b = 3, c = 5 maosavi ast ba 3.3333
```

خط شماره (۱۴) مثال فوق، مقدار موجود در متغیر d را در خروجی استاندارد چاپ می‌کند.

با قرار دادن مقادیر دیگر در متغیرهای مثال فوق، برنامه را می‌توان برای اعداد دیگر اجرا کرد.

۵ـ۲ـ۱ قوانین نام‌گذاری متغیرها

در زبان C++ اسامی متغیرها می‌تواند شامل حروف a تا z، حروف A تا Z، اعداد 0 تا 9 و علامت زیرخط (_) باشد. به طوری‌که شروع آن با یکی از حروف فوق و یا علامت زیرخط باشد (بنابراین نام یک متغیر نمی‌تواند با عدد شروع شود). طول اسامی نیز حداکثر می‌تواند ۳۲ حرف باشد. ذکر این نکته ضروری است که زبان C++ نسبت به کوچک یا بزرگ بودن حروف حساس است. به عبارت دیگر اسامی به‌کار رفته در جمله زیر متفاوت از اسامی موجود در برنامه فوق است:

```
int A, B, C;
```

نکته: استفاده از کلمات کلیدی برای نامگذاری متغیرها مجاز نیست. زیرا کلمات کلیدی اسامی هستند که معنی خاصی برای کامپایلر دارند. مثلا کامپایلر با خواندن کلمه کلیدی int متوجه می‌شود که باید یک متغیر از نوع عدد صحیح تعریف کند. جدول شماره ۴ لیست تمام کلمات کلیدی زبان C++ را نشان می‌دهد.

۶ـ۲ـ۱ تبدیل نوع (type casting)

همانطورکه در مثال ۵ مشاهده کردید، عبارت ریاضی موجود در خط شماره (۱۰) یک مقدار اعشاری تولید می‌کرد. به همین دلیل متغیر d را از نوع عدد اعشاری تعریف نمودیم. حال فرض کنید که به عمد یا به اشتباه این متغیر را از نوع عدد صحیح (یعنی int) تعریف می‌کردیم. یعنی به جای خط شماره (۶) خط زیر را در برنامه می‌نوشتیم:

جدول ۴: لیست کلمات کلیدی زبان C++ (برگرفته از کتاب دایتل و دایتل)

C++ Keywords

Keywords common to the C and C++ programming languages

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

C++-only keywords

and	and_eq	asm	bitand	bitor
bool	catch	class	compl	const_cast
delete	dynamic_cast	explicit	export	false
friend	inline	mutable	namespace	new
not	not_eq	operator	or	or_eq
private	protected	public	reinterpret_cast	static_cast
template	this	throw	true	try
typeid	typename	using	virtual	wchar_t
xor	xor_eq			

```
int d;
```

اتفاقی که رخ می‌داد آن بود که مقدار 3.3333 که به عنوان نتیجه عبارت موجود در خط شماره (۱۰) محاسبه شده بود، به مقدار 3 تبدیل می‌شد. در واقع در چنین حالتی یک *تبدیل نوع ضمنی* انجام شده و قسمت اعشاری عدد حذف می‌شد (بنابراین با قرار دادن یک مقدار اعشاری در یک متغیر از نوع صحیح، آن عدد گرد نمی‌شود بلکه قسمت اعشار آن از بین می‌رود).

نوع دیگری از تبدیل نوع وجود دارد که به آن تبدیل *نوع صریح* گفته می‌شود و در آن به طور صریح نوع نتیجه را به نوع مورد نظرمان تبدیل کنیم. برای این منظور در زبان C++ عملگری تحت عنوان عملگر تبدیل نوع تعریف شده است که نحوه استفاده از آن به شکل زیر است:

```
static_cast<نوع مورد نظر>(عملوند)
```

همان طوری که مشاهده می‌کنید، این عملگر فقط یک عملوند دارد و استفاده از آن موجب می‌شود که یک کپی موقت از عملوند مشخص شده در داخل پرانتز متناسب با نوع مورد نظر ایجاد شود. به عنوان مثال برای انجام یک تقسیم اعشاری در مثال‌های ۴ و ۵ می‌توانیم به جای قرار دادن عدد 3. در مخرج، نوع عبارت موجود در صورت یا مخرج را به نوع `float` تبدیل کنیم که برای مثال ۴ در خط شماره (۸) به شکل زیر:

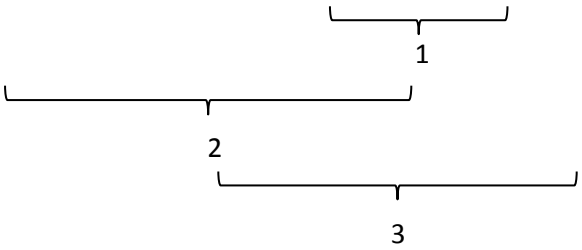
```
static_cast<float>(2 + 3 + 5)/3
```

و برای مثال ۴ در خط شماره (۱۰) نیز به شکل زیر خواهد شد:

```
static_cast<float>(a + b + c)/3
```

نکته: الویت عملگر تبدیل نوع بیشتر از تمام عملگرها و کمتر از پرانتز است. به عنوان مثال در خط فوق الویت‌ها به شکل زیر است:

```
static_cast<float>(a + b + c)/3
```



بنابراین در صورتی که آن را به شکل زیر بنویسیم، هرچند که تبدیل نوع صریح انجام می‌شود، اما نتیجه اعشاری تولید نمی‌شود.

```
static_cast<float>((a + b + c)/3)
```

زیرا طبق الویت عملگرها، پس از محاسبه جمع سه متغیر، ابتدا عمل تقسیم انجام می‌شود و پس از آن عملگر تبدیل نوع اعمال خواهد شد.

تمرین: اگر عبارت فوق را به شکل زیر بنویسیم آیا باز هم نتیجه اعشاری تولید خواهد شد؟ (برنامه مربوطه را نوشته و نتیجه را بررسی کنید)

```
(static_cast<float>(a) + b + c)/3
```

۷ـ۲ـ۱ کار با ورودی و خروجی در زبان C++

در مثال ۵ از بخش قبلی، وظیفه اصلی برنامه محاسبه میانگین سه عدد ۲، ۳ و ۵ بود. حال فرض کنید که کاربر برنامه (یعنی کسی که برنامه را اجرا می‌کند) بخواهد میانگین اعداد دیگری را محاسبه کند. قطعا این برنامه پاسخگوی نیاز کاربر نخواهد بود. اولین راه حلی که برای این مشکل به نظر می‌رسد آن است که خطوط (۷) تا (۹) برنامه را برای انتساب مقادیر جدید کاربر تغییر دهیم. به عنوان مثال خطوط زیر مقادیر ۱۵، ۱۰ و ۲۹ مقادیر متفاوتی را در اختیار برنامه قرار می‌دهد:

```
a = 15;
b = 10;
c = 29;
```

اما مشکلی که در راه حل فوق وجود دارد آن است که اولا همواره برنامه‌نویس و کاربر باید با هم در ارتباط باشند تا برنامه‌نویس بتواند برنامه را طبق خواسته کاربر تغییر دهد. دوما برای مقادیر جدید متغیرها، برنامه باید از ابتدا کامپایل و اجرا شود.

از آنجایی که کاربران معمولا نسخه اجرایی برنامه‌ها را دریافت می‌کنند، مقادیر مختلف مورد نظر آنها باید در هنگام اجرا به برنامه داده شوند. پس نیاز به دستوری است که بتواند هنگام اجرای برنامه مقادیری را از کاربر دریافت و در متغیرهای مربوطه قرار دهد. برای این منظور در زبان C++ شیئی تحت عنوان cin وجود دارد که به ورودی استاندارد (یعنی همان صفحه کلید) اشاره می‌کند. با استفاده از این شیء، برنامه می‌تواند مقدار (یا مقادیری) را از طریق صفحه کلید دریافت و در متغیر (یا متغیرهایی) قرار دهد. در ادامه برای آشنایی بیشتر با این شیء cin، مثالی معادل با مثال ۵ آورده شده است که در آن مقادیر متغیرهای a، b و c از کاربر دریافت می‌شود.

مثال ۶:

برنامه‌ای بنویسید که مقادیر متغیرهای a، b و c را از ورودی استاندارد دریافت کند و میانگین آن‌ها را محاسبه و در صفحه نمایش چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 // Panjomin Barnamye C++
4 int main(){
5     int a, b, c;
6     float d;
7     cout << "meghdar motagayyer a ra vared konid:";
8     cin >> a;
9     cout << "meghdar motagayyer b ra vared konid:";
10    cin >> b;
11    cout << "meghdar motagayyer c ra vared konid:";
12    cin >> c;
13    d = static_cast<float>(a + b + c)/3;
14    cout << "Miangin se adad "
15    << a << "," << b << "," << c
16    << " = "
17    << d
18    << endl;
19    return 0;
20 }
```

خروجی برنامه

در اینجا فرض می‌کنیم که کاربر مقادیر 15، 10 و 29 را به ترتیب برای متغیرهای a، b و c وارد کند:

```
meghdar motagayyer a ra vared konid:15
meghdar motagayyer b ra vared konid:10
meghdar motagayyer c ra vared konid:29
Miangin se adad 2,3,5 = 3.33333
```

توضیح برنامه

خطوط شماره (۷)، (۹) و (۱۱) با چاپ پیغامی در صفحه نمایش از کاربر می‌خواهند که مقادیر متغیرهای مربوطه را وارد کنند.

خطوط شماره (۸)، (۱۰) و (۱۲) نحوه استفاده از شیء `cin` را برای دریافت مقادیر کاربر و قرار دادن آن‌ها در متغیرهای مربوطه نشان می‌دهند. همان طور که مشاهده می‌کنید ابتدا نام شیء `cin`، سپس علامت `<<` (علامت درج)و پس از آن نام متغیرها آورده شده است. هنگام اجرای برنامه این شیء منتظر دریافت ورودی از صفحه کلید می‌شود و تا زمانی که کاربر مقداری را وارد نکند، اجرای برنامه ادامه نمی‌یابد.

همان طور که در **خط شماره (۱۳)** مشاهده می‌کنید، یک تبدیل نوع صریح بر روی مجموع مقادیر متغیرهای `a`، `b` و `c` انجام شده است.

در برنامه فوق می‌توانستیم به جای نوشتن سه جمله مجزا برای چاپ پیغام در خروجی و همچنین سه جمله مجزا برای دریافت مقدار کاربر، آن‌ها را به شکل زیر بنویسیم:

```
cout << "meghdar motagayyerhaye a, b, c ra vared konid:";
cin >> a >> b >> c;
```

در این حالت خروجی برنامه به شکل زیر خواهد بود:

```
meghdar motagayyerhaye a, b, c ra vared konid:15 10 29
Miangin se adad 2,3,5 = 3.33333
```

برای کردن ورودی‌ها کاربر می‌تواند مقادیر خود را با یک فاصله و یا با زدن کلید enter از هم جدا کند.

نکته: استفاده از علامت , برای جدا کردن ورودی‌ها در دستور `cin` از نظر کامپایلر صحیح است اما از نظر منطقی نتیجه درستی تولید نخواهد کرد.

نکته: همانند شیء `cout`، تعریف شیء `cin` نیز در کتابخانه `iostream` و فضای نام `std` قرار دارد. بنابراین برای استفاده از آن لازم است با استفاده از دستور `include` این کتابخانه را به برنامه بیفزاییم. همچنین فضای نام `std` نیز باید به کمک جمله `using namespace std;` به برنامه افزوده شود. البته به جای افزودن این جمله به برنامه می‌توانستیم عبارت `std::` را قبل از نام شیء `cin` استفاده کنیم:

```
std::cout << "meghdar motagayyerhaye a, b, c ra vared konid:";
```

```
std::cin >> a >> b >> c;
```

همان طور که در مثال فوق مشاهده می‌کنید، خروجی برنامه برای مقادیر 2، 3 و 5 برابر با 3.33333 می‌شود. در حالی که میانگین این سه عدد در ریاضی برابر با 3.3333333... (دنباله نامحدودی از ارقام 3)خواهد شد. دلیل این امر آن است که فضای اختصاص یافته برای چنین مقداری در حافظه محدود است. پس بدیهی است که مقادیر اعداد اعشاری ذخیره شده در کامپیوتر به صورت تقریبی خواهند بود.

مثال ۷:

برنامه‌ای بنویسید که مقدار قاعده کوچک، قاعده بزرگ و ارتفاع یک ذوزنقه را دریافت کرده و مساحت آن را محاسبه کند (مساحت ذوزنقه =(قاعده کوچک + قاعده بزرگ)× ارتفاع × $\frac{1}{2}$).

```
1 #include <iostream>
2 using namespace std;
3 // Sheshomin Barnamye C++
4 int main(){
5     int ghaede_koochack, ghaede_bozorg, ertefa;
6     float masahat;
7     cout << "meghdar ghaedeye koochack, ghaedeye bozorg va ertefa
      ra vared konid:";
8     cin >> ghaede_koochack >> ghaede_bozorg >> ertefa;
9     mashat = static_cast<float>((ghaede_koochack +
ghaede_bozorg)*ertefa)/2;
10    cout << "masahat zoozanaghe = "
11    << masahat
12    << endl;
13    return 0;
14 }
```

خروجی برنامه

در اینجا فرض می‌کنیم که کاربر مقادیر 2، 4 و 6 را به ترتیب برای قاعده کوچک، قاعده بزرگ و ارتفاع وارد کند:

```
meghdar ghaedeye koochack, ghaedeye bozorg va ertefa ra vared konid:
2 4 6
masahat zoozanaghe =18
```

توضیح برنامه

در این برنامه سعی شده است از اسامی متناسبی برای متغیرها استفاده شود. تمام این اسامی طبق قوانین زبان C++ نامگذاری شده‌اند.

مثال ۸:

برنامه‌ای بنویسید که مقدار متغیرهای a و b را از ورودی دریافت کند و به کمک یک متغیر دیگر مقادیر آن‌ها را با هم عوض کند.

```
1 #include <iostream>
2 using namespace std;
3 // Haftomin Barnamye C++
4 int main(){
5     int a, b, c;
6     cout << "meghdar a va b ra vared konid:";
7     cin >> a >> b;
8     c = a;
9     a = b;
10    b = c;
11    cout << "meghdar jadid a = " << a
12    << " va b = " << b << endl;
13    return 0;
14 }
```

خروجی برنامه

در اینجا فرض می‌کنیم که کاربر مقادیر 10 و 11 را به ترتیب برای a و b وارد کند:

```
meghdar a va b ra vared konid:10 20
meghdar jadid a = 20 va b = 10
```

توضیح برنامه

در این مثال متغیر c یک متغیر کمکی است که به طور موقت مقدار قبلی متغیر a را در خود نگهداری می‌کند. برای مقادیر فوق، در خط شماره (۸) مقدار متغیر c برابر با 10 می‌شود. در خط شماره (۹) متغیر a حاوی مقدار موجود در متغیر b یعنی مقدار 20 خواهد شد و در نهایت در خط شماره (۱۰) مقداری که به طور موقت در متغیر c نگهداری می‌شد (یعنی مقدار قبلی متغیر a) در متغیر b قرار می‌گیرد.

تمرین: آیا می‌توان بدون استفاده از متغیر اضافی، محتوای دو متغیر a و b را با یکدیگر عوض کرد (خودتان فکر کنید و برنامه آن را بنویسید).

مثال ۹:

برنامه‌ای بنویسید که مقدار زمان را برحسب ثانیه دریافت کند و آن را بر حسب ساعت، دقیقه و ثانیه چاپ کند (برای حل این مساله ابتدا باید تعداد ساعت‌های موجود در مقدار ورودی را با محاسبه جزء صحیح تقسیم آن بر عدد ۳۶۰۰ بدست آوریم. سپس حاصل را از مقدار اولیه کسر کنیم و تعداد دقایق موجود در حاصل تفاضل را با محاسبه جزء صحیح تقسیم آن بر عدد ۶۰ بدست آوریم. در نهایت با کسر تعداد دقایق از مقدار حاصل تفاضل قبلی، تعداد ثانیه‌ها را محاسبه می‌کنیم).

```
1 #include <iostream>
2 using namespace std;
3 // Hashtomin Barnamye C++
4 int main(){
5     int time, hour, minute, second;
6     cout << "meghdar zaman ra vared konid:";
7     cin >> time;
8     hour = time / 3600;
9     time = time - (hour*3600);
10    minute = time / 60;
11    second = time - (minute*60);
12    cout << "hour = " << hour
13    << " minute = " << minute
14    << " second = " << second
15    << endl;
16    return 0;
17 }
```

خروجی برنامه

در اینجا فرض می‌کنیم که کاربر مقدار 3600 ثانیه را به عنوان ورودی وارد کند:

```
meghdar zaman ra vared konid:3600
hour = 1 minute = 0 second = 0
```

توضیح برنامه

در خط شماره (۸) تعداد ساعت‌های کامل موجود در مقدار دریافتی با تقسیم متغیر time بر عدد 3600 بدست می‌آید. با توجه به این که هم صورت و هم مخرج کسر هر دو عدد صحیح هستند، تقسیم به صورت عدد صحیح انجام می‌شود که عملکرد آن شبیه به جزء صحیح خواهد بود.

در خط شماره (۹) مقدار محاسبه شده برای ساعت از مقدار اولیه متغیر time کسر می‌شود تا مقدار دقیقه پس از جدا کردن تعداد ساعت‌ها بدست آید.

در خط شماره (۱۰) تعداد دقایق موجود در مقدار جدید متغیر time با تقسیم صحیح آن بر عدد 60 محاسبه می‌شود.

در خط شماره (۱۱) مقدار محاسبه شده برای دقیقه از مقدار موجود در متغیر time کسر می‌شود تا مقدار ثانیه پس از جدا کردن تعداد دقایق بدست آید.

تمرین: اگر در مثال فوق، به جای خطوط شماره (۸) و (۱۰) خطوط زیر را می‌نوشتیم، نتیجه برنامه چه می‌شد؟

```
hour = time / 3600.;
minute = time / 60.;
```

۸ـ۲ـ۱ تنظیم دقت اعداد اعشاری

در مثال‌هایی که تاکنون بیان شده‌اند، با انواع داده int و float آشنا شدیم. در زبان C++ نوع دیگری از داده با نام double وجود دارد که برای نگهداری اعداد اعشاری استفاده می‌شود. تفاوت این نوع داده با نوع float در میزان دقت آن است. در واقع نوع داده float اعداد اعشاری را با دقت ۷ رقم و نوع داده double با دقت ۱۵ رقم اعشار نگهداری می‌کند. بنابراین در برنامه‌هایی که نیاز به دقت بالایی برای محاسبات اعداد اعشاری وجود دارند، می‌توان از نوع double استفاده کرد. از طرفی دیگر، در زبان C++ تمام اعداد ثابت اعشاری (همانند 13.75، 36.000045 و ...) از نوع double در نظر گرفته می‌شوند. به همین دلیل در مثال‌های بعدی این جزوه از نوع double به جای نوع float استفاده نموده‌ایم.

زبان C++ مستقل از اینکه برنامه‌نویس از کدام یک از انواع float یا double برای مقادیر اعشاری خود در برنامه استفاده می‌کند، توابعی را برای تعیین دقت این مقادیر در اختیار او قرار می‌دهد. توابع setprecision() و fixed از جمله این توابع هستند که می‌توانند دقت یک مقدار اعشاری را به هنگام چاپ آن تنظیم کنند. این تابع در کتابخانه استاندارد iomanip قرار دارد. برای آشنایی با نحوه استفاده از این تابع، مثال ۶ را به کمک آن بازنویسی نموده‌ایم:

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4 // Nohomin Barnamye C++
```

```
5 int main(){
6     int a, b, c;
7     float d;
8     cout << "meghdar motagayyerhaye a, b, c ra vared konid:";
9     cin >> a >> b >> c;
10    d = static_cast<float>(a + b + c)/3;
11    cout << "Miangin se adad "
12    << a << "," << b << "," << c
13    << " = "
14    << setprecision(2) << fixed << d
15    << endl;
16    return 0;
17 }
```

خروجی برنامه

برنامه با مقادیر مثال ۶ یعنی اعداد 2، 3 و 5 اجرا می‌کنیم:

```
meghdar motagayyerhaye a, b, c ra vared konid:15 10 29
Miangin se adad 2,3,5 = 3.33
```

توضیح برنامه

همانطور که در خروجی برنامه مشاهده می‌کنید، نتیجه بدست آمده برای این سه مقدار که در مثال ۶ برابر با عدد `3.33333` می‌شد، در اینجا به دو رقم اعشار گرد شده است.

با توجه به اینکه توابع `setprecision` و `fixed` در کتابخانه استاندارد `iomanip` قرار دارند، برای استفاده از آن‌ها باید این فایل کتابخانه‌ای را با یک دستور پیش پردازنده `include` به برنامه اضافه کنیم. **خط شماره (۱)** بیانگر این امر است.

نحوه استفاده از توابع `setprecision` و `fixed` در **خط شماره (۱۴)** بیان شده است. همان طور که مشاهده می‌کنید، قبل از چاپ مقدار متغیر `d` در خروجی استاندارد، ابتدا تابع `setprecision` به همراه عدد 2 به عنوان تعداد ارقام اعشار و پس از آن تابع `fixed` آورده شده است. این دو تابع به کمک یکدیگر عدد مورد نظر را به دو رقم اعشار تنظیم می‌کنند.

نکته: تابع `fixed` همواره باید بعد از تابع `setprecision` قرار گیرد.

نکته: در صورت عدم استفاده از تابع `fixed` پس از تابع `setprecision`، نقطه اعشار نیز به عنوان دقت عدد مورد نظر محسوب خواهد شد. به عنوان مثال اگر خط شماره (۱۴) از برنامه فوق را به شکل زیر تغییر دهیم، خروجی با یک رقم اعشار چاپ خواهد شد:

```
14    << setprecision(2) << d
```

خروجی برنامه:

```
meghdar motagayyerhaye a, b, c ra vared konid:15 10 29
Miangin se adad 2,3,5 = 3.3
```

نکته: همان طور که در مثال ۶ مشاهده کردید، در صورت عدم استفاده از تابع `fixed` و `setprecision`، دقت پیش فرض مقادیر اعشاری با احتساب نقطه اعشار برابر با ۶ خواهد بود.

۹ـ۲ـ۱ آشنایی با کاراکترها در زبان C++

زبان C++ علاوه بر انواع صحیح و اعشاری، امکان نمایش و استفاده از حروف کوچک، بزرگ و حتی نمادهای خاص روی صفحه کلید را نیز در برنامه‌ها فراهم ساخته است. در این زبان برای نگهداری یک حرف و یا به عبارت بهتر یک *کاراکتر*، از نوع داده‌ای char استفاده می‌شود. متغیر `ch` در خط زیر، توانایی نگهداری هریک از حروف را خواهد داشت:

```
char ch;
ch = 'A';
```

برای تعیین مقدار یک متغیر کاراکتری، کاراکتر مورد نظر را در داخل دو علامت ' ' قرار می‌دهیم. البته مقادیر این متغیرها را می‌توان از ورودی استاندارد نیز دریافت کرد. اما نکته‌ای که باید به آن توجه کرد آن است که اولین کاراکتری که کاربر وارد کند، در این متغیر قرار خواهد گرفت و در صورت وارد کردن کاراکترهای اضافی، مابقی آن‌ها نادیده گرفته خواهند شد.

نکته: در دنیای کامپیوتر تمام کاراکترها دارای یک شماره به نام کد اسکی[۷] هستند که یک عدد صحیح مثبت است. این کدها در جدولی به نام جدول کد اسکی نگهداری می‌شوند. به عنوان مثال کد متناظر با کاراکتر A در جدول کدهای اسکی برابر با عدد 65 است.

نکته: زبان C++ امکان دسترسی کاربران را به کد اسکی کاراکترها فراهم ساخته است. به عبارت دیگر، برنامه‌نویس مجاز است برای نمایش و استفاده از کاراکترها از کد اسکی آن‌ها نیز استفاده کند. در ادامه برای شرح بهتر این مفهوم یک مثال آورده شده است.

[۷] ASCII code

مثال ۱۰

برنامه‌ای بنویسید که با دریافت یک کاراکتر از کاربر، کد اسکی آن را در صفحه نمایش چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 // Dahomin Barnamye C++
4 int main(){
5     char ch;
6     cout << "character mored nazar ra vared konid:";
7     cin >> ch;
8     cout << “ascii code baraye ”
9          << “’” << ch << “’ = “
10         << static_cast<int>(ch)
11         << endl;
12    return 0;
13 }
```

خروجی برنامه

برنامه را با کاراکتر A اجرا می‌کنیم:

```
character mored nazar ra vared konid:A
ascii code baraye 'A' = 65
```

یکبار دیگر برنامه را با کاراکتر # اجرا می‌کنیم:

```
character mored nazar ra vared konid:#
ascii code baraye '#' = 35
```

توضیح برنامه

همانطور که در خروجی برنامه مشاهده می‌کنید، کد اسکی متناسب با هر یک از کاراکترها در خروجی چاپ شده است.

در **خط شماره (۵)** یک متغیر کاراکتری به نام ch برای نگهداری مقدار دریافتی از کاربر تعریف شده است.

خط شماره (۷) کاراکتر دریافتی را در متغیر ch قرار می‌دهد.

برای دستیابی به کد اسکی کاراکتر دریافتی، در **خط شماره (۱۰)** یک عمل تبدیل نوع صریح از نوع کاراکتری به نوع صحیح انجام شده است.

۱۰ـ۲ـ۱ آشنایی با رشته‌ها در زبان C++

به دنباله کاراکترها در زبان C++، یک رشته[۸۸] گفته می‌شود. اما برخلاف نوع داده کاراکتری، هیچ نوع اولیه‌ای برای رشته‌ها در این زبان وجود ندارد و به جای آن از اشیای نوع کلاس string برای نگهداری و نمایش رشته‌ها استفاده می‌شود. به عنوان مثال شیء `str` در خط زیر، توانایی نگهداری یک رشته را خواهد داشت:

```
string str;
str = “salam”;
```

برای تعیین مقدار یک شیء رشته، دنباله مورد نظر را در داخل دو علامت ”“ قرار می‌دهیم. البته مقادیر این اشیاء را می‌توان از ورودی استاندارد نیز دریافت کرد. اما نکته‌ای که باید به آن توجه کرد آن است که هنگام وارد کردن مقدار رشته، کاراکتر فضای خالی (space) به معنای پایان رشته خواهد بود. در ادامه برای آشنایی بیشتر با رشته‌ها یک مثال ساده آورده شده است.

مثال ۱۱

برنامه‌ای بنویسید که با دریافت یک نام و نام خانوادگی کاربر، یک پیغام خوشامد گویی در صفحه نمایش چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 // Yazdahomin Barnamye C++
4 int main(){
5     string name, family;
6     cout << "nam van am khanevadegi shoma chist?";
7     cin >> name >> family;
8     cout << name << “ “ << family
9          << “ aziz be donyaye c++ khosh amadid.”;
10    return 0;
11 }
```

خروجی برنامه

```
nam van am khanevadegi shoma chist?mohammad mohammdi
```

[۸۸] string

```
mohammad mohammdi aziz be donyaye c++ khosh amadid.
```

توضیح برنامه

خط شماره (۵) شامل تعریف دو شیء `name` و `family` برای نگهداری نام و نام خانوادگی کاربر است.

در **خط شماره (۷)** مقادیر نام و نام خانوادگی کاربر دریافت و در متغیرهای مربوطه قرار داده می‌شود.

در **خط شماره (۸)** مقادیر دو شیء `name` و `family` در خروجی استاندارد چاپ می‌شود.

یادآوری: عبارت‌هایی که در دستور چاپ خروجی در داخل دو علامت "" قرار داده می‌شوند نیز یک رشته محسوب می‌شوند.

۱۱ـ۲ـ۱ ساختارهای تصمیم در زبان C++

در تمام مثال‌هایی که تاکنون بررسی نموده‌ایم، یک مسیر برای اجرای برنامه وجود داشته است. به عبارت دیگر تمام دستورات موجود در بدنه اصلی برنامه از ابتدا تا انتها کامپایل و اجرا می‌شد. اما در برنامه‌های واقعی ممکن است بخشی از دستورات برنامه وابسته به درست بودن یک شرط باشند. به عنوان مثال ممکن است برنامه‌ای بنویسیم که فقط برای اعداد مثبت قابل استفاده باشد. در این حالت اجرای دستورات برنامه منوط به شرط مثبت بودن عدد دریافتی خواهد بود. برای چنین مواردی زبان C++ دستوری به نام `if` را در اختیار برنامه‌نویس قرار می‌دهد تا به کمک آن بتواند پس از بررسی یک شرط، در مورد مسیر اجرای برنامه تصمیم‌گیری کند.

به طور کلی ساختار `if` به شکل زیر است:

```
if(عبارت شرطی)
        دستور;
else
        دستور;
```

در صورتی که عبارت شرطی داخل پرانتز، ارزش درستی داشته باشد (یعنی شرط موجود در آن صحیح باشد)، دستور (یا دستورات) بعد از `if` اجرا می‌شود و در غیر این صورت، دستور یا دستورات بعد از `else` اجرا خواهد شد. دستور یا دستوراتی که بعد از `if` و `else` نوشته می‌شوند، می‌توانند هر دستور معتبری در زبان C++ باشند. علاوه بر این اگر تعداد این دستورات بیش از یک دستور باشد، باید آن‌ها را داخل علامت‌های {} نوشت:

```
if(عبارت شرطی){
        دستور ۱;
        دستور۲;
        ...
```

```
}
else{
      دستور ۱;
      دستور۲;
      ...
}
```

مثال ۱۲

برنامه‌ای بنویسید که عددی صحیح از ورودی دریافت کند و بررسی کند که عدد مثبت است یا منفی.

```
1 #include <iostream>
2 using namespace std;
3 //Davazdahomin Barnamye C++
4 int main(){
5     int adad;
6     cout << "yek adad vared konid:";
7     cin >> adad;
8     if(adad >= 0)
9           cout << "adad mosbat ast." << endl;
10    else
11          cout << "adad manfi ast." << endl;
12    return 0;
13 }
```

خروجی برنامه

در اینجا برنامه را برای دو مقدار 10- و 10 اجرا می‌کنیم (یعنی برنامه را دوبار اجرا می‌کنیم)، اجرای اول:

```
yek adad vared konid:10
adad mosbat ast.
```

اجرای دوم:

```
yek adad vared konid:-10
adad manfi ast.
```

توضیح برنامه

خط شماره (۸) یک دستور `if` است که در یک عبارت شرطی، مقدار موجود در متغیر `adad` را با عدد صفر مقایسه می‌کند. همان طور که مشاهده می‌کنید برای مقایسه بزرگترمساوی از علامت‌های =< بدون فاصله بین آن‌ها استفاده شده است. به این علامت‌ها، عملگر رابطه‌ای گفته می‌شود که در بخش‌های بعدی توضیح خواهیم داد.

خط شماره (۹) در صورتی اجرا خواهد شد که عبارت شرطی موجود در دستور `if`، ارزش درستی داشته باشد (یعنی نتیجه مقایسه صحیح باشد).

خط شماره (۱۰) حاوی دستور `else` است.

خط شماره (۱۱) در صورتی اجرا خواهد شد که عبارت شرطی موجود در دستور `if`، ارزش درستی نداشته باشد (یعنی نتیجه مقایسه صحیح نباشد).

اگر در برنامه فوق، دستور `else` را حذف کنیم، یعنی خطوط (۸) تا (۱۱) را به شکل زیر تغییر دهیم:

```
if(adad >= 0)
      cout << "adad mosbat ast." << endl;
cout << "adad manfi ast." << endl;
```

آنگاه برنامه برای اعداد مثبت، به صورت صحیح اجرا نخواهد شد. به عنوان مثال فرض کنید که کاربر عدد 10 را وارد کند. با توجه این که عبارت شرطی دستور `if` برای این عدد، ارزش درستی خواهد داشت، دستور `cout` موجود در بدنه این ساختار اجرا می‌شود. سپس برنامه به اجرای خود ادامه داده و دستور `cout` بعدی را نیز اجرا خواهد کرد. زیرا اجرای آن وابسته به هیچ شرطی نیست. درنهایت خروجی برنامه برای این عدد به شکل زیر خواهد بود:

```
yek adad vared konid:10
adad mosbat ast.
adad manfi ast.
```

بنابراین در برنامه فوق، وجود دستور `else` اجباری است. اما در برخی از برنامه‌ها نیازی به نوشتن این دستور نداریم. مثال زیر گویای این مطلب است.

مثال ۱۳

برنامه‌ای بنویسید که عددی صحیح از ورودی دریافت کند و قدر مطلق آن را در خروجی چاپ کند (در زبان C++ عملگر مستقیمی برای محاسبه قدر مطلق وجود ندارد).

```
1 #include <iostream>
2 using namespace std;
3 //Sizdahomin Barnamye C++
```

```
4 int main(){
5     int adad;
6     cout << "yek adad vared konid:";
7     cin >> adad;
8     if(adad < 0)
9           adad = (-1)*adad;
10    cout << "adad = " << adad << endl;
11    return 0;
12 }
```

خروجی برنامه

در اینجا برنامه را برای دو مقدار 10- و 10 اجرا می‌کنیم (یعنی برنامه را دوبار اجرا می‌کنیم)، اجرای اول:

```
yek adad vared konid:10
adad = 10
```

اجرای دوم:

```
yek adad vared konid:-10
adad = 10
```

توضیح برنامه

خط شماره (۸) یک دستور `if` با شرطی متناسب است که وظیفه بررسی منفی بودن عدد را بر عهده دارد. در عبارت شرطی از علامت > به عنوان عملگر کوچکتر استفاده شده است.

خط شماره (۹) وقتی اجرا می‌شود که عبارت شرطی `if` برای عدد وارد شده، ارزش درستی داشته باشد (یعنی عدد وارد شده منفی باشد). همان طور که مشاهده می‌کنید، در صورتی که عدد کاربر منفی باشد، آن را در عدد 1- ضرب می‌کنیم تا مقدار آن مثبت شود.

خط شماره (۱۰) وابسته به هیچ شرطی نیست و قطعا اجرا خواهد شد و مقدار مثبت عدد را در خروجی چاپ خواهد کرد.

نکته: نیاز یا عدم نیاز به دستور `else` به نحوه برنامه‌نویسی شما بستگی دارد. اما نکته‌ای که هرگز نباید فراموش شود آن است که هیچ دستور `else` بدون `if`ای نباید در برنامه وجود داشته داشته باشد (چون کامپایلر هر دستور `else` را با یک دستور `if` مطابقت می‌دهد و اگر `else` بدون `if` وجود داشته باشد، خطا صادر خواهد کرد).

سؤال: زبان C++ تابعی با نام `abs` در فایل‌های کتابخانه‌ای `cmath` و `math.h` تعریف کرده است که می‌تواند قدر مطلق یک عدد را محاسبه کند. در صورتی که بخواهیم از این تابع برای حل مثال فوق استفاده کنیم، نیازی به

نوشتن دستور if نخواهیم داشت و حتما باید یکی از این دو فایل cmath یا math.h را با کمک دستور پیش‌پردازنده #include به برنامه اضافه کنیم. این برنامه را خودتان بنویسید.

برای نوشتن یک عبارت شرطی در دستور if، از عملگرهای رابطه‌ای استفاده می‌شود که نمونه‌ای از آن‌ها را در مثال‌های فوق مشاهده کردید. جدول شماره ۵ لیست عملگرهای رابطه‌ای در زبان C++ را نشان داده است.

تذکر: در بین عملگرهای جدول شماره ۵، آن‌هایی که حاوی دو علامت هستند، نباید بین علامت‌های آن‌ها فاصله وجود داشته باشد. به عنوان مثال نباید بین دو علامت > و = در عملگر => فاصله‌ای وجود داشته باشد.

جدول ۵: عملگرهای رابطه‌ای در زبان C++

علامت مورد استفاده	مفهوم آن
==	مساوی بودن
!=	نامساوی بودن
<	کوچکتر بودن
>	بزرگتر بودن
>=	بزرگتر یا مساوی بودن
<=	کوچکتر یا مساوی بودن

نکته: به طور کلی، عملگرهای رابطه‌ای برای مقایسه و یافتن رابطه دو عبارت مورد استفاده قرار می‌گیرند. یعنی این دو عبارت می‌توانند عبارات ریاضی و یا حتی عبارات رابطه‌ای دیگر باشند.

مثال ۱۴

برنامه‌ای بنویسید که عددی صحیح از ورودی دریافت کند و تشخیص دهد که زوج است یا فرد (برای تشخیص زوج بودن یک عدد کافی است که بخش‌پذیر بودن آن عدد را بر عدد ۲ بررسی کنیم).

```
1 #include <iostream>
2 using namespace std;
3 //Chahardahomin Barnamye C
4 int main(){
5     int adad;
6     cout << "yek adad vared konid:";
7     cin >> adad;
8     if(adad % 2 == 0)
9           cout << "adad zoj ast." << endl;
```

```
10    else
11          cout << "adad fard ast." << endl;
12    return 0;
13 }
```

خروجی برنامه

در اینجا برنامه را برای دو مقدار 25 و 512 اجرا می‌کنیم (یعنی برنامه را دوبار اجرا می‌کنیم)، اجرای اول:

```
yek adad vared konid:25
adad fard ast.
```

اجرای دوم:

```
yek adad vared konid:512
adad zoj ast.
```

توضیح برنامه

خط شماره (۸) یک دستور `if` است که در یک عبارت شرطی، بخش‌پذیر بودن مقدار موجود در متغیر `adad` را بر عدد 2 بررسی می‌کند. همان طور که مشاهده می‌کنید، بخش‌پذیر بودن را با استفاده از عملگر % می‌سنجیم. به عبارت دیگر اگر حاصل باقیمانده عدد دریافتی بر عدد ۲ برابر با صفر باشد، عدد زوج خواهد بود. در این‌جا برای مقایسه حاصل باقیمانده مقدار موجود در `adad` بر عدد 2، از عملگر == استفاده کرده‌ایم.

تذکر: اگر به جای علامت‌های == در عبارت شرطی مثال فوق، از علامت = برای مقایسه استفاده کنیم، نتیجه اشتباهی بدست خواهیم آورد.

در مثال فوق، عبارت شرطی حاوی عملگرهای ریاضی و رابطه‌ای است. اما نیازی به قرار دادن عبارت ریاضی محاسبه باقیمانده در داخل پرانتز نیست. زیرا تقدم عملگرهای ریاضی از عملگرهای رابطه‌ای بیشتر است. در جدول ۶، تقدم این عملگرها آورده شده است.

جدول ۶: قوانین تقدم عملگرهای ریاضی و رابطه‌ای در زبان C++

نام عملگر	علامت مربوطه در زبان C/C++	الویت بین خودشان
پرانتز	()	از چپ به راست
ضرب، تقسیم و باقیمانده	*، / و %	از چپ به راست
جمع و تفریق	+ و -	از چپ به راست
کوچکتر، کوچکترمساوی، بزرگتر و بزرگتر مساوی	>، =>، < و =<	از چپ به راست
مساوی و نامساوی	==، =!	از چپ به راست

مثال ۱۵

برنامه‌ای بنویسید که دو عدد صحیح از ورودی دریافت کند و تقسیم اعشاری عدد بزرگتر را بر عدد کوچکتر بدست آورد.

```
1 #include <iostream>
2 using namespace std;
3 //Panzdahomin Barnamye C++
4 int main(){
5     int x, y;
6     float z;
7     cout << "do adad vared konid:";
8     cin >> x >> y;
9     if(x >= y){
10          z = (float)x / y;
11          cout << "x / y = " << z << endl;
12    }
13    else{
14          z = (float)y / x;
15          cout << "y / x = " << z << endl;
16    }
17    return 0;
18 }
```

خروجی برنامه

در اینجا برنامه را برای دو مقدار 10 و 15 اجرا می‌کنیم:

```
do adad vared konid:10 15
y / x = 1.5
```

توضیح برنامه

در این برنامه هر چند که در **خط شماره (۵)** متغیرهای x و y از نوع عدد صحیح تعریف می‌شوند، اما متغیر z که حاصل تقسیم اعشاری این دو عدد را نگهداری می‌کند، در **خط شماره (۶)** از نوع عدد اعشاری تعریف می‌شود. چون اگر از نوع صحیح تعریف شود، طبق نکته‌ای که قبلا ذکر شده است، قسمت اعشاری عدد از بین خواهد رفت.

خط شماره (۱۰) در صورتی که عدد موجود در متغیر x بزرگتر از y باشد، تقسیم اعشاری x بر y را با استفاده از یک عمل تبدیل نوع بدست می‌آورد.

خط شماره (۱۴) در صورتی که عدد موجود در متغیر y بزرگتر از x باشد، تقسیم اعشاری y بر x را با استفاده از یک عمل تبدیل نوع بدست می‌آورد.

همان طورکه در مثال فوق مشاهده می‌کنید، چون هر یک از قسمت‌های `if` و `else` حاوی دو دستور هستند، از علامت‌های {} بعد از هر دستور استفاده شده است.

مثال ۱۶

برنامه‌ای بنویسید که با دریافت سه عدد از ورودی به عنوان ضرایب معادله درجه دو ($ax^2 + bx + c$)، اعلام کند که این معادله ریشه حقیقی دارد یا نه (کافی است که مقدار دلتا را با استفاده از فرمول $\Delta = b^2 - 4ac$ بیابیم و مثبت بودن Δ را ارزیابی کنیم).

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 //Shanzdahomin Barnamye C++
5 int main(){
6     int a, b, c;
7     float delta;
8     cout << "zarayeb a, b va c ra vared konid:";
9     cin >> a >> b >> c;
10    delta = b*b - 4*a*c;
11    if(delta >= 0)
12          cout << "moadele rishe darad." << endl;
13    else
14          cout << "moadele rishe nadarad." << endl;
15    return 0;
16 }
```

خروجی برنامه

برنامه را یکبار برای مقادیر 2، 3 و 1 به عنوان ضرایب معادله درجه دو اجرا می‌کنیم:

```
zarayeb a, b va c ra vared konid:2 3 1
```

```
moadele rishe darad.
```

بار دیگر برای مقادیر 2، 2 و 1 اجرا می‌کنیم که به صورت زیر خواهد شد:

```
zarayeb a, b va c ra vared konid:2 2 1
moadele rishe nadarad.
```

توضیح برنامه

در **خط شماره (۱۰)** یک عبارت ریاضی برای محاسبه مقدار دلتا نوشته شده و نتیجه آن در متغیر عدد اعشاری `delta` قرار داده می‌شود. همان طور که قبلا هم اشاره شد، در زبان C/C++ عملگر مستقیمی برای محاسبه مقدار توان وجود ندارد. به همین دلیل برای محاسبه عبارت b^2، مقدار موجود در متغیر `b` را در خودش ضرب کرده‌ایم. البته زبان C++ برای این منظور یک تابع به نام `pow` را در اختیار برنامه‌نویس قرار می‌دهد. با توجه به این که این تابع در هر دو فایل کتابخانه‌ای `math.h` و `cmath` تعریف شده است، برای استفاده از آن باید ابتدا یکی از این دو فایل را با یک دستور پیش‌پردازنده `#include` به برنامه افزود. در صورت استفاده از این تابع در عبارت موجود در خط شماره (۲)، این خط به شکل زیر تغییر می‌یابد:

```
delta = pow(b,2) - 4*a*c;
```

همان طور که مشاهده می‌کنید، ابتدا نام تابع، سپس مقدار پایه و توان در داخل پرانتز آورده می‌شوند.

در **خط شماره (۱۱)** مقدار متغیر `delta` را با عدد صفر مقایسه می‌کنیم و پیغام متناسب را در خروجی چاپ می‌کنیم.

۱۲ـ۲ـ۱ ساختار `if` تو در تو

همان طور که قبلا هم اشاره شد، هر دستور معتبر در زبان C++ را می‌توان به عنوان یک دستور در بدنه دستور `if` و `else` قرار داد. بنابراین خود دستورات `if` و `else` نیز در صورت نیاز می‌توانند در بدنه یک دستور `if` یا `else` دیگر قرار گیرند. در ادامه برای فهم بهتر این مطلب مثالی آورده شده است.

مثال ۱۷

برنامه‌ای بنویسید که ابتدا یک عدد صحیح از ورودی دریافت کند، سپس برابر، کوچکتر یا بزرگتر بودن آن را با عدد صفر بررسی کند.

```
1 #include <iostream>
2 using namespace std;
3 //Hefdahomin Barnamye C++
4 int main(){
5    int a;
```

```
6     cout << "yek adad vared konid:";
7     cin >> a;
8     if(a == 0)
9          cout << "adad sefr ast." << endl;
10    else if(a > 0)
11         cout << "adad bozorgtar az sefr ast." << endl;
12    else
13         cout << "adad koochektar az sefr ast." << endl;
14    return 0;
15 }
```

خروجی برنامه

در اینجا برنامه را برای مقادیر 10، 0 و 10- اجرا می‌کنیم. اجرای اول:

```
yek adad vared konid:10
adad bozorgtar az sefr ast
```

اجرای دوم:

```
yek adad vared konid:0
adad sefr ast.
```

اجرای سوم:

```
yek adad vared konid:-10
adad koochektar az sefr ast.
```

توضیح برنامه

در **خط شماره (۸)** یک دستور `if` وجود دارد که مقدار موجود در متغیر `a` را با عدد صفر مقایسه می‌کند. اگر نتیجه مقایسه درست باشد، خط شماره (۹) اجرا می‌شود.

خط شماره (۱۰) حاوی بخش `else` دستور شماره (۱۱) است. همان طور که مشاهده می‌کنید، این بخش، خود یک دستور `if` است که در صورتی که عبارت شرطی موجود در خط شماره (۸) ارزش درستی نداشته باشد، اجرا خواهد شد و شرط بعدی را بررسی خواهد کرد.

خط شماره (۱۲) حاوی بخش `else` دستور `if` موجود در خط شماره (۱۰) است. بنابراین اگر عبارت موجود در خط شماره (۱۰) ارزش نادرستی داشته باشد، دستور `cout` موجود در خط شماره (۱۳) اجرا می‌شود.

یادآوری این نکته در این جا ضروری است که هر دستور else باید متعلق به یک دستور if باشد. پس اگر خطوط (۸) تا (۱۳) برنامه فوق را به اشتباه به شکل زیر بنویسیم، قطعا یک خطای کامپایلری خواهیم داشت. زیرا کامپایلر نمی‌تواند برای یکی از elseها یک if معادل بیابد.

```
if(a == 0)
      cout << "adad sefr ast." << endl;
else
      cout << "adad bozorgtar az sefr ast." << endl;
      else
      cout << "adad koochektar az sefr ast." << endl;
```

مثال ۱۸

برنامه‌ای بنویسید که با دریافت سه عدد از ورودی به عنوان ضرایب معادله درجه دو ($ax^2 + bx + c$)، ریشه‌های آن را بیابد(پس از محاسبه دلتا، اگر مقدار آن بزرگتر از صفر بود دو ریشه به شکل $\frac{-b \pm \sqrt{b^2-4ac}}{2a}$ و اگر برابر با صفر بود، یک ریشه مضاعف به شکل $\frac{-b}{2a}$ خواهیم داشت).

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 //Hejdahomin Barnamye C
5 int main(){
6     int a, b, c;
7     float delta;
8     float rishe1, rishe2;

9     cout << "zarayeb a, b va c ra vared konid:";
10    cin >> a >> b >> c;
11    delta = b*b - 4*a*c;
12    if(delta > 0){
13          cout << "moadele do rishe darad:\n";
14          rishe1 = ((-1)*b - sqrt(delta))/(2*a);
15          rishe2 = ((-1)*b + sqrt(delta))/(2*a);
16          cout << "rishe1 = " << rishe1 << " va "
17          << "rishe2 = " << rishe2 << endl;
```

```
18    }
19    else if(delta == 0){
20          cout << "moadele yek rishe darad:\n";
21          rishe1 = ((-1)*b)/(2*a);
22          cout << "rishe = " << rishe1 << endl;
23    }
24    else
25          cout << "moadele rishe nadarad." << endl;
26    return 0;
27 }
```

خروجی برنامه

برنامه را یکبار برای مقادیر 2، 3 و 1 به عنوان ضرایب معادله درجه دو اجرا می‌کنیم:

```
zarayeb a, b va c ra vared konid:2 3 1
moadele do rishe darad:
rishe1 = -1 va rishe2 = -0.5
```

بار دیگر برای مقادیر 1، 2 و 1 اجرا می‌کنیم که به صورت زیر خواهد شد:

```
zarayeb a, b va c ra vared konid:1 2 1
moadele yek rishe darad:
rishe = -1
```

توضیح برنامه

در **خط شماره (۸)** برای نگهداری ریشه یا ریشه‌های احتمالی ضرایب، دو متغیر اعشاری `rishe1` و `rishe2` تعریف شده است.

اگر شرط موجود **در خط شماره (۱۲)** صحیح باشد، دو ریشه خواهیم داشت.

خطوط شماره (۱۴) و (۱۵) که اجرای آن‌ها وابسته به شرط موجود در خط شماره (۸) است، وظیفه محاسبه دو ریشه معادله درجه دو را بر عهده دارند. برای محاسبه جذر دلتا از تابع `sqrt` استفاده شده است. به همین دلیل لازم است تابع کتابخانه‌ای `cmath` به برنامه افزوده شود.

خط شماره (۱۹) تک ریشه‌ای بودن معادله را بررسی می‌کند.

خط شماره (۲۱) بدون ریشه بودن معادله را محاسبه می‌کند.

۱۳ـ۲ـ۱ مفهوم ارزش درستی و ارزش نادرستی در عبارت‌های شرطی

در مثال‌هایی که تاکنون بررسی کرده‌ایم، جمله "اگر عبارت شرطی موجود در دستور if ارزش درستی داشته باشد" بسیار تکرار شده است. اما مفهوم ارزش درستی یا نادرستی شرح داده نشده است. در زبان C++ منظور از ارزش نادرستی، *مقدار صفر* و منظور از ارزش درستی، *مقدار غیر صفر* است. با توجه به این مطلب، روند اجرای یک دستور if در زمان اجرای برنامه به صورت زیر خواهد بود:

۱. عبارت داخل پرانتز ارزیابی می‌شود.
۲. در صورتی که نتیجه ارزیابی نادرست باشد، مقدار صفر و گرنه مقداری غیر صفر به جای عبارت داخل پرانتز در نظر گرفته می‌شود.
۳. اگر مقدار داخل پرانتز، غیر صفر باشد دستور وابسته به if اجرا می‌شود. درغیر اینصورت دستور وابسته به else اجرا خواهد شد.

به عنوان مثال، در برنامه‌ای که برای مثال ۱۷ نوشته شد، فرض کنید که کاربر مقادیر 1، 2 و 1 را به عنوان ضرایب چندجمله‌ای وارد کند. ابتدا در خط شماره (۱۲) عبارت شرطی موجود در دستور if ارزیابی می‌شود. با توجه به این که حاصل جذر دلتا بزرگتر از صفر نیست، ارزش این عبارت نادرست و برابر با صفر خواهد شد. بنابراین، دستورات وابسته به آن اجرا نمی‌شود. اجرای برنامه با ارزیابی عبارت شرطی دستور if خط شماره (۱۹) ادامه می‌یابد. با توجه به اینکه جذر دلتا برابر با صفر است، مقداری غیر صفر به عنوان ارزش عبارت در نظر گرفته می‌شود و دستورات وابسته به آن از جمله خط شماره (۲۱) اجرا می‌شود.

نکته: با توجه به مفهوم ارزش درستی و ارزش نادرستی، قطعه برنامه‌های زیر صحیح هستند:

```
if(1)
    دستور;
else
    دستور;
```

در این قطعه برنامه، عبارت شرطی دستور if همواره ارزش درستی دارد، چون حاوی مقداری غیر صفر است. بنابراین همواره دستور موجود در بدنه if اجرا خواهد شد و دستور وابسته به else اجرا نمی‌شود (قرار دادن عدد ۱ به جای شرط دستور if مرسوم نیست. اما در حلقه‌ها از آن برای ایجاد حلقه بینهایت استفاده می‌شود). قطعه برنامه زیر نیز درست است:

```
if(0)
    دستور;
else
    دستور;
```

نکته: با توجه به مفهوم ارزش درستی و نادرستی، اگر بخواهیم در عبارت شرطی دستور `if`، تساوی یا عدم تساوی یک عبارت را با عدد صفر مقایسه کنیم، استفاده از عملگرهای `==` و `=!` الزامی نیست. برای مقایسه از نظر عدم تساوی با صفر، کافی است که عبارت مورد نظر را به تنهایی داخل پرانتز قرار دهیم. برای مقایسه از نظر تساوی با صفر نیز کافی است که قبل از آن عبارت، یک علامت `!` قرار دهیم. به عنوان مثال، قطعه برنامه زیر نامساوی بودن مقدار متغیر `a` را با عدد صفر مقایسه می‌کند

```
if(a)
    دستور;
else
    دستور;
```

و معادل با برنامه زیر است:

```
if(a != 0)
    دستور;
else
    دستور;
```

طبق نکته فوق، قطعه برنامه زیر مساوی بودن مقدار متغیر `a` را با عدد صفر مقایسه می‌کند

```
if(!a)
    دستور;
else
    دستور;
```

و معادل با برنامه زیر است:

```
if(a == 0)
    دستور;
else
    دستور;
```

علامت `!` که *عملگر نقیض* نامیده می‌شود، یک *عملگر منطقی* است (در مورد عملگرهای منطقی در بخش بعدی صحبت خواهیم کرد). وظیفه این عملگر در یک عبارت شرطی، معکوس کردن ارزش عبارت پس از خود است. یعنی اگر آن عبارت، ارزش نادرستی داشته باشد، عملگر نقیض، ارزش آن را به مقدار درست تبدیل می‌کند و بالعکس.

به عنوان مثالی دیگر، برنامه زیر معادل با مثال ۱۴ است:

```
1 #include <iostream>
2 using namespace std;
3 //Chahardahomin Barnamye C++
4 int main(){
5     int adad;
6     cout << "yek adad vared konid:";
7     cin >> adad;
8     if(!(adad % 2))
9          cout << "adad zoj ast." << endl;
10    else
11         cout << "adad fard ast." << endl;
12    return 0;
13    }
```

در خط شماره (۸)، برای بررسی زوج بودن مقدار موجود در متغیر adad، ابتدا حاصل باقیمانده متغیر بر عدد 2 محاسبه می‌شود. پس از آن، نتیجه با عدد صفر مقایسه می‌شود. اگر حاصل باقیمانده برابر با صفر باشد (یعنی ارزش نادرستی داشته باشد)، اعمال عملگر نقیض بر روی آن موجب می‌شود که کل عبارت شرطی دستور if ارزش درستی پیدا کند و دستور وابسته به آن اجرا شود. در غیر اینصورت عدد دریافتی فرد بوده و دستور بخش else اجرا خواهد شد.

۱۴ـ۲ـ۱ ترکیب عبارت‌های شرطی در دستور if

در تمام مثال‌هایی که تاکنون بررسی کرده‌ایم، عبارت شرطی موجود در دستور if تنها حاوی یک شرط بود. به عبارت دیگر اجرای دستور (یا دستورات) ساختار if وابسته به برقرار بودن یک شرط بوده است. اما در برخی از مواقع لازم است چندین شرط برای انجام یک دستور بررسی شوند. در این مواقع نیاز به ترکیب چندین عبارت شرطی در یک عبارت شرطی داریم. برای فهم بهتر این موضوع مثال زیر را دنبال کنید.

مثال ۱۹

برنامه‌ای بنویسید که نمره یک دانشجو را از ورودی دریافت کند و سطح آن را با توجه به موارد زیر با حروف A تا D تعیین کند.

17 < نمره <= 20 : A

14 < نمره <= 17 : B

10 < نمره <= 14: C

نمره <= 10 : D

```
1 #include <iostream>
2 using namespace std;
3 //Noozdahomin Barnamye C++
4 int main(){
5     float nomreh;
6     cout << "nomreh ra vared konid:";
7     cin >> nomreh;
8     if(nomreh > 17 && nomreh <= 20)
9           cout << "A\n";
10    else if(nomreh > 14 && nomreh <= 17)
11          cout << "B\n";
12    else if(nomreh > 10 && nomreh <= 14)
13          cout << "C\n";
14    else
15          cout << "D\n";
16    return 0;
17 }
```

خروجی برنامه

برنامه را با نمره 17.5 اجرا می‌کنیم:

```
nomreh ra vared konid:17.75
A
```

توضیح برنامه

در **خط شماره (۵)** یک متغیر اعشاری به نام nomreh برای نگهداری نمره دریافتی از کاربر تعریف شده است.

خط شماره (۸) حاوی یک دستور if است که وظیفه مقایسه مقدار موجود در متغیر nomreh را با بازه نمره‌های 17 تا 20 بر عهده دارد. همان طور که مشاهده می‌کنید، عبارت شرطی داخل پرانتز خود حاوی دو عبارت شرطی مختلف است که با استفاده از دو علامت && با یکدیگر ترکیب شده‌اند. مفهوم این عبارت شرطی آن است که اگر هر دو عبارت شرطی دارای ارزش درستی باشند، ارزش کل عبارت درست خواهد بود.

به همین ترتیب **خط شماره (۱۰)** مقدار موجود در متغیر nomreh را با بازه نمره‌های 14 تا 17 و **خط شماره (۱۲)** با بازه نمره‌های 10 تا 14 مقایسه می‌کند.

در صورتی‌که نمره دریافتی از کاربر در هیچ‌کدام از بازه‌های تعیین شده نباشد، **خط شماره (۱۴)** اجرا خواهد شد.

تذکر: شاید به نظر می‌رسد که بتوان عبارت‌های شرطی خطوط (۸) تا (۱۲) را به شکل زیر نوشت:

```
8 if(17 < nomreh <= 20)
9     cout << "A\n";
10 else if(14 < nomreh <= 17)
11    cout << "B\n";
12 else if(10 < nomreh <= 17)
13    cout << "C\n";
```

این دستورات از نظر کامپایلر صحیح هستند، اما به هنگام اجرا نتیجه درستی تولید نخواهند کرد. فرض کنید که هنگام اجرای برنامه، کاربر عدد 14.5 را وارد کند. هنگامی‌که نوبت به اجرای خط شماره (۸) می‌رسد، با توجه به این‌که تقدم عملگرهای رابطه‌ای موجود در این خط با هم برابر و از سمت چپ به راست است (طبق جدول ۲ـ۵)، ابتدا مقدار موجود در متغیر nomreh با عدد 17 مقایسه می‌شود. چون 14.5 بیشتر از 17 نیست، پس ارزش این عبارت نادرست و برابر با صفر می‌شود. سپس عملگر رابطه‌ای دوم اجرا شده و مقدار صفر با عدد 20مقایسه می‌شود و در نتیجه کل عبارت شرطی ارزش درست خواهد داشت. پس دستور وابسته به if اجرا شده و مقدار A در خروجی استاندارد چاپ می‌شود که نتیجه نادرستی است.

برای ترکیب عبارت‌های شرطی در دستور if، از عملگرهای منطقی استفاده می‌شود که نمونه‌ای از آن را در مثال فوق مشاهده کردید. جدول شماره ۷ لیست عملگرهای منطقی در زبان C++ را نشان داده است.

جدول ۷: عملگرهای منطقی در زبان C++

علامت مورد استفاده	نام آن
&&	و
||	یا
!	نقیض

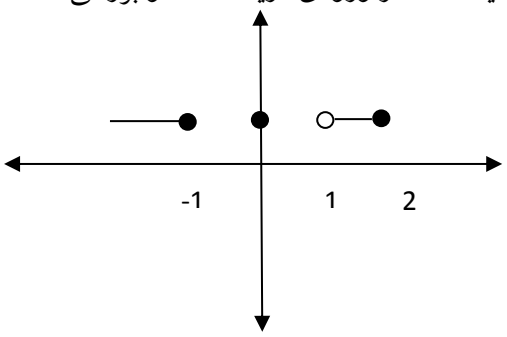
شرح عملگرهای منطقی:

- **عملگر نقیض:** در مورد رفتار و کاربرد عملگر نقیض (!) در بخش قبلی توضیح داده شد.
- **عملگر &&** : اگر عملگر && بین دو عبارت شرطی قرار گیرد، ارزش کل عبارت وقتی درست خواهد بود که ارزش هر دو عبارت درست باشد. یعنی اگر حتی یکی از عبارت‌ها دارای ارزش نادرستی باشد، کل عبارت ارزش نادرستی خواهد داشت.

- **عملگر ||**: اگر عملگر || بین دو عبارت شرطی قرار گیرد، ارزش کل عبارت وقتی درست خواهد بود که حداقل ارزش یکی از عبارت‌ها درست باشد. اما اگر هر دو عبارت ارزش نادرستی داشته باشند، کل عبارت ارزش نادرستی خواهد داشت.

مثال ۲۰

برنامه‌ای بنویسید که یک عدد از ورودی دریافت کند و بررسی کند که آیا متعلق به دامنه تابع زیر است یا نه.




```
1 #include <iostream>
2 using namespace std;
3 //Bistomin Barnamye C++
4 int main(){
5     float x;
6     cout << "x ra vared konid:";
7     cin >> x;
8     if(x <= -1 || !x || x > 1 && x <= 2)
9          cout << "x dar damane gharar darad.\n";
10    else
11         cout << "x dar damane gharar nadarad.\n";
12    return 0;
13 }
```

خروجی برنامه

برنامه را با عدد 1.5 اجرا می‌کنیم:

```
x ra vared konid:1.5
x dar damane gharar darad.
```

توضیح برنامه

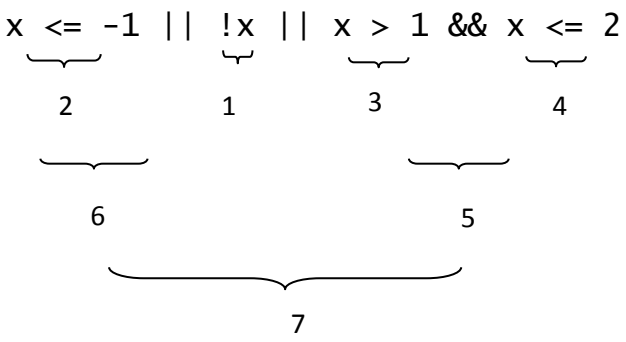
با توجه به نمودار تابع، چندین عبارت شرطی را در **خط شماره (۸)** ترکیب کرده‌ایم. همان طور که مشاهده می‌کنید از عملگرهای منطقی برای این منظور استفاده شده است. در بین عبارت‌های موجود در این خط، عبارت `x !` وجود دارد که با توجه به نکته بخش قبل معادل با `x==0` است.

نکته: در یک عبارت شرطی همانند مثال فوق، امکان دارد چندین عملگر از نوع عملگرهای رابطه‌ای، ریاضی و منطقی وجود داشته باشد. برای چنین حالت‌هایی تقدم عملگرها چگونه خواهد بود؟ پاسخ این سؤال را در جدول ۸ مشاهده می‌کنید.

جدول ۸: قوانین تقدم عملگرهای ریاضی، رابطه‌ای و منطقی در زبان C++

نام عملگر	علامت مربوطه در زبان C/C++	الویت بین خودشان
پرانتز	()	از چپ به راست
نقیض	!	از راست به چپ
ضرب، تقسیم و باقیمانده	*، / و %	از چپ به راست
جمع و تفریق	+ و -	از چپ به راست
کوچکتر، کوچکترمساوی، بزرگتر و بزرگتر مساوی	<، =<، > و =>	از چپ به راست
مساوی و نامساوی	==، =!	از چپ به راست
و	&&	از چپ به راست
یا	||	از چپ به راست

طبق جدول فوق، برای مثال شماره ۱۹، تقدم عملگرهای موجود در عبارت شرطی خط شماره (۸) به شکل زیر خواهد بود:



۱۵ـ۲ـ۱ ساختار switch

در بخش‌های قبلی با ساختار `if` تودرتو و کاربرد آن آشنا شدیم. با استفاده از این ساختار می‌توان به ازای مقادیر مختلف یک عبارت، دستورات مختلفی اجرا کرد. اما در حالتی که تعداد مواردی که عبارت مورد نظر قرار است با آن‌ها مقایسه شود، زیاد باشند، بهتر است به جای `if` تودرتو، از ساختار `switch` استفاده کرد. این ساختار به شکل زیر است:

```
switch(عبارت){
      case مورد ۱ :
            دستور ۱ ;
            دستور ۲ ;
            ...
            break;
      case مورد ۲ :
            دستور ۱ ;
            دستور ۲ ;
            ...
            break;
      . . .
      default:
            دستور ۱ ;
            دستور ۲ ;
            ...
            break;
}
```

عبارتی که داخل پرانتز دستور switch قرار می‌گیرد، می‌تواند یک متغیر یا هر عبارت معتبر دیگری در زبان C++ باشد. موردی که بعد از کلمه کلیدی case قرار می‌گیر د، مقداری است که نتیجه عبارت دستور switch با آن مقایسه خواهد شد. در صورت تطابق این عبارت با یکی از موارد، دستور یا دستورات بدنه case اجرا خواهند شد. همان طور که مشاهده می‌کنید، آخرین دستوری که در بدنه case قرار می‌گیرد، دستور break است. این دستور که به عنوان یکی از دستورات *انتقال کنترل غیرشرطی* شناخته می‌شود، کنترل اجرای برنامه را از دستور switch خارج می‌کند (در مورد دستورات انتقال کنترل غیر شرطی در بخش ساختارهای تکرار بیشتر صحبت خواهیم کرد). در صورتی که هنگام اجرای برنامه، نتیجه عبارت با هیچ یک از مواردی که با دستور case مشخص شده‌اند، مطابقت نداشته باشد، دستور default اجرا خواهد شد.همان طوری که مشاهده می‌کنید، نیازی به نوشتن یک مقدار بعد از این دستور وجود ندارد. برای آشنایی بیشتر با این ساختار توجه شما را به مثال زیر جلب می‌کنیم.

مثال ۲۱

برنامه‌ای بنویسید که با دریافت یک عدد صحیح به عنوان شماره یک روز از هفته، نام آن روز را در خروجی چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 //Bisto Yekomin Barnamye C++
4 int main(){
5     int x;
6     cout << "shomareye rooz ra vared konid:";
7     cin >> x;
8     if(x == 1)
9          cout << "emrooz shanbe ast.\n";
10    else if(x == 2)
11         cout << "emrooz yek shanbe ast.\n";
12    else if(x == 3)
13         cout << "emrooz do shanbe ast.\n";
14    else if(x == 4)
15         cout << "emrooz se shanbe ast.\n";
16    else if(x == 5)
17         cout << "emrooz chahar shanbe ast.\n";
18    else if(x == 6)
19         cout << "emrooz panj shanbe ast.\n";
20    else if(x == 7)
21         cout << "emrooz jome ast.\n";
22    else
23         cout << "adad vared shode namotabar ast.\n";
24    return 0;
25 }
```

خروجی برنامه

برنامه را با عدد 1 اجرا می‌کنیم:

```
shomareye rooz ra vared konid:1
emrooz shanbe ast.
```

توضیح برنامه

همان طور که مشاهده می‌کنید، برنامه از ساختار `if` تودرتو برای مقایسه مقدار دریافتی متغیر `x` با اعداد `1` تا `7` استفاده می‌کند. **خطوط شماره (۸) تا (۲۰)** بیانگر دستورات `if` مربوطه هستند. در صورتی که عدد وارد شده، متناظر با هیچ کدام از شرط‌ها نباشد، برنامه در **خط شماره (۲۲)** پیغام متناسبی چاپ می‌کند.

برنامه فوق را می‌توان با استفاده از ساختار `switch` به شکل زیر بازنویسی کرد:

```
1 #include <iostream>
2 using namespace std;
3 //Bisto Yekomin Barnamye C++
4 int main(){
5     int x;
6     cout << "shomareye rooz ra vared konid:";
7     cin >> x;
8     switch(x){
9           case 1:
10                cout << "emrooz shanbe ast.\n";
11                break;
12          case 2:
13                cout << "emrooz yek shanbe ast.\n";
14                break;
15          case 3:
16                cout << "emrooz do shanbe ast.\n";
17                break;
18          case 4:
19                cout << "emrooz se shanbe ast.\n";
20                break;
21          case 5:
22                cout << "emrooz chahr shanbe ast.\n";
23                break;
24          case 6:
```

```
25              cout << "emrooz panj shanbe ast.\n";
26              break;
27         case 7:
28              cout << "emrooz jome ast.\n";
29              break;
30         default:
31              cout << "adad vared shode namotabar ast.\n";
32              break;
33    }
34 return 0;
35 }
```

خروجی برنامه

برنامه را با عدد 1 اجرا می‌کنیم:

```
shomareye rooz ra vared konid:1
emrooz shanbe ast.
```

توضیح برنامه

در **خط شماره (۸)** برنامه، متغیر x به عنوان عبارت مورد ارزیابی در داخل پرانتز دستور switch آورده شده است (یک متغیر به تنهایی می‌تواند یک عبارت محسوب شود).

خط شماره (۹) همانند خط شماره (۸) در برنامه قبلی، مقدار متغیر x را از نظر تساوی با عدد 1 مقایسه می‌کند. اگر با یکدیگر مساوی باشند، خطوط شماره (۱۰) و (۱۱) به عنوان دستورات وابسته دستور case اجرا می‌شوند.

خط شماره (۱۰) پیغام متناسبی در خروجی چاپ می‌کند.

خط شماره (۱۱) موجب خروج اجرای برنامه از دستور switch می‌شود. زیرا تطابق مورد نظر یافته شده و دیگری نیازی به بررسی سایر دستورات case نیست.

خطوط شماره (۱۲) تا (۲۷) عمل مقایسه مقدار متغیر x را با بقیه موارد انجام می‌دهند.

خط شماره (۳۰) شامل دستور default است و در حالتی اجرا خواهد شد که مقدار دریافتی متغیرx با هیچ‌کدام از موارد ذکر شده در دستورات case مطابقت نداشته باشد. همان طور که مشاهده می‌کنید پیغامی که در آخرین دستور else از برنامه قبلی وجود داشت، در بدنه این دستور قرار گرفته است.

نکته: در ساختار switch فقط می‌توان عبارت داخل پرانتز را از نظر تساوی با مقادیر مشخص شده در دستورات case مقایسه کرد. به عبارت دیگر، امکان بررسی بزرگتر، کوچکتر یا نامساوی بودن عبارت داخل پرانتز با مقادیر مشخص شده در دستورات case وجود ندارد.

نکته: مقدار مشخص شده در یک دستور case، نباید با مقادیر caseهای دیگر یکسان باشد.

نکته: حتی اگر تعداد دستورات وابسته به دستور case بیش از یک دستور باشد، نیازی به قرار دادن آکولاد نیست.

نکته: نوشتن دستور default در بدنه دستور switch اجباری نیست و کاملا وابسته به نیاز برنامه است.

نکته: در صورتی که دستور break به عنوان آخرین دستور بدنه یک case نوشته نشود، عملکردی مشابه با عملگر || بین مقدار تعیین شده توسط آن case و case بعدی رخ می‌دهد. برای فهم بهتر این نکته به مثال بعدی توجه کنید.

مثال ۲۲

برنامه‌ای بنویسید که با دریافت دو عدد صحیح و یک کاراکتر به عنوان نماد یک عملگر، عمل متناظر با آن کاراکتر را بر روی دو عدد انجام دهد.

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 //Bisto Dovvomin Barnamye C++
5 int main(){
6     char ch;
7     int x, y;
8     cout << "do adad ra vared konid:";
9     cin >> x >> y;
10    cout << "amalgar mored nazar ra vared konid:";
11    cin >> ch;
12    switch(ch){
13          case '+':
14                cout << "x + y = " << x + y << endl;
15                break;
16          case '-':
17                cout << "x - y = " << x - y << "\n";
18                break;
```

```
19          case '*':
20               cout << "x * y = " << x * y << "\n";
21               break;
22          case '/':
23          case '\\':
24               cout << "x/ y = " << (float)x / y << "\n";
25               break;
26          case '^':
27               cout << "x ^ y = " << pow(x,y) << "\n";
28               break;
29          default:
30               cout << "amalgar namotabar ast.\n";
31               break;
32     }
33     return 0;
34 }
```

خروجی برنامه

برنامه را با اعداد 2 و 3، یکبار با کاراکتر + به عنوان عملگر و کاراکترهای / و \ را به عنوان عملگر تقسیم اجرا می‌کنیم. اجرای اول:

```
do adad ra vared konid:2 3
amalgar mored nazar ra vared konid:+
x + y = 5
```

اجرای دوم:

```
do adad ra vared konid:2 3
amalgar mored nazar ra vared konid:\
x / y = 0.666667
```

توضیح برنامه

در **خط شماره (۶)** برنامه، یک متغیر کاراکتری با نام ch از نوع char تعریف می‌شود تا کاراکتری را که کاربر به عنوان عملگر مربوطه وارد می‌کند، در خود نگهداری کند.

در خط شماره (۱۱) مقدار متغیر ch از ورودی استاندارد دریافت می‌شود.

در خط شماره (۱۲)، یک دستور switch وجود دارد که وظیفه مقایسه مقدار متغیر ch را با موارد مشخص شده در خطوط شماره (۱۳)، (۱۶)، (۱۹)، (۲۲)، (۲۳) و (۲۶) بر عهده دارد. همان طور که مشاهده می‌کنید، مقادیر تعیین شده در برابر دستورات case به صورت کاراکتری هستند. به این معنی که داخل علامت‌های '' قرار گرفته می‌شوند.

خط شماره (۲۲) شامل یک دستور case است که شامل دستور break نیست و بلافاصله پس از آن یک دستور case دیگر در خط شماره (۲۳) قرار گرفته است. طبق نکته‌ای که قبلا ذکر شد، در این حالت عملکردی مشابه با عملگر || بین مقادیر تعیین شده توسط دو دستور case پشت سر هم انجام می‌شود. به این معنی که اگر کاراکتر وارد شده توسط کاربر برابر با '/' و یا '\' باشد، برنامه عمل تقسیم اعشاری متغیرهای x و y را در **خط شماره (۲۴)** انجام می‌دهد.

خط شماره (۲۶) در صورتی اجرا خواهد شد که عملگر وارد شده توسط کاربر، عملگر توان باشد. در این صورت همان طور که مشاهده می‌کنید، در **خط شماره (۲۷)**، تابع pow برای انجام عمل توان فراخوانی شده است.

نکته: برای استفاده از مقدار کاراکتر \ (منظور back slash) لازم است قبل از آن یک علامت \ دیگر نوشته شود. به عنوان مثال خط زیر، مقدار این کاراکتر را با استفاده از دستور cout در خروجی استاندارد چاپ می‌کند:

```
cout << "\\";
```

در صورتی که دستور فوق را به شکل زیر بنویسید با خطای کامپایلری مواجه خواهید شد. زیرا کاراکتر \ معنی خاصی برای کامپایلر دارد (به عنوان مثال: \n برای انتقال اشاره‌گر به خط بعدی، \t برای انتقال اشاره‌گر به اندازه ۸ تا فضای خالی).

```
cout << "\";
```

با توجه به نکته فوق، مقدار تعیین شده برای دستور case در **خط شماره (۲۳)** به شکل '\\' نوشته می‌شود و منظور از آن یک کاراکتر \ است.

خط شماره (۲۹) شامل دستور default است و در حالتی اجرا خواهد شد که مقدار دریافتی متغیر ch با هیچ‌کدام از موارد ذکر شده در دستورات case مطابقت نداشته باشد.

سؤال: مثال فوق را با ساختار if تودرتو بازنویسی کنید.

۱۶ـ۲ـ۱ ساختارهای تکرار در زبان C++

با استفاده از مفاهیمی که تاکنون در مورد زبان C++ مطرح کرده‌ایم، به راحتی می‌توان برنامه‌هایی نوشت که شامل انواع عبارات ریاضی باشند. همچنین در صورتی که اجرای بخشی از دستورات برنامه وابسته به برآورده شدن شرایط خاصی باشد، می‌توان با استفاده از ساختارهای تصمیم if-else و switch شرایط مورد نظر را بررسی کرد.

در دنیای واقعی مسائلی وجود دارند که راه حل آن‌ها شامل عملی است که با اجرای چندباره آن نتیجه نهایی حاصل می‌شود. به عنوان مثال برای تشخیص اول بودن یک عدد صحیح اولین راه حل آن است که بخش‌پذیری عدد مورد نظر را بر تمام اعداد بازه ۱ تا خود آن عدد بررسی کرد. در اینجا هسته اصلی برنامه، عمل بررسی بخش‌پذیری عدد مورد نظر بر عددی دیگر است. اما نکته‌ای که وجود دارد آن است که برای رسیدن به راه حل نهایی لازم است این عمل، چندین بار (یعنی به تعداد اعداد موجود در بازه ۱ تا خود آن عدد) تکرار شود. به عنوان مثال دیگر فرض کنید که بخواهیم قبول یا مردود شدن تعدادی دانشجو را در یک درس مشخص بررسی کنیم. ابتدا این مساله را برای حالتی که فقط یک دانشجو وجود دارد، بررسی می‌کنیم:

```
1 #include <iostream>
2 using namespace std;
3 //Bisto Sevvomin Barnamye C++
4 int main(){
5     double nomreh;
6     cout << "nomreye dars ra vared konid:";
7     cin >> nomreh;
8     if(nomreh >= 10)
9          cout << "ghabool ast.\n";
10    else
11         cout << "mardood ast.\n";
12    return 0;
13 }
```

همان طور که مشاهده می‌کنید، با دریافت یک نمره از ورودی، در صورتی که مقدار آن بیشتر از 10 باشد، پیغامی مبنی بر قبول شدن دانشجو و در غیر این صورت پیغام مردود شدن او در خروجی چاپ می‌شود. حال فرض کنید که بخواهیم این برنامه را برای تعداد ۴ دانشجو تکرار کنیم. برای این منظور می‌توان خطوط (۶) تا (۱۱) را به تعداد ۴ بار بازنویسی کرد. شاید این روش برای تعداد کم، قابل استفاده باشد. اما اگر تعداد دانشجویان را مثلا به عدد ۱۰۰ افزایش دهیم، یک برنامه بسیار طولانی خواهیم داشت که بسیاری از دستورات آن تکراری است. راه حل منطقی که در اینجا به ذهن می‌رسد آن است که خطوط (۶) تا (۱۱) یکبار نوشته شوند، اما به تعداد ۱۰۰ بار اجرا شوند. به عبارت دیگر اگر می‌توانستیم اجرای برنامه را بعد از خط شماره (۱۱) دوباره به خط شماره (۶) بازگردانیم و این عمل را ۱۰۰ بار تکرار کنیم، مشکل حل می‌شد.

زبان C++ مانند بسیاری از زبان‌های دیگر حاوی دستوری به نام goto است که قادر است اجرای برنامه را از یک نقطه به هر نقطه دیگری از آن منتقل کند. نحوه از این دستور برای مثال فوق به شکل زیر خواهد بود:

```
1 #include <iostream>
```

```
2 using namespace std;
3 //Bisto Chaharomin Barnamye C++
4 int main(){
5     double nomreh;
6     int i = 0;
7     L:cout << "yek nomreh vared konid:";
8     cin >> nomreh;
9     if(nomreh >= 10)
10          cout << "ghabool ast.\n";
11    else
12          cout << "mardood ast.\n";
13    i = i + 1;
14    if(i < 4)
15          goto L;
16    return 0;
17 }
```

خروجی برنامه

برنامه را با نمرات 10، 2.75، 15 و 5.5 اجرا می‌کنیم:

```
yek nomreh vared konid:10
ghabool ast.
yek nomreh vared konid:2.75
mardood ast.
yek nomreh vared konid:15
ghabool ast.
yek nomreh vared konid:5.5
mardood ast.
```

توضیح برنامه

همان طورکه در برنامه فوق مشاهده می‌کنید، عمل دریافت یک نمره از کاربر و بررسی آن بدون این‌که دستورات مربوطه بازنویسی شود، به تعداد ۴ بار اجرا شده است.

در **خط شماره (۶)** یک متغیر عدد صحیح به نام `i` با مقدار اولیه `0` تعریف شده است که برای نگهداری تعداد دفعات اجرای دستورات خطوط (۷) تا (۱۲) استفاده می‌شود.

خط شماره (۷) شامل دستور چاپ و کاراکتری به نام `L` است که با یک علامت کولن (`:`) از یکدیگر جدا شده‌اند. کاراکتر `L` در اینجا نقش برچسبی را دارد که بیانگر نقطه شروع قسمتی از برنامه است که چندین بار تکرار خواهد شد. نامی که برای برچسب انتخاب می‌شود، باید از قوانین نامگذاری متغیرها تبعیت کند.

خطوط (۸) تا (۱۲) هسته اصلی برنامه و قسمتی است که باید تکرار شود.

برای اطمینان از این که دستورات مورد نظر به تعداد ۴ بار تکرار شده‌اند، پس از هربار اجرای خطوط (۷) تا (۱۲) در **خط شماره (۱۳)** یک واحد به مقدار موجود در `i` افزوده می‌شود و در **خط شماره (۱۴)** مقدار این متغیر با عدد 4 مقایسه می‌شود. اگر مقدار آن کمتر از 4 باشد، در **خط شماره (۱۵)** دستور `goto` اجرا شده و برنامه به خطی که حاوی برچسب `L` است، پرش می‌کند و بدین ترتیب بار دیگر دستورات خطوط (۷) تا (۱۵) اجرا خواهند شد. در غیر اینصورت اجرای برنامه با خط بعد از دستور `goto` ادامه می‌یابد.

همان طور که در مثال فوق مشاهده می‌کنید، دستور `goto` منجر به ایجاد حلقه‌ای از اجرا در برنامه شده است. در این حالت به متغیر `i` که وظیفه نگهداری تعداد دفعات اجرای حلقه را برعهده دارد، *شمارنده حلقه* گفته می‌شود.

اگرچه دستور `goto` می‌تواند منجر به تکرارتعدادی از دستورات برنامه شود، امّا استفاده از آن خوانایی برنامه را کاهش خواهد داد. به همین دلیل در زبان C++ سه نوع دستور دیگر با نام‌های `for`، `while` و `do while` وجود دارند که وظیفه ایجاد تکرار یا حلقه در برنامه را دارند. برای استفاده از این دستورها کافی است که برنامه‌نویس بخش تکراری برنامه را شناسایی کند و آن‌ها را در بدنه یکی از این سه دستور قرار دهد. بدین ترتیب هنگام اجرای برنامه دستورهای مورد نظر به جای یکبار اجرا، چندین بار اجرا خواهند شد. در ادامه شرح این ساختارها آورده شده است.

۱ـ۱۶ـ۲ـ۱ ساختار تکرار for

دستور `for` یکی از دستورهای زبان C++ است که می‌تواند امکان تکرار بخشی از دستورات برنامه را فراهم کند. ساختار این دستور به شکل زیر است:

```
for(مقداردهی اولیه شمارنده حلقه ; شرط حلقه ; گام حلقه){
    دستور ۱;
    دستور ۲;
    ...
}
```

در این ساختار ابتدا کلمه کلیدی for و پس از آن سه عبارت مختلف که با علامت ; از هم جدا می‌شوند در داخل یک پرانتز آورده می‌شوند. عبارت اول، عمل مقداردهی اولیه شمارنده حلقه را انجام می‌دهد که فقط یکبار اجرا می‌شود. همان طور که قبلا هم اشاره شد، شمارنده حلقه، متغیری از نوع عدد صحیح است که وظیفه نگهداری تعداد دفعات اجرای حلقه را بر عهده دارد. البته وجود این عبارت زمانی ضروری است که برای پیاده‌سازی حلقه، از متغیر مجزایی برای شمارش استفاده شود.

عبارت دوم دستور for، یک عبارت شرطی است که در صورت درست بودن آن، دستورات بدنه حلقه اجرا خواهند شد. همانند عبارت‌های شرطی ساختار if-else، این عبارت می‌تواند حاوی عملگرهای ریاضی، رابطه‌ای و منطقی باشد که در هر بار ا اجرای حلقه ارزیابی می‌شود.

بالاخره عبارت سوم دستور for، دستوری است که اجرای حلقه را پیش برده و به پایان آن نزدیک می‌کند. در صورتی که از متغیر شمارنده استفاده شده باشد، گام حلقه دستوری خواهد بود که مقدار شمارنده را تغییر می‌دهد. این عبارت پس از اجرای تمام دستورات موجود در بدنه حلقه، اجرا خواهد شد.

همانند دستور if، در صورتی که تعداد دستورات موجود در بدنه دستور for بیش از یک دستور باشد، باید آن‌ها را در داخل علامت‌های {} قرار دهیم. برای آشنایی با عملکرد این دستور، مثال اخیر را با استفاده از آن بازنویسی می‌کنیم:

```
1 #include <iostream>
2 using namespace std;
3 //Bisto Panjomin Barnamye C++
4 int main(){
5     double nomreh;
6     int i;
7     for(i = 0; i < 4 ; i++){
8           cout << "yek nomreh vared konid:";
9           cin >> nomreh;
10          if(nomreh >= 10)
11                cout << "ghabool ast.\n";
12          else
13                cout << "mardood ast.\n";
14    }
15    return 0;
16 }
```

خروجی برنامه

برنامه را با مقادیر 10، 2.75، 15 و 5.5 اجرا می‌کنیم و متوجه می‌شویم که نتیجه اجرای آن، همانند مثال قبلی خواهد بود:

```
yek nomreh vared konid:10
ghabool ast.
yek nomreh vared konid:2.75
mardood ast.
yek nomreh vared konid:15
ghabool ast.
yek nomreh vared konid:5.5
mardood ast.
```

در جدول زیر مقادیر متغیرهای برنامه پس از هر دور اجرای حلقه نشان داده شده است. همان طور که مشاهده می‌کنید، با رسیدن مقدار متغیر i به عدد 4، اجرای برنامه پایان می‌پذیرد.

	مقدار متغیر nomreh	مقدار متغیر i
مقدار اولیه یا مقدار دریافتی از کاربر	نامعلوم	0
پس از اجرای دستورات مورد تکرار	10	1
پس از اجرای دستورات مورد تکرار	2.75	2
پس از اجرای دستورات مورد تکرار	15	3
پس از اجرای دستورات مورد تکرار	5.5	4

توضیح برنامه

در این برنامه تنها با نوشتن یک دستور for و قرار دادن دستورات مورد نظر در بدنه آن، اجرای این دستورات را ۴ بار تکرار نموده‌ایم.

همانند مثال قبلی **خط شماره (۶)** حاوی متغیری است که برای شمارش تعداد دفعات اجرای دستورات مورد تکرار استفاده می‌شود.

خط شماره (۷) نحوه استفاده از دستور for را نشان می‌دهد. همان طور که مشاهده می‌کنید، سه عبارت مختلف در داخل پرانتز وجود دارند که با استفاده از علامت ; از یکدیگر جدا شده‌اند. اولین عبارت، وظیفه مقداردهی اولیه متغیر i (شمارنده حلقه) را برعهده دارد. دومین عبارت، شرط i < 4 است که قبل از اجرای دستورات بدنه حلقه ارزیابی می‌شود. در صورت درست بودن این شرط تمام دستورات **خطوط شماره (۸) تا (۱۳)** اجرا خواهند شد. پس از یک دور از اجرای دستورات بدنه حلقه، عبارت سوم داخل پرانتز اجرا می‌شود. این دستور که معادل با دستور

i=i+1 است، مقدار شمارنده را یک واحد افزایش می‌دهد. در صورتی که عبارت شرطی حلقه نادرست باشد، اجرای برنامه به بعد از حلقه و **خط شماره (۱۵)** منتقل خواهد شد.

نکته: مقدار اولیه شمارنده حلقه می‌تواند هر مقدار عدد صحیحی باشد. اما معمولا عدد صفر را به عنوان مقدار اولیه در نظر می‌گیرند.

نکته: در صورتی که مقدار اولیه متغیر شمارنده عددی غیر صفر باشد، باید شرط حلقه به تناسب آن تنظیم شود. به عنوان مثال اگر در برنامه فوق مقدار اولیه متغیر i را برابر 1 در نظر بگیریم آن‌گاه شرط حلقه را باید به شکل زیر تغییر دهیم:

```
6     int i ;
7     for(i = 1 ; i <= 4 ; i++){
```

نکته: عمل مقداردهی اولیه شمارنده حلقه را می‌توان قبل از دستور for انجام داد. به عنوان مثال خطوط شماره (۶) و (۷) از برنامه فوق را می‌توان به شکل زیر تغییر داد:

```
6     int i  = 0;
7     for(; i < 4 ; i++){
```

نکته: در برخی از کامپایلرها، برنامه‌نویس می‌تواند متغیر شمارنده را در همان عبارت اول دستور for تعریف و مقداردهی کند. به عنوان مثال در برنامه فوق می‌توان خط شماره (۶) را حذف کرده و خط شماره (۷) را به شکل زیر تغییر داد:

```
7     for(int i = 0; i < 4 ; i++){
```

نکته: وجود هیچ یک از عبارت‌های داخل پرانتز دستور for الزامی نیست. به عنوان مثال قطعه برنامه زیر معادل با برنامه فوق است.

```
6     int i = 0;
7     for(; i < 4 ;){
8           cout << "yek nomreh vared konid:";
9           cin >> nomreh;
10          if(nomreh >= 10)
11                cout << "ghabool ast.\n";
12          else
13                cout << "mardood ast.\n";
14          i++;
15    }
```

اما نوشتن دو علامت ; ; در داخل پرانتز الزامی است.

نکته: در صورتی که دستور را به شکل for(;;) نوشته شود معادل با یک حلقه بی‌نهایت است که در ادامه جزوه بررسی خواهد شد.

مثال ۲۳

برنامه‌ای بنوسید که اعداد بازه ۱ تا ۱۰۰ را در خروجی چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 //Bisto Sheshomin Barnamye C++
4 int main(){
5     int i;
6     for(i = 1 ; i <= 100 ; i++)
7           cout << i << " ";
8     return 0;
9 }
```

خروجی برنامه

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66
67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99 100
```

توضیح برنامه

هدف از آوردن این مثال، اشاره به این موضوع است که متغیر شمارنده علاوه بر نقش شمارش می‌تواند در دستورهای بدنه حلقه نیز استفاده شود. کما اینکه مقدار متغیر i در این مثال در **خط شماره (۷)** برای چاپ در خروجی مورد استفاده قرار می‌گیرد. به مقدار اولیه متغیر i و شرط حلقه توجه کنید. با توجه به این که بدنه این حلقه تنها حاوی یک دستور است، نیازی به استفاده از علائم {} نیست.

مثال ۲۴

برنامه‌ای بنوسید که اعداد زوج بازه ۱ تا ۱۰۰ را در خروجی چاپ کند.

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

```
3 //Bisto Haftomin Barnamye C++
4 int main(){
5     int i;
6     for(i = 1 ; i <= 100 ; i++){
7           if(i % 2 == 0)
8                 cout << i << " ";
9     }
10    return 0;
11 }
```

خروجی برنامه

```
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44
46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86
88 90 92 94 96 98 100
```

توضیح برنامه

در این مثال، قبل از چاپ مقدار شمارنده `i` در خروجی، زوج بودن آن بررسی می‌شود. البته این مثال را به شکل زیر نیز می‌توان نوشت:

```
1 #include <iostream>
2 using namespace std;
3 //Bisto Haftomin Barnamye C++
4 int main(){
5     int i;
6     for(i = 2 ; i <= 100 ; i += 2)
7           cout << i << " ";
8     return 0;
9 }
```

تفاوت این برنامه با برنامه قبلی در نحوه نوشتن عبارت‌های اول و سوم داخل پرانتز دستور `for` است. همان طور که در **خط شماره (۶)** مشاهده می‌کنید، مقدار اولیه متغیر `i` برابر با عدد `2` در نظر گرفته شده است. علاوه بر این، پس از هر بار اجرای دستورات حلقه، مقدار شمارنده حلقه دو واحد افزایش می‌یابد که معادل با دستور زیر است:

```
i = i + 2
```

تمرین: برنامه فوق را برای اعداد فرد بازنویسی کنید.

مثال ۲۵

برنامه‌ای بنویسید که کد اسکی کاراکترهای A تا E را در خروجی استاندارد چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 //Bisto Hashtomin Barnamye C++
4 int main(){
5     char ch;
6     for(ch = 'A' ; ch <= 'E' ; ch++)
7           cout << ch << ":" << static_cast<int>(ch) << endl;
8     return 0;
9 }
```

خروجی برنامه

```
A:65
B:66
C:67
D:68
E:69
```

توضیح برنامه

همان طور که در این مثال مشاهده می‌کنید، شمارنده حلقه یک متغیر کاراکتری است که در هر دور از اجرای حلقه، شامل کاراکتر بعدی جدول اسکی است. **خط شماره (۷)** کد اسکی متناظر با کاراکتر موجود در متغیر ch را در هر بار از اجرای حلقه در خروجی استاندارد چاپ می‌کند.

در مثال‌هایی که تاکنون بررسی کردیم، تعداد دفعات اجرای حلقه عدد ثابت و مشخصی بود که مقدار آن قبل از اجرای دور بعدی حلقه، با مقدار شمارنده مقایسه می‌شد (در مثال فوق: i <= 100). اما در مسائل واقعی ممکن است لازم باشد که کاربر تعداد دفعات تکرار حلقه را تعیین کند. در این حالت نیاز به متغیری داریم که تعداد مورد نظر کاربر را در خود نگهداری کند. به مثال زیر توجه کنید.

مثال ۲۶

برنامه‌ای بنویسید که با دریافت نمره تعدادی دانشجو، قبول یا مردود شدن آن‌ها را بررسی کند (در این مثال، تعداد دانشجویان عدد نامعلومی است که مقدار آن توسط کاربر تعیین می‌شود).

```
1 #include <iostream>
2 using namespace std;
3 //Bisto Nohomin Barnamye C++
4 int main(){
5     double nomreh;
6     int i, tedad;
7     cout << "daneshjooyan chand nafar hastand? ";
8     cin >> tedad;
9     for(i = 0; i < tedad ; i++){
10          cout << "yek nomreh vared konid:";
11          cin >> nomreh;
12          if(nomreh >= 10)
13               cout << "ghabool ast.\n";
14          else
15               cout << "mardood ast.\n";
16    }
17    return 0;
18 }
```

خروجی برنامه

برنامه را با تعداد 2 دانشجو و نمرات 5 و 15 اجرا می‌کنیم:

```
daneshjooyan chand nafar hastand? 2
yek nomreh vared konid:5
mardood ast.
yek nomreh vared konid:15
ghabool ast.
```

در جدول زیر مقادیر متغیرهای برنامه پس از هر دور اجرای حلقه نشان داده شده است:

	مقدار متغیر nomreh	مقدار متغیر i	مقدار متغیر tedad
مقدار اولیه یا مقدار دریافتی از کاربر	نامعلوم	0	2
پس از اجرای دستورات حلقه	5	1	2
پس از اجرای دستورات حلقه	15	2	2

همان طورکه در سطر سوم از جدول مشاهده می‌کنید، با برابر شدن مقدار متغیر i با مقدار متغیر tedad، اجرای حلقه پایان می‌پذیرد.

توضیح برنامه

تفاوت این مثال با مثال مشابه قبلی آن در این است که در **خط شماره (۶)** علاوه بر متغیر i، یک متغیر عدد صحیح به نام tedad برای نگهداری تعداد دانشجویان در نظر گرفته شده است. در ابتدای هر دور از اجرای حلقه، مقدار متغیر i با مقدار متغیر tedad مقایسه می‌شود. بنابراین تعداد دفعات اجرای حلقه برابر با عددی خواهد بود که کاربر تعیین می‌کند.

اگر به برنامه فوق دقت کنید، متوجه می‌شوید که بدون داشتن متغیر i نیز می‌توان تعداد دفعات اجرای حلقه را شمارش کرد. به عبارت دیگر متغیر tedad علاوه بر تعیین تعداد نهایی اجرای حلقه، می‌تواند نقش شمارنده را نیز بازی کند. در این حالت برنامه فوق به شکل زیر بازنویسی می‌شود:

```
1 #include <iostream>
2 using namespace std;
3 //Bisto Nohomin Barnamye C++
4 int main(){
5     double nomreh;
6     int tedad;
7     cout << "daneshjooyan chand nafar hastand? ";
8     cin >> tedad;
9     for(;tedad > 0 ; tedad--){
10          cout << "yek nomreh vared konid:";
11          cin >> nomreh;
12          if(nomreh >= 10)
13               cout << "ghabool ast.\n";
14          else
15               cout << "mardood ast.\n";
16    }
17    return 0;
18 }
```

خروجی برنامه

همانند مثال قبلی برنامه را با تعداد 2 دانشجو و نمرات 5 و 15 اجرا می‌کنیم:

```
daneshjooyan chand nafar hastand? 2
yek nomreh vared konid:5
mardood ast.
yek nomreh vared konid:15
ghabool ast.
```

در جدول زیر مقادیر متغیرهای برنامه پس از هر دور اجرای حلقه نشان داده شده است:

	مقدار متغیر nomreh	مقدار متغیر tedad
مقدار اولیه یا مقدار دریافتی از کاربر	نامعلوم	2
پس از اجرای دستورات حلقه	5	1
پس از اجرای دستورات حلقه	15	0

همان طورکه در سطر سوم از جدول مشاهده می‌کنید، با برابر شدن مقدار متغیر tedad با عدد 0 اجرای حلقه پایان می‌پذیرد.

توضیح برنامه

همان طورکه در **خط شماره (۶)** مشاهده می‌کنید، نیازی به تعریف متغیر i در این برنامه وجود ندارد/

با توجه به این که مقدار متغیر tedad توسط کاربر تعیین می‌شود، دستور for **خط شماره (۹)**، فاقد عبارت مقداردهی اولیه آن است.

با حذف متغیر i، شرط اجرای حلقه به مقایسه مقدار متغیر tedad با عدد 0 تغییر می‌کند. بنابراین لازم است پس از هر دور حلقه، یک واحد از مقدار این متغیر کاسته شود. این مطلب را در عبارت سوم دستور for مشاهده می‌کنید که معادل با عبارت زیر است:

```
tedad = tedad - 1;
```

البته روش فوق تنها زمانی قابل استفاده است که پس از اتمام اجرای حلقه، دیگر نیازی به مقدار متغیر tedad نداشته باشیم. به عنوان مثال فرض کنید که برنامه فوق را طوری تغییر دهیم که علاوه بر تعیین قبولی یا مردودی دانشجویان، میانگین نمرات آن‌ها را نیز محاسبه کند. با توجه به این که برای محاسبه میانگین نمرات لازم است مجموع نمرات بر تعداد آن‌ها تقسیم شود، پس دانستن تعداد نمرات یعنی همان تعداد دانشجویان الزامی است. به عبارت دیگر، نمی‌توان همزمان از متغیر tedad به عنوان شمارنده حلقه نیز استفاده کرد و نیاز به متغیر دیگری است. برای برای درک این مطلب به مثال زیر توجه کنید.

مثال ۲۷

برنامه‌ای بنویسید که با دریافت نمره تعدادی دانشجو، علاوه بر تعیین قبولی یا مردودی دانشجویان، میانگین نمرات آن‌ها را نیز محاسبه کند و در خروجی چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 //Seeyomin Barnamye C++
4 int main(){
5     double nomreh, majmoo = 0, miangin;
6     int i, tedad;
7     cout << "daneshjooyan chand nafar hastand? ";
8     cin >> tedad;
9     for(i = 0; i < tedad ; i++){
10          cout << "yek nomreh vared konid:";
11          cin >> nomreh;
12          majmoo += nomreh;
13    }
14    miangin = majmoo / tedad;
15    cout << "miangin = " << miangin << endl;
16    return 0;
17 }
```

خروجی برنامه

برنامه را با تعداد 4 دانشجو و نمرات 10، 2.75، 15 و 5.5 اجرا می‌کنیم:

```
daneshjooyan chand nafar hastand? 4
yek nomreh vared konid:10
yek nomreh vared konid:2.75
yek nomreh vared konid:15
yek nomreh vared konid:5.5
miangin = 8.3125
```

در جدول زیر مقادیر متغیرهای برنامه پس از هر دور اجرای حلقه نشان داده شده است:

	nomreh	i	tedad	majmoo	miangin
مقدار اولیه یا مقدار دریافتی از کاربر	نامعلوم	0	4	0	نامعلوم
پس از اجرای دستورات حلقه	10	1	4	10	نامعلوم
پس از اجرای دستورات حلقه	2.75	2	4	12.75	نامعلوم
پس از اجرای دستورات حلقه	15	3	4	27.75	نامعلوم
پس از اجرای دستورات حلقه	5.5	4	4	33.25	نامعلوم

با اتمام اجرای حلقه، مقدار متغیر miangin محاسبه و برابر با 8.3125می‌شود.

توضیح برنامه

همان طور که در این برنامه مشاهده می‌کنید، از متغیر i به عنوان شمارنده حلقه استفاده شده است و مقدار متغیر tedad برای محاسبه میانگین نمرات بدون تغییر باقی مانده است.

در **خط شماره (۵)** برنامه، علاوه بر متغیر nomreh که برای نگهداری نمره یک دانشجو استفاده می‌شود، متغیرهای اعشاری majmoo و miangin به ترتیب برای نگهداری مجموع و میانگین نمرات دانشجویان تعریف شده‌اند.

در **خط شماره (۶)**، دو متغیر صحیح i و tedad به ترتیب برای کنترل تعداد دفعات اجرای حلقه و نگهداری تعداد دانشجویان تعریف شده‌اند.

در **خط شماره (۹)**، شرط اجرای حلقه ارزیابی شده و در صورتی که صحیح باشد، دستورات حلقه اجرا می‌شوند.

در **خط شماره (۱۲)**، حاصل جمع مقدار موجود در متغیر majmoo با مقدار دریافتی برای نمره که در متغیر nomreh قرار دارد، در متغیر majmoo قرار می‌گیرد. دستوری که در این خط وجود دارد معادل با دستور زیر است:

```
majmoo = majmoo + nomreh;
```

برای دور اول حلقه، مقدار موجود در متغیر majmoo برابر با مقدار اولیه آن یعنی 0 است که پس از جمع بسته شدن با اولین نمره دریافتی، برابر با آن نمره خواهد شد.

خط شماره (۱۴)، میانگین نمرات را در یک عبارت ریاضی با تقسیم مقدار متغیر majmoo بر متغیر tedad محاسبه می‌کند.

نکته: اگر در برنامه فوق کاربر تعداد دانشجویان را به اشتباه یا عمد برابر با صفر وارد کند، در زمان کامپایل خطایی پیش نمی‌آید اما در زمان اجرای برنامه خطای تقسیم بر صفر نشان داده می‌شود. برای پیشگیری از این حالت بهتر است قبل از انجام تقسیم و محاسبه میانگین، صفر نبودن مخرج کسر بررسی شود.

مثال ۲۸

برنامه‌ای بنویسید که با دریافت نمره تعدادی دانشجو، بیشترین نمره را بیابد (برای یافتن بیشترین نمره کافی است که پس از دریافت هر نمره، آن را با بیشترین نمره قبلی مقایسه کرد).

```
1 #include <iostream>
2 using namespace std;
3 //Seeyo Yekomin Barnamye C++
4 int main(){
5     double nomreh, bishtarin = 0;
6     int i = 0, tedad;
7     cout << "daneshjooyan chand nafar hastand? ";
8     cin >> tedad;
9     for(i = 0; i < tedad ; i++){
10          cout << "yek nomreh vared konid:";
11          cin >> nomreh;
12          if(nomreh >= bishtarin)
13               bishtarin = nomreh;
14    }
15    cout << "bishtarin nomreh = " << bishtarin << "\n";
16    return 0;
17 }
```

خروجی برنامه

برنامه را با تعداد 4 دانشجو و نمرات 5، 12.5، 19 و 16.75 اجرا می‌کنیم:

```
daneshjooyan chand nafar hastand? 4
yek nomreh vared konid:5
yek nomreh vared konid:12.5
yek nomreh vared konid:19
yek nomreh vared konid:16.75
bishtarin nomreh = 19
```

در جدول زیر مقادیر متغیرهای برنامه پس از هر دور اجرای حلقه نشان داده شده است:

bishtarin	tedad	i	nomreh	
0	4	0	نامعلوم	**مقدار اولیه یا مقدار دریافتی از کاربر**
5	4	1	5	**پس از اجرای دستورات حلقه**
12.5	4	2	12.5	**پس از اجرای دستورات حلقه**
19	4	3	19	**پس از اجرای دستورات حلقه**
19	4	4	16.75	**پس از اجرای دستورات حلقه**

همان طورکه در سطر پنجم از جدول مشاهده می‌کنید، با برابر شدن مقدار متغیر `i` با مقدار متغیر `tedad`، اجرای حلقه پایان می‌پذیرد در حالی‌که متغیر `bishtarin` حاوی مقدار 19 است.

توضیح برنامه

در این برنامه همواره بیشترین نمره دریافتی در متغیری قرار دارد که مقدار آن با نمره‌های دریافتی جدید مقایسه می‌شود. در صورت بیشتر بودن نمره جدید، مقدار متغیر بیشترین نمره تغییر می‌کند.

در **خط شماره (۵)** برنامه، متغیری اعشاری به نام `bishtarin` تعریف شده است که برای نگهداری بیشترین نمره استفاده می‌شود. مسلم است که در ابتدای اجرای برنامه، این متغیر باید حاوی کمترین مقدار در بازه معتبر نمره‌ها یعنی عدد صفر باشد.

در **خط شماره (۱۳)** مقدار نمره دریافتی در متغیر `nomreh` با بیشترین نمره دورهای قبلی حلقه که در متغیر `bishtarin` قرار دارد، مقایسه می‌شود. در صورتی که نمره جاری بزرگتر از بیشترین نمره باشد، در **خط شماره (۱۴)** مقدار متغیر `bishtarin` برابر با این نمره خواهد شد و در غیر اینصورت بدون تغییر باقی خواهد ماند. با توجه به این که مقدار اولیه متغیر `bishtarin` برابر با 0 است، قطعا اولین نمره دریافتی به عنوان بیشترین نمره در نظر گرفته شده و در این متغیر قرار خواهد گرفت.

تمرین: سعی کنید برنامه محاسبه کمترین نمره بین نمرات دانشجویان را بنویسید.

مثال ۲۹

برنامه‌ای بنویسید که عمل ضرب دو عدد صحیح مثبت `x` و `y` (یعنی $x \times y$) را با جمع‌های متوالی انجام دهد (برای حل این مساله کافی است که عدد `x` را به تعداد `y` بار با خودش جمع کنید).

```
1 #include <iostream>
2 using namespace std;
3 //Seeyo Dovvomin Barnamye C++
4 int main(){
5     int x, y, z = 0;
6     int i;
```

```
7     cout << "x va y ra vared konid:";
8     cin >> x >> y;
9     for(i = 0 ; i < y ; i++)
10          z += x;
11    cout << x << " * " <<  y << " = " << z << "\n";
12    return 0;
13 }
```

خروجی برنامه

برنامه را با اعداد 2 و 3 اجرا می‌کنیم:

```
x va y ra vared konid:2 3
2 * 3 = 6
```

در جدول زیر مقادیر متغیرهای برنامه پس از هر دور اجرای حلقه نشان داده شده است:

	i	y	z	x
مقدار اولیه یا مقدار دریافتی از کاربر	0	3	0	2
پس از اجرای دستورات حلقه	1	3	2	2
پس از اجرای دستورات حلقه	2	3	4	2
پس از اجرای دستورات حلقه	3	3	6	2

ستون سوم از جدول، حاصل جمع مقدار قبلی متغیر z را با متغیر x نشان می‌دهد. همان طورکه در سطر چهارم مشاهده می‌کنید، با برابر شدن مقدار متغیر i با مقدار متغیر y، اجرای حلقه پایان می‌پذیرد در حالی‌که متغیر z حاوی مقدار 6 به عنوان حاصلضرب مورد نظر است.

توضیح برنامه

همان طورکه مشاهده می‌کنید، برای حل این مثال از حلقه استفاده شده است. زیرا عمل جمع‌های متوالی متغیر x با خودش، عملی تکراری است که به تعداد y بار رخ می‌دهد.

در **خط شماره (۵)**، متغیرهای x و y برای نگهداری دو عدد دریافتی و متغیر z به عنوان یک متغیر کمکی در محاسبه جمع‌های متوالی متغیر x به‌کار می‌روند. در انتهای حلقه مقدار موجود در این متغیر، بیانگر حاصل ضرب دو متغیر x و y خواهد بود. بدیهی است که مقدار اولیه متغیر z باید برابر با صفر باشد.

در **خط شماره (۶)** همانند مثال‌های قبلی، متغیر i را به عنوان شمارنده حلقه تعریف نمودیم. زیرا لازم است تعداد دفعات انجام عمل جمع را نگهداری کنیم.

در **خط شماره (۹)**، دستور for را مشاهده می‌کنید که در هر بار اجرای آن، مقدار متغیر i با مقدار متغیر y مقایسه می‌شود. زیرا همان طور که اشاره شد، جمع‌های متوالی x با خودش به تعداد y بار انجام می‌شود.

خط شماره (۱۰) حاوی دستوری است که در هر بار اجرای حلقه، مقدار موجود در متغیر z را با مقدار موجود در متغیر x جمع می‌کند و معادل با دستور زیر است:

```
z = z + x;
```

پس از پایان یافتن حلقه، در **دستور شماره (۱۱)** مقدار متغیر z به عنوان حاصل ضرب متغیرهای x و y در خروجی استاندارد چاپ می‌شود.

تمرین: سعی کنید برنامه عمل به توان رساندن دو عدد صحیح مثبت x و y (یعنی x^y) را به‌کمک ضرب‌های متوالی بنویسید.

مثال ۳۰

برنامه‌ای بنویسید که ۱۰ جمله اول سری فیبوناچی را در خروجی چاپ کند (دو جمله اول سری فیبوناچی برابر با 1 است و پس از آن هر جمله این سری برابر با مجموع دو جمله قبلی خود است).

```
1 #include <iostream>
2 using namespace std;
3 //Seeyo Sevvomin Barnamye C++
4 int main(){
5     int x1 = 1, x2 = 1, x3;
6     int i;
7     cout << x1 << " " << x2;
8     for(i = 0 ; i < 8 ; i++){
9           x3 = x1 + x2;
10          cout << " " << x3;
11          x1 = x2;
12          x2 = x3;
13    }
14    return 0;
15 }
```

خروجی برنامه

```
1 1 2 3 5 8 13 21 34 55
```

در جدول زیر مقادیر متغیرهای برنامه پس از اجرای هر دور از حلقه نشان داده شده است:

	x3	x1	x2	i
مقدار اولیه یا مقدار دریافتی از کاربر	نامعلوم	1	1	0
پس از اجرای دستورات حلقه	2	1	2	1
پس از اجرای دستورات حلقه	3	2	3	2
پس از اجرای دستورات حلقه	5	3	5	3
پس از اجرای دستورات حلقه	8	5	8	4
پس از اجرای دستورات حلقه	13	8	13	5
پس از اجرای دستورات حلقه	21	13	21	6
پس از اجرای دستورات حلقه	34	21	34	7
پس از اجرای دستورات حلقه	55	34	55	8

مقدار x3 در ستون دوم جدول برابر با حاصل جمع مقادیر قبلی متغیرهای x1 و x2 است. مقادیر جدید متغیرهای x1 و x2 در ستون‌های سوم و چهارم جدول به ترتیب برابر با مقادیر قبلی متغیرهای x2 و x3 است. حلقه زمانی پایان می‌یابد که مقدار متغیر i برابر با عدد 8 شود.

توضیح برنامه

خط شماره (۵) شامل تعریف متغیرهای صحیح x1 با مقدار اولیه 1، x2 با مقدار اولیه 1 و x3 است. متغیر x1 برای نگهداری جمله اول سری، متغیر x2 برای نگهداری جمله دوم و متغیر x3 برای نگهداری جمله جدید آن به‌کار می‌روند.

در **خط شماره (۷)** دو جمله اول سری فیبوناچی بدون هیچگونه محاسبه‌ای در خروجی چاپ می‌شوند.

خط شماره (۸)، آغاز تعریف یک ساختار for برای تولید ۸ جمله بعدی سری فیبوناچی است (چون دو جمله اول آن قبلا چاپ شده‌اند).

در **خط شماره (۹)**، جمله جدید سری فیبوناچی از حاصل جمع دو جمله قبلی این سری‌که در متغیرهای x1 و x2 قرار دارند، محاسبه شده و در متغیر x3 قرار می‌گیرد. سپس در **خط شماره (۱۰)** این جمله به همراه یک فاصله از جمله قبلی خود در خروجی استاندارد چاپ می‌شود.

خطوط شماره (۱۱) و (۱۲)، مقادیر متغیرهای x1 و x2 برای دورهای بعدی حلقه محاسبه می‌کنند.

مثال ۳۱

برنامه‌ای بنویسید که یک عدد صحیح از ورودی دریافت کند و تشخیص دهد که اول است یا نه (عدد اول عددی است که به غیر از خودش و یک بر هیچ عدد دیگری بخش‌پذیر نباشد. اگر عدد ۱ و خود عدد را کنار بگذاریم، با

تقسیم‌های متوالی عدد دریافتی بر اعداد بازه ۲ تا یک واحد کمتر از عدد، می‌توان وجود مقسوم علیه‌های عدد را بررسی کرد. در صورتی که حتی یک مقسوم علیه وجود داشته باشد، عدد مورد نظر، اول نخواهد بود).

```
1 #include <iostream>
2 using namespace std;
3 //Seeyo Chaharomin Barnamye C++
4 int main(){
5     int adad, tedad = 0, i;
6     cout << "adad mored nazar ra vared konid: ";
7     cin >> adad;
8     for(i = 2 ; i < adad ; i++)
9          if(adad % i == 0){
10              tedad++;
11              break;
12         }
13    if(adad < 2 || tedad > 0)
14         cout << "adad mored nazar avval nist.\n";
15    else
16         cout << "adad mored nazar avval ast.\n";
17    return 0;
18 }
```

خروجی برنامه

برنامه را با عدد 5 اجرا می‌کنیم:

```
adad mored nazar ra vared konid: 5
adad mored nazar avval ast.
```

در جدول زیر مقادیر متغیرهای برنامه پس از اجرای هر دور از حلقه نشان داده شده است:

i	tedad	adad	
2	0	5	**مقدار اولیه یا مقدار دریافتی از کاربر**
3	0	5	**پس از اجرای دستورات حلقه**
4	0	5	**پس از اجرای دستورات حلقه**
5	0	5	**پس از اجرای دستورات حلقه**

مقدار موجود در ستون سوم جدول فوق در صورتی که مقدار موجود در متغیر adad بر مقدار موجود در متغیر i بخش‌پذیر باشد، یک واحد افزایش می‌یابد. اما همان طور که مشاهده می‌کنید، حلقه برای تمام اعداد بازه ۲ تا یک واحد کمتر از مقدار موجود در متغیر adad اجرا شده است. اما چون هیچ مقسوم علیه‌ای وجود ندارد، مقدار متغیر tedad تغییر نکرده و عدد مورد نظر اول خواهد بود.

برنامه را با یک عدد خیلی بزرگ همانند 2147483646 اجرا می‌کنیم:

```
adad mored nazar ra vared konid: 2147483646
adad mored nazar avval nist.
```

در جدول زیر مقادیر متغیرهای برنامه پس از اجرای هر دور از حلقه نشان داده شده است:

	adad	tedad	i
مقدار اولیه یا مقدار دریافتی از کاربر	2147483646	0	2
پس از اجرای دستورات حلقه	2147483646	1	2

همان طور که مشاهده می‌کنید، به محض یافتن اولین مقسوم علیه، به دلیل استفاده از دستور break اجرای حلقه خاتمه یافته و بقیه موارد بررسی نمی‌شوند.

توضیح برنامه

دستور break که در بخش‌های قبلی جزوه معرفی شد، یکی از دستورات انتقال کنترل غیر شرطی است که بدون بررسی هیچ گونه شرطی، کنترل اجرای برنامه را از یک ساختار تصمیم switch و یا یک ساختار تکرار مانند دستور for خارج می‌کند. استفاده از این دستور موجب کاهش چشمگیر زمان اجرای برنامه فوق برای اعداد خیلی بزرگی خواهد بود که اولین مقسوم علیه آن‌ها، عدد بسیار کوچکی است.

در **خط شماره (۵)**، علاوه بر متغیر adad که عدد دریافتی از کاربر را نگهداری می‌کند، متغیر tedad برای نگهداری تعداد مقسوم علیه‌های متغیر adad و متغیر i برای تولید اعداد بازه $[2, adad - 1]$ تعریف شده‌اند.

خط شماره (۸)، آغازگر حلقه for برای بررسی بخش‌پذیر بودن مقدار موجود در متغیر adad بر اعداد بازه $[2, adad - 1]$ است.

در **خط شماره (۹)** بخش‌پذیر بودن عدد دریافتی بر مقدار موجود در متغیر i بررسی می‌شود. در صورت بخش-پذیری، مقدار متغیر tedad در **خط شماره (۱۰)** یک واحد افزایش می‌یابد و دستور break در **خط شماره (۱۱)** اجرا می‌شود. در غیر اینصورت مقدار متغیر i برای اجرای دور بعدی حلقه یک واحد افزایش می‌یابد.

پس از اتمام اجرای حلقه، **خطوط شماره (۱۳) تا (۱۶)** وظیفه بررسی و چاپ اول بودن عدد موجود در متغیر adad را بر عهده دارند.

طبق **خط شماره (۱۳)** در صورتی که عدد مورد نظر کاربر کوچکتر از عدد 2 باشد و یا بر عددی در بازه $[2, adad-1]$ بخش‌پذیر نباشد، عدد اول است و پیغام **خط شماره (۱۴)** چاپ خواهد شد. در غیر اینصورت پیغام **خط شماره (۱۶)** نشان داده می‌شود.

در تمام مثال‌هایی که برای حلقه‌ها بررسی شد، تعداد دفعات اجرای دستورات حلقه از همان ابتدا مشخص بود. به عنوان مثال در محاسبه میانگین N نمره، تعداد دفعات اجرای حلقه برابر با N است. یا در مثال محاسبه حاصل ضرب دو عدد به کمک جمع‌های متوالی، تعداد دفعات انجام عمل جمع برابر با عدد دوم است. اما در برخی از مسائل ممکن است تعداد دفعات اجرای حلقه عدد ثابت و مشخصی نباشد. برای این گونه از موارد بهتر است از دستور `while` یا `do while` استفاده کرد. در ادامه هر یک از این ساختارها به طور مختصر شرح داده شده‌اند.

۲ـ۱۶ـ۲ـ۱ ساختار تکرار `while`

ساختار این دستور به شکل زیر است:

```
While(عبارت شرطی){
        دستور ۱;
        دستور ۲;
        ...
}
```

در این ساختار ابتدا کلمه کلیدی `while`، سپس شرط حلقه و پس از آن دستورات بدنه حلقه آورده می‌شوند. این دستورات زمانی اجرا خواهند شد که عبارت شرطی ابتدای حلقه ارزش درستی داشته باشد. اگر تعداد دستورات بدنه حلقه بیش از یک دستور باشد، باید آن‌ها در داخل دو علامت `{}` قرار داد. در ادامه می‌توانید نحوه استفاده از این دستور را برای مثال بررسی نمرات دانشجویان مشاهده کنید:

```
1 #include <iostream>
2 using namespace std;
3 //Seeyo Chaharomin Barnamye C++
4 int main(){
5     double nomreh;
6     int i = 0, tedad;
7     cout << "daneshjooyan chand nafar hastand? ";
8     cin >> tedad;
9     while(i < tedad){
10          cout << "yek nomreh vared konid:";
```

```
11          cin >> nomreh;
12          if(nomreh >= 10)
13                cout << "ghabool ast.\n";
14          else
15                cout << "mardood ast.\n";
16          i++;
17    }
18    return 0;
19 }
```

همان طور که مشاهده می‌کنید، دستور while و for بسیار شبیه به هم هستند، با این تفاوت که عمل تعریف و مقداردهی اولیه شمارنده حلقه در دستور while قبل از حلقه و عمل افزایش مقدار شمارنده در داخل حلقه انجام می‌شود. بنابراین عملا تمام برنامه‌هایی که با دستور for پیاده‌سازی می‌شوند، با دستور while نیز قابل پیاده‌سازی هستند (وبالعکس). با توجه به ساختار دستور while، کاربرد اصلی این دستور بیشتر برای مواردی است که تعداد دفعات اجرای حلقه از همان ابتدا مشخص نشده باشد، بنابراین نیازی به تعریف شمارنده و تغییر مقدار آن نخواهیم داشت. در ادامه چند مثال را آورده شده است.

مثال ۳۲

برنامه‌ای بنویسید که یک عدد صحیح از ورودی دریافت کند و مقلوب ارقام آن را در خروجی چاپ کند (برای این منظور کافی است که عدد دریافتی را به صورت متوالی بر عدد ۱۰ تقسیم کرده و باقیمانده تقسیم‌ها را در خروجی چاپ کنیم).

```
1 #include <iostream>
2 using namespace std;
3 //Seeyo Panjomin Barnamye C++
4 int main(){
5     int adad, ragham;
6     cout << "adad mored nazar ra vared konid: ";
7     cin >> adad;
8     cout << "maghloob adad mored nazar = ";
9     while(adad > 0){
10          ragham = adad % 10;
11          cout << ragham;
12          adad /= 10
```

```
13    }
14    return 0;
15 }
```

خروجی برنامه

برنامه را با عدد 123456 اجرا می‌کنیم:

```
adad mored nazar ra vared konid: 123456
maghloob adad mored nazar = 654321
```

در جدول زیر مقادیر متغیرهای برنامه پس از هر دور اجرای حلقه نشان داده شده است:

adad	ragham	
123456	نامعلوم	**مقدار اولیه یا مقدار دریافتی از کاربر**
12345	6	**پس از اجرای دستورات حلقه**
1234	5	**پس از اجرای دستورات حلقه**
123	4	**پس از اجرای دستورات حلقه**
12	3	**پس از اجرای دستورات حلقه**
1	2	**پس از اجرای دستورات حلقه**
0	1	**پس از اجرای دستورات حلقه**

ستون دوم از جدول فوق، باقیمانده تقسیم مقدار موجود در متغیر adad را بر عدد 10 نشان می‌دهد و ستون سوم آن، خارج قسمت مقدار قبلی این متغیر بر عدد 10 است. همان طورکه سطر هفتم از جدول نشان می‌دهد، با برابر شدن مقدار متغیر adad با عدد 0 اجرای حلقه پایان می‌پذیرد.

توضیح برنامه

در **خط شماره (۵)** دو متغیر عدد صحیح با نام‌های adad و ragham تعریف شده است که به ترتیب برای نگهداری عدد دریافتی از کاربر و باقیمانده تقسیم آن بر عدد ۱۰ به‌کار می‌روند.

با توجه به اینکه ممکن است کاربر عدد 0 را به عنوان عدد مورد نظر خود وارد کند، در **خط شماره (۹)** این مساله را بررسی کرده و در **خط شماره (۱۰)** مقدار 0 را به عنوان مقلوب عدد دریافتی چاپ می‌کنیم (یادآوری: دستور if(!adad) معادل با دستور if(adad==0) است).

خطوط شماره (۱۱) تا (۱۴) هسته اصلی الگوریتم را تشکیل می‌دهند.

خط شماره (۱۱) شامل دستور for و آغازگر حلقه است. همان طورکه مشاهده می‌کنید، این دستور فقط شامل دو بخش شرط و گام حلقه است. زیرا در این مثال، نیازی به وجود یک متغیر مجزا تحت عنوان شمارنده نیست و

عدد دریافتی از کاربر همزمان وظیفه شمارش تعداد دفعات اجرای حلقه را نیز بر عهده دارد. گام حلقه، دستوری است که خارج قسمت تقسیم متغیر adad بر عدد 10 را برای استفاده در دورهای بعدی حلقه محاسبه کرده و در متغیر adad قرار می‌دهد. این دستور معادل با دستور زیر است:

```
adad = adad / 10;
```

در **خط شماره (۱۲)** با محاسبه باقیمانده متغیر adad بر عدد 10، رقم یکان آن جدا شده و در متغیر ragham قرار می‌گیرد.

در **خط شماره (۱۳)** محتوای متغیر ragham در خروجی استاندارد چاپ می‌شود. اما برای این که بتوان رقم‌های جدا شده عدد را کنار یکدیگر چاپ کرد، نباید از کاراکترهای کنترلی \n، \t و یا حتی یک فاصله در دستور cout استفاده کرد.

تمرین: سعی کنید برنامه شمارش تعداد رقم‌های عدد دریافتی را نیز بنویسید.

۳ـ۱۶ـ۲ـ۱ ساختار تکرار do while

ساختار این دستور به شکل زیر است:

```
do{
      دستور ۱;
      دستور ۲;
      ...
}while(عبارت شرطی);
```

همان طور که مشاهده می‌کنید، این ساختار با کلمه کلیدی do آغاز می‌شود و پس از آن بلافاصله دستور یا دستوراتی که قرار است تکرار شوند، نوشته می‌شوند. در صورتی که تعداد این دستورات بیش از یک دستور باشد، باید آن‌ها را در بین دو علامت } و { قرار داد. در انتها کلمه کلیدی while به همراه یک عبارت شرطی آورده می‌شود. هنگام اجرای برنامه تمام دستورات بعد از do اجرا می‌شوند، مگر این که عبارت شرطی حلقه ارزش درستی نداشته باشد. در ادامه می‌توانید نحوه استفاده از این دستور را برای مثال بررسی نمرات دانشجویان مشاهده کنید:

```
1 #include <iostream>
2 using namespace std;
3 //Seeyo Haftomin Barnamye C++
4 int main(){
5     double nomreh;
6     int i = 0, tedad;
```

```
7     cout << "daneshjooyan chand nafar hastand? ";
8     cin >> tedad;
9     do{
10          cout << "yek nomreh vared konid:";
11          cin >> nomreh;
12          if(nomreh >= 10)
13               cout << "ghabool ast.\n";
14          else
15               cout << "mardood ast.\n";
16          i++;
17    } while(i < tedad);
18    return 0;
19 }
```

این دستور نیز بسیار شبیه به دستور `for` و `while` است. اما تفاوت آن در این است که شرط حلقه در انتهای آن بررسی می‌شود. به عبارت دیگر دستورات بدنه حلقه حداقل یکبار اجرا خواهند، حتی اگر شرط حلقه در همان دور اول نادرست باشد. به عنوان مثال فرض کنید که در برنامه فوق، کاربر تعداد دانشجویان را 0 وارد کند. در این حالت تا رسیدن به دستور `while` و بررسی شرط حلقه، تمام دستورات داخل حلقه اجرا خواهند شد. در حالی از نظر منطقی، نباید چنین اتفاقی رخ می‌داد. بنابراین در استفاده از این دستور دقت کرد.

۴ـ۱۶ـ۲ـ۱ ساختارهای تکرار نامعین (بی‌نهایت)

همه مثال‌هایی که تاکنون بررسی کردیم در دسته ساختارهای تکرار معین قرار می‌گیرند. زیرا در تمام آن‌ها حداکثر تعداد دفعات اجرای حلقه از همان ابتدا معلوم شده بود. اما در برخی از مثال‌ها ممکن است نیازی به مشخص کردن تعداد دفعات تکرار نباشد و به عنوان مثال ادامه یا خاتمه حلقه وابسته به نظر کاربر باشد. مثلا فرض کنید که در برنامه تشخیص قبولی یا مردودی دانشجویان یک درس، تعداد دانشجویان از همان ابتدا معلوم نشده باشد و حلقه تا زمانی ادامه یابد که کاربر بخواهد نمره دیگری وارد کند. این نوع از ساختارها را *ساختارهای تکرار نامعین یا بی‌نهایت* گویند. در این ساختارها، عبارت شرطی حلقه همواره ارزش درستی دارد. به عنوان مثال تمام دستورهای زیر بیانگر حلقه تکرار نامعین هستند:

```
for(;;){
    ...
}

while(1){
```

```
    ...
}

do{
    ...
}while(1);
```

البته معمولا از دستور while برای حلقه تکرار بی‌نهایت استفاده می‌شود. در ادامه، برنامه تشخیص قبولی و مردودی دانشجویان را با استفاده از یک حلقه بی‌نهایت بازنویسی کرده‌ایم:

```
1 #include <iostream>
2 using namespace std;
3 //Seeyo Hashtomin Barnamye C++
4 int main(){
5     double nomreh;
6     string pasokh;
7     while(1){
8           cout << "nomreh ra vared konid:";
9           cin >> nomreh;
10          if(nomreh >= 10)
11                cout << "ghabool ast.\n";
12          else
13                cout << "mardood ast.\n";
14          cout << "nomreye digari vojood darad (yes/no)? ";
15          cin >> pasokh;
16          if(pasokh == “no”)
17                break;
18    }
19    cout << "payan.\n";
20    return 0;
21 }
```

خروجی برنامه

برنامه را با سه نمره 5 و 15 و 11.75 اجرا می‌کنیم:

```
nomreh ra vared konid:5
mardood ast.
nomreye digari vojood darad (yes/no)? yes
nomreh ra vared konid:15
ghabool ast.
nomreye digari vojood darad (yes/no)? yes
nomreh ra vared konid:11.75
mardood ast.
nomreye digari vojood darad (yes/no)? no
payan.
```

در جدول زیر مقادیر متغیرهای برنامه پس از هر دور اجرای حلقه نشان داده شده است:

	مقدار متغیر nomreh	مقدار متغیر pasokh
مقدار اولیه یا مقدار دریافتی از کاربر	نامعلوم	نامعلوم
پس از اجرای دستورات حلقه	5	yes
پس از اجرای دستورات حلقه	15	yes
پس از اجرای دستورات حلقه	11.75	no

همان طورکه در سطر سوم از جدول مشاهده می‌کنید، با واردکردن مقدار no برای متغیر pasokh اجرای حلقه پایان می‌پذیرد.

توضیح برنامه

در برنامه فوق عمل دریافت و بررسی نمره کاربر می‌تواند تا بی‌نهایت ادامه یابد. مگر این که کاربر با توجه به سوالی که از او پرسیده می‌شود، خواستار پایان دادن به اجرای دستورات باشد.

در **خط شماره (۶)** یک متغیر رشته‌ای تحت عنوان pasokh برای نگهداری پاسخ کاربر در نظر گرفته شده است.

خط شماره (۷) آغازگر یک ساختار while بی‌نهایت است که در طی آن یک نمره از کاربر دریافت شده و بررسی می‌شود.

در **خطوط شماره (۱۴) و (۱۵)** سوالی مبنی بر تمایل کاربر به ادامه حلقه از او پرسیده می‌شود و پاسخ آن در متغیر pasokh قرار داده می‌شود.

در **خط شماره (۱۶)** پاسخ کاربر بررسی می‌شود. در صورتی که کاربر مقدار no را به عنوان پاسخ منفی وارد کرده باشد، دستور break در **خط شماره (۱۷)** اجرا شده و کنترل از حلقه خارج می‌شود. در غیر اینصورت، حلقه به اجرای خود ادامه خواهد داد.

در **خط شماره (۱۹)** دستوری مبنی بر چاپ پیغام پایان در خروجی استاندارد اجرا می‌شود.

در برنامه فوق فرض کرده‌ایم که کاربر خود می‌داند که باید نمره‌ها را از بازه [1,20] انتخاب کند. به همین دلیل صحت نمره دریافت شده را بررسی نکرده‌ایم. اما برای تکمیل برنامه، می‌توان کاربر را مجبور کرد که حتما نمره صحیحی وارد کند. برای حل این مساله راه حل‌های مختلفی می‌تواند وجود داشته باشد، اما با توجه به این که از یک حلقه بی‌نهایت برای نوشتن برنامه استفاده کرده‌ایم، می‌توان کنترل اجرای برنامه را به عقب بازگرداند و از کاربر خواست که نمره دیگری وارد کند. برای این منظور در زبان C++ دستوری با نام دستور continue وجود دارد که همانند دستور break یک دستور انتقال کنترل غیر شرطی است. با این تفاوت که کنترل اجرا را به ابتدای حلقه باز می‌گرداند. در این حالت برنامه فوق به شکل زیر بازنویسی می‌شود:

```
1 #include <iostream>
2 using namespace std;
3 //Seeyo Nohomin Barnamye C++
4 int main(){
5     float nomreh;
6     string pasokh;
7     while(1){
8           cout << "nomreh ra vared konid:";
9           cin >> nomreh;
10          if(nomreh < 0 || nomreh > 20){
11                cout << "nomreh namotabar ast,";
12                << "lotfan deghat konid.\n";
13                continue;
14          }
15          if(nomreh >= 10)
16                cout << "ghabool ast.\n";
17          else
18                cout << "mardood ast.\n";
19          cout << "nomreye digari vojood darad (yes/no)? ";
20          cin >> pasokh;
```

```
21          if(pasokh == “no”)
22                  break;
23      }
24      cout << "payan.\n";
25      return 0;
26 }
```

توضیح برنامه

در این برنامه پس از دریافت یک نمره از کاربر، در **خط شماره (۱۰)** صحت آن با مقایسه مقدار متغیر `nomreh` با اعداد `0` و `20` بررسی می‌شود. در صورتی که مقدار متغیر خارج از بازه [1,20] باشد، پیغامی متناسب در **خطوط شماره (۱۱) و (۱۲)** چاپ می‌شود. سپس در **خط شماره (۱۳)** کنترل اجرای برنامه به ابتدای حلقه باز می‌گردد تا کاربر نمره دیگری وارد کند. تا زمانی که مقدار متغیر `nomreh` در بازه مورد نظر نباشد، دستور `continue` مانع از اجرای دستورات بعدی خواهد شد.

نکته: وجه مشترک دستور `break` و دستور `continue` در این است که به محض اجرای هر کدام از آن‌ها، دستورات بعدی حلقه اجرا نخواهد شد. اما نقطه تفاوت آن‌ها در این است که دستور `break` موجب انتقال کنترل به خارج از حلقه می‌شود. در حالی که دستور `continue` اجرا را به ابتدای حلقه منتقل می‌کند.

۵ـ۱۶ـ۲ـ۱ ساختارهای تکرار تو در تو

همان طور که قبلا هم اشاره شد، دستوراتی که در بدنه یک ساختار تکرار آورده می‌شوند، می‌توانند هر دستور معتبری در زبان C++ باشند. بنابراین ساختارهای تکرار خود نیز می‌توانند در بدنه ساختار تکرار دیگر قرار گیرند. در این حالت یک ساختار تکرار تو در تو خواهیم داشت که می‌تواند در حل برخی از مسائل مفید واقع شود. در ادامه برای فهم بهتر این مطلب، مثال‌هایی آورده شده است.

مثال ۳۳

برنامه‌ای بنویسید که با دریافت تعدادی عدد صحیح از ورودی استاندارد، اول بودن یا نبودن آن‌ها را تشخیص دهد (برای حل این مساله کافی است که با یک حلقه بی‌نهایت تعدادی عدد از ورودی دریافت کرده و با یک حلقه داخلی اول بودن یا نبودن آن را مشخص کنیم).

```
1 #include <iostream>
2 using namespace std;
3 //Chelomin Barnamye C++
4 int main(){
5     int adad, tedad = 0, i = 2;
```

```
6     string pasokh;
7     while(1){
8           cout << "adad mored nazar ra vared konid: ";
9           cin >> adad;
10          for(i = 2 ; i < adad ; i++)
11                if(adad % i == 0){
12                      tedad++;
13                      break;
14                }
15          if(adad < 2 || tedad > 0)
16                cout << "adad mored nazar avval nist.\n";
17          else
18                cout << "adad mored nazar avval ast.\n";
19          tedad = 0;
20          cout << "edame midahid (yes/no)? ";
21          cin >> pasokh;
22          if(pasokh == “no”)
23                break;
24    }
25    cout << "payan.\n";
26    return 0;
27    }
```

خروجي برنامه

برنامه را با اعداد 10 و 11 اجرا می‌کنیم:

```
adad mored nazar ra vared konid: 10
adad mored nazar avval nist.
edame midahid (yes/no)? yes
adad mored nazar ra vared konid: 11
adad mored nazar avval ast.
edame midahid (yes/no)? no
```

```
payan.
```

در جدول زیر مقادیر متغیرهای برنامه پس از اجرای هر دور از حلقه بیرونی و داخلی نشان داده شده است:

	adad	tedad	i	pasokh
مقدار اولیه یا مقدار دریافتی از کاربر	نامعلوم	0	2	نامعلوم
پس از اجرای دستورات حلقه داخلی	10	1	2	نامعلوم
پس از اجرای دستورات حلقه بیرونی	10	0	2	yes
پس از اجرای دستورات حلقه داخلی	11	0	3	yes
پس از اجرای دستورات حلقه داخلی	11	0	4	yes
پس از اجرای دستورات حلقه داخلی	11	0	5	yes
پس از اجرای دستورات حلقه داخلی	11	0	6	yes
پس از اجرای دستورات حلقه داخلی	11	0	7	yes
پس از اجرای دستورات حلقه داخلی	11	0	8	yes
پس از اجرای دستورات حلقه داخلی	11	0	9	yes
پس از اجرای دستورات حلقه داخلی	11	0	10	yes
پس از اجرای دستورات حلقه بیرونی	11	0	2	no

در جدول فوق اجرای حلقه بیرونی و حلقه داخلی به صورت تواما نشان داده شده است. همان طور که مشاهده می‌کنید، زمانی که متغیر adad شامل مقدار 10 است، حلقه داخلی فقط یکبار اجرا می‌شود. زیرا در همان دور اول، بخش‌پذیر بودن این عدد بر عدد 2 اثبات شده و دستور break منجر به خروج از حلقه داخلی می‌شود. در حلقه بیرونی مقدار متغیر tedad به مقدار اولیه خود یعنی مقدار 0 بازگردانده می‌شوند (سطر سوم جدول). زمانی که کاربر در انتهای حلقه بیرونی رشته "yes" را به عنوان مقدار متغیر pasokh وارد می‌کند، حلقه بی‌نهایت به اجرای خود ادامه می‌دهد و عدد بعدی را از کاربر درخواست می‌کند. به ازای عدد 11 حلقه داخلی به تعداد ۹ بار اجرا می‌شود. سپس حلقه بیرونی اجرا شده و با وارد کردن رشته "no" توسط کاربر، حلقه بی‌نهایت شکسته شده و اجرا پایان می‌یابد.

توضیح برنامه

همان طور که مشاهده می‌کنید برنامه فوق، ترکیب برنامه محاسبه اول بودن یک عدد با یک حلقه بی‌نهایت برای دریافت اعداد از ورودی است.

در **خطوط شماره (۵) و (۶)** متغیرهای مورد نیاز برنامه تعریف شده‌اند.

خط شماره (۷) آغاز تعریف حلقه تکرار بی‌نهایت برای دریافت اعداد مورد نظر کاربر است.

خط شماره (۱۰) آغاز حلقه‌ای است که وظیفه محاسبه تعداد مقسوم علیه‌های عدد مورد نظر را برعهده دارد.

در صورتی که عدد دریافتی بر یکی از اعداد بازه $[2, adad - 1]$ بخش‌پذیر باشد، **خطوط شماره (۱۲) و (۱۳)** اجرا شده و ادامه برنامه به **خط شماره (۱۵)** منتقل می‌شود. در اجرای حلقه for ادامه می‌یابد.

در **خطوط شماره (۱۹)** مقدار متغیر tedad برای استفاده در دورهای بعدی حلقه بیرونی و بررسی عدد بعدی کاربر، به حالت اولیه خود باز می‌گردد.

مثال ۳۴

برنامه‌ای بنویسید که جدول ضرب 10 در 10 را در خروجی چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 //Chehelo Yekomin Barnamye C++
4 int main(){
5     int i, j;
6     for(i = 1 ; i <= 10 ; i++) {
7           for(j = 1 ; j <= 10 ; j++)
8                 cout << i * j << "\t";
9           cout << "\n";
10    }
11    return 0;
12 }
```

خروجی برنامه

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

توضیح برنامه

در **خط شماره (۵)** دو متغیر صحیح `i` و `j` به ترتیب برای نگهداری تعداد دفعات تکرار حلقه بیرونی و داخلی تعریف شده است.

خط شماره (۶) آغاز تعریف یک حلقه `for` است که با اجرای آن، دستور `for` دوم در **خط شماره (۷)** و دستور `cout` در **خط شماره (۹)** به تعداد `10` بار اجرا می‌شوند. شمارنده این حلقه (یعنی متغیر `i`) اولین عدد عمل ضرب را برای جدول ضرب فراهم می‌کند.

خط شماره (۷) شامل تعریف ساختار `for` دوم برای فراهم کردن عدد دوم در عمل ضرب است.

خط شماره (۸)، حاصل ضرب متغیرهای `i` و `j` را به همراه یک `tab` (۸ فاصله) در خروجی چاپ می‌شود.

دستور موجود در **خط شماره (۹)** موجب می‌شود که پس از اجرای کامل حلقه `for` دوم، اشاره‌گر به خط بعدی منتقل شود.

۱۷ـ۲ـ۱ آرایه‌ها در زبان C++

در برخی از موارد، برای پیاده‌سازی یک برنامه لازم است چندین مقدار به طور همزمان در حافظه ذخیره و مورد بررسی قرار گیرند. به عنوان مثال فرض کنید که می‌خواهیم میانگین و واریانس ۱۰ عدد صحیح را بیابیم. برای محاسبه میانگین، کافی است که پس از دریافت هر عدد از کاربر، مقدار آن را به مجموع کل بیفزاییم و پس از اتمام حلقه، مقدار مجموع را بر تعداد اعداد تقسیم کنیم. طبق روشی که در آمار برای واریانس وجود دارد، محاسبه واریانس نیازمند بدست آوردن حاصل تفاضل میانگین اعداد از هر عدد است. بنابراین لازم است تک تک اعدادی که از کاربر دریافت شده اند، بار دیگر مورد دسترسی قرار گیرند. برای این منظور ساده‌ترین راه حل آن است که تعداد ۱۰ متغیر عدد صحیح تعریف کنیم و هر عدد دریافتی را در متغیر مربوطه قرار دهیم. اما این راه حل برای حالتی که بخواهیم برنامه را برای تعداد بیشتری از اعداد توسعه دهیم، منطقی نخواهد بود. برای حل این گونه از مسائل، می‌توان از *آرایه‌ها* در C++ استفاده کرد. آرایه، مجموعه پیوسته و پشت سر همی از خانه‌های حافظه است که همگی دارای نوع یکسان و نام مشترک هستند. یک آرایه به شکل زیر تعریف می‌شود:

نوع عناصر آرایه نام آرایه [اندازه آرایه] ;

همان طور که در این تعریف مشاهده می‌کنید، ابتدا نوع عناصر آرایه، سپس نام آرایه و در نهایت اندازه یا همان تعداد عناصر آرایه در داخل دو علامت [] آورده می‌شود. نوع عناصر آرایه می‌تواند یکی از انواع شناخته شده در زبان C++ باشد (مثال: `int`، `float`، `char`، `long` و ...). نام آرایه همانند اسامی متغیرها تعریف می‌شود و اندازه آرایه یک عدد صحیح مثبت خواهد بود. همانند متغیرها، مقادیر عناصر آرایه را نیز می‌توان به هنگام مقداردهی اولیه تعیین کرد و یا در طی یک دستور ورودی از کاربر دریافت کرد.

برای دسترسی به هریک از خانه‌ها (یا همان عناصر آرایه)، کافی است که نام آرایه را به همراه یک عدد صحیح مثبت که بیانگر محل قرارگیری عنصر مورد نظر در آرایه است، به کار ببریم. به این عدد صحیح که برای دسترسی به یک عنصر از آرایه استفاده می‌شود، *اندیس آرایه* گفته می‌شود. اندیس آرایه می‌تواندعدد صفر تا یک واحد کمتر از طول

آرایه باشد. با توجه به این که یک آرایه معمولا شامل بیش از یک عنصر است، برای بررسی همه عناصر آن نیاز به یک ساختار حلقه خواهیم داشت. در ادامه برای شرح بهتر این مفاهیم تعدادی مثال آورده شده است.

مثال ۳۵

برنامه‌ای بنویسید که یک آرایه ۱۰ تایی از اعداد صحیح تعریف کند و آن را مقداردهی اولیه کند. سپس مقادیر آن را یک به یک در خروجی استاندارد چاپ کند.

```
#include <iostream>
using namespace std;
int main(){
	int array[10] = {1,-2,36,412,-5,6,17,8,1099,-100};	(1)
	for(int i = 0 ; i < 10 ; i++){				(2)
		cout << array[i] << " ";			(3)
	}
	return 0;
}
```

خروجی برنامه

```
1 -2 36 412 -5 6 17 8 1099 -100
```

توضیح برنامه

در **خط شماره (۱)**، یک آرایه با نام `array` با اندازه `10` عنصر از نوع عدد صحیح تعریف شده است. همان طور که مشاهده می‌کنید، تعداد ۱۰ عدد صحیح در داخل دو علامت `{}` به عنوان مقادیر اولیه خانه‌های آرایه آورده شده است.

در**خط شماره (۲)**، یک دستور `for` تعریف شده است که عناصر آرایه را یک به یک در خروجی استاندارد چاپ می‌کند. در این دستور، متغیر `i` به عنوان اندیس آرایه محسوب می‌شود و با افزایش مقدار آن از 0 تا 9، می‌توان به تک تک عناصر آرایه دسترسی یافت. همان طور که مشاهده می‌کنید، تعریف این متغیر در داخل دستور `for` آورده شده است. این عمل در زبان C++ کاملا قانونی است، اما در زبان C منجر به ایجاد خطای کامپایلری خواهد شد.

در **خط شماره (۳)**، محتوای خانه `i`ام آرایه (مقدار `array[i]`) به همراه یک کاراکتر فاصله، در خروجی استاندارد نمایش داده می‌شود. بنابراین در هر دور از حلقه، یکی از عناصر آرایه در خروجی چاپ خواهد شد.

نکته: هنگام مقداردهی اولیه آرایه‌ها، در صورتی که تعداد مقادیری که در داخل دو علامت `{}` آورده می‌شوند کمتر از اندازه آرایه باشد، مقدار بقیه خانه‌های آرایه برابر با صفر خواهد شد.

نکته: هنگام مقداردهی اولیه آرایه‌ها، در صورتی که تعداد مقادیری که در داخل دو علامت {} آورده می‌شوند بیشتر از اندازه آرایه باشد، با یک خطای کامپایلری مواجه خواهیم شد.

نکته: در صورتی که برنامه بخواهد به یک اندیس خارج از محدوده آرایه یعنی 0 تا یک واحد کمتر از طول آرایه دسترسی یابد، هرچند که کامپایلر اعلام خطا نمی‌کند، اما نتیجه نادرستی بدست خواهد آمد.

نکته: اندازه آرایه همواره در طول برنامه ثابت است. به عبارت دیگر، پس از تعیین اندازه یک آرایه در هنگام تعریف آن، تغییر نخواهد کرد.

نکته: اندازه آرایه را می‌توان با یک *ثابت* نیز مشخص کرد. یک ثابت، نامی است که برای یک مقدار در نظر گرفته شده و با استفاده از دستور پیش‌پردازنده `define` تعریف می‌شود. ثوابت را می‌توان در بخش‌های مختلف برنامه استفاده کرد. به طوری که با دسترسی به یک ثابت، مقدار تعریف شده برای آن در نظر گرفته خواهد شد. به عنوان مثال اگر بخواهیم در برنامه فوق از یک ثابت برای نگهداری اندازه آرایه استفاده کنیم، باید آن را به شکل زیر تغییر دهیم:

```
#include <iostream>
#define SIZE 10                                              (1)
using namespace std;
int main(){
      int array[SIZE] = {1,-2,36,412,-5,6,17,8,1099,-100};   (2)
      for(int i = 0 ; i < SIZE ; i++){                       (3)
            cout << array[i] << " ";
      }
      return 0;
}
```

خروجی برنامه

خروجی این برنامه، همانند مثال قبل خواهد بود.

توضیح برنامه

در **خط شماره (۱)**، ثابتی تحت عنوان `SIZE` به وسیله دستور پیش‌پردازنده `define` تعریف شده است. این ثابت، نامی است که برای مقدار 10 در نظر گرفته‌ایم. پس هر قسمت از برنامه که به این ثابت دست یابد، مقدار 10 با آن جایگزین خواهد شد. به عنوان مثال، در **خط شماره (۲)** از مقدار ثابت `SIZE` به عنوان اندازه آرایه و در خط شماره (۳) برای شرط حلقه استفاده شده است.

نکته: برخلاف متغیرها، معمولا ثابت‌ها را با حروف بزرگ می‌نویسند.

مثال ۳۶

برنامه‌ای بنویسید که تعداد ۱۰ عدد صحیح را از ورودی استاندارد دریافت کند و آن‌ها را در یک آرایه قرار دهد. سپس عناصر آرایه را یک به یک در خروجی استاندارد چاپ کند.

```
#include <iostream>
using namespace std;
int main(){
	int array[10];
	for(int i = 0 ; i < 10 ; i++){
		cout << "yek adad vared konid:";
		cin >> array[i];                                    (1)
	}
	for(int i = 0 ; i < 10 ; i++){
		cout << array[i] << " ";
	}
	return 0;
}
```

خروجی برنامه

برنامه را با اعداد مثال قبلی اجرا می‌کنیم:

```
yek adad vared konid:1
yek adad vared konid:-2
yek adad vared konid:36
yek adad vared konid:412
yek adad vared konid:-5
yek adad vared konid:6
yek adad vared konid:17
yek adad vared konid:8
yek adad vared konid:1099
yek adad vared konid:-100
```

```
1 -2 36 412 -5 6 17 8 1099 -100
```

توضیح برنامه

این مثال مشابه با مثال قبلی است. با این تفاوت که مقادیر عناصر آرایه به کمک تابع `cin` در **خط شماره (۱)** از کاربر دریافت می‌شود.

مثال ۳۷

برنامه‌ای بنویسید که تعداد ۱۰ عدد صحیح را از ورودی استاندارد دریافت کند و آن‌ها را در یک آرایه قرار دهد. سپس عناصر آرایه را به ترتیب معکوس در خروجی استاندارد چاپ کند.

```
#include <iostream>
using namespace std;
int main(){
	int array[10];
	for(int i = 0 ; i < 10 ; i++){
		cout << "yek adad vared konid:";
		cin >> array[i];
	}
	for(int i = 0 ; i < 10 ; i++){                    (1)
		cout << array[9-i] << " ";                    (2)
	}
	return 0;
}
```

خروجی برنامه

برنامه را با اعداد مثال قبلی اجرا می‌کنیم:

```
yek adad vared konid:1
yek adad vared konid:-2
yek adad vared konid:36
yek adad vared konid:412
yek adad vared konid:-5
yek adad vared konid:6
yek adad vared konid:17
```

```
yek adad vared konid:8
yek adad vared konid:1099
yek adad vared konid:-100
-100 -1099 8 17 6 -5 412 36 -2 1
```

توضیح برنامه

این مثال مشابه با مثال قبلی است. با این تفاوت که مقادیر عناصر آرایه به ترتیب عکس در خروجی چاپ می‌شود. زیرا با افزایش مقدار متغیر `i` در حلقه موجود **در خط شماره (۱)**، عبارت مورد استفاده برای اندیس آرایه در **خط شماره (۲)**، منجر به دسترسی به عناصر آرایه از آخر به اول خواهد شد.

خطوط شماره (۱) و (۲) از برنامه فوق را به شکل زیر نیز می‌توان نوشت:

```
for(int i = 9 ; i >= 0; i--){
      cout << array[i] << " ";
}
```

در این حالت مقادیر متغیر `i` در حلقه `for` به طور اتوماتیک دسترسی معکوس به عناصر آرایه را فراهم می‌کنند.

مثال ۳۸

برنامه‌ای بنویسید که تعداد ۱۰ عدد صحیح را از ورودی استاندارد دریافت کند و آن‌ها را در آرایه `a` قرار دهد. پس از آن یک عدد اعشاری از ورودی دریافت کند و حاصل ضرب آن عدد را در تک تک عناصر آرایه `a` محاسبه و در آرایه `b` قرار دهد.

```
#include <iostream>
using namespace std;
int main(){
      int a[10];                                          (1)
      float b[10];                                        (2)
      float adad;                                         (3)
      for(int i = 0 ; i < 10 ; i++){                      (4)
            cout << "yek adad vared konid:";
            cin >> a[i];
      }
      cout << "zarib ashari ra vared konid:";
      cin >> adad;                                        (5)
```

```
	cout << "****anasor araye b****" << " ";
	for(int i = 0 ; i < 10 ; i++){
		b[i] = adad * a[i];                    (6)
		cout << b[i] << " ";                   (7)
	}
	return 0;
}
```

خروجی برنامه

برنامه را با اعداد 1 تا 10 به عنوان عناصر آرایه و عدد 2.5 به عنوان ضریب اجرا می‌کنیم:

```
yek adad vared konid:1
yek adad vared konid:2
yek adad vared konid:3
yek adad vared konid:4
yek adad vared konid:5
yek adad vared konid:6
yek adad vared konid:7
yek adad vared konid:8
yek adad vared konid:9
yek adad vared konid:10
zarib ashari ra vared konid:2.5
****anasor araye b****
2.5 5 7.5 10 12.5 15 17.5 20 22.5 25
```

توضیح برنامه

در **خط شماره (۱)**، آرایه a از نوع عدد صحیح برای نگهداری عناصر آرایه اول و در **خط شماره (۲)**، آرایه b از نوع عدد اعشاری برای ذخیره‌سازی عناصر آرایه دوم تعریف شده‌اند.

در **خط شماره (۳)**، متغیر عدد اعشاری adad برای نگهداری عدد مورد نظر برای انجام عمل ضرب در آرایه a تعریف شده است.

در حلقه موجود در **خط شماره (۴)**، مقادیر عناصر آرایه a از ورودی استاندارد دریافت می‌شود.

در **خط شماره (۵)** مقدار متغیر adad به عنوان ضریب اعشاری، از ورودی دریافت می‌شود.

در **خط شماره (۶)**، مقدار عنصر iام آرایه b برابر با حاصل ضرب مقدار موجود در متغیر adad و عنصر iام آرایه a می‌شود و در **خط شماره (۷)** مقدار آن به همراه یک کاراکتر فاصله در خروجی چاپ می‌شود.

مثال ۳۹

برنامه‌ای بنویسید که تعداد ۱۰ عدد صحیح را از ورودی استاندارد دریافت کند و آن‌ها را در یک آرایه قرار دهد. سپس بزرگترین عنصر و محل آن را در آرایه بیابد.

```
#include <iostream>
using namespace std;
int main(){
      int array[10];
      int max = 0, position;                                    (1)
      for(int i = 0 ; i < 10 ; i++){                            (2)
            cout << "yek adad vared konid:";
            cin >> array[i];
      }
      for(int i = 0 ; i < 10 ; i++){                            (3)
            if(array[i] >= max){                                (4)
                  max = array[i];                               (5)
                  position = i + 1;                             (6)
            }
      }
      cout << "max = " << max
            << " in position = " << position;                   (7)
      return 0;
}
```

خروجی برنامه

برنامه را با اعداد مثال ۸ـ۲ اجرا می‌کنیم:

```
yek adad vared konid:1
yek adad vared konid:-2
```

```
yek adad vared konid:36
yek adad vared konid:412
yek adad vared konid:-5
yek adad vared konid:6
yek adad vared konid:17
yek adad vared konid:8
yek adad vared konid:1099
yek adad vared konid:-100
max = 1099 in position = 9
```

توضیح برنامه

در **خط شماره (۱)**، دو متغیر عدد صحیح به نام‌های max و position به ترتیب برای نگهداری بیشترین مقدار موجود در آرایه و محل قرارگیری آن در نظر گرفته شده است.

حلقه موجود در **خط شماره (۲)**، مقادیر عناصر آرایه array را از ورودی استاندارد دریافت می‌کند.

در **خط شماره (۳)** ساختار حلقه دیگری برای یافتن بزرگترین عنصر آرایه و محل آن در آرایه آمده است.

خط شماره (۴)، عنصر iام از array را با مقدار متغیر max مقایسه می‌کند. در صورتی که این مقایسه ارزش درستی داشته باشد، در **خط شماره (۵)** این عنصر به عنوان بزرگترین مقدار موجود در بین عناصر 0 تا i در نظر گرفته می‌شود. همچنین در **خط شماره (۶)** مقدار متغیر i به عنوان محل قرارگیری بزرگترین عنصر، در متغیر position قرار می‌گیرد. البته چون مقدار اندیس آرایه از عدد 0 آغاز می‌شود، در این خط ابتدا یک واحد به مقدار اندیس i افزوده می‌شود و سپس در متغیر position قرار می‌گیرد تا معنی بهتری داشته باشد.

در نهایت در **خط شماره (۷)**، مقدار متغیر max و position در خروجی چاپ می‌شود.

مثال ۴۰

برنامه‌ای بنویسید که تعداد ۱۰ عدد صحیح را از ورودی استاندارد دریافت کند و آن‌ها را در یک آرایه قرار دهد. سپس یک عدد صحیح را به عنوان یک کلید از ورودی دریافت کند و آن را در آرایه جستجو کند. در صورتی که این عدد در آرایه وجود داشته باشد، محل قرارگیری آن را در خروجی چاپ کند.

```
#include <iostream>
using namespace std;
int main(){
      int array[10];
```

```
	int key, position;                                          (1)
	for(int i = 0 ; i < 10 ; i++){                              (2)
		cout << "yek adad vared konid:";
		cin >> array[i];
	}
	cout << "adad mored nazar baraye jostojo?";
	cin >> key;                                                 (3)
	for(int i = 0 ; i < 10 ; i++){                              (4)
		if(array[i] == key){                                    (5)
			position = i + 1;                                   (6)
			cout << "adad mored nazar dar khaneye "
				<< position << " araye gharar dadrad.\n";       (7)
			break;                                              (8)
		}
	}
	return 0;
}
```

خروجی برنامه

برنامه را با اعداد مثال ۸ـ۲ اجرا می‌کنیم:

```
yek adad vared konid:1
yek adad vared konid:-2
yek adad vared konid:36
yek adad vared konid:412
yek adad vared konid:-5
yek adad vared konid:6
yek adad vared konid:17
yek adad vared konid:8
yek adad vared konid:1099
yek adad vared konid:-100
adad mored nazar baraye jostojo?-5
```

adad mored nazar dar khaneye 5 araye gharar dadrad.

توضیح برنامه

در **خط شماره (۱)**، دو متغیر عدد صحیح به نام‌های adad و position به ترتیب برای نگهداری عدد مورد جستجو و محل قرارگیری آن در آرایه تعریف شده است.

حلقه موجود در **خط شماره (۲)**، مقادیر عناصر آرایه array را از ورودی استاندارد دریافت می‌کند.

در **خط شماره (۳)**، عدد مورد نظر کاربر برای جستجو در آرایه از ورودی استاندارد دریافت می‌شود.

در **خط شماره (۴)** ساختار حلقه دیگری برای جستجوی عدد مورد نظر کاربر و یافتن محل قرارگیری آن در آرایه آمده است.

خط شماره (۵)، عنصر iام از array را با مقدار متغیر adad مقایسه می‌کند. در صورتی که این مقایسه ارزش درستی داشته باشد (یعنی عنصر مورد نظر در آرایه وجود داشته باشد)، در **خط شماره (۶)** متغیر i یک واحد افزایش یافته و به عنوان محل قرارگیری عنصر مورد جستجو در آرایه، در متغیر position قرار می‌گیرد.

در **خط شماره (۷)**، پیغامی مبنی بر این که عنصر مورد جستجوی کاربر در آرایه وجود دارد و همچنین مقدار متغیر position در خروجی استاندارد چاپ می‌شود.

با یافتن عنصر مورد جستجو در آرایه، نیازی به بررسی بقیه عناصر آرایه نیست. به همین دلیل در **خط شماره (۸)**، دستور break اجرا می‌شود و عمل جستجوی آرایه پایان می‌پذیرد.

مثال ۴۱

برنامه‌ای بنویسید که تعداد ۱۰ عدد صحیح را از ورودی استاندارد دریافت کند و آن‌ها را در یک آرایه قرار دهد. سپس میانگین عناصر آرایه را محاسبه و در خروجی چاپ کند.

```
#include <iostream>
using namespace std;
int main(){
      int array[10];
      float majmoo = 0, miangin;                              (1)
      for(int i = 0 ; i < 10 ; i++){                          (2)
            cout << "yek adad vared konid:";
            cin >> array[i];
            majmoo += array[i];                               (3)
      }
```

```
    miangin = majmoo / 10;                                  (4)
    cout << "miangin  = " << miangin << "\n";               (5)
    return 0;
}
```

خروجی برنامه

```
yek adad vared konid:1
yek adad vared konid:2
yek adad vared konid:3
yek adad vared konid:4
yek adad vared konid:5
yek adad vared konid:6
yek adad vared konid:7
yek adad vared konid:8
yek adad vared konid:9
yek adad vared konid:10
miangin  = 5.5
```

توضیح برنامه

در **خط شماره (۱)**، دو متغیر عدد اعشاری به نام‌های majmoo و miangin به ترتیب برای نگهداری مجموع و میانگین اعداد در نظر گرفته شده است.

با اجرای دستور for در **خط شماره (۲)**، محتوای هر یک از عناصر آرایه از ورودی دریافت شده و در **خط شماره (۳)** به مقدار موجود در متغیر majmoo افزوده می‌شود.

در **خط شماره (۴)**، مقدار متغیر majmoo که برابر با مجموع همه عناصر آرایه است بر عدد 10 تقسیم شده و در متغیر miangin قرار می‌گیرد.

در **خط شماره (۵)**، مقدار محاسبه شده برای میانگین در خروجی چاپ می‌شود.

مثال ۴۲

برنامه‌ای بنویسید که تعداد ۱۰ عدد صحیح را از ورودی استاندارد دریافت کند و واریانس آن‌ها را محاسبه کند (فرمول محاسبه واریانس: $\frac{\sum_{i=0}^{n}(x_i-\mu)^2}{n}$، در این فرمول منظور از μ همان میانگین اعداد است).

```
#include <iostream>
```

```
#include <cmath>
using namespace std;
int main(){
      int array[10];
      float majmoo = 0, miangin, variance;                    (1)
      for(int i = 0 ; i < 10 ; i++){                          (2)
            cout << "yek adad vared konid:";
            cin >> array[i];
            majmoo += array[i];
      }
      miangin = majmoo / 10;                                  (3)
      majmoo = 0;                                             (4)
      for(i = 0 ; i < 10 ; i++){                              (5)
            majmoo += pow((array[i]-miangin),2);              (6)
      }
      variance = majmoo / 10;                                 (7)
      cout << " miangin = " << miangin
            << " va variance = " << variance << "\n";         (8)
      return 0;
}
```

خروجي برنامه

```
yek adad vared konid:1
yek adad vared konid:2
yek adad vared konid:3
yek adad vared konid:4
yek adad vared konid:5
yek adad vared konid:6
yek adad vared konid:7
yek adad vared konid:8
yek adad vared konid:9
```

```
yek adad vared konid:10
miangin  = 5.5 va variance = 8.25
```

توضیح برنامه

با توجه به این که در محاسبه واریانس اعداد، از مقدار میانگین اعداد استفاده می‌شود، در این برنامه ابتدا مقدار میانگین اعداد محاسبه می‌شود.

در **خط شماره (۱)**، علاوه بر متغیرهای اعشاری majmoo و miangin که در مثال قبلی شرح داده شدند، متغیر دیگری با نام variance برای نگهداری مقدار واریانس نیز تعریف شده است.

ساختار حلقه موجود در **خط شماره (۲)** همانند مثال قبلی، مجموع عناصر آرایه را برای بدست آوردن میانگین آن‌ها محاسبه می‌کند. سپس در **خط شماره (۳)**، میانگین عناصر آرایه محاسبه شده و در متغیر miangin قرار داده می‌شود.

با توجه به این که برای محاسبه واریانس اعداد بار دیگر نیاز به جمع بستن تعدادی عدد داریم، می‌توانیم از متغیر majmoo که در محاسبه میانگین استفاده نموده بودیم، برای این منظور استفاده نماییم. بنابراین نیازی به تعریف متغیر دیگر نخواهیم داشت. تنها نکته‌ای که باید به آن توجه داشته باشیم، آن است که مقدار این متغیر را به مقدار 0 تنظیم کنیم. این عمل را در **خط شماره (۴)** مشاهده می‌کنید.

خط شماره (۵)، حلقه‌ای است که به تک تک عناصر آرایه دسترسی می‌یابد و از مقدار آن‌ها در فرمول واریانس استفاده می‌کند. همان طور که ملاحظه می‌کنید، برای محاسبه توان دوم حاصل تفریق عنصر iام array و مقدار میانگین، از تابع ریاضی pow استفاده شده است.

در **خط شماره (۷)**، عمل محاسبه واریانس با تقسیم مقدار موجود در متغیر majmoo بر تعداد عناصر آرایه که در اینجا برابر با 10 است، تکمیل می‌شود.

در نهایت در **خط شماره (۸)**، مقدار محاسبه شده برای واریانس در خروجی استاندارد چاپ می‌شود.

در تمام مثال‌هایی که تاکنون برای آرایه‌ها بررسی کردیم، طول آرایه از همان ابتدا عدد مشخص و ثابتی بود. در نسخه استاندارد C/C++، امکان تعریف آرایه‌ای که طول آن توسط یک متغیر تعیین شود نیز به وجود آمده است. در ادامه مثالی در این مورد بیان شده است.

مثال ۴۳

برنامه‌ای بنویسید که ابتدا طول آرایه را از کاربر دریافت کند و یک آرایه به طول عدد دریافت شده تعریف کند. سپس با توجه به طول آرایه، تعدادی عدد صحیح از ورودی استاندارد دریافت کند و در آرایه قرار دهد و در نهایت عناصر آرایه را یک به یک در خروجی استاندارد چاپ کند.

```
#include <iostream>
```

```
using namespace std;
int main(){
	int tool;                                          (1)
	cout << "tedad adad ra vared koind:";
	cin >> tool;

	int array[tool];                                   (2)
	for(int i = 0 ; i < tool ; i++){                   (3)
		cout << "yek adad vared konid:";
		cin >> array[i];
	}
	for(int i = 0 ; i < tool ; i++){                   (4)
		cout << array[i] << " ";
	}
	return 0;
}
```

خروجی برنامه

برنامه را برای 5 عدد 1 تا 5 اجرا می‌کنیم:

```
tedad adad ra vared koind:5
yek adad vared konid:1
yek adad vared konid:2
yek adad vared konid:3
yek adad vared konid:4
yek adad vared konid:5
```

توضیح برنامه

همان طورکه مشاهده می‌کنید در این برنامه طول آرایه از ابتدا تعیین نشده است و مقدار آن از کاربر دریافت می‌شود.

در **خط شماره (۱)**، متغیر عدد صحیح `tool` برای نگهداری مقدار دریافتی از کاربر برای طول آرایه تعریف شده است.

با توجه به عدد دریافتی برای طول، در **خط شماره (۲)** یک آرایه به اندازه مقدار موجود در متغیر tool تعریف شده است و طبق همین عدد، حلقه‌های موجود در **خطوط شماره (۳) و (۴)** نیز به اندازه مقدار موجود در متغیر tool اجرا خواهند شد.

۱ـ۱۷ـ۲ـ۱ ماتریس‌ها یا همان آرایه‌های دو بعدی

تمام مثال‌هایی که تاکنون در مورد آرایه‌ها بررسی کردیم، مربوط به آرایه‌های یک بعدی بودند. اما در زبان C++ امکان تعریف آرایه‌هایی با ابعاد بالاتر نیز وجود دارد. در این بخش با آرایه‌های دو بعدی (ماتریس) آشنا می‌شویم. یک آرایه دو بعدی مانند یک جدول، حاوی سطر و ستون است و به شکل زیر تعریف می‌شود:

نوع عناصر آرایه نام آرایه [تعداد سطرها] [تعداد ستون‌ها] ;

همان طور که در این تعریف مشاهده می‌کنید، از دو جفت براکت برای مشخص کردن تعداد سطرها و ستون‌های آرایه استفاده می‌شود. بنابراین برای دسترسی به عناصر ماتریس نیاز به دو اندیس خواهیم داشت. همچنین برای بررسی همه عناصر یک ماتریس، نیاز به دو حلقه تو در تو خواهیم داشت.

مثال ۴۴

برنامه‌ای بنویسید که یک ماتریس 3×2 از اعداد صحیح تعریف کند و آن را مقداردهی اولیه کند. سپس مقادیر آن را یک به یک در خروجی استاندارد چاپ کند.

```
#include <iostream>
using namespace std;
int main(){
      int array[2][3] = {{1,-2,36},{412,-5,6}};                (1)
      for(int i = 0 ; i < 2 ; i++){                            (2)
            for(int j = 0 ; j < 3 ; j++){
                  cout << array[i][j] << " ";                  (3)
            cout << "\n";                                      (4)
      }
      return 0;
}
```

خروجی برنامه

```
1 -2 36
412 -5 6
```

توضیح برنامه

در **خط شماره (۱)**، یک آرایه دو بعدی از نوع عدد صحیح به نام array تعریف شده است که شامل 2 سطر و 3 ستون است. مقادیر اولیه این آرایه شامل ۶ عدد صحیح است که در داخل دو علامت {} آورده شده‌اند. به طوری که سه عدد اول در داخل یک جفت براکت و سه عدد دوم در داخل یک جفت براکت دیگر قرار دارند.

خط شماره (۲)، دو حلقه for تو در تو را نشان می‌دهد که به کمک آن‌ها همه عناصر آرایه دو بعدی array مورد دستیابی قرار می‌گیرند و در **خط شماره (۳)** در خروجی استاندارد چاپ می‌شوند.

برای این که پس از چاپ عناصر یک سطر، عناصر بعدی در خط بعدی چاپ شوند، در **خط شماره (۴)** یک دستور cout به همراه کاراکتر کنترلی \n آورده شده است.

مثال ۴۵

برنامه‌ای بنویسید که یک ماتریس 3×2 از اعداد صحیح تعریف کند و مقدار عناصر آن را از ورودی استاندارد دریافت کند. سپس مقادیر آن‌ها را یک به یک در خروجی استاندارد چاپ کند.

```
#include <iostream>
using namespace std;
int main(){
	int array[2][3];
	for(int i = 0 ; i < 2 ; i++){
		for(int j = 0 ; j < 3 ; j++){
		cout << "yek adad vared konid:";
		cin >> array[i][j];                                    (1)

	}
	for(int i = 0 ; i < 2 ; i++){
		for(int j = 0 ; j < 3 ; j++){
			cout << array[i][j] << " ";
		cout << "\n";
	}
	return 0;
}
```

خروجی برنامه

برنامه را با اعداد مثال قبلی اجرا می‌کنیم:

```
yek adad vared konid:1
yek adad vared konid:-2
yek adad vared konid:36
yek adad vared konid:412
yek adad vared konid:-5
yek adad vared konid:6
1 -2 36
412 -5 6
```

توضیح برنامه

این مثال مشابه با مثال قبلی است. با این تفاوت که مقادیر عناصر ماتریس به کمک تابع `cin` در **خط شماره (۱)** از کاربر دریافت می‌شود.

مثال ۴۶

برنامه‌ای بنویسید که تعدد واحد و نمره ۵ درس یک دانشجو را از ورودی استاندارد دریافت کند و در یک ماتریس قرار دهد. سپس معدل او را محاسبه کند..

```
#include <iostream>
using namespace std;
int main(){
      float majmoo_vahedha = 0, majmoo_nomarat = 0;          (1)
      float a[5][2];                                         (2)
      float moadel;                                          (3)
      for(int i = 0; i < 5 ; i++){                           (4)
            cout << "tedad vahed? ";
            cin >> a[i][0];                                  (5)
            majmoo_vahedha += a[i][0];                       (6)
      }
      for(int i = 0 ; i < 5 ; i++){                          (7)
            cout << "nomreh ? ";
```

```
            cin >> a[i][1];                                    (8)
      }
      for(int i = 0; i < 5 ; i++){                             (9)
            for(int j = 0 ; j < 2 ; j++){
                  cout << a[i][j] << " ";
            }
            cout << "\n";
      }
      for(int i = 0; i < 5 ;i++){                              (10)
            majmoo_nomarat += (a[i][0] * a[i][1]);             (11)
      }
      moadel = majmoo_nomarat / majmoo_vahedha;                (12)
      cout << "moadel = " << moadel;                           (13)
      return 0;
}
```

خروجی برنامه

برنامه را با ورودی‌های زیر اجرا می‌کنیم:

```
tedad vahed? 1
tedad vahed? 2
tedad vahed? 3
tedad vahed? 2
tedad vahed? 1
nomreh ? 20
nomreh ? 19
nomreh ? 10
nomreh ? 16.5
nomreh ? 14
1 20
2 19
3 10
2 16.5
```

```
1 14
moadel = 15
```

توضیح برنامه

در **خط شماره (۱)**، دو متغیر اعشاری تحت عنوان majmoo_vahedha و majmoo_nomarat به ترتیب برای نگهداری مجموع واحدهای درسی و مجموع نمرات وزن‌دار دروس در نظر گرفته شده است.

در **خط شماره (۲)**، یک ماتریس 2×5 از نوع عدد اعشاری برای نگهداری تعداد واحدها و نمرات ۵ درس تعریف شده است. ستون اول این ماتریس، تعداد واحدهای هر درس و ستون دوم آن شامل نمره درس است.

خط شماره (۳) شامل تعریف یک متغیر عدد اعشاری با نام moadel برای نگهداری معدل محاسبه شده دانشجو است.

خط شماره (۴) بیانگر حلقه‌ای است که در طی آن، تعداد واحدهای هر درس از ورودی استاندارد دریافت می‌شود (خط شماره (۵)) و در ستون اول ماتریس قرار داده می‌شود.

در **خط شماره (۶)** با دریافت تعداد واحد هر درس، مقدار آن با تعداد واحدهای قبلی جمع و در متغیر majmoo_vahedha قرار داده می‌شود تا بعدا برای محاسبه معدل از مقدار آن استفاده شود.

همانند تعداد واحد درس، نمره هر درس نیز در داخل حلقه‌ای که در **خط شماره (۷)** آمده است، از ورودی استاندارد دریافت می‌شود و در محل مربوطه در ستون دوم ماتریس قرار می‌گیرد (خط شماره (۸)).

در **خط شماره (۹)** دو حلقه تو در تو برای چاپ مقادیر ماتریس وجود دارد که مقادیر هر سطر را که شامل تعداد واحد و نمره یک درس است، در خطی جداگانه چاپ می‌کند.

خط شماره (۱۰)، آغازگر حلقه‌ای است که در **خط شماره (۱۱)** از آن، تعداد واحد هر درس در ستون اول در نمره آن درس در ستون دوم ضرب شده و با مقدار قبلی متغیر majmoo_nomarat جمع می‌شود. بدین ترتیب این متغیر شامل مجموع نمرات وزن‌دار دروس دانشجو خواهد بود.

در **خط شماره (۱۱)** معدل دانشجو با تقسیم کردن مقدار متغیر majmoo_nomarat بر مقدار متغیر majmoo_vahedha محاسبه شده و در متغیر moadel قرار می‌گیرد.

در نهایت در **خط شماره (۱۲)**، معدل دانشجو که در متغیر moadel قرار دارد، در خروجی استاندارد چاپ می‌شود.

۱۸ـ۲ـ۱ توابع در زبان C++

تاکنون با برخی از توابع از پیش‌تعریف شده ریاضی که در کتابخانه‌های استاندارد C++ قرار داشتند، آشنا شدیم و از آن‌ها در برخی از مثال‌های خود استفاده کردیم. توابع sqrt، pow، floor، ceil و بسیاری از توابع دیگر که درکتابخانه math.h (و cmath) قرار دارند، جزو این توابع محسوب می‌شوند. علاوه بر توابع از پیش تعریف-شده، برنامه‌نویس می‌تواند طبق ضروت و نیازهای برنامه خود، توابع جدیدی نیز تعریف کند. به عنوان مثال تابع

main یکی از توابعی است که توسط برنامه‌نویس تعریف می‌شود. برای آشنایی با نحوه تعریف توابع ابتدا با با برخی از مفاهیم اولیه تابع آشنا می‌شویم.

تابع، برنامه‌ای است که برای هدف خاصی تعریف می‌شود و می‌تواند در تابع main و دیگر توابع مورد استفاده قرار گیرد. نحوه تعریف یک تابع به شکل زیر است:

```
نوع نتیجه تابع  نام تابع (نام و نوع پارامترهای تابع) {
        دستور ۱ ;
        دستور ۲ ;
        . . .
}
```

در تعریف فوق، سه ویژگی مهم تابع را مشاهده می‌کنید که به شرح زیر هستند:

۱. **نام تابع:** همانند متغیرها، توابع نیز باید دارای نام باشند. اسامی توابع باید طبق قوانین نام‌گذاری متغیرها انتخاب شده باشند (یادآوری: اسامی متغیرها می‌تواند ترکیبی از حروف A-Z، a-z، 0-9 و کاراکتر _ باشد) و بیانگر عملکرد و وظیفه توابع باشند.
۲. **پارامترهای تابع:** منظور از پارامترهای تابع، ورودی‌ها یا اطلاعاتی هستند که یک تابع برای انجام وظیفه خود به آن‌ها نیاز دارد. هنگام تعریف یک تابع، نوع و نام این پارامترها باید مشخص شود. نوع پارامترها یکی از انواع موجود در زبان C++ (همانند نوع int، float، char و نوع void که در این بخش شرح داده خواهد شد) و نام آن‌ها همانند نام متغیرها است.
۳. **نتیجه تابع:** یک تابع ممکن است پس از انجام وظیفه خود،نتیجه یا مقداری را به تابع استفاده کننده از آن برگرداند. نوع مقدار بازگشتی یک تابع باید در تعریف آن آورده شود. این نوع می‌تواند یکی از انوع موجود در زبان C++ باشد.

پس از تعیین ویژگی‌های فوق برای یک تابع، لیست دستورات مورد نیاز برای پیاده‌سازی هدف یک تابع در بدنه آن آورده می‌شود. این دستورات می‌توانند هر دستور معتبری در زبان C++ باشند.

همان طور که قبلا هم اشاره شد، پس از تعریف یک تابع می‌توان آن را در توابع دیگر استفاده کرد. به استفاده کردن از یک تابع در تابع دیگر، *فراخوانی تابع* گویند که به شکل زیر است:

```
نام تابع (مقادیر مورد نیاز تابع) ;
```

همان طور که مشاهده می‌کنید، ابتدا نام تابع و سپس مقادیر مورد نیاز برای انجام وظیفه تابع، داخل پرانتز آورده می‌شود. در صورتی که تابع مورد نظر یک نتیجه هم تولید کند، می‌توان از مقدار آن هنگام فراخوانی تابع استفاده کرد. در ادامه مثال‌هایی برای شرح بهتر این مفاهیم آورده شده است.

مثال ۴۷

برنامه‌ای بنویسید که طول و عرض یک مستطیل را از ورودی دریافت کند و به کمک یک تابع، مساحت آن را محاسبه کند.

```
#include <iostream>
using namespace std;
int masahat(int tool, int arz){                                    (1)
      int m;                                                       (2)
      m = tool * arz;                                              (3)
      return m;                                                    (4)
}
int main(){                                                        (5)
      int x, y;                                                    (6)
      cout << "tool va arz mostatil ra vared konid:";              (7)
      cin >> x >> y;                                               (8)
      cout << "masahat mostatil = " << masahat(x,y) << "\n";       (9)
      return 0;
}
```

خروجی برنامه

فرض می‌کنیم که کاربر مقادیر 15 و 10 را به ترتیب برای طول و عرض وارد کند:

```
tool va arz mostatil ra vared konid:15 10
masahat mostatil = 150
```

توضیح برنامه

در این برنامه، تعریف تابع masahat که وظیفه محاسبه مساحت مستطیل را بر عهده دارد، قبل از تابع main آورده شده است. در تابع main که پس از تابع masahat تعریف شده است، مقادیر طول و عرض مستطیل از کاربر دریافت شده و به تابع masahat ارسال می‌شود.

خط شماره (۱) آغاز تعریف تابع masahat است. این تعریف نشان می‌دهد که تابعی با نام masahat یک نتیجه از نوع عدد صحیح بازمی‌گرداند و حاوی دو پارامتر از نوع عدد صحیح با نام‌های tool و arz است.

خطوط شماره (۲) تا (۴)، بیانگر دستورات موجود در بدنه تابع masahat هستند. در **خط شماره (۲)** از این دستورات، متغیر عدد صحیح m برای نگهداری مقدار محاسبه شده مساحت در نظر گرفته شده است. در **خط شماره**

(۳) مساحت مستطیل با استفاده از مقادیر پارامترهای tool و arz محاسبه می‌شود و در متغیر m قرار داده می‌شود. در **خط شماره (۴)** به کمک دستور return مقدار موجود در متغیر m به عنوان نتیجه تابع بازگردانده می‌شود.

خط شماره (۵) تابع main را تعریف می‌کند.

برنامه در **خطوط شماره (۶) تا (۸)** متغیرهای x و y را تعریف کرده و مقادیر آن‌ها را از کاربر دریافت می‌کند.

خط شماره (۹) تابع masahat را در داخل دستور cout فراخوانی کرده و مقادیر متغیرهای x و y را به عنوان مقادیر پارامترهای tool و arz به این تابع ارسال می‌کند. همچنین با توجه به این که تابع masahat یک نتیجه تولید می‌کند، مقدار بازگشتی آن توسط دستور cout در خروجی چاپ می‌شود.

نکته: هنگام فراخوانی یک تابع باید به تعداد، نوع و ترتیب پارامترهای آن دقت کرد تا مقادیر درستی به تابع ارسال شود.

نکته: علاوه بر پارامترهای تابع که از مقادیر آن‌ها در دستورات بدنه تابع استفاده می‌شود، طبق نیاز تابع می‌توان متغیرهای جدیدی در آن تعریف کرد (به عنوان مثال در تابع masahat، علاوه بر پارامترهای tool و arz، متغیر جدید m نیز در تابع تعریف شده است).

نکته: در صورتی که تابع نتیجه‌ای تولید کند، وجود دستور return که مقدار نتیجه را برگرداند در بین دستورات تابع الزامی است (به عنوان مثال، خط شماره (۴) در تابع masahat).

نکته: در صورتی که تابع نتیجه‌ای تولید کند، نوع نتیجه تابع که در تعریف آن آورده می‌شود باید با نوع مقداری که توسط دستور return بازگردانده می‌شود، یکسان باشد (به عنوان مثال، نوع متغیر m که در تابع masahat توسط return بازگردانده می‌شود، با نوع نتیجه تابع که در تعریف آن آمده است، یکسان است).

نکته: در صورتی که تابع نتیجه‌ای تولید کند، از مقدار بازگشتی آن می‌توان در عبارت‌های ریاضی، رابطه‌ای و دستور cout استفاده کرد (به عنوان مثال، در خط شماره (۹) از تابع main از مقدار بازگشتی تابع masahat در یک دستور cout استفاده شده است).

در مثال فوق، تابع masahat را می‌توان به شکل زیر نیز تعریف کرد، زیرا حاصل ضرب متغیرهای tool و arz منجر به تولید یک نتیجه از نوع عدد صحیح خواهد شد که با نوع بازگشتی تابع یکسان است.

```
int masahat(int tool, int arz){
      return (tool * arz);
}
```

مثال ۴۸

برنامه‌ای بنویسید که سه عدد صحیح از ورودی دریافت کند و به کمک یک تابع، میانگین آن‌ها را محاسبه کند.

```
#include <iostream>
using namespace std;
float miangin(int adad1, int adad2, int adad3){          (1)
      float m;                                           (2)
      m = (float)(adad1 + adad2 + adad3)/3;              (3)
      return m;                                          (4)
}
int main(){
      int x, y, z;                                       (5)
      cout << "se adad sahih vared konid:";
      cin >> x >> y >> z;
      cout << "miangin = " << miangin(x,y,z) << "\n";     (6)
      return 0;
}
```

خروجی برنامه

فرض می‌کنیم که کاربر مقادیر 2، 3 و 5 را وارد کند:

```
se adad sahih vared konid:2 3 5
miangin = 3.3333
```

توضیح برنامه

خط شماره (۱) آغاز تعریف تابعی به نام `miangin` است. این تابع حاوی سه پارامتر عدد صحیح با نام‌های `adad1`، `adad2` و `adad3` است و یک نتیجه از نوع عدد اعشاری باز می‌گرداند.

خط شماره (۲) از دستورات تابع `miangin`، بیانگر تعریف یک متغیر عدد اعشاری به نام `m` است که برای نگهداری مقدار میانگین به کار می‌رود.

در **خط شماره (۳)** میانگین پارامترهای تابع با انجام یک تقسیم اعشاری محاسبه شده و در متغیر `m` قرار داده می‌شود (یادآوری: در صورتی که تمام عملوندهای یک عملگر از نوع عدد صحیح باشند، نتیجه اعمال آن عملگر بر روی عملوندها از نوع عدد صحیح خواهد شد. مگر این که با تبدیل نوع، نوع عبارت مورد نظر را تغییر داد).

در **خط شماره (۴)** به کمک دستور `return` مقدار موجود در متغیر `m` به عنوان نتیجه تابع بازگردانده می‌شود.

در **خط شماره (۵)** سه متغیر `x`، `y` و `z` را برای نگهداری مقادیر دریافت شده از کاربر تعریف کرده‌ایم.

خط شماره (۶) تابع `miangin` را در داخل یک دستور `cout` فراخوانی کرده و مقدار بازگشتی آن را در خروجی استاندارد چاپ می‌کند.

همان طور که در مثال فوق مشاهده می‌کنید، مقدار بازگشتی تابع از نوع عدد اعشاری است. به همین دلیل متغیر `m` را نیز از نوع اعشاری تعریف نموده‌ایم.

مثال ۴۹

برنامه‌ای بنویسید که سه عدد صحیح از ورودی دریافت کند و به کمک یک تابع، بزرگترین عدد بین آن‌ها را بیابد.

```
#include <iostream>
using namespace std;
int bozorgtarin(int adad1, int adad2, int adad3){          (1)
      int max;                                             (2)
      if(adad1 >= adad2 && adad1 >= adad3)                 (3)
            max = adad1;
      else if(adad2 >= adad1 && adad2 >= adad3)
            max = adad2;
      else
            max = adad3;
      return max;                                          (4)
}
int main(){
      int x, y, z;
      cout << "se adad sahih vared konid:";
      cin >> x >> y >> z;
      cout << "bozorgtarin = "
            << bozorgtarin(x,y,z) << "\n";                 (5)
      return 0;
}
```

خروجی برنامه

فرض می‌کنیم که کاربر مقادیر 2، 3 و 5 را وارد کند:

```
se adad sahih vared konid:2 3 5
```

```
bozorgtarin = 5
```

توضیح برنامه

خط شماره (۱) آغاز تعریف تابعی به نام `bozorgtarin` است. این تابع حاوی سه پارامتر عدد صحیح با نام‌های `adad1`، `adad2` و `adad3` است و یک نتیجه از نوع عدد صحیح باز می‌گرداند.

خط شماره (۲) از دستورات تابع `bozorgtarin` ، بیانگر تعریف یک متغیر عدد صحیح به نام `max` است که برای نگهداری بزرگترین مقدار از بین سه مقدار به کار می‌رود.

در **خط شماره (۳)** آغاز یک ساختار `if-else` است که مقادیر پارامترها را با یکدیگر مقایسه کرده و بزرگترین مقدار را در متغیر `max` قرار می‌دهد.

در **خط شماره (۴)** به کمک دستور `return` مقدار موجود در متغیر `max` به عنوان نتیجه تابع بازگردانده می‌شود.

خط شماره (۵) تابع `bozorgtarin` را در داخل یک دستور `cout` فراخوانی کرده و مقدار بازگشتی آن را در خروجی استاندارد چاپ می‌کند.

همان طور که در مثال فوق مشاهده می‌کنید، از ساختارهای تصمیم نیز می‌توان برای پیاده‌سازی وظیفه تابع مورد نظر استفاده کرد.

تابع `bozorgtarin` را می‌توان به شکل زیر نیز تعریف کرد:

```
int bozorgtarin(int adad1, int adad2, int adad3){
      if(adad1 >= adad2 && adad1 >= adad3)                (1)
            return adad1;                                 (2)
      else if(adad2 >= adad1 && adad2 >= adad3)           (3)
            return adad2;                                 (4)
      else                                                (5)
            return adad3;                                 (6)

}
```

در تابع فوق به جای استفاده از یک دستور `return` برای بازگرداندن مقدار بزرگترین عدد، دستور `return` را در بدنه ساختار `if-else` تکرار کرده‌ایم. در این حالت به محض درست بودن یکی از عبارت‌های شرطی در خطوط شماره (۱) و شماره (۳) و یا اجرای بخش `else` در خط شماره (۵)، دستور `return` مربوطه در خطوط (۲)، (۴) و (۶) اجرا خواهد شد.

مثال ۵۰

برنامه‌ای بنویسید که یک عدد صحیح سه رقمی از ورودی دریافت کند و به کمک یک تابع، تشخیص دهد که عدد پالیندروم (متقارن) است یا نه (پالیندروم عددی است که از طرف چپ و راست به یک شکل خوانده می‌شود). برای حل این مساله کافی است که ارقام عدد را تفکیک کرده و رقم اول و سوم آن را با یکدیگر مقایسه کنیم.

```
#include <iostream>
using namespace std;
void palindrome(int adad){                                          (1)
      int x, y, z;                                                  (2)
      x = adad / 100;                                               (3)
      y = adad % 100;                                               (4)
      z = y % 10;                                                   (5)
      if(x == z)                                                    (6)
            cout << "adad " << adad << " palindrome ast.\n";       (7)
      else
            cout << "adad " << adad << " palindrome nist.\n";      (8)
}
int main(){
      int x;
      cout << "yek adad se raghami va sahih vared konid:";
      cin >> x;
      palindrome(x);                                                (9)
      return 0;
}
```

خروجی برنامه

فرض کنید که کاربر مقدار 353 را وارد کند:

```
yek adad se raghami va sahih vared konid:353
adad 353 palindrome ast.
```

توضیح برنامه

خط شماره (۱) آغاز تعریف تابعی به نام palindrome است. این تابع حاوی یک پارامتر عدد صحیح به نام adad است و بر خلاف مثال‌های قبلی هیچ نتیجه‌ای تولید نمی‌کند. به همین دلیل برای مشخص کردن نوع نتیجه تابع، از کلمه کلیدی void استفاده شده است.

در **خط شماره (۲)** متغیرهای صحیح x، y و z برای استفاده در تفکیک ارقام پارامتر adad معرفی شده‌اند.

در **خط شماره (۳)**، با تقسیم مقدار پارامتر adad بر عدد 100 مقدار رقم صدگان عدد جدا شده و در متغیر x قرار می‌گیرد.

در **خط شماره (۴)**، باقیمانده تقسیم پارامتر adad بر عدد 100 که یک عدد دو رقمی شامل ارقام یکان و دهگان پارامتر adad است، محاسبه شده و در متغیر y قرار می‌گیرد.

در **خط شماره (۵)**، با بدست آوردن باقیمانده عدد دو رقمی موجود در متغیر y بر عدد 10، رقم یکان پارامتر adad جدا شده و در متغیر z قرار می‌گیرد.

دستور if در **خط شماره (۵)**، مقدار متغیر x و z را که به ترتیب حاوی رقم‌های یکان و دهگان مقدار موجود در پارامتر adad هستند، با یکدیگر مقایسه می‌کند. در صورتی که مقایسه ارزش درستی داشته باشد، دستور موجود در **خط شماره (۷)** اجرا می‌شود و پیغام متقارن بودن عدد در خروجی چاپ می‌شود. در غیر اینصورت **خط شماره (۸)** اجرا شده و پیغام دیگری نمایش داده خواهد شد.

خط شماره (۹) در تابع main، نحوه فراخوانی تابع palindrome را نشان می‌دهد. با توجه به این که تابع palindrome هیچ مقداری باز نمی‌گرداند، دستور فراخوانی آن به صورت مستقل آورده شده است.

تابع palindrome نمونه‌ای از توابعی است که هیچ مقداری را به تابع فراخواننده آن (در مثال فوق، تابع main) باز نمی‌گردانند. در این گونه از موارد، نوع مقدار بازگشتی تابع از نوع void خواهد بود. به هنگام فراخوانی این گونه از توابع، نمی‌توان آن‌ها را در داخل یک عبارت محاسباتی، رابطه‌ای و حتی دستور cout استفاده کرد.

مثال ۵۱

برنامه‌ای بنویسید که N عدد صحیح سه رقمی از ورودی دریافت کند و به کمک یک تابع، تشخیص دهد که اعداد دریافتی پالیندروم هستند یا نه.

```
#include <iostream>
using namespace std;
void palindrome(int adad){
      int x, y, z;
      x = adad / 100;
```

```
	y = adad % 100;
	z = y % 10;
	if(x == z)
		cout << "adad " << adad << " palindrome ast.\n";
	else
		cout << "adad " << adad << " palindrome nist.\n";
}

int main(){
	int x, N;
	cout << “tedad adad mored nazar ra vared konid:  “;
	cin >> N;                                                    (1)

	for(int i = 0 ; i < N ; i++){                                (2)
		cout << "yek adad se raghami va sahih vared konid:";
		cin >> x;
		palindrome(x);                                       (3)
	}
	return 0;
}
```

خروجی برنامه

فرض کنید که کاربر مقدار 2 را برای متغیر N وارد کند. سپس برای دور اول حلقه، مقدار 353 و برای دور دوم مقدار 466 را در نظر بگیرد:

```
yek adad se raghami va sahih vared konid:353
adad 353 palindrome ast.
yek adad se raghami va sahih vared konid:466
adad 466 palindrome nist.
```

توضیح برنامه

در **خط شماره (۱)**، مقدار متغیر N را از کاربر دریافت می‌شود.

خط شماره (۲) آغازگر یک ساختار for است که امکان دریافت N عدد صحیح و بررسی پالیندروم بودن آن را بر عهده دارد. در هر دور از اجرای این حلقه، یک عدد صحیح از کاربر دریافت می‌شود و در **خط شماره (۳)** به تابع palindrome ارسال می‌شود.

مثال ۵۲

برنامه‌ای بنویسید که N عدد صحیح از ورودی دریافت کند و به کمک یک تابع، تشخیص دهد که اعداد دریافتی اول هستند یا نه.

```
#include <iostream>
using namespace std;
void avval(int adad){                                          (1)
      int tedad = 0, i;                                        (2)
      for(i = 2 ; i < adad ; i++)                              (3)
            if(adad % i == 0){                                 (4)
                  tedad++;                                     (5)
                  break;                                       (6)
            }
      if(adad < 2 || tedad > 0)                                (7)
            cout << "adad mored nazar avval nist.\n";          (8)
      else
            cout << "adad mored nazar avval ast.\n";           (9)
}
int main(){
      int x, N;
      cout << “tedad adad mored nazar ra vared konid:  “;
      cin >> N;                                                (10)

      for(int i = 0 ; i < N ; i++){                            (11)
            cout << "yek adad vared konid:";
            cin >> x;
            avval(x);                                          (12)

      }
```

```
    return 0;
}
```

خروجی برنامه

فرض کنید که کاربر مقدار 2 را برای متغیر N وارد کند. سپس برای دور اول حلقه، مقدار 13 و برای دور دوم مقدار 299 را در نظر بگیرید:

```
yek adad vared konid:13
adad 13 avval ast.
yek adad vared konid:299
adad 299 avval nist.
```

توضیح برنامه

در این برنامه وظیفه تشخیص اول بودن عدد بر عهده تابع avval است.

خط شماره (۱) آغاز تعریف تابع avval است. این تابع یک عدد صحیح به عنوان پارامتر دریافت می‌کند و هیچ مقداری باز نمی‌گرداند.

در **خط شماره (۲)** از تابع avval، متغیر صحیح i برای مشخص کردن اعداد بازه $[2, adad-1]$ و متغیر tedad برای نگهداری تعداد مقسوم علیه‌های پارامتر adad تعریف شده است.

خطوط شماره (۳) تا (۶) بیانگر حلقه‌ای هستند که با بررسی حاصل باقیمانده مقدار پارامتر adad بر متغیر i، تعداد مقسوم علیه‌های adad را محاسبه و در متغیر tedad قرار می‌دهد.

در **خط شماره (۷)**، مقدار متغیر tedad برای بررسی اول بودن پارامتر بررسی می‌شود. در صورتی که مقدار این متغیر بیشتر از صفر باشد و یا مقدار موجود در پارامتر adad کمتر از 2 باشد، عدد مورد نظر اول نیست و در غیر اینصورت اول خواهد بود.

همانند مثال قبلی در تابع main این مثال نیز، حلقه ای داریم که تعداد N بار اجرا شده و هر بار یک عدد از ورودی دریافت می‌کند. اما این بار در **خط شماره (۱۲)** تابع avval برای بررسی عدد دریافت شده فراخوانی می‌شود. همانند تابع palindrome، تابع avval نیز هیچ مقداری باز نمی‌گرداند و وظیفه آن بررسی اول بودن پارامتر دریافتی و چاپ یک پیغام متناسب در خروجی استاندارد است.

همان طور که در مثال فوق مشاهده می‌کنید، از ساختارهای تکرار نیز می‌توان برای پیاده‌سازی وظیفه تابع مورد نظر استفاده کرد.

مثال ۵۳

برنامه‌ای بنویسید که N عدد صحیح از ورودی دریافت کند و به کمک یک تابع، فاکتوریل عدد مورد نظر را محاسبه کند ($n! = n \times n-1 \times ... \times 1$).

```
#include <iostream>

using namespace std;

long long factorial(int adad){
      (1)

      long long fact = 1;
      (2)

      for(; adad >= 1 ; adad--){
      (3)

            fact = fact * adad;
      (4)

      }

      return fact;                                                    (5)

}

int main(){

      int x, N;

      cout << “tedad adad mored nazar ra vared konid:  “;

      cin >> N;
      (6)

      for(int     i     =     0     ;     i     <     N     ;     i++){
            (7)

            cout << "yek adad vared konid:";

            cin >> x;

            cout << x << "! = " << factorial(x) << "\n";
      (8)

      }

      return 0;

}
```

خروجی برنامه

فرض کنید که کاربر مقدار 2 را برای متغیر N وارد کند. سپس برای دور اول حلقه، مقدار 4 و برای دور دوم مقدار 10 را در نظر بگیرد:

```
yek adad vared konid:4
4! = 24
yek adad vared konid:10
10! = 3628800
```

توضیح برنامه

در این برنامه وظیفه محاسبه فاکتوریل عدد بر عهده تابع `factorial` است.

خط شماره (۱) آغاز تعریف تابع `factorial` است. این تابع یک عدد صحیح به عنوان پارامتر دریافت می‌کند و مقدار فاکتوریل آن را محاسبه و به تابع فراخواننده باز می‌گرداند. با توجه به این که مقدار فاکتوریل عدد دریافتی، ممکن است یک عدد صحیح بسیار بزرگ باشد، نوع مقدار بازگشتی تابع `factorial` را `long long` در نظر گرفته‌ایم.

در **خط شماره (۲)** از تابع `factorial`، یک متغیر صحیح از نوع `long long` به نام `fact` برای نگهداری مقدار محاسبه شده برای فاکتوریل تعریف شده است. مقدار اولیه این متغیر برابر با `1` است.

خط شماره (۳) بیانگر حلقه‌ای است که به تعداد `adad` بار اجرا شده و هر بار در**خط شماره (۴)** مقدار متغیر `adad` را در مقدار قبلی متغیر `fact` ضرب می‌کند.

در نهایت در **خط شماره (۵)** مقدار محاسبه شده برای فاکتوریل متغیر `adad` که در متغیر `fact` قرار دارد، به تابع فراخواننده بازگردانده می‌شود.

در تابع `main` حلقه ای داریم که تعداد `N` بار اجرا شده و هر بار یک عدد از ورودی دریافت می‌کند. در **خط شماره (۸)** تابع `factorial` در داخل یک دستور `cout` فراخوانی شده و مقدار فاکتوریل محاسبه شده برای عدد به شکل مناسبی در خروجی چاپ می‌شود.

نکته: نوع `long long` در زبان C++ برای نگهداری اعداد صحیح بسیار بزرگ مورد استفاده قرار می‌گیرد.

محل تعریف تابع

در تمام مثال‌هایی که برای توابع مطرح شدند، توابع مورد نظر قبل از تابع `main` تعریف شدند. در زبان C++ امکان تعریف توابع بعد از تابع `main` و یا حتی در یک فایل جداگانه نیز وجود دارد. در حالت اول، پیاده‌سازی تابع مورد نظر پس از تابع `main` آورده می‌شود، اما لازم است الگوی آن قبل از تابع `main` معرفی شود. الگوی یک تابع به شکل زیر است:

```
نام و نوع پارامترهای تابع) نام تابع نوع نتیجه تابع(;
```

البته در الگوی فوق نیازی به نوشتن نام پارامترها نیست. به عنوان مثال اگر بخواهیم تعریف تابع masahat در مثال ۱ را قبل از تابع main بنویسیم، باید آن را به شکل زیر بازنویسی کنیم:

```
#include <iostream>
using namespace std;
int masahat(int, int);                                    (1)
int main(){
      int x, y;
      cout << "tool va arz mostatil ra vared konid:";
      cin >> x >> y;
      cout << "masahat mostatil = " << masahat(x,y) << "\n";
      return 0;
}
int masahat(int tool, int arz){                           (2)
      int m;
      m = tool * arz;
      return m;
}
```

خروجی برنامه

نتیجه این برنامه همانند مثال ۱ است.

توضیح برنامه

تنها تفاوت این برنامه با برنامه مثال ۱ در این است که پیاده‌سازی کامل تابع بعد از تابع main آورده شده است و فقط الگوی تابع، قبل از تابع main آورده شده است.

خط شماره (۱) بیانگر الگوی تابع masahat است که شامل نوع مقدار بازگشتی تابع، نام تابع و نوع پارامترهای تابع است که به یک علامت ; ختم شده است.

خط شماره (۲) آغاز تعریف کامل تابع masahat است.

در روشی دیگر می‌توان تعریف تابع مورد نظر را در فایل دیگری قرار داد و در داخل برنامه فراخواننده تابع، فایل مورد نظر را به وسیله یک دستور پیش‌پردازنده به برنامه افزود. به عنوان مثال می‌توانیم تعریف تابع masahat در مثال ۱ را در یک فایل به نام mostatil.h قرار دهیم و برنامه را به شکل زیر بازنویسی کنیم:

```
#include <iostream>
```

```
#include "mostatil.h"                                    (1)
using namespace std;
int main(){
      int x, y;
      cout << "tool va arz mostatil ra vared konid:";
      cin >> x >> y;
      cout << "masahat mostatil = " << masahat(x,y) << "\n";
      return 0;
}
```

خروجی برنامه

نتیجه این برنامه همانند مثال ۱ است.

توضیح برنامه

در **خط شماره (۱)**، فایل `mostatil.h` که حاوی تعریف تابع `masahat` است به کمک دستور پیش‌پردازنده `include` به برنامه افزوده می‌شود. نکته‌ای که باید به آن توجه کرد این است که پس از دستور `include`، نام فایل مربوطه باید داخل دو علامت `""` قرار داده شود.

بخش دوم:

شیءگرایی

مقدمه

در بخش قبلی جزوه با روش برنامه‌نویسی ساخت یافته در زبان C++ آشنا شدیم. در این روش، بخش‌های مختلف یک برنامه را به کمک عبارت‌ها و جملات و با به کارگیری ساختارهای تصمیم و تکرار پیاده‌سازی کردیم. علاوه بر برنامه‌نویسی ساخت یافته، زبان C++ امکانات شیوه برنامه‌نویسی شیءگرا را نیز در اختیار برنامه‌نویس قرار داده است.

شیءگرایی موجب تولید سریع برنامه‌هایی می‌شود که از قابلیت خوانایی و تغییرپذیری بالایی برخوردار هستند و می‌توانند در برنامه‌های دیگر نیز مورد استفاده قرار گیرند. در دنیای برنامه‌نویسی شیءگرا، دامنه مساله شامل تعدادی شیء[9] خواهد بود که اتفاقا معادل‌هایی در دنیای واقعی دارند. به عنوان مثال در مساله محاسبه معدل نمرات تعدادی از دانشجویان، می‌توان دانشجو را به عنوان یک شیء در نظر گرفت.

هر شیء دارای *ویژگی‌هایی* است که در تمام اشیاء هم نوع او نیز وجود دارد. به عنوان مثال، شیء دانشجو دارای ویژگی‌هایی همانند نام و نام خانوادگی و لیست دروس اخذ شده است که در تمام دانشجویان وجود خواهد داشت. خود درس را نیز می‌توان به عنوان یک شیء در نظر گرفت که دارای ویژگی نام درس، تعداد واحد درس و نمره اخذ شده برای آن است. علاوه بر ویژگی‌ها، هر شیء یک یا چند وظیفه نیز خواهد داشت. به عنوان مثال یکی از وظایف شیء دانشجو می‌تواند محاسبه و نمایش معدل دروس اخذ شده باشد(البته این مثال را به شکل بهتری نیز می‌توان بیان کرد که فعلا برای سادگی همین گونه در نظر بگیرید). ویژگی‌های یک شیء به صورت یک متغیر و وظایف آن به صورت یک تابع تعریف می‌شوند. در فصل قبلی با نحوه استفاده از یک تابع (همانند توابع `pow`، `sqrt` و ...) آشنا شدید، اما چگونگی تعریف آن بیان نشد. در این فصل با نحوه تعریف یک تابع نیز آشنا خواهید شد.

در ادامه ابتدا به بیان برخی از مفاهیم و تعاریف اولیه شیءگرایی می‌پردازیم، سپس با ارائه مثال‌هایی قابلیت‌های زبان C++ را برای پیاده‌سازی این مفاهیم شرح خواهیم داد.

۱ـ۲ مفاهیم اولیه شیء گرایی

در فصل گذشته، تمام عبارات، جملات و ساختارهای تکرار و تصمیم به صورت یکجا در داخل تابع `main` نوشته می‌شدند، اما یک برنامه شیءگرا علاوه بر تابع `main` شامل تعریف تعدادی *کلاس* نیز خواهد بود. کلاس، قالبی است که اشیاء هم نوع از روی آن ساخته می‌شوند. به عبارت دیگر اشیای هم‌نوع نمونه‌های یک کلاس هستند. به عنوان مثال، می‌توان کلاسی تحت عنوان دانشجو تعریف کرد که اشیاء دانشجو از روی آن ساخته شوند. ساختار یک کلاس به شکل زیر است:

```
class نام‌کلاس{
    private/public:
```

[9] object

ویژگی‌ها

private/public:

وظایف

};

در ساختار فوق ابتدا کلمه کلیدی `class`، سپس نام مورد نظر برای کلاس و پس از آن علامت‌های { و } برای تعیین محدوده تعریف کلاس آورده می‌شوند. البته نوشتن علامت ; در انتهای تعریف کلاس ضروری است. در بدنه کلاس ویژگی‌ها و وظایف آن آورده می‌شوند.

ویژگی‌های کلاس متغیرهایی هستند که ابعاد داده‌ای کلاس را توصیف می‌کنند. به همین دلیل به آن‌ها *اعضای داده‌ای*[10] نیز می‌گویند. مثلا کلاس دانشجو ممکن است یک ویژگی نام دانشجو داشته باشد که متغیری از نوع رشته است. علاوه بر ویژگی‌ها، یک کلاس می‌تواند یک یا چند وظیفه نیز بر عهده داشته باشد. به عنوان مثال کلاس دانشجو می‌تواند وظیفه نمایش اطلاعات دانشجو را بر عهده داشته باشد. وظایف کلاس به صورت یکسری تابع پیاده‌سازی می‌شوند که به آن‌ها توابع عضو[11] نیز می‌گویند.

یک کلاس زمانی مفید خواهد بود که بتواند به دیگر کلاس‌ها یا توابع خدمتی ارائه کند. به عبارت دیگر، در یک برنامه شیء گرا لازم است از داخل تابع `main` و یا از داخل کلاس‌های دیگر به اعضای داده‌ای و توابع عضو یک کلاس دست یافت و از آن‌ها استفاده کرد. برای این منظور زبان C++ دو روش دسترسی `private` و `public` را در اختیار برنامه‌نویس قرار می‌دهد. در روش `private` اعضای داده‌ای و توابع عضو کلاس، فقط در داخل آن قابل استفاده هستند. اما در روش `public` تمام کلاس‌ها و توابع خارج از کلاس نیز می‌توانند به اعضای داده‌ای و توابع عضو آن دست یابند. در ساختار فوق نحوه استفاده از دو روش نشان داده شده است. همان طور که مشاهده می‌کنید، کافی است که کلمه کلیدی `private` یا `public` را به همراه علامت کولن (:) قبل از تعریف متغیرها و توابع مورد نظر بنویسیم.

تعریف یک کلاس باید قبل از تابع `main` آورده شود و آنچه که داخل تابع `main` وجود خواهد داشت، یک یا چند شیء از نوع کلاس مورد نظر خواهد بود. نحوه تعریف یک شیء از نوع یک کلاس به شکل زیر است:

نام کلاس نام شیء;

همان طور که مشاهده می‌کنید، تعریف یک شیء از نوع یک کلاس همانند تعریف یک متغیر از نوع یکی از انواع شناخته شده همانند `int`، `float` و ... است. بنابراین نامگذاری آن‌ها نیز باید از قواعد نامگذاری متغیرها تبعیت کند.

[10] Data member
[11] Member function

برای دستیابی به اعضای داده‌ای و توابع عضو یک شیء در تابع main نیاز به عملگری به نام عملگر انتخاب عضو است. این عملگر که با علامت نقطه (.) نشان داده می‌شود، پس از نام شیء و قبل از عنوان عضو داده‌ای یا تابع مورد نظر آورده می‌شود:

عنوان عضو داده‌ای. نام شیء

عنوان تابع عضو. نام شیء

یادآوری می‌کنیم که فقط اعضای داده‌ای و توابعی که نوع دسترسی آن‌ها عمومی (public) باشد، به شکل فوق قابل استفاده خواهند بود. برای درک بهتر این مفاهیم به مثال‌هایی که در ادامه آورده شده است، توجه کنید. با توجه به این که اعضای داده‌ای کلاس همان ویژگی‌های کلاس هستند، در طی مثال‌های این بخش این دو کلمه را جابجا استفاده کرده‌ایم.

مثال ۱

برنامه‌ای بنویسید که کلاسی تحت عنوان Student تعریف کند که شامل ویژگی‌های عمومی نام و نام خانوادگی دانشجو باشد. سپس در تابع main یک نمونه شیء از آن بسازد.

```
1 #include <iostream>
2 using namespace std;
3 //Panjaho Chahromin Barnameye C++
4 class Student{
5    public:
6          string first_name, last_name;
7 };
8 int main(){
9    Student student1;
10   return 0;
11 }
```

خروجی برنامه

این برنامه فاقد خروجی است.

توضیح برنامه

هدف از آوردن این مثال، آشنایی شما با نحوه تعریف و استفاده از یک کلاس است. در این مثال کلاسی تحت عنوان کلاس Student داریم که حاوی دو ویژگی (عضو داده‌ای) عمومی به نام‌های first_name و last_name است که برای نگهداری نام و نام خانوادگی دانشجو تعریف شده‌اند. این کلاس هیچ تابعی ندارد.

خط شماره (۴) آغاز تعریف کلاس `Student` است.

در **خط شماره (۵)** کلمه `public` را به همراه علامت کولن (`:`) مشاهده می‌کنید که بیانگر عمومی بودن اعضای داده‌ای کلاس `Student` هستند.

خط شماره (۶) متغیرهای `first_name` و `last_name` را از نوع رشته تعریف می‌کند.

در **خط شماره (۸۷)**، محدوده تعریف کلاس بسته می‌شود.

خط شماره (۸)، آغاز تعریف تابع `main` است.

در **خط شماره (۹)**، یک شیء تحت عنوان `student1` از نوع کلاس `Student` ساخته شده است.

در مثال فوق هر دو ویژگی کلاس `Student` از نوع `public` تعریف شده‌اند و از طریق شیء `student1` قابل دسترس هستند. منظور از قابل دسترس بودن آن است که در تابع `main` می‌توان برای این ویژگی‌ها مقدار تعیین کرد و حتی از مقادیر آن‌ها در قسمت‌های دیگر برنامه استفاده کرد. مثال زیر بیانگر این موضوع است.

مثال ۲

برنامه‌ای بنویسید که یک نمونه شیء از کلاس `Student` بسازد. سپس مقادیر هر دو ویژگی آن را تعیین کرده و در خروجی استاندارد چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 //Panjaho Panjomin Barnameye C++
4  class Student{
5     public:
6          string first_name, last_name;
7  };
8 int main(){
9     Student student1;
10    student1.first_name = “ali”;
11    student1.last_name = “alavi”;
12    cout << student1.first_name << “ “
13         << student1.last_name
14         << “ yek daneshjoo ast.”
15         << endl;
16    return 0;
17 }
```

خروجی برنامه

```
ali alavi yek daneshjoo ast.
```

توضیح برنامه

همان طور که مشاهده می‌کنید به راحتی به کمک عملگر انتخاب عضو می‌توانیم به اعضای داده‌ای عمومی کلاس دست یافته و مقدار آن‌ها را تعیین و استفاده کنیم.

خطوط شماره (۱۰) و (۱۲) ویژگی‌های شیء `student1` را مقداردهی می‌کنند. برای این منظور در خطوطی جداگانه ابتدا نام شیء `student1` سپس عملگر انتخاب عضو(`.`) و پس از آن عنوان هریک از ویژگی‌ها به همراه مقداری متناسب آورده شده‌اند. همان طور که مشاهده می‌کنید، مقداردهی ویژگی‌های یک شیء همانند مقداردهی یک متغیر عادی است.

خط شماره (۱۲) آغاز دستور چاپی است که از مقادیر ویژگی‌های شیء برای چاپ یک پیغام استفاده کرده است. همانند عمل مقداردهی، دسترسی به مقادیر ویژگی‌های شیء `student1` نیز به کمک عملگر انتخاب عضو انجام شده است.

مثال فوق بیانگر کلاسی است که متناظر با یک موجودیت در دنیای واقعی است. اما در برنامه‌نویسی شیء گرا ممکن است نیاز به تعریف کلاس‌هایی باشد که هیچ معادل قابل لمسی در دنیای واقعی نداشته باشند. مثال زیر بیانگر این موضوع است.

مثال ۳

برنامه‌ای بنویسید که کلاسی تحت عنوان `Course` تعریف کند که شامل ویژگی‌های عمومی شماره درس، عنوان درس و تعداد واحدهای آن باشد. سپس یک نمونه شیء از آن بسازد و مقادیر ویژگی‌های آن‌ها را تعیین کرده و در خروجی چاپ کند.

```
1 #include <iostream>
2 using namespace std;
3 //Panjaho Sheshomin Barnameye C++
4 class Course{
5     public:
6           int identifier, number;
7           string title;
8 };
9 int main(){
10    Course course1;
11    course1.id = 1000;
12    course1.title = “programming”;
```

```
13    dars1.number = 3;
14    cout << “dars ” << course1.title
15          << “ be shomareye ” << course1.identifier
16          << “ va tedad vahed ” << course1.number
17          << “ ast.”
18          << endl;
19    return 0;
20 }
```

خروجی برنامه

```
dars programming be shomareye 1000 va tedad vahed 3 ast.
```

توضیح برنامه

کلاس Course حاوی سه عضو داده‌ای به نام‌های identifier، title و number است که هر سه به صورت عمومی تعریف شده‌اند. ویژگی identifier برای نگهداری شماره درس، ویژگی title برای نگهداری عنوان درس و ویژگی number برای نگهداری تعداد واحدهای درس تعریف شده‌اند. همانند مثال قبلی، در این مثال نیز در تابع main به هریک از ویژگی‌ها دست یافته و مقدار آن‌ها را تعیین نموده‌ایم. سپس از مقادیر آن‌ها در چاپ خروجی استفاده کرده‌ایم.

در برنامه‌های فوق می‌توان مقادیر ویژگی‌های کلاس را از ورودی استاندارد دریافت کرد. به عنوان مثال، تابع main برنامه فوق را می‌توان به شکل زیر نیز تغییر داد:

```
9 int main(){
10    Course course1;
11    cout << “shomare dars ra vared konid: “;
12    cin >> course1.identifier;
13    cout << “nam dars ra vared konid: “;
14    cin >> course1.title;
15    cout << “tedad vahed dars ra vared konid: “;
16    cin >> course1.number;
17    cout << “dars ” << dars1.title
18          << “ be shomareye ” << dars1.identifier
19          << “ va tedad vahed ” << dars1.number
20          << “ ast.”
21          << endl;
22    return 0;
```

```
23 }
```

خروجی برنامه

```
shomare dars ra vared konid: 1000
nam dars ra vared konid: programming
tedad vahed dars ra vared konid: 3
dars programming be shomareye 1000 va tedad vahed 3 ast.
```

۲ـ۲ تعریف یک تابع عضو برای کلاس

همان طورکه قبلا هم اشاره شد، کلاس‌ها می‌توانند علاوه بر ویژگی‌ها، وظایفی نیز برعهده داشته باشند. وظایف یک کلاس از طریق توابع پیاده‌سازی می‌شوند. قبلا با نحوه تعریف و فراخوانی یک تابع آشنا شدیم. همانند ویژگی‌های کلاس، توابع کلاس نیز می‌ـ توانند از نوع `private` و یا `public` تعریف شوند. در صورتی که نحوه دسترسی به تابع عضو یک کلاس از نوع `public` باشد، به راحتی می‌توان در تابع `main` از آن استفاده کرد. در ادامه مثال‌هایی در این رابطه مشاهده می‌کنید.

مثال ۴

برنامه‌ای بنویسید که یک تابع عضو عمومی برای نمایش اطلاعات دانشجو به کلاس `Student` بیفزاید. سپس یک نمونه شیء از کلاس بسازد و تابع نمایش اطلاعات آن را فراخوانی کند.

```
1 #include <iostream>
2 using namespace std;
3 //Panjaho Haftomin Barnameye C++
4  class Student{
5    public:
6          string first_name, last_name;
7          void print(){
8               cout << “man ” << first_name << “ “
9                    << last_name
10                   << “ yek daneshjoo hastam.” << endl;
11         }
12  };
13 int main(){
14   Student student1;
15   cout << “nam daneshjoo ra vared konid: “;
16   cin >> student1.first_name;
```

```
17    cout << "nam khanevadegi daneshjoo ra vared konid: ";
18    cin >> student1.last_name;
19    student1.print();
20    return 0;
21 }
```

خروجی برنامه

```
nam daneshjoo ra vared konid: ali
nam khanevadegi daneshjoo ra vared konid: alavi
man ali alavi yek daneshjoo hastam.
```

توضیح برنامه

همان طور که مشاهده می‌کنید کلاس Student علاوه بر ویژگی‌های عمومی first_name و last_name دارای یک تابع عمومی به نام print است. وظیفه این تابع، چاپ اطلاعات دانشجو در خروجی استاندارد است.

خط شماره (۷) آغاز تعریف تابع print برای نمایش اطلاعات دانشجو در خروجی استاندارد است. این تابع هیچ نتیجه‌ای به فراخواننده خود (یعنی تابع main) باز نمی‌گرداند به همین دلیل نوع مقدار بازگشتی آن را با کلمه void مشخص نموده‌ایم. با توجه به این‌که تابع print نیازی به اطلاعات ورودی ندارد، هیچ پارامتری برای آن در نظر گرفته نشده است.

خط شماره (۸) دستور چاپ اطلاعات دانشجو در خروجی است. همان طور که مشاهده می‌کنید، برای استفاده از مقادیر اعضای داده‌ای کلاس در تابع عضو همان کلاس، ذکر نام متغیر مربوطه کافی است.

در تابع main پس از دریافت مقادیر ویژگی‌های کلاس از ورودی استاندارد در **خطوط (۱۵) تا (۱۸)**، در **خط شماره (۱۹)** تابع print شیء student1 به کمک عملگر انتخاب عضو فراخوانی می‌شود. این خط بیانگر نحوه فراخوانی یک تابع بدون پارامتر و بدون مقدار بازگشتی است.

نکته: همان طور که در مثال فوق نیز مشاهده می‌کنید، در صورتی که نوع دسترسی به توابع عضو و ویژگی‌های کلاس یکسان باشد، یکبار نوشتن کلمه private یا public در کلاس کافی است.

۳ـ۲ توابع get و set

در مثال‌های فوق، ویژگی‌های یا همان اعضای داده‌ای کلاس به صورت public تعریف شده‌اند. به همین دلیل تمام اشیایی که از نوع این دو کلاس ساخته می‌شوند، می‌توانند به این اعضا دسترسی داشته باشند. اما در شیءگرایی یک اصل مهم به نام اصل پنهان‌سازی اطلاعات[۱۲] وجود دارد که طبق آن بهتر است داده‌های (اعضای داده‌ای) یک کلاس از دسترسی مستقیم کلاس‌ها و توابع دیگر پنهان باقی بمانند. تا از این طریق پیچیدگی‌های احتمالی کار با این داده‌ها از دید کلاس‌های بیرونی مخفی باقی بماند. به همین دلیل معمولا برنامه‌نویسان نوع دسترسی به ویژگی‌های

[۱۲] Information hiding

کلاس را از نوع private در نظر می‌گیرند. در این حالت فقط توابع عضو کلاس می‌توانند به طور مستقیم به این ویژگی‌ها دسترسی داشته باشند و به عنوان مثال نمی‌توان در تابع main و به کمک عملگر انتخاب عضو، مقدار آن‌ها را تنظیم یا استفاده کرد.

در برنامه‌نویسی شیءگرا برای خواندن و تغییر مقدار اعضای داده‌ای private یک کلاس معمولا از دو تابع عضو به نام‌های توابع get و set استفاده می‌شود. این دو تابع عمومی، امکان دسترسی غیر مستقیم توابع خارج از کلاس را به ویژگی‌های خصوصی یک کلاس فراهم می‌کنند. برای درک بهتر این مفاهیم به مثال‌های بعدی توجه کنید.

مثال ۵

برنامه‌ای بنویسید که دو تابع عضو عمومی به نام‌های توابع get و set را به ترتیب برای خواندن و تغییر مقدار اعضای داده‌ای کلاس به کلاس Student بیفزاید. سپس یک نمونه شیء از کلاس بسازد و توابع آن را فراخوانی کند.

```
1 #include <iostream>
2 using namespace std;
3 //Panjaho Hashtomin Barnameye C++
4  class Student{
5    private:
6          string first_name, last_name;
7    public:
8          string getFirstName(){
9                return first_name;
10         }
11         void setFirstName(string f){
12               first_name = f;
13         }
14         string getLastName(){
15               return last_name;
16         }
17         void setLastName(string l){
18               last_name = l;
19         }
20         void print(){
```

```
21              cout << “man ” << getFirstName()
22              << “ “ << getLastName()
23              << “ yek daneshjoo hastam.” << endl;
24          }
25  };
26 int main(){
27    Student student1;
28    student1.setFirstName(“ali”);
29    student1.setLastName(“alavi”);
30    student1.print();
31    return 0;
32 }
```

خروجی برنامه

```
man ali alavi yek daneshjoo hastam.
```

توضیح برنامه

همان طور که مشاهده می‌کنید در این مثال ویژگی‌های کلاس از نوع `private` تعریف شده‌اند. پس نمی‌توان در تابع `main` آن‌ها را مقداردهی کرد. به همین دلیل به ازای هر یک از ویژگی‌های`first_name` و `last_name` یک تابع `set` متناسب برای تنظیم مقدار متغیر و یک تابع `get` برای برگرداندن مقدار متغیر تعریف شده‌اند.

در **خط شماره (۵)** نوع دسترسی دو عضو داده‌ای کلاس از نوع `private` تعریف شده است. بنابراین فقط توابع داخل کلاس می‌توانند به این اعضا دسترسی داشته باشند و استفاده از آن‌ها در خارج از کلاس، موجب خطای کامپایلری خواهد شد.

در **خط شماره (۷)** نوع دسترسی توابع `get` و `set` تمام ویژگی‌ها و همچنین تابع `print` به صورت `public` تعریف شده است. بنابراین علاوه بر محدوده کلاس، تمام توابع خارج از کلاس از جمله تابع `main` نیز می‌توانند این توابع را فراخوانی کنند.

خط شماره (۸) نحوه تعریف یک تابع `get` را برای یک برگرداندن مقدار یک ویژگی از کلاس را نشان می‌دهد. با توجه به این که وظیفه این تابع بازگرداندن مقدار متغیر `first_name` است، آن را `getFirstName` نامگذاری کرده‌ایم. همچنین چون متغیر `first_name` یک رشته است، نوع مقدار بازگشتی تابع را `string` در نظر گرفته‌ایم.

خط شماره (۹) تنها دستور موجود در بدنه تابع است. این دستور مقدار متغیر `first_name` را به کمک دستور `return` به فراخواننده تابع بازمی‌گرداند.

خط شماره (۱۱) بیانگر تعریف یک تابع `set` برای تنظیم مقدار یک ویژگی از کلاس است. با توجه به این که وظیفه این تابع فقط تنظیم مقدار متغیر `first_name` است، آن را `setFirstName` نامگذاری کرده و نوع

مقدار بازگشتی آن را void در نظر گرفته‌ایم. علاوه بر این، یک متغیر رشته‌ای به نام f را نیز به عنوان پارامتر تابع در داخل پرانتز تعریف کرده‌ایم.

خط شماره (۱۲) به عنوان تنها دستور تابع، مقدار پارامتر f را در متغیر first_name کپی می‌کند.

خطوط شماره (۱۴) تا (۱۹) نحوه تعریف توابع get و set متناسب با ویژگی last_name را نشان می‌دهند.

خط شماره (۲۰) بیانگر تعریف تابع print برای نمایش اطلاعات دانشجو است. همان طورکه مشاهده می‌کنید، برای خواندن مقادیر متغیرهای first_name و last_name از توابع get مربوطه استفاده شده است. البته تابع print به عنوان یک تابع عضو کلاس در Student اجازه دسترسی مستقیم به این متغیرها را دارد و صرفا برای آشنایی شما با نحوه استفاده از تابع get به این شکل تعریف شده است.

خطوط شماره (۲۸) تا (۲۹) با فراخوانی توابع set مربوط به هریک از ویژگی‌ها، مقدار آن‌ها را تنظیم می‌کنند.

در **خط شماره (۳۰)** تابع print فراخوانی شده و اطلاعات دانشجو در خروجی استاندارد چاپ می‌شود.

در مثال فوق می‌توان مقادیر ویژگی‌های کلاس را از ورودی استاندارد دریافت کرد. برای این منظور تابع main را به شکل زیر می‌توان تغییر داد:

```
26 int main(){
27    Student student1;
28    string fname, lname;
29    cout << “nam daneshjoo ra vared konid: “;
30    cin >> fname;
31    cout << “nam khanevadegi daneshjoo ra vared konid: “;
32    cin >> lname;
33    student1.setFirstName(fname);
34    student1.setLastName(lname);
35    student1.print();
36    return 0;
37 }
```

خروجی برنامه

```
nam daneshjoo ra vared konid: ali
nam khanevadegi daneshjoo ra vared konid: alavi
man ali alavi yek daneshjoo hastam.
```

۴ـ۲ توابع سازنده (Constructor)

همان طور که در مثال‌های فوق نیز مشاهده می‌کنید، قبل از فراخوانی تابع print برای چاپ اطلاعات دانشجو، مقادیر ویژگی‌های کلاس به کمک توابع set مربوطه تنظیم شده‌اند. حال اگر برنامه‌نویس به اشتباه یا به عمد، بدون تعیین مقادیر ویژگی‌های کلاس تابع print را فراخوانی کند، خروجی برنامه چه خواهد بود؟ پاسخ این سوال بستگی به کامپایلری دارد که برنامه‌نویس برای کامپایل برنامه استفاده می‌کند. به عنوان مثال نتیجه اجرای این برنامه در کامپایلر devC++ حاوی یک رشته خالی برای ویژگی‌های کلاس است:

```
man yek daneshjoo hastam
```

این نتیجه نشان می‌دهد که در صورت عدم تعیین مقدار اولیه برای یک متغیر از نوع string در کامپایلر devC++، مقدار پیش فرض رشته خالی برای آن در نظر گرفته می‌شود. بنابراین با توجه به این که کامپایلر مورد استفاده چه مقدار پیش فرضی برای انواع داده در نظر گرفته است، پاسخ‌های متفاوتی برای این سوال وجود خواهد داشت.

برای پیشگیری از این گونه اشتباهات، می‌توان مقادیر ویژگی‌های یک شیء را در همان لحظه ساخت آن مشخص کرد. برای این منظور از توابعی به نام *توابع سازنده* استفاده می‌شود. سازنده تابعی است که در داخل کلاس تعریف می‌شود و به هنگام ساخت یک شیء از روی کلاس مورد نظر فراخوانی می‌شود. این تابع که معمولا وظیفه مقداردهی اولیه اعضای داده‌ای کلاس را بر عهده دارد، باید همنام با کلاس باشد و هیچ مقداری باز نمی‌گرداند. برای درک این مفاهیم، به مثال زیر توجه نمایید.

مثال ۶

برنامه‌ای بنویسید که یک تابع سازنده برای تعیین مقادیر اولیه ویژگی‌های کلاس Student به آن اضافه کند.

```
1 #include <iostream>
2 using namespace std;
3 //Panjaho Nohomin Barnameye C++
4 class Student{
5     private:
6           string first_name, last_name;
7     public:
8           Student(string fname, string lname){
9                 setFirstName(fname);
10                setLastName(lname);
11          }
12          string getFirstName(){
13                return first_name;
14          }
```

```
15          void setFirstName(string f){
16                first_name = f;
17          }
18          string getLastName(){
19                return last_name;
20          }
21          void setLastName(string l){
22                last_name = l;
23          }
24          void print(){
25                cout << “man ” << getFirstName() << “ “
26                << getLastName()
27                      << “ yek daneshjoo hastam.” << endl;
28          }
29   };
30 int main(){
31     Student student1(”ali”,”alavi”);
32     student1.print();
33     Student student2(”mohammad”,”mohammadi”);
34     student2.print();
35     return 0;
36 }
```

خروجی برنامه

```
man ali alavi yek daneshjoo hastam.
man mohammad mohammadi yek daneshjoo hastam.
```

توضیح برنامه

این مثال حاوی تعریف یک سازنده با دو پارامتر رشته‌ای برای تعیین مقادیر ویژگی‌های کلاس `Student` است. علاوه بر این در این مثال، دو نمونه شیء با نام‌های `student1` و `student2` از روی کلاس ایجاد شده‌اند.

خطوط شماره (۸) تا (۱۱) بیانگر تعریف یک تابع سازنده برای کلاس `Student` است. در تعریف این تابع نکاتی وجود دارد که باید به آن توجه کرد. ۱) نحوه دسترسی به این تابع از نوع `public` است تا به راحتی بتوان از توابع خارج از کلاس آن را فراخوانی کرد. ۲) نام این تابع باید با نام کلاس یکسان باشد. ۳) این تابع هیچ مقداری باز نمی‌گرداند و البته نیازی به نوشتن کلمه `void` هم در تعریف آن وجود ندارد.

خطوط شماره (۳۱) و (۳۳) نحوه تعریف اشیاء `student1` و `student2` را با فراخوانی تابع سازنده تعریف شده در کلاس نشان می‌دهند. همان طور که مشاهده می‌کنید ابتدا نام کلاس، سپس نام شیء مورد نظر و پس از آن، پارامترهای تابع سازنده در داخل یک پرانتز نوشته می‌شوند.

نکته: هر کلاس زبان C++ به طور ضمنی شامل سازنده‌ای است که هیچ پارامتری دریافت نمی‌کند و بدنه آن حاوی هیچ دستوری نیست.

نکته: یک کلاس می‌تواند شامل تعریف چندین تابع سازنده باشد که همگی همنام هستند. البته تعداد و نوع پارامترهای آن‌ها باید متفاوت باشند. در این صورت باید هنگام تعریف شیئی از نوع کلاس، سازنده مورد نظر را مشخص نمود.

۵ـ۲ وراثت (Inheritance)

وراثت یک مفهوم کلیدی در برنامه‌نویسی شیء گرا است. به کمک این ویژگی می‌توان کلاس‌هایی تعریف کرد که ویژگی‌ها و رفتارهای کلاس‌های موجود را دریافت می‌کنند و آن‌ها را برای ایجاد اشیائی خاص‌تر توسعه می‌دهند. به عنوان مثال یک دانشجو به نوبه خود یک انسان است که دارای ویژگی‌های نام و نام خانوادگی است. به عبارت بهتر چون دانشجو یک انسان است پس به صورت خودکار دارای نام و نام خانوادگی خواهد بود. علاوه بر این یک دانشجو می‌تواند در طول ترم تعدادی درس اخذ کند. از طرف دیگر استاد هم یک انسان است که دارای نام و نام خانوادگی است و وهمچنین می‌تواند در طول یک ترم تعدادی درس ارائه کند. با دقت در این مثال می‌توان به این نتیجه رسید که ویژگی‌های نام و نام خانوادگی جزو ویژگی‌های مشترک دانشجو و استاد هستند که به واسطه انسان بودن آن‌ها وجود دارد. در این حالت می‌توان یک کلاس کلی‌تر به نام `Person` با ویژگی‌های نام و نام خانوادگی تعریف کرد و با استفاده از مفهوم وراثت، این ویژگی‌ها را به کلاس‌های دانشجو و استاد منتقل کرد. در این حالت، کلاس `Person` را *کلاس اصلی*[۱۳] یا پدر و کلاس‌های دانشجو و استاد *را کلاس‌های مشتق شده*[۱۴] یا فرزند گویند. نحوه تعریف به ارث‌بری یک کلاس از کلاسی دیگر به شکل زیر است:

```
class نام کلاس فرزند :public/private نام کلاس پدر{
    بدنه کلاس
}
```

همان طور که مشاهده می‌کنید ابتدا نام کلاس فرزند سپس علامت کولن (`:`)، پس از آن نحوه به ارث‌بری و در نهایت نام کلاس اصلی آورده می‌شود. نحوه به ارث‌بری در ساختار فوق یکی از روش‌های `public` و `private` است. مفهوم این کلمات همانند مفهوم دسترسی برای اعضای داده‌ای و توابع عضو است. به عبارت دیگر نوع به ارث‌بری `private` موجب می‌شود که کلاس فرزند فقط به ویژگی‌ها و توابع عمومی کلاس پدر دسترسی داشته باشد. اما در نوع `public`، اجازه دسترسی به تمام ویژگی‌ها و توابع عمومی کلاس پدر برای کلاس فرزند مهیا است. برای درک بهتر این مفاهیم به مثال زیر توجه کنید.

---

[۱۳] Base class
[۱۴] Derived class

مثال ۷

برنامه‌ای بنویسید که ابتدا کلاسی تحت عنوان Person با ویژگی‌های نام و نام خانوادگی تعریف کند. سپس دو کلاس دیگر به نام‌های Student و Teacher تعریف کند که ویژگی‌های نام و نام خانوادگی خود را از کلاس Person به ارث می‌برند.

```
1 #include <iostream>
2 using namespace std;
3 //Shastomin Barnameye C++
4 class Person{
5     private:
6           string first_name;
7           string last_name;
8     public:
9           void setFirstName(string fname){
10                first_name = fname;
11          }
12          void setLastName(string lname){
13                last_name = lname;
14          }
15          string getFirstName(){
16                return first_name;
17          }
18          string getLastName(){
19                return last_name;
20          }
21          void print(){
22                cout << “man “ << getFirstName()
23                << “  “ << getLastName()
24                << ” yek ensan hastam.” << endl;
25          }
26 };
27 class Student: public Person{
28 };
```

```
29 class Teacher: public Person{
30 };
31 int main(){
32    Person person1;
33    person1.setFirstName("ali");
34    person1.setLastName("alavi");
35    person1.print();
36    Student student1;
37    student1.setFirstName("mohammad");
38    student1.setLastName("mohammadi");
39    student1.print();
40    Teacher teacher1;
41    teacher1.setFirstName("ahmad");
42    teacher1.setLastName("ahmadi");
43    teacher1.print();
44    return 0;
45 }
```

خروجی برنامه

```
man ali          alavi yek ensan hastam.
man mohammad     mohammadi yek ensan hastam.
man ahmad        ahmadi yek ensan hastam.
```

توضیح برنامه

در این مثال سه کلاس تحت عناوین Person، Student و Teacher تعریف شده‌اند که کلاس‌های Student و Teacher ویژگی‌های خود را از کلاس Person به ارث برده‌اند. همان طور که مشاهده می‌کنید این کلاس‌ها خود حاوی هیچ ویژگی یا تابعی نیستند و از ویژگی‌ها و توابع کلاس Person استفاده می‌کنند.

خط شماره (۴) آغاز تعریف کلاس Person است که حاوی دو عضو داده‌ای first_name و last_name برای نگهداری ویژگی‌های نام و نام خانوادگی است.

خطوط شماره (۲۷) و (۲۹) به ترتیب حاوی تعریف کلاس‌های Student و Teacher هستند که نحوه به ارث بری این کلاس‌ها را از کلاس Person نشان می‌دهند. استفاده از کلمه public برای مشخص کردن نحوه به ارث‌بری کلاس‌های Student و Teacher، موجب شده است که تمام ویژگی‌ها و توابع تعریف شده در کلاس Teacher به راحتی در این کلاس‌ها مورد دسترسی قرار گیرند.

خطوط شماره (۳۲) تا (۳۵) شیئی به نام person1 را از نوع کلاس Person تعریف کرده و ویژگی‌های نام و نام خانوادگی آن را با مقادیر مناسب مقداردهی می‌کنند. سپس تابع print آن را فراخوانی کرده و مقادیر ویژگی‌های این شیء را در خروجی چاپ می‌کنند.

خط شماره (۳۶) شیئی با نام student1 از نوع کلاس Student تعریف می‌کند. سپس همان طور که مشاهده می‌کنید در خطوط **شماره (۳۷) تا (۳۹)** توابع setFirstName، setLastName و print علیرغم این که تعریف آن‌ها در کلاس Student وجود ندارد فراخوانی شده‌اند و این امر منجر به فراخوانی توابع موجود در کلاس Person خواهد شد.

خطوط شماره (۴۰) تا (۴۳) شیئی به نام teacher1 را از نوع کلاس Teacher تعریف کرده و توابع setFirstName، setLastName و print را از کلاس اصلی (Person) فراخوانی می‌کنند.

در مثال فوق کلاس Person کلاس پدر و کلاس‌های Student و Teacher کلاس‌های فرزند هستند.

همان طور که در ابتدای این بخش اشاره شد، کلاس‌های فرزند علاوه بر به ارث بردن ویژگی‌ها و توابع کلاس پدر، می‌توانند شامل ویژگی‌ها یا توابع دیگری نیز باشند. همچنین این کلاس‌ها می‌توانند توابع به ارث رسیده از کلاس پایه را توسعه دهند. به مثال‌های زیر توجه کنید.

مثال ۸

مثال ۷ را با تغییر متناسب تابع print در کلاس‌های Student و Teacher بازنویسی کنید.

```
1 #include <iostream>
2 using namespace std;
3 //Shasto Yekomin Barnameye C++
4 class Person{
5    private:
6          string first_name;
7          string last_name;
8    public:
9          void setFirstName(string fname){
10               first_name = fname;
11         }
12         void setLastName(string lname){
13               last_name = lname;
14         }
15         string getFirstName(){
16               return first_name;
```

```
17          }
18          string getLastName(){
19                return last_name;
20          }
21          void print(){
22                cout << “man “ << getFirstName()
23                << “  “ << getLastName()
24                << ” yek ensan hastam.” << endl;
25          }
26 };
27 class Student: public Person{
28    public:
29          void print(){
30                cout << “man “ << getFirstName()
31                << “  “ << getLastName()
32                << ” yek daneshjoo hastam.” << endl;
33          }
34 };
35 class Teacher: public Person{
36    public:
37          void print(){
38                cout << “man “ << getFirstName()
39                << “  “ << getLastName()
40                << ” yek modarres hastam.” << endl;
41          }
42 };
43 int main(){
44    Person person1;
45    person1.setFirstName("ali");
46    person1.setLastName("alavi");
47    person1.print();
48    Student student1;
49    student1.setFirstName("mohammad");
```

```
50    student1.setLastName("mohammadi");
51    student1.print();
52    Teacher teacher1;
53    teacher1.setFirstName("ahmad");
54    teacher1.setLastName("ahmadi");
55    teacher1.print();
56    return 0;
57 }
```

خروجی برنامه

```
man ali  alavi yek ensan hastam.
man mohammad mohammadi yek daneshjoo hastam.
man ahmad ahmadi yek modarres hastam.
```

توضیح برنامه

در این مثال تابع print در کلاس‌های Student و Teacher مجددا تعریف شده است.

خطوط شماره (۲۹) تا (۳۳) از کلاس Student **و (۳۷) تا (۴۱)** از کلاس Teacher محتوای این تابع را به طور متناسب برای کلاس‌های مربوطه تغییر می‌دهند.

بنابراین فراخوانی‌های این تابع در **خطوط شماره (۵۱) و (۵۵)** منجر به فراخوانی تابع موجود در هر کلاس خواهد شد.

مثال ۹

مثال ۸ را با افزودن ویژگی‌های مقطع تحصیلی در کلاس Student و آخرین مدرک تحصیلی در کلاس Teacher بازنویسی کنید.

```
1 #include <iostream>
2 using namespace std;
3 //Shasto Dovvomin Barnameye C++
4 class Person{
5     private:
6           string first_name;
7           string last_name;
8     public:
9           void setFirstName(string fname){
10                first_name = fname;
```

```
11          }
12          void setLastName(string lname){
13                last_name = lname;
14          }
15          string getFirstName(){
16                return first_name;
17          }
18          string getLastName(){
19                return last_name;
20          }
21          void print(){
22                cout << “man “ << getFirstName()
23                << “  “ << getLastName()
24                << ” yek ensan hastam.” << endl;
25          }
26 };
27 class Student: public Person{
28    private:
29          string level;
30    public:
31          string getLevel(){
32                return level;
33          }
34          void setLevel(string l){
35                level = l;
36          }
37          void print(){
38                cout << “man “ << getFirstName()
39                << “  “ << getLastName()
40                << ” yek daneshjoo dar maghtaa ”
41                << getLevel() << “ hastam.” << endl;
42          }
43 };
```

```
44 class Teacher: public Person{
45   private:
46         string degree;
47   public:
48         string getDegree(){
49               return degree;
50         }
51         void setDegree(string d){
52               degree = d;
53         }
54         void print(){
55               cout << “man “ << getFirstName()
56               << “  “ << getLastName()
57               << ” yek modarres ba madrak ”
58               << getDegree() << “ hastam.” << endl;
59         }
60 };
61 int main(){
62   Person person1;
63   person1.setFirstName("ali");
64   person1.setLastName("alavi");
65   person1.print();
66   Student student1;
67   student1.setFirstName("mohammad");
68   student1.setLastName("mohammadi");
69   student1.setLevel(“karshenasi”);
70   student1.print();
71   Teacher teacher1;
72   teacher1.setFirstName("ahmad");
73   teacher1.setLastName("ahmadi");
74   teacher1.setDegree(“phd”);
75   teacher1.print();
76   return 0;
```

```
77 }
```

خروجی برنامه

```
man ali          alavi yek ensan hastam.
man mohammad     mohammadi yek daneshjoo dar maghtaa karshenasi hastam.
man ahmad        ahmadi yek modarres ba madrak phd hastam.
```

توضیح برنامه

در این مثال ویژگی‌های جدیدی تحت عناوین Level و Degree برای نگهداری مقطع و مدرک تحصیلی در کلاس‌های Student و Teacher تعریف شده‌اند.

خطوط شماره (۲۹) و (۴۶) شامل تعریف ویژگی‌های Level و Degree با نوع دسترسی private در کلاس‌های Student و Teacher هستند.

خطوط شماره(۳۱) تا (۳۶) از کلاس Student و **خطوط شماره (۴۸) و (۵۳)** از کلاس Teacher شامل تعریف توابع get و set متناسب با ویژگی‌های تعریف شده جدید هستند.

به همین ترتیب توابع print هر دو کلاس با توجه به ویژگی جدید هریک از کلاس‌ها اصلاح شده‌اند. در **خط شماره (۴۱)** تابع getLevel و در **خط شماره (۵۸)** تابع getDegree در دستور چاپ اطلاعات مربوطه فراخوانی شده‌اند.

خط شماره (۶۹) تابع setLevel را برای مقداردهی اولیه متغیر Level از کلاس Student و **خط شماره (۷۴)** تابع setDegree را برای مقداردهی اولیه متغیر Degree از کلاس Teacher فراخوانی کرده‌اند.

در صورتی که در وراثت کلاس پدر حاوی یک تابع سازنده پارامتردار باشد، می‌توان آن را در تابع سازنده فرزند فراخوانی کرد. برای این منظور باید تابع سازنده کلاس فرزند را به شکل زیر تعریف کرد:

عنوان پارامترهای متناسب) عنوان تابع سازنده پدر: (لیست پارامترها به همراه نوع آن‌ها) عنوان تابع سازنده فرزند (با تابع سازنده پدر

همان طور که مشاهده می‌کنید، ابتدا عنوان تابع سازنده کلاس فرزند، سپس لیست پارامترهای تابع به همراه نوع آن‌ها (که شامل پارامترهای تابع سازنده پدر نیز می‌باشند)، سپس علامت کولن (:) آورده می‌شود. پس از این علامت، تابع سازنده کلاس پدر فراخوانی شده و مقادیر پارامترهای آن تعیین می‌شوند. برای درک این مفهوم به مثالی که در ادامه آورده شده است، توجه نمایید.

مثال ۱۰

مثال ۹ را با افزودن یک تابع سازنده پارامتردار به کلاس Person بازنویسی کنید.

```
1 #include <iostream>
```

```
2 using namespace std;
3 //Shasto Sevvomin Barnameye C++
4 class Person{
5    private:
6          string first_name;
7          string last_name;
8    public:
9          Person(string fname, string lname){
10               first_name = fname;
11               last_name = lname;
12         }
13         void setFirstName(string fname){
14               first_name = fname;
15         }
16         void setLastName(string lname){
17               last_name = lname;
18         }
19         string getFirstName(){
20               return first_name;
21         }
22         string getLastName(){
23               return last_name;
24         }
25         void print(){
26               cout << “man “ << getFirstName()
27               << “  “ << getLastName()
28               << ” yek ensan hastam.” << endl;
29         }
30 };
31 class Student: public Person{
32   private:
33         string level;
34   public:
```

```
35          Student(string fname, string lname,
36          string l):Person(fname,lname){
37                setLevel(l);
38          }
39          string getLevel(){
40                return level;
41          }
42          void setLevel(string l){
43                level = l;
44          }
45          void print(){
46                cout << “man “ << getFirstName()
47                << “  “ << getLastName()
48                << ” yek daneshjoo dar maghtaa ”
49                << getLevel() << “ hastam.” << endl;
50          }
51 };
52 class Teacher: public Person{
53    private:
54          string degree;
55    public:
56          Teacher(string fname, string lname,
57          string d):Person(fname,lname){
58                setDegree(d);
59          }
60          string getDegree(){
61                return degree;
62          }
63          void setDegree(string d){
64                degree = d;
65          }
66          void print(){
67                cout << “man “ << getFirstName()
```

```
68              << “  “ << getLastName()
69              << ” yek modarres ba madrak ”
70              << getDegree() << “ hastam.” << endl;
71         }
72 };
73 int main(){
74    Person person1(“ali”,”alavi”);
75    person1.print();
76    Student student1(“mohammad”,”mohammadi”,”karshensi”);
77    student1.print();
78    Teacher teacher1(“ahmad”,”ahmadi”,”phd”);
79    teacher1.print();
80    return 0;
81 }
```

خروجی برنامه

```
man ali          alavi yek ensan hastam.
man mohammad     mohammadi yek daneshjoo dar maghtaa karshensi hastam.
man ahmad        ahmadi yek modarres ba madrak phd hastam.
```

توضیح برنامه

در این مثال یک تابع سازنده با دو پارامتر رشته‌ای برای مقداردهی اولیه ویژگی‌های first_name و last_name در کلاس Person تعریف شده است. علاوه بر این در هریک از کلاس‌های Student و Teacher نیز یک تابع سازنده با پارامترهای متناسب تعریف شده است.

خط شماره (۹) آغاز تعریف تابع سازنده کلاس Person است که به کمک آن مقادیر اولیه ویژگی‌های first_name و last_name مشخص می‌شوند.

خطوط شماره (۳۵) و (۵۶) نحوه تعریف توابع سازنده کلاس‌های Student و Teacher را نشان می‌دهند. همان طور که مشاهده می‌کنید این توابع دارای سه پارامتر رشته‌ای برای تعیین مقادیر ویژگی‌های نام و نام خانوادگی و ویژگی اضافی هر کلاس هستند. با توجه به این که مقادیر ویژگی‌های first_name و last_name را می‌توان به کمک تابع سازنده کلاس Person تعیین کرد، به هنگام تعریف توابع سازنده کلاس‌های Student و Teacher آن را فراخوانی کرده و مقدار پارامتر fname و lname را در اختیار آن قرار داده‌ایم. مقدار پارامتر سوم توابع سازنده هر دو کلاس فرزند در داخل توابع سازنده هر یک از کلاس مصرف شده است.

در **خطوط شماره (۷۴) تا (۷۶) و (۷۸)** به ترتیب توابع سازنده کلاس `Person`، کلاس `Student` و کلاس `Teacher` به هنگام ساخت اشیای مورد نظر فراخوانی شده‌اند.