جزوه‌ی آموزشیِ برنامه‌نویسی برای آمادگی در المپیاد کامپیوتر

تهیه و تنظیم : عرفان عبدی

ویرایش : جواد عابدی

نسخه‌ی دوم

فهرست

مقدمه

در این جزوه‌ی آموزشی تلاش شده به صورت مختصر دانش‌پژوهان با زبان برنامه‌نویسی C++ آشنا شوند. مرحله‌ی سوم المپیاد کامپیوتر به صورت برنامه‌نویسی برگزار می‌شود. این آزمون از تعدادی مساله تشکیل شده که پاسخ نهایی هرکدام یک عدد خواهد بود. حل هر مساله معمولا نیازمند محاسبات پیچیده‌ای است که به کمک برنامه‌نویسی می‌توان پاسخ را بدست آورد.

به عنوان مثال فرض کنید پاسخ مساله برابر با مجموع اعداد اول بین یک تا یک میلیون باشد. بدیهتا شما نمی‌توانید روی کاغذ این عدد را محاسبه کنید، اما با نوشتن چند خط کد ساده می‌توان این مقدار را بدست آورد.

هنگامی که کد مساله‌ای را نوشتید می‌توان توسط ابزارهایی که به آنها compiler گفته می‌شود کد را به یک برنامه‌ی اجرایی تبدیل و خروجیِ آن را مشاهده کرد. اگر در کدِ نوشته شده تمام قواعد برنامه نویسی رعایت شده باشد compiler با موفقیت آن را به فایل اجرایی تبدیل می‌کند ولی در غیر اینصورت خطاهای کد را نمایش می‌دهد و شما می‌توانید پس از رفع مشکلات دوباره کد خود را compile کنید.

شما می‌توانید کد خود را در هر جایی روی کامپیوتر حتی در notepad بنویسید ولی ابزارهای استانداردی برای برنامه‌نویسی وجود دارد که علاوه بر ویرایش، امکانات اضافه‌ای در اختیار شما قرار می‌دهد.

نرم‌افزارهای دیگری نیز هستند که این دو قابلیت را همزمان برای شما فراهم می‌کنند، یعنی هم می‌توانید در آن به آسانی کد بزنید و هم کد را در همانجا compile کنید از جمله نرم‌افزارهای کم‌حجم و ساده برای برنامه‌نویسی برنامه‌ی devcpp می‌باشد که می‌توانید آنرا از آدرس زیر دریافت کنید:

http://www.bloodshed.net/devcpp.html

شما در هر محیطی کد بزنید فرقی در اجرای برنامه‌ی نهایی ندارد ولی برای آنکه از یک رویه پیروی کنیم در این جزوه فرض می‌کنیم در devcpp کد می‌زنیم.

آشنایی با C++

برای آغاز، نرم افزار devcpp را باز کرده و Source File → New → File را انتخاب کنید. حال در صفحه‌ی روبرو قطعه کد زیر را بنویسید:

```cpp
#include <iostream>

using namespace std;

int main(){
    cout << 777 << endl;
    system("pause");
}

```

Code1_1.cpp

سپس از منوی Execute گزینه ی Compile & Run را انتخاب کنید یا اینکه از کلید F9 را فشار دهید.

سپس شما باید صفحه‌ی زیر را ببینید:

```
777
Press any key to continue . . .
```

در خط اول خروجی عدد ۷۷۷ چاپ شده است. به جای ۷۷۷ عدد دیگری را بنویسید و نتیجه را مشاهده کنید.

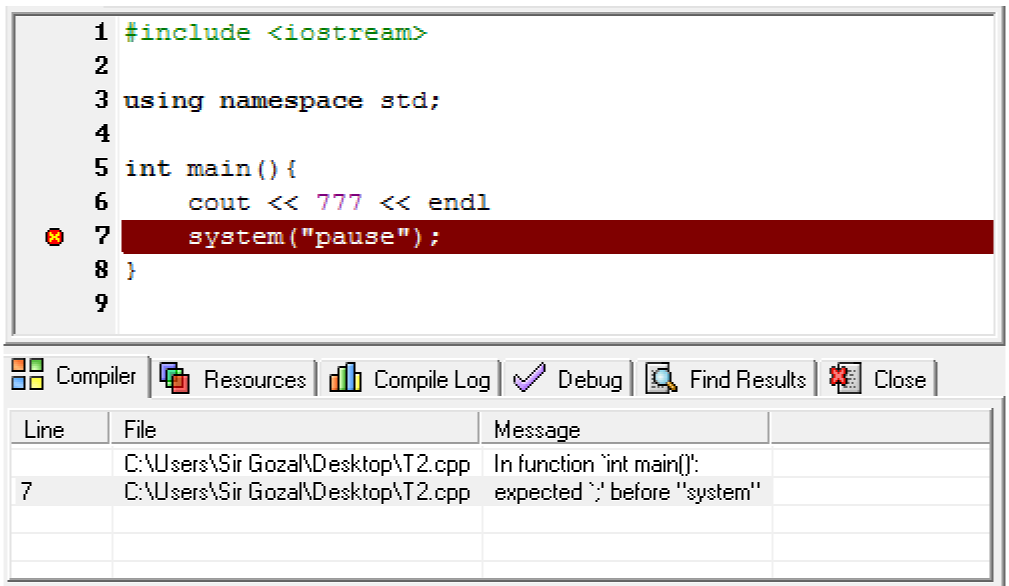
دستور cout برای نمایش مقادیر در خروجی است.

خطا در کامپایل

فرض کنید کدی که نوشته‌اید خطای دستوری داشته باشد در این صورت هنگامی که آنرا کامپایل می‌کنید اجرا نخواهد شد. برای مثال در کد قبلی از انتهای خط چهارم یعنی:

cout << 777 << endl;

; را بردارید و دوباره برنامه را اجرا کنید، خطای زیر را مشاهده خواهید کرد:



Code1_1.cpp

همانطور که می‌بینید در پایین صفحه خطاهای دستوری کد نوشته شده است.

به ازای هر خطا، ابتدا خطی که خطا در آن مشاهده شده، سپس نام فایل و در نهایت توضیحاتی درباره‌ی نوع خطا نوشته شده است. شما می‌توانید با مراجعه به کد و توضیحات داده شده مشکلات خود را برطرف کرده و دوباره آن را کامپایل کنید.

متغیرها

در خیلی از موارد نیاز به ذخیره مقادیر داریم تا در ادامه از آنها استفاده کنیم، به این منظور می‌توانید از متغیرها استفاده کنید.

در زبان C++ متغیرها انواع مختلفی می‌توانند داشته باشند. برای اینکه متغیری از جنس اعداد صحیح تعریف کنیم به این صورت عمل می‌کنیم:

```
int   a;
```

حال ما یک متغیر از جنس اعداد صحیح به نام a داریم و می‌توانیم از آن استفاده کنیم. به مثال زیر توجه کنید:

```cpp
#include <iostream>

using namespace std;

int main(){
    int a = 5;
    int b;
    b = 9;
    cout << a << endl;
    cout << a + b << endl;
    a = 17;
    cout << a << endl;
    system("pause");
}

```

Code1_2.cpp

در این مثال ابتدا متغیر a با مقدار ۵ و سپس متغیر b با مقدار ۹ تعریف شده است.

در ادامه متغیر a را در خروجی چاپ کرده‌ایم. پس انتظار داریم در خط اولِ خروجی مقدار ۵ چاپ شود و پس از آن مقدار a + b یعنی ۱۴ را در خروجی ببینیم.

بعد از آن مقدار متغیر a را به ۱۷ تغییر داده‌ایم و دوباره a را چاپ کرده‌ایم که در خروجی مقدار ۱۷ را مشاهده خواهیم کرد:

خروجی:

```
5
14
17
Press any key to continue . . .
```

ورودی خواندن

همانطور که در مثال‌های بالا دیدید متغیرهایی تعریف کردیم و به آنها مقدار دادیم.

ما می‌توانیم با استفاده از دستور cin مقدارِ متغیرهای خود را از ورودی بخوانیم (به همان سادگی که متغیر را در خروجی چاپ می‌کردیم). به مثال زیر توجه کنید:

```cpp
#include <iostream>

using namespace std;

int main(){
    int a, b;
    cin >> a;
    cin >> b;
    cout << a + b << endl;
    system("pause");
}

```

Code1_3.cpp

همانطور که می‌بینید در این مثال ابتدا دو متغیر a و b را تعریف کرده‌ایم.

سپس در خط بعد از ورودی مقدار a را خوانده‌ایم و در خط بعد آن مقدار b را دریافت کرده‌ایم.

و در نهایت مجموع آنها را در خروجی چاپ کرده‌ایم.

خروجی به شکل زیر خواهد بود:

```
10
23
33
Press any key to continue . . .
```

همانطور که می‌بینید در زمان اجرای برنامه خط اول به a مقدار ۱۰ و به b مقدار ۲۳ را داده‌ایم.

دستورات شرطی:

اگر بخواهیم دستوراتی در شرایط خاصی اجرا شوند می‌توانیم از if استفاده کنیم، ساختار if به شکل روبرو است:

If (شرط مورد نظر){

دستوراتی که باید انجام شود

}

برای مثال فرض کنید می‌خواهیم عددی را از ورودی بخوانیم، در صورتی که کوچکتر و یا مساوی ۲۰ بود آن عدد را چاپ کنیم و در غیر این صورت (یعنی اگر بزرگتر از ۲۰ بود) عدد ۵۰ را چاپ کنیم:

```
#include <iostream>

using namespace std;

int main(){
    int n;
    cin >> n;
    if(n <= 20){
        cout << n << endl;
    }
    if(n > 20){
        cout << 50 << endl;
    }
    system("pause");
}

```

Code1_4.cpp

توجه کنید برای نوشتن شرطِ برابریِ دو عبارت باید از == استفاده کنیم. به مثال زیر توجه کنید:

```
if(n==2){
    cout<<1<<endl;
}
```

دستور بالا به این معنی است که اگر n برابر ۲ بود شرط برقرار است.

همچنین برای نوشتن شرطِ نابرابریِ دو عبارت باید از عملگر =! استفاده کنیم.

حلقه‌ها:

اگر بخواهیم دستوراتی را چند بار به صورت مکرر انجام دهیم از حلقه‌ها استفاده می‌کنیم. ساختار کلی یک حلقه به شکل زیر است:

for(**تعریف متغیر و مقداردهی اولیه ; شرط خاتمه‌ی حلقه ; اتفاقی که پس از اجرای دستورات داخل رخ می‌دهد**){

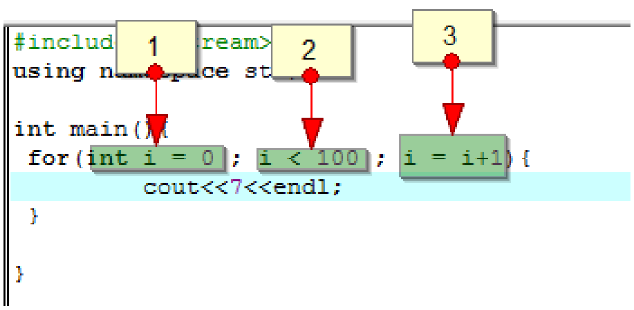
اعمالی که باید اتفاق بیفتند

}

مثلا فرض کنید می‌خواهیم ۱۰۰ بار عدد ۷ را چاپ کنیم. برنامه‌ی زیر را برای آن می‌نویسیم:

در برنامه‌ی روبرو ابتدا یک متغیر به اسم i تعریف کرده‌ایم. و مقدار اولیه آن را برابر صفر قرار داده‌ایم.

سپس شرط حلقه را چک می‌کنیم یعنی بررسی می‌کنیم مقدار i کمتر از ۱۰۰ است یا خیر. اگر این شرط برقرار بود داخل بدنه‌ی حلقه می‌شویم و عدد ۷ را چاپ می‌کنیم.



Code1_5.cpp

سپس به قسمت سوم حلقه بازمی‌گردیم و دستوری که آنجا هست را اجرا می‌کنیم یعنی i را برابر $i + 1$ قرار می‌دهیم در واقع یک واحد به آن اضافه کرده و شرط حلقه یا همان قسمت دوم آن را چک می‌کنیم. اگر برقرار بود داخل حلقه می‌شویم و این فرایند را آن قدر انجام می‌دهیم تا شرط قسمت دوم حلقه دیگر برقرار نباشد.

چند مثال:

با استفاده از دستوراتی که تا بدین لحظه فرا گرفتیم، می‌توانیم به حل چند مثال بپردازیم:

<u>مثال اول</u>: می‌خواهیم برنامه‌ای بنویسیم که عدد n را از ورودی بخواند و اعداد 1 تا n را در خروجی چاپ کند.

```cpp
#include <iostream>

using namespace std;

int main(){
    int n;
    cin >> n;
    for(int i = 1; i <= n; i = i + 1){
        cout << i << endl;
    }
    system("pause");
}

```

Example1_1.cpp

خروجی:

```
7
1
2
3
4
5
6
7
Press any key to continue . . .
```

مثال دوم: می‌خواهیم ابتدا یک عدد از ورودی بخوانیم و به تعداد آن، عدد طبیعی از ورودی دریافت کرده و از بین آنها عدد بیشینه را چاپ کنیم. مثلا فرض کنید در ابتدا عدد ۵ و سپس ۵ عدد وارد کنیم. بعد از آن اعداد ۴، ۱، ۱۰، ۷ و ۳ را وارد کرده باشیم. در نتیجه خروجی باید ۱۰ باشد:

```cpp
#include <iostream>

using namespace std;

int main(){
    int n;
    cin >> n;
    int max = 0;
    int input;
    for(int i = 0; i < n; i = i + 1){
        cin >> input;
        if(input > max){
            max = input;
        }
    }
    cout << max << endl;
    system("pause");
}

```

Example1_2.cpp

همانطور که می‌بینید ابتدا متغیر n را از ورودی خوانده‌ایم و سپس دو متغیر تعریف کرده‌ایم، یکی برای اینکه ماکسیمم را در آن قرار دهیم و دیگری متغیر ورودی است.

سپس در یک حلقه که n دفعه اجرا می‌شود هر بار یک ورودی دریافت می‌کنیم، اگر از ماکسیممی که تا به الان خوانده‌ایم بیشتر بود ماکسیمم را برابر آن قرار می‌دهیم و در نهایت بعد از خاتمه‌ی حلقه مقدار ماکسیمم را چاپ می‌کنیم.

خروجی:

```
5
4
1
10
7
3
10
Press any key to continue . . .
```

مثال سوم: می‌خواهیم عددی را از ورودی بخوانیم اگر آن عدد اول بود در خروجی یک چاپ کنیم و اگر اول نبود صفر چاپ کنیم.

```cpp
#include <iostream>

using namespace std;

int main(){
    int n;
    cin >> n;
    int cnt = 0;
    for(int i = 2; i < n; i = i + 1){
        if(n % i == 0){
            cnt = cnt + 1;
        }
    }
    if(cnt == 0){
        cout << 1 << endl;
    }
    if(cnt != 0){
        cout << 0 << endl;
    }
    system("pause");
}

```

Example1_3.cpp

ابتدا عدد n را می‌خوانیم. سپس به ازای تمام اعداد بین ۲ تا 1 - n چک می‌کنیم اگر n حداقل بر یکی از آنها بخشپذیر باشد یعنی اول نیست ولی اگر بر هیچکدام بخشپذیر نبود، اول خواهد بود. لازم به ذکر است برای اینکه باقیمانده‌ی عددی را بر عدد دیگر محاسبه کنیم می‌توانیم از عملگر ٪ استفاده کنیم. برای اینکار:

یک متغیر به اسم cnt تعریف می‌کنیم که تعداد مقسوم‌علیه‌های n در بین اعداد ۲ تا n-1 را در آن ذخیره کنیم.

سپس یک حلقه از ۲ تا 1 - n اجرا می‌کنیم و...

خروجی:

```
17
1
Press any key to continue . . .
```

چند متغیر دیگر

تا به اینجا با متغیر از نوع int که برای ذخیره‌سازی اعداد صحیح است آشنا شدیم. لازم به ذکر است اعداد از نوعِ int یک محدوده‌ی خاص دارند و اعداد بیرون این محدوده نمی‌توانند باشند. در جدول زیر چند نوع متغیر جدید به همراه محدوده و کاربردشان را نوشته‌ایم:

نام متغیر	کاربرد	محدوده
Int	اعداد صحیح	-2^{31} تا $2^{31}-1$
long long	اعداد صحیح بزرگ	-2^{63} تا $2^{63}-1$
Float	اعداد اعشاری	3.4E +/- 38
Double	اعداد اعشاری بزرگ	1.7E +/- 30

توابع و کاربردهای آن

تا اینجا هر کدی زده‌ایم شامل کلماتی ثابت در خط اول و دوم بوده‌اند:

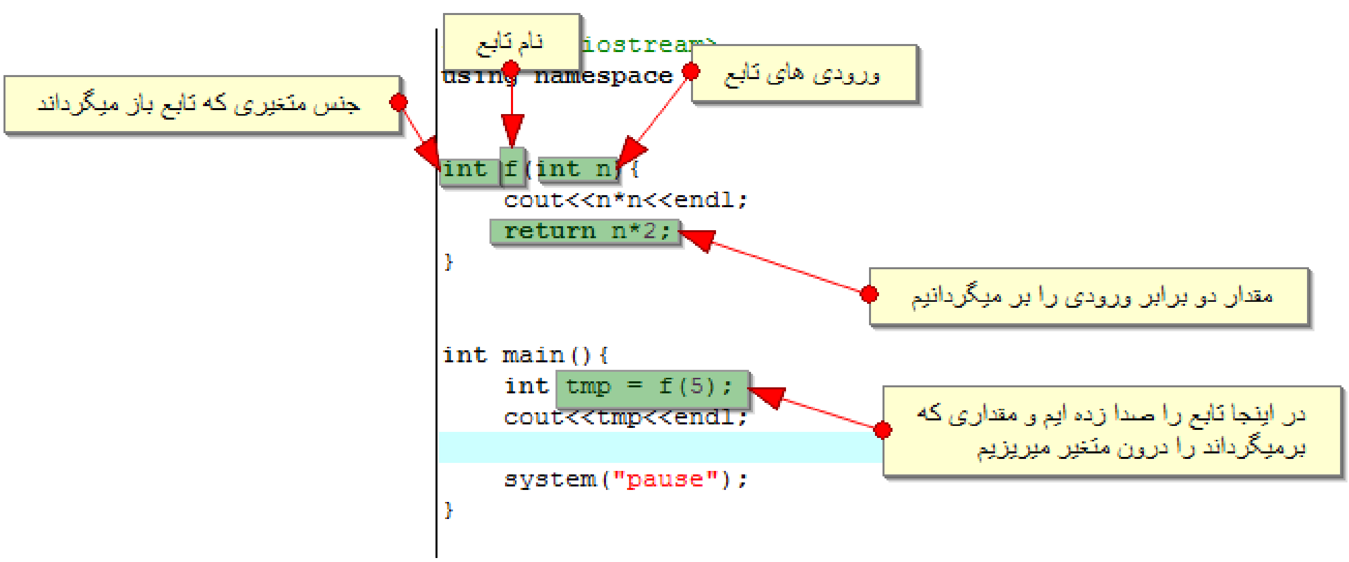
```
#include<iostream>
using namespace std;
```

اینکه include چه کاری می‌کند را در ادامه خواهیم دید، اما پس از این دو خط دستورات خود را همیشه در داخل

```
int main(){
        دستورات مورد نظر
}
```

تابعی به نام main می‌نوشتیم. در واقع هنگامی که برنامه‌ی ما اجرا می‌شود از اولین خطِ درون main شروع می‌شود تا به انتهای آن برسد.

حال می‌خواهیم در برنامه توابع دیگری تعریف کنیم. با یک مثال شروع می‌کنیم:



برای هر تابعی که تعریف می‌کنیم ابتدا باید جنس متغیری که به عنوان خروجی برمی‌گرداند را مشخص کنیم. در مثال بالا خروجی از نوع int است. می‌توانیم از هر نوع دیگری از متغیرها نیز استفاده کنیم و درصورتیکه بخواهیم خروجی نداشته باشیم، باید به جای آنها از void استفاده کنیم، سپس نام تابع را می‌نویسیم که در مثال f نوشته‌ایم و در نهایت

اگر بخواهیم ورودی داشته باشیم به تعداد دلخواه ورودی قرار می‌دهیم (ابتدا نوع متغیر و سپس نام آن). در نهایت داخل {} دستورات مورد نظر را می‌نویسیم.

در برنامه هر جا که بخواهیم می‌توانیم تابع را صدا بزنیم و تمام دستوراتی که درون آن است اجرا می‌شود. در این مثال در خط اولِ تابع main، متغیر tmp را تعریف کرده‌ایم و آن را برابر خروجیِ تابع f با ورودی ۵ قرار داده‌ایم. یعنی ابتدا تابع f با ورودی ۵ صدا زده می‌شود، پس هنگامی که تابع می‌خواهد اجرا شود مقدار ورودی خود یعنی همان n را برابر ۵ قرار می‌دهد و آن تابع را اجرا می‌کند. پس ابتدا ۵×۵ یا ۲۵ را چاپ می‌کند و پس از آن مقدار ۵×۲ را برمی‌گرداند که مقدار متغیر tmp را برابر ۱۰ قرار می‌دهد و در نهایت ۱۰ را در main چاپ می‌کنیم.

حال برای اینکه بیشتر با تابع آشنا شویم چند مثال می‌زنیم:

<u>مثال چهارم</u>: می‌خواهیم یک تابع بنویسیم که به آن دو عدد بدهیم و آن تابع جمع آنها را حساب کند.

```
#include <iostream>

using namespace std;

int add(int a, int b){
    int c = a + b;
    return c;
}

int main(){
    int x;
    x = add(5, 10);
    cout << x << endl;
    x = add(20, 13);
    cout << x << endl;
    system("pause");
}

```

Example1_4.cpp

ما در این برنامه x را برابر حاصل add(5,10) قرار می‌دهیم تا پس از اجرای آن مقدار x برابر ۱۵ شود. سپس همین کار را به ازای $a = 20$ و $b = 13$ انجام می‌دهیم.

خروجی:

```
15
33
Press any key to continue . . .
```

مثال پنجم: برنامه‌ای بنویسید که عدد n را از ورودی بگیرد و یک مثلث با ستاره‌ها چاپ کند که طول ضلع آن n باشد.

برای مثال اگر n برابر ۳ باشد باید شکل زیر چاپ شود:

*

**

***

برای اینکه این سوال را حل کنیم و همچنین از تابع استفاده کرده باشیم یک تابع می‌نویسیم که یک ورودی بگیرد و به تعداد آن ورودی ستاره چاپ کند سپس در main این تابع را با ورودی‌های مختلف در یک حلقه n بار صدا می‌زنیم.

```cpp
#include <iostream>

using namespace std;

void f(int n){
    for(int i = 0; i < n; i++){
        cout << "*";
    }
    cout << endl;
}

int main(){
    int x;
    cin >> x;
    for(int i = 1; i <= x; i++){
        f(i);
    }
    system("pause");
}

```

Example1_5.cpp

خروجی:

```
4
*
**
***
****
Press any key to continue . . .
```

مثال ششم: فرض کنید ابتدا از ورودی عدد n را می‌خوانیم و سپس n زوج عدد دریافت می‌کنیم. می‌خواهیم به ازای هر زوج عدد داده شده ماکسیمم آن دو را چاپ کنیم.

برای حل این سوال یک تابع می‌نویسیم که دو ورودی بگیرد و ماکسیمم آنها را بازگرداند سپس آنرا چاپ می‌کنیم.

```
#include <iostream>

using namespace std;

int max(int a, int b){
    if(a > b){
        return a;
    }
    return b;
}

int main(){
    int n;
    cin >> n;
    int c, d;
    for(int i = 0; i < n; i++){
        cin >> c >> d;
        cout << max(c, d) << endl;
    }
    system("pause");
}

```

Example1_6.cpp

ابتدا یک تابع تعریف کرده‌ایم که به آن دو عدد می‌دهیم و در آن شرطی گذاشته‌ایم که اگر a بزرگتر از b بود a و در غیر اینصورت b را به عنوان خروجی برگرداند. هنگامی که در تابع به return می‌رسیم تابع همان جا تمام می‌شود و خروجی‌اش را همانجا که آنرا صدا زده‌ایم قرار می‌دهد.

خروجی:

```
5
1 10
10
11 12
12
43 10
43
2 2
2
12 100
100
Press any key to continue . . .
```

آرایه‌ها

در بسیاری از موارد می‌خواهیم تعداد زیادی داده نگه داریم، به نظر منطقی نمی‌آید که ۱۰۰ متغیر عدد صحیح به شکل زیر تعریف کنیم:

int a0, a1, a2, a3, a4, …, a99;

زبان سی برای این منظور آرایه را معرفی می‌کند، دستور نوشتن آرایه به شکل زیر است:

int a[100];

در واقع این دستور ۱۰۰ متغیر با نام‌های a[0] ,a[1],a[2] ,…a[99] تعریف می‌کند و ما برای مثال برای دسترسی به آنها می‌توانیم به فرمت زیر استفاده کنیم:

a[0] = 54;

a[10] = 59 * 4 * a[0];

<u>مثال هفتم</u>: برنامه‌ای بنویسید که ۱۰ عدد را از ورودی بخواند و آنها را به ترتیب عکس درخروجی چاپ کند.

برای حل این سوال یک آرایه تعریف می‌کنیم و ۱۰ ورودی خود را به ترتیب در آن می‌ریزیم. سپس آنها را به ترتیب معکوس چاپ می‌کنیم.

```
#include <iostream>

using namespace std;

int main(){
    int arr[20];
    for(int i = 0; i < 10; i++){
        cin >> arr[i];
    }
    for(int i = 9; i >= 0; i--){
        cout << arr[i] << " ";
    }
    cout << endl;
    system("pause");
}

```

Example1_7.cpp

خروجی:

```
1 4 2 10 23 8 5 3 9 13
13 9 3 5 8 23 10 2 4 1
Press any key to continue . . .
```

مثال هشتم: می‌خواهیم ابتدا از ورودی یک عدد را بخوانیم و در متغیر n بریزیم. سپس n عدد از ورودی دریافت کرده و در نهایت آنها را به صورت مرتب شده در خروجی چاپ کنیم. مثلا اگر ورودی ها به صورت زیر باشند:

5

1 10 4 2 4

خروجی باید به شکل زیر باشد:

1 2 4 4 10

```cpp
#include <iostream>

using namespace std;

int main(){
    int arr[1000];
    int n;
    cin >> n;
    for(int i = 0; i < n; i++)
        cin >> arr[i];
    for(int i = 0; i < n; i++){
        for(int j = 1; j < n; j++){
            if(arr[j] < arr[j - 1]){
                int tmp = arr[j];
                arr[j] = arr[j - 1];
                arr[j - 1] = tmp;
            }
        }
    }
    for(int i = 0; i < n; i++){
        cout << arr[i] << " ";
    }
    cout << endl;
    system("pause");
}

```

Example1_8.cpp

برای اینکار ما یک آرایه تعریف می‌کنیم و اعداد را درون آن می‌ریزیم. از عدد دوم شروع می‌کنیم، اگر از عدد قبل خود کوچکتر بود آن دو را با هم جابه‌جا می‌کنیم سپس این کار را برای عدد سوم انجام می‌دهیم تا به عدد آخر برسیم. بدین ترتیب بزرگترین عدد در خانه‌ی آخر قرار می‌گیرد. اگر این کار را n بار انجام دهیم کل اعداد مرتب خواهند شد.

خروجی:

```
5
1 10 4 2 4
1 2 4 4 10
Press any key to continue . . .
```

توابع بازگشتی

همانطور که بالاتر با توابع آشنا شدیم، می‌توان آنها را تعریف کرد و هر جا خواستیم از آنها استفاده کنیم. حال می‌خواهیم یک تابع را دورن خودش صدا بزنیم، به چنین تابعی (که در تابع، خود را صدا می‌زند) تابع بازگشتی می‌گویند. برای مثال می‌خواهیم تابع فیبوناچی را به صورت بازگشتی بنویسیم:

```
#include <iostream>

using namespace std;

int fib(int n){
    if(n <= 1)
        return 1;
    return fib(n - 1) + fib(n - 2);
}

int main(){
    int n;
    cin >> n;
    cout << "f(n) => " << fib(n) << endl;
    system("pause");
}

```

Code1_6.cpp

همانطور که می‌بینید ما یک تابع به نام fib داریم که انتظار داریم مقدار عدد فیبوناچی nام را بدهد. هنگامی که به آن ورودی n را می‌دهیم، برای تولید آن، مانند استقرا عمل می‌کنیم:

فرض می‌کنیم توابع با nهای کوچکتر درست کار می‌کنند. تابع را با استفاده از nهای کوچکتر می‌نویسیم، همچنین برای اینکه تابع خاتمه یابد مانند استقرا باید پایه داشته باشیم که آن هم شرطی است که در اول تابع قرار داده‌ایم.

خروجی:

```
5
f(n) => 8
Press any key to continue . . .
```

معرفی توابع پیش نوشته‌شده‌ی کاربردی

بعضی از توابع در برنامه‌های مختلف بسیار کاربرد دارند و به همین دلیل برنامه‌ی آنها از قبل به شکل بهینه نوشته شده‌اند و شما می‌توانید از آنها استفاده کنید. چند تابع کاربردی را در زیر معرفی می‌کنیم. ابتدا برای اینکه از این توابع استفاده کنیم باید کتابخانه‌ای که تابع در نوشته شده است را به برنامه اضافه کنیم که اینکار با نوشتن دستور:

#include <نام کتابخانه‌ی موردنظر>

انجام می‌شود.

اگر توجه کرده باشید در ابتدای تمام برنامه‌های قبلی برای استفاده از دستورات cin و cout، کتابخانه‌ی iostream را اضافه کرده‌ایم.

در زیر بعضی از توابع کاربردیِ کتابخانه‌ی cmath را آورده‌ایم. پس برای استفاده از آنها در ابتدای برنامه باید:

```
#include <cmath>
```

را بنویسیم.

کاربرد	نام تابع
مقدار کسینوس a را بر می‌گرداند	cos(a)
مقدار سینوس a را برمی‌گرداند	sin(a)
مقدار تانژانت a را برمی‌گرداند	tan(a)
مقدار لگاریتم عدد a در پایه ده را می‌دهد	log10(a)
مقدار a به توان b را برمی‌گرداند	pow(a,b)
مقدار جذرaرا برمی‌گرداند	sqrt(a)
سقف عددa را برمی‌گرداند	ceil(a)
کف عدد aرا بر می‌گرداند	floor(a)
قدر مطلق a را برمی‌گرداند	fabs(a)

حال توابع کتابخانه‌ی algorithm را مورد بررسی قرار می‌دهیم. قبل از آن فرض کنید آرایه ی زیر را تعریف کرده‌ایم:

```
int arr[1000];
```

نام تابع	کاربرد
swap(a,b)	این تابع دو متغیر را می‌گیرد و مقدار آنها را با هم عوض می‌کند
fill(arr,arr+n,c)	این تابع مقدار arr[0] تا arr[n - 1] را برابر c قرار می‌دهد
sort(arr,arr+n)	این تابع مقادیر داخل خانه های ۰ تا n - 1 آرایه arr را مرتب می‌کند
min(a,b)	این تابع مقدار مینیمم دو عدد a و b را برمی‌گرداند
max(a,b)	این تابع مقدار ماکسیمم دو عدد a و b را برمی‌گرداند

خواندن ورودی از فایل و نوشتن خروجی در آن

امکان دارد در برنامه‌ای بخواهید ورودی خود را به جای اینکه داخل ترمینال وارد کنید از فایل بخوانید. مثلا مسئله‌ای به شما یک فایل بزرگ داده که ورودی سوال است. برای خواندن اطلاعات از فایل اگر در cmdی ویندوز و یا در ترمینالِ لینوکس برنامه‌ی خود را اجرا می‌کنید می‌توانید به راحتی و بدون هیچ تغییری در ساختار کد خود نام فایل اجرایی را نوشته و ورودی را به آن بدهید. برای مثال فرض کنید نام فایل اجرایی برنامه‌ی شما a.out باشد و نام فایل ورودی inp باشد. می‌توانید با دستور زیر فایل inp را به برنامه‌ی خود بدهید.

در لینوکس:

```
./a.out < inp
```

در ویندوز:

```
a.exe < inp
```

اما اگر برنامه‌ی خود را در محیط devcpp نوشته و اجرا می‌کنید باید در ابتدا کتابخانه‌ی fstream را باز کرده و قبل از تابع main دستور زیر را وارد کنید:

```
ifstream fin ("inp");
```

توجه کنید "inp" نام همان فایلی است که می‌خواهید باز کنید، و این فایل باید همانجا باشد که کد خود را در devcpp ذخیره کرده‌اید. حال از این به بعد به سادگی از fin مانند cin استفاده می‌کنید.

مثال نهم: برنامه‌ای بنویسید که از فایلی به نام sum دو عدد بخواند و جمع آنها را در خروجی چاپ کند.

```
#include<iostream>
#include<fstream>

using namespace std;

ifstream fin("sum");

int main(){
    int a, b;
    fin >> a >> b;
    cout << a + b << endl;
    system("pause");
}

```

Example1_9.cpp

هم چنین برای نوشتن خروجی در فایل باید قبل از main با توجه به نام فایل مورد نظر کد زیر را اضافه کنید:

```
ofstream fout ("out");
```

حال از این به بعد همانند cout می‌توانید از fout استفاده کنید.

تذکر: روش استفاده از fstream همانند دو حالتی است که در بالا برای ترمینال لینوکس و cmdی ویندوز گفته شد.

کار با رشته

برای کار با رشته‌ها شما را با متغیری جدید به نام char آشنا می‌کنیم، در این متغیر تک بایتی می‌توان یک حرف را ذخیره کرد. به مثال زیر توجه کنید:

```
char tmp;

tmp = 'a';

cout << tmp << endl;

tmp = 'b';

cout << tmp << endl;
```

در این مثال ما یک کاراکتر تعریف کرده‌ایم و سپس مقدار آن را برابر مقدار کاراکتر a قرار داده‌ایم. توجه کنید برای اینکه کاراکتر a را درون tmp بریزیم باید a را بین ' ' قرار دهیم، سپس هنگامی که tmp را چاپ می‌کنیم a در خروجی نوشته می‌شود و بعد از آن نیز b در خروجی چاپ خواهد شد.

هنگامی که می‌نویسید tmp = 'a'، یک عدد درون tmp ذخیره می‌شود که نمایانگر کاراکتر a می‌باشد و نکته‌ی مهم این است که عدد متناظر با حروف بر حسب ترتیب الفبایی آنهاست. به قطعه کد زیر توجه کنید:

```
tmp = 'a';

for (int  i = 0; i < 5; i ++){

        cout << tmp << endl;

        tmp = tmp + 1;

}
```

خروجی این چنین خواهد بود:

```
a

b

c
```

d

e

به ازای هر علامت در کامپیوتر، عددی بین ۰ تا ۲۵۵ به آن اختصاص می‌یابد. در جدول زیر عدد مربوط به هر علامت مشخص گردیده است:

CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX
[NUL]	0	00		32	20	@	64	40	`	96	60
[SOH]	1	01	!	33	21	A	65	41	a	97	61
[STX]	2	02	"	34	22	B	66	42	b	98	62
[ETX]	3	03	#	35	23	C	67	43	c	99	63
[EOT]	4	04	$	36	24	D	68	44	d	100	64
[ENQ]	5	05	%	37	25	E	69	45	e	101	65
[ACK]	6	06	&	38	26	F	70	46	f	102	66
[BEL]	7	07	'	39	27	G	71	47	g	103	67
[BS]	8	08	(	40	28	H	72	48	h	104	68
[HT]	9	09	)	41	29	I	73	49	i	105	69
[LF]	10	0A	*	42	2A	J	74	4A	j	106	6A
[VT]	11	0B	+	43	2B	K	75	4B	k	107	6B
[FF]	12	0C	,	44	2C	L	76	4C	l	108	6C
[CR]	13	0D	-	45	2D	M	77	4D	m	109	6D
[SO]	14	0E	.	46	2E	N	78	4E	n	110	6E
[SI]	15	0F	/	47	2F	O	79	4F	o	111	6F
[DLE]	16	10	0	48	30	P	80	50	p	112	70
[DC1]	17	11	1	49	31	Q	81	51	q	113	71
[DC2]	18	12	2	50	32	R	82	52	r	114	72
[DC3]	19	13	3	51	33	S	83	53	s	115	73
[DC4]	20	14	4	52	34	T	84	54	t	116	74
[NAK]	21	15	5	53	35	U	85	55	u	117	75
[SYN]	22	16	6	54	36	V	86	56	v	118	76
[ETB]	23	17	7	55	37	W	87	57	w	119	77
[CAN]	24	18	8	56	38	X	88	58	x	120	78
[EM]	25	19	9	57	39	Y	89	59	y	121	79
[SUB]	26	1A	:	58	3A	Z	90	5A	z	122	7A
[ESC]	27	1B	;	59	3B	[	91	5B	{	123	7B
[FS]	28	1C	<	60	3C	\	92	5C	|	124	7C
[GS]	29	1D	=	61	3D	]	93	5D	}	125	7D
[RS]	30	1E	>	62	3E	^	94	5E	~	126	7E

CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX
€	128	80		160	A0	À	192	C0	à	224	E0
(n/a)	129	81	¡	161	A1	Á	193	C1	á	225	E1
‚	130	82	¢	162	A2	Â	194	C2	â	226	E2
ƒ	131	83	£	163	A3	Ã	195	C3	ã	227	E3
„	132	84	¤	164	A4	Ä	196	C4	ä	228	E4
…	133	85	¥	165	A5	Å	197	C5	å	229	E5
†	134	86	¦	166	A6	Æ	198	C6	æ	230	E6
‡	135	87	§	167	A7	Ç	199	C7	ç	231	E7
ˆ	136	88	¨	168	A8	È	200	C8	è	232	E8
‰	137	89	©	169	A9	É	201	C9	é	233	E9
Š	138	8A	ª	170	AA	Ê	202	CA	ê	234	EA
‹	139	8B	«	171	AB	Ë	203	CB	ë	235	EB
Œ	140	8C	¬	172	AC	Ì	204	CC	ì	236	EC
(n/a)	141	8D		173	AD	Í	205	CD	í	237	ED
Ž	142	8E	®	174	AE	Î	206	CE	î	238	EE
(n/a)	143	8F	¯	175	AF	Ï	207	CF	ï	239	EF
(n/a)	144	90	°	176	B0	Ð	208	D0	ð	240	F0
‘	145	91	±	177	B1	Ñ	209	D1	ñ	241	F1
’	146	92	²	178	B2	Ò	210	D2	ò	242	F2
“	147	93	³	179	B3	Ó	211	D3	ó	243	F3
”	148	94	´	180	B4	Ô	212	D4	ô	244	F4
•	149	95	µ	181	B5	Õ	213	D5	õ	245	F5
–	150	96	¶	182	B6	Ö	214	D6	ö	246	F6
—	151	97	·	183	B7	×	215	D7	÷	247	F7
˜	152	98	¸	184	B8	Ø	216	D8	ø	248	F8
™	153	99	¹	185	B9	Ù	217	D9	ù	249	F9
š	154	9A	º	186	BA	Ú	218	DA	ú	250	FA
›	155	9B	»	187	BB	Û	219	DB	û	251	FB
œ	156	9C	¼	188	BC	Ü	220	DC	ü	252	FC
(n/a)	157	9D	½	189	BD	Ý	221	DD	ý	253	FD
ž	158	9E	¾	190	BE	Þ	222	DE	þ	254	FE

تا الان آموخته‌ایم یک کاراکتر را نگه‌داری کنیم. در صورتی که آرایه‌ای از کاراکترها بگیریم می‌توانیم یک رشته را نیز ذخیره کنیم. برای مثال به کد زیر توجه کنید:

```
char [100] tmp;

tmp = "Long live IRAN";

cout << tmp << endl;
```

در این کد ما ابتدا یک آرایه از کاراکترها تعریف کرده‌ایم و یک رشته را در آن ریخته‌ایم (توجه کنید برای ریختن یک رشته با بیش از یک حرف، به جای ' ' باید از " " استفاده شود) سپس آن‌را چاپ کرده‌ایم. در واقع اگر خانه‌های آرایه‌ی tmp را بررسی کنید به شکل زیر خواهند بود:

tmp[0] = 'L', tmp[1] = 'o', tmp[2] = 'n', … , tmp[13] = 'N', tmp[14] = 0

حال ما می‌توانیم هرکدام از این خانه‌ها را به صورت مستقیم تغییر دهیم. مثلا می‌توانیم کاراکتر اول را با کد tmp[0]='l' تغییر دهیم و در صورت چاپ tmp در خروجی long live IRAN به جای Long live IRAN چاپ خواهد شد.

برای خواندن رشته از ورودی نیز ابتدا یک آرایه تعریف می‌کنیم و به سادگی آنرا با cin می‌خوانیم:

```
cin >> tmp;
```

<u>نکته‌ی مهم این است که وقتی رشته‌ی شما خوانده و درون tmp ریخته شد اولین خانه بعد از رشته‌ی شما در آرایه برابر صفر می‌شود</u> و این نشانگر پایان رشته است و در چاپ کردن این صفر خاتمه‌دهنده‌ی رشته‌ی چاپ شونده است.

برای مثال کد زیر را در نظر بگیرید:

```
char tmp[100]="Olympiad in informatics";

cout<<tmp<<endl;

tmp[8]=0;

cout<<tmp<<endl;
```

خروجی این برنامه به شکل زیر خواهد بود:

```
Olympiad in informatics

Olympiad
```