```
local x in main is 5
```

در برنامه فوق سه تابع با نام های **useStaticLocal, useGlobal, useLocal** داریم. متغیر x تعریف شده در ابتدای برنامه با مقدار **۱** به عنوان یک متغیر عمومی می باشد و دارای حوزه فایل است. در تابع **main**، متغیر **x** با مقدار **۵** تعریف شده است. لذا متغیر عمومی **x** با مقدار **۱** نادیده گرفته می شود و هنگام اجرای دستور **cout** ، متغیر **x** با مقدار **۵** در خروجی چاپ می شود. در بلوک درونی، متغیر **x** با مقدار **۷** تعریف شده است. لذا **x** عمومی و **x** محلی تابع **main** نادیده گرفته می شوند و تنها **x** با مقدار **۷** توسط دستور **cout** چاپ می گردد. پس از آنکه بلوک حوزه **x** با مقدار **۷** به اتمام می رسد، دوباره **x** محلی تابع **main** با مقدار **۵** نمایان می گردد.

تابع **useLocal** متغیر محلی **x** را با مقدار **۲۵** در خود تعریف می کند. هنگامی که این تابع در برنامه فراخوانی می شود، تابع ، متغیر **x** را چاپ می کند، سپس یک واحد به آن اضافه کرده و دوباره **x** را چاپ می کند. هر بار که این تابع فراخوانی می شود، متغیر **x** با مقدار **۲۵** در آن تعریف می شود و هنگام خروج از تابع از بین می رود.

تابع **useStaticLocal** متغیرمحلی **x** را از نوع **static** تعریف کرده و با عدد **۵۰** مقداردهی می کند و سپس آن را چاپ کرده و یک واحد به آن اضافه می کند و دوباره چاپش می کند. اما هنگام خروج از تابع مقدار **x** از بین نمی رود. و با فراخوانی مجدد تابع ، مقدار قبلی متغیر **x** محلی ، برای این تابع موجود می باشد و دوباره از نو مقداردهی نمی شود. در اینجا هنگامی که تابع دوباره فراخوانی می شود، **x** حاوی **۵۱** خواهد بود.

تابع **useGlobal** هیچ متغیری را در خود تعریف نمی کند. لذا هنگامی که به متغیر **x** مراجعه می کند ، متغیر**x** عمومی مورد استفاده قرار می گیرد. هنگامی که این تابع فراخوانی می شود مقدار متغیر **x** عمومی چاپ می شود. سپس در **۱۰** ضرب شده و دوباره چاپ می گردد. هنگامی که برای بار دوم تابع **useGlobal** فراخوانی می شود **x** عمومی حاوی عدد **۱۰۰** می باشد.

پس از اینکه برنامه هر یک از توابع فوق را دوبار فراخوانی کرد ، مجددا متغیر **x** تابع **main** با مقدار **۵** چاپ می گردد و این نشان می دهد که هیچ یک از توابع ، تاثیری روی متغیر محلی تابع **main** نداشتند.

### آرگومانهای پیش فرض توابع

در برنامه نویسی ممکن است تابعی را به دفعات با آرگومانهای یکسانی صدا بزنیم . در چنین حالتی ، برنامه نویس می تواند برای آرگومانهای تابع ، مقداری را به عنوان پیش فرض قرار دهد . هنگامی که در فراخوانی توابع ، آرگومان دارای مقدار پیش فرض حذف شده باشد ، کامپایلر مقدار پیش فرض آن آرگومان را به تابع خواهد فرستاد .

آرگومان های پیش فرض باید سمت راستی ترین آرگومان های تابع باشند . هنگامی که تابعی با بیش از یک آرگومان فراخوانی می شود ، اگر آرگومان حذف شده سمت راستی ترین آرگومان نباشد ، آنگاه همه آرگومانهای سمت راست آن

آرگومان نیز باید حذف شوند . آرگومان های پیش فرض باید در اولین جایی که نام تابع آورده می شود ( که معمولاً در پیش تعریف تابع است ) مشخص شوند .

مقادیر پیش فرض می توانند اعداد ، مقادیر ثابت ، متغیرهای عمومی و یا خروجی تابع دیگر باشند .

برنامه زیر نحوه مقدار دهی به آرگومان های پیش فرض و نیز نحوه فراخوانی تابع با مقدار پیش فرض را نشان می دهد . در این برنامه حجم جعبه ای محاسبه می شود .

```
#include <iostream.h>

// function prototype that specifies default arguments
int boxVolume(int length=1, int width=1, int height=1);

int main()
{
  //no arguments--use default values for all dimensions
  cout <<"The default box volume is: "<<boxVolume();

  //specify length; default width and height
  cout <<"\n\nThe volume of a box with length 10,\n"
       <<"width 1 and height 1 is: "<<boxVolume(10);

  //specify length and width; default height
  cout <<"\n\nThe volume of a box with length 10,\n"
       <<"width 5 and height 1 is: "<<boxVolume(10,5);

  //specify all arguments
  cout <<"\n\nThe volume of a box with length 10,\n"
       <<"width 5 and height 2 is: "<<boxVolume(10,5,2)
       <<endl;

  return 0; // indicates successful termination

} //end main

// function boxVolume calculates the volume of a box
int boxVolume( int length, int width, int height )
{
  return length * width * height;
} // end function boxVolume
```

خروجی برنامه فوق به صورت زیر می باشد .

```
The default box volume is: 1
```

```
The volume of a box with length 10,
width 1 and height 1 is: 10

The volume of a box with length 10,
width 5 and height 1 is: 50

The volume of a box with length 10,
width 5 and height 2 is: 100
```

در پیش تعریف تابع **boxVolume** به هر یک از سه آرگومان تابع مقدار پیش فرض ۱ داده شده است . توجه داشته باشید که مقادیر پیش فرض باید در پیش تعریف تابع نوشته شوند ، ضمناً نوشتن نام آرگومان های تابع در پیش تعریف الزامی نیست و در برنامه فوق اینکار تنها برای خوانایی بیشتر انجام گرفته است ، البته توصیه می شود که شما نیز از این شیوه استفاده کنید . به عنوان مثال پیش فرض تابع **boxVolume** در برنامه فوق را می توانستیم به صورت زیر نیز بنویسیم :

```
int boxVolume (int = 1 , int = 1 , int = 1 );
```

در اولین فراخوانی تابع **boxVolume** در برنامه فوق هیچ آرگومانی به آن داده نشده است لذا هر سه مقدار پیش فرض آرگومان ها مورد استفاده قرار می گیرد و حجم جعبه عدد ۱ می شود . در دومین فراخوانی آرگومان **length** ارسال می گردد ، لذا مقادیر پیش فرض آرگومان های **width** و **height** استفاده می شوند . در سومین فراخوانی آرگومان های **width** و **length** ارسال می گردند لذا مقادیر پیش فرض آرگومان **height** مورد استفاده قرار می گیرد . در آخرین فراخوانی هر سه آرگومان ارسال می شوند لذا از هیچ مقدار پیش فرضی استفاده نمی شود .

پس هنگامی که یک آرگومان به تابع فرستاده می شود ، آن آرگومان به عنوان **length** در نظر گرفته می شود و هنگامی که دو آرگومان به تابع **boxVolume** فرستاده می شود تابع آنها را به ترتیب از سمت چپ به عنوان آرگومان **length** و سپس **width** در نظر می گیرد و سرانجام هنگامی که هرسه آرگومان فرستاده می شود به ترتیب از سمت چپ در **length** و **width** و **height** قرار می گیرند .

### عملگر یگانی تفکیک حوزه

همانطور که در مبحث قوانین حوزه دیدید تعریف متغیرهای محلی و عمومی با یک نام در برنامه امکان پذیر می باشد . زبان **C++** عملگر یگانی تفکیک دامنه (::) را برای امکان دستیابی به متغیر عمومی همنام با متغیر محلی ، در اختیار ما قرار داده است . توجه داشته باشید که این عملگر تنها قادر به دستیابی به متغیر عمومی در حوزه فایل می باشد . ضمناً متغیر عمومی بدون نیاز به این عملگر نیز قابل دستیابی می باشد ؛ به شرط آنکه متغیرمحلی همنام با متغیر عمومی ، در برنامه به کار برده نشود . استفاده از عملگر (::) همراه نام متغیر عمومی ، در صورتی که نام متغیر عمومی برای متغیر دیگری به کار برده نشده باشد ، اختیاری است . اما توصیه می شود که برای اینکه بدانید از متغیر عمومی استفاده می کنید از این عملگر همواره در کنار نام متغیر عمومی استفاده کنید . برنامه زیر نحوه کاربرد عملگر (::) را نشان می دهد .

```
#include <iostream.h>

float pi=3.14159;

void main( )
{
     int pi=::pi;
     cout << "Local pi is : " << pi << endl;
     cout << "Global pi is : " << ::pi << endl;
}
```

خروجی برنامه به صورت زیر می باشد :

```
Local pi is : 3
Global pi is : 3.14159
```

در برنامه فوق متغیر عمومی **pi** از نوع **float** تعریف شده است و در تابع متغیر محلی **pi** از نوع **int** با مقدار اولیه **pi** عمومی مقدار دهی می شود . توجه داشته باشید که برای دستیابی به مقدار **pi** عمومی از عملگر یگانی تفکیک حوزه (::) استفاده شد . پس از مقدار دهی به **pi** محلی ، توسط دستور **cout** ، متغیر **pi** محلی که حاوی عدد **۳** است چاپ می گردد و در خط بعدی متغیر **pi** عمومی که حاوی **۳,۱۴۱۵۹** می باشد چاپ خواهد شد .

### ارسال آرگومان ها به تابع ، با ارجاع

تا به حال ، در تمام توابعی که نوشتیم آرگومان ها با مقدار به توابع فرستاده می شدند . این بدان معناست که هنگامی که توابع با آرگومانها فرا خوانی می شدند ، چیزی که ما به عنوان ورودی تابع ارسال می کردیم مقدار یا عدد بود و هرگز خود متغیر به تابع فرستاده نشد ، به عنوان مثال تابع **maximum** در مبحث تعریف توابع را به صورت زیر فراخوانی می کنیم :

```
int a=5, b=6, c=7, max;
max = maximum(a,b,c);
```

کاری که در اینجا صورت می گیرد فراخوانی تابع و فرستادن مقادیر موجود در **a** و**b** و**c** یعنی ۵ و ۶ و ۷ به تابع می باشد . و خود متغیرها فرستاده نمی شوند .

```
int maximum (int x  , int y  , int z)
              5↑     6↑     7↑
max = maximum (  a  ,    b  ,    c )
```

بدین صورت هنگامی که تابع **maximum** فراخوانی می شود ، مقدار متغیرهای **x** و**y** و**z** به ترتیب برابر ۵ و ۶ و ۷خواهند شد و هرگونه تغییری روی متغیرهای **x** و**y** و**z** در تابع ، تأثیری روی متغیرهای **a** و**b** و**c** نخواهد داشت . زیرا خود متغیرهای **a** و**b** و**c** به تابع فرستاده نشده اند بلکه مقادیر موجود در آنها به تابع ارسال گشته اند .

در برنامه نویسی مواردی پیش می آید که بخواهید از داخل تابع ، مقادیر متغیرهای خارجی را تغییر دهیم ، به عنوان مثال در تابع **maximum** مقدار متغیر **a** را از داخل تابع تغییر دهیم . برای نیل به این هدف باید از روش ارسال آرگومان ها با ارجاع استفاده کنیم . برای آنکه آرگومان تابعی با ارجاع فرستاده شود ، کافی است در پیش تعریف تابع بعد از تعیین نوع آرگومان یک علامت (**&**) بگذاریم و نیز در تعریف تابع قبل از نام آرگومان یک علامت (**&**) قرار دهیم . برای آشنایی با نحوه ارسال آرگومان ها با ارجاع به برنامه زیر توجه کنید .

```
#include <iostream.h>

void duplicate (int & , int & );

void main ( )
{
    int a=1 , b=2 ;
    cout << "a = " << a << " and b = " << b << endl;
    duplicate (a,b);
    cout << "a = " << a << " and b = " << b << endl;
}

void duplicate (int &x , int &y)
{
    x*=2;
    y*=2;
}
```

خروجی برنامه به صورت زیر می باشد .

```
a = 1 and b = 2
a = 2 and b = 4
```

در برنامه فوق متغیرهای **a** و**b** به تابع ارسال می گردند و سپس در دو ضرب می شوند. در این برنامه مقدار متغیرهای **a** و**b** فرستاده نمی شود بلکه خود متغیر فرستاده می شود و لذا هنگامی که دستورهای

```
x*=2;
y*=2;
```

اجرا می گردند مقادیر دو متغیر**a** و**b** دو برابر می شود . در حقیقت **x** و**y** مانند نام مستعاری برای **a** و**b** می باشند .

```
void duplicate (int &x , int &y)
                  a↕      b↕
     duplicate (     a ,      b)
```

هنگامی که متغیری با ارجاع فرستاده می شود هر گونه تغییری که در متغیر معادل آن در تابع صورت گیرد عیناً آن تغییر بر روی متغیر ارسالی نیز اعمال می گردد .

**مثال :** تابعی بنویسید که دو متغیر را به عنوان ورودی دریافت کرده و مقادیر آنها را جابه جا کند . از این تابع در برنامه ای استفاده کنید .

```
#include <iostream.h>
void change (int & , int &);
int main ( )
 {
      int a=1 , b=2 ;
      cout << "a is " << a << " and b is " << b << endl;
      change (a,b);
      cout << "a is " << a << " and b is " << b << endl;
      return 0;
  }
void change (int &x , int &y)
 {
      int temp;
      temp = y;
      y = x;
      x = temp;
 }
```

خروجی برنامه به صورت زیر است :

```
a is 1 and b is 2
a is 2 and b is 1
```

برنامه فوق مقادیر دو متغیر **a** و**b** را توسط **change** با شیوه ارسال آرگومان با ارجاع جابه جا می کند .

یکی دیگر از کاربردهای ارسال با ارجاع ، دریافت بیش از یک خروجی از تابع می باشد ، به عنوان مثال تابع **prevnext** در برنامه زیر مقادیر صحیح قبل و بعد از اولین آرگومان را به عنوان خروجی بر می گرداند .

```
#include <iostream.h>
```

```
void prevnext (int ,int & , int &);

void main ( )
{
    int x = 100 , y , z ;
    cout << "The input of prevnext function is "
         << x << endl;
    prevnext (x,y,z) ;
    cout << "previous =" << y <<",Next =" << z;
}

void prevnext (int input , int & prev , int & next)
{
    prev = input - 1 ;
    next = input + 1 ;
}
```

خروجی برنامه فوق به صورت زیر می باشد .

```
The input of prevnext function is 100
previous =99,Next =101
```

همانطور که مشاهده می کنید آرگومان **input** مقدار داده موجود در متغیر **x** را دریافت می کند ولی آرگومان های **prev** و**next** خود متغیرهای **y** و**z** را دریافت می کنند . لذا تغییرات روی متغیر**prev** و**next** بر روی **y** و**z** انجام می گیرد و توسط تابع مقدار دهی می شوند .

**گرانبار کردن توابع (استفاده از یک نام برای چند تابع)**

**C++** استفاده از یک نام را برای چند تابع ، هنگامی که توابع از نظر نوع آرگومان ها ، تعداد آرگومان ها یا ترتیب قرار گرفتن نوع آرگومان ها با هم متفاوت باشند را امکان پذیر کرده است این قابلیت ، گرانبار کردن توابع نامیده می شود . هنگامی که یک تابع گرانبار شده فراخوانی می شود کامپایلر با مقایسه نوع ، تعداد و ترتیب آرگومان ها تابع درست را انتخاب می کند . معمولاً از توابع گرانبار شده برای ایجاد چند تابع با نامهای یکسان که کار یکسانی را بر روی انواع داده ای متفاوتی انجام می دهند استفاده می شود . به عنوان مثال اکثر توابع ریاضی زبان **C++** برای انواع داده ای متفاوت گرانبار شده اند . گرانبار کردن توابعی که کار یکسانی را انجام می دهند برنامه را قابل فهم تر و خواناتر می سازد . برنامه زیر نحوه به کار گیری توابع گرانبار شده را نشان می دهد .

```
#include <iostream.h>

int square( int );
double square( double );
```

```
void main()
{
  // calls int version
  int intResult = square( 7 );
  // calls double version
  double doubleResult = square( 7.5 );

  cout << "\nThe square of integer 7 is "
       << intResult
       << "\nThe square of double 7.5 is "
       << doubleResult
       << endl;

} // end main

// function square for int values
int square( int x )
{
  cout <<"Called square with int argument: "
       << x << endl;
  return x * x;
} // end int version of function square

// function square for double values
double square( double y )
{
  cout <<"Called square with double argument: "
       << y << endl;
  return y * y;
} // end double version of function square
```

خروجی برنامه به صورت زیر می باشد .

```
Called square with int argument: 7
Called square with double argument: 7.5

The square of integer 7 is 49
The square of double 7.5 is 56.25
```

برنامه فوق برای محاسبه مربع یک عدد صحیح (**int**) و یک عدد اعشاری (**double**) از تابع گرانبارشده **square** استفاده می کند .هنگامی که دستور:

```
int intResult = square (7) ;
```

اجرا می گردد تابع **square** با پیش تعریف :

```
int square (int) ;
```

فراخوانی می شود و هنگامی که دستور :

```
double doubleResult = square (7.5) ;
```

اجرا می گردد تابع **square** با پیش تعریف :

```
double square (double );
```

فراخوانی می شود .

**نکته :** توجه داشته باشید که توابع گرانبار شده الزامی ندارند که وظیفه یکسانی را انجام دهند . و ممکن است کاملاً با هم تفاوت داشته باشند ، ولی توصیه می شود که توابعی را گرانبار کنید که یک کار را انجام می دهند .

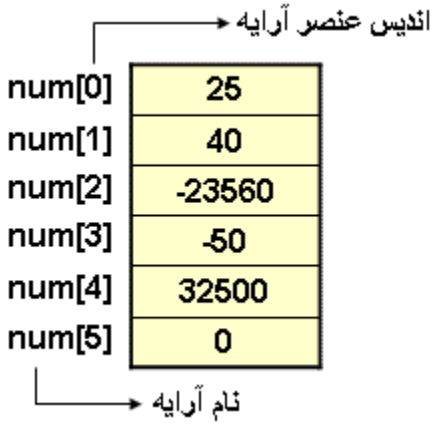
### اعلان آرایه ها

یک آرایه مجموعه ای از خانه ها متوالی حافظه می باشد که دارای یک نام و یک نوع می باشند . به هر یک از این خانه ها یک عنصر آرایه گفته می شود . برای دستیابی به یک عنصر آرایه ، باید نام آرایه و شمارنده آن خانه را مشخص کنیم . لذا عناصر آرایه توسط متغیری به نام اندیس مشخص می شوند به همین دلیل، آرایه ها را متغیرهای اندیس دار نیز می گویند . نام آرایه ، از قواعد نام گذاری متغیرها پیروی می کند . نوع آرایه نیز یکی از انواع داده ذکر شده در مبحث مفاهیم حافظه و انواع داده ای می باشد . اعلان آرایه ها به صورت زیر است :

```
نام آرایه [طول آرایه]  نوع داده آرایه ;
```

به عنوان مثال دستور زیر آرایه ای به طول ۶ ، با نام **num** را از نوع **int** ایجاد می کند .

```
int num [6];
```




توجه داشته باشید که تمام عناصر دارای نام یکسانی می باشند و تنها با اندیس از هم تفکیک می شوند . به عنوان مثال عنصر با اندیس **۲** دارای مقدار **۲۳۵۶۰**- می باشد ، ضمناً اندیس عناصر از **۰** شروع می شود .

استفاده از یک عبارت محاسباتی به جای اندیس عناصر امکان پذیر می باشد ، به عنوان مثال با فرض اینکه متغیر **a** حاوی **۲** و متغیر **b** حاوی **۳** باشد ، دستور زیر:

```
num [a+b] + = 3;
```

سه واحد به عنصر **num [5]** اضافه خواهد کرد و این عنصر حاوی عدد **۳** می گردد . برای چاپ مجموع سه عنصر اول آرایه می توانید از دستور زیر استفاده کنید :

```
cout << num[0] + num[1] + num[2] << endl;
```

برای تقسیم عنصر چهارم آرایه بر **۲** و قرار دادن حاصل در متغیر **x** از دستور زیر می توانید استفاده کنید :

```
x = num[3]/2;
```

**نکته :** توجه داشته باشید که عنصر چهارم آرایه با عنصر شماره چهار ( با اندیس چهار ) متفاوت می باشد . همانطور که در دستور فوق دیدید عنصر شماره چهار دارای اندیس سه می باشد ، دلیل این امر اینست که اندیس گذاری از صفر شروع می شود . در آرایه فوق عنصر چهارم آرایه **num[3]=-50** می باشد ولی عنصر شماره چهار ( با اندیس چهار ) **num[4]=32500** می باشد .

همانند متغیرها چند آرایه را نیز می توان توسط یک دستور تعریف کرد :

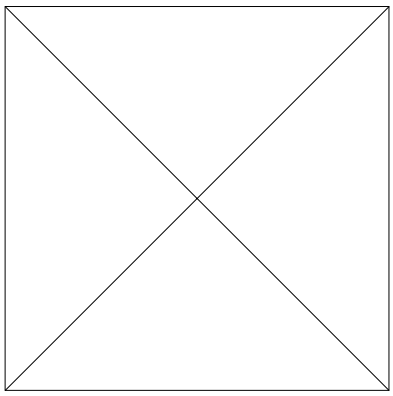
```
int b[100] , x[27] ;
```

دستور فوق ۱۰۰ خانه از نوع عدد صحیح را برای آرایه با نام **b** و ۲۷ خانه از نوع عدد صحیح را برای آرایه با نام **x** در نظر می گیرد .

برای مقدار دهی اولیه به هر یک از عناصر آرایه می توانید از شیوه زیر استفاده کنید :

```
int n[5] = {32 , 27 , 64 , 18 , 95 }
```

دستور فوق عناصر آرایه **n** را به صورت زیر مقدار دهی می کند .



اگر طول آرایه هنگام تعریف آرایه تعیین نشده باشد و لیست مقدار عناصر نوشته شود ، همانند دستور زیر :

```
int n[] = { 1 , 2 , 3 , 4 , 5 }
```

در این صورت کامپایلر به تعداد عناصر لیست ، خانه حافظه برای آرایه در نظر می گیرد ، مثلاً در دستور فوق ۵ خانه حافظه برای آرایه **n** در نظر گرفته می شود .

راه دیگری که برای مقدار دهی اولیه به عناصر آرایه وجود دارد استفاده از روش زیر است :

```
int num[10] = {0}
```

دستور فوق ۱۰ خانه حافظه برای آرایه **num** در نظر می گیرد و مقادیر همه آنها را صفر می کند . توجه داشته باشید که اگر از دستور زیر استفاده کنیم :

```
int num[10] = {1}
```

تمامی عناصر مقدار ۱ را نمی گیرند بلکه عنصر اول آرایه یک می شود و بقیه عناصر مقدار صفر را می گیرند .

در تعریف آرایه دیدید که طول آرایه را با عدد صحیح ثابتی تعیین می کنیم . هر جا که از عدد ثابتی استفاده می شود ، متغیر ثابت نیز می توان به کار برد . متغیرهای ثابت به صورت زیر تعریف می شوند :

```
const     نام متغیر ثابت = مقدار متغیر          نوع داده متغیر  ;
```

به عنوان مثال :

```
const  int  arraySize = 10;
```

دستور فوق عدد ۱۰ را به متغیر **arraySize** ثابت انتساب می دهد . توجه داشته باشید که مقدار یک متغیر ثابت را در طول برنامه نمی توان تغییر داد و نیز متغیر ثابت در هنگام تعریف شدن ، مقدار اولیه اش نیز باید تعیین گردد . به متغیرهای ثابت ، متغیرهای فقط خواندنی نیز گفته می شود . کلمه "متغیر ثابت" یک کلمه مرکب ضد و نقیض می باشد چون کلمه متغیر متضاد ثابت می باشد و این اصطلاحی است که برای اینگونه متغیرهای در اکثر زبانهای برنامه نویسی به کار می رود . برنامه زیر نحوه تعریف یک متغیر ثابت را نشان می دهد :

```
#include <iostream.h>

void main()
{
   const int x = 7;

   cout << "The value of constant variable x is: "
        << x << endl;

}
```

برنامه فوق عدد ۷ را در متغیر ثابت **x** قرار می دهد و توسط دستور **cout** آنرا چاپ می کند . همانطور که گفتیم مقدار متغیر ثابت در هنگام تعریف باید تعیین گردد و نیز ثابت قابل تغییر نمی باشد ، به برنامه زیر توجه کنید .

```
#include <iostream.h>

void main()
{
   const int x;

   x=7;
}
```

برنامه فوق هنگام کامپایل شدن دو پیغام خطا خواهد داد ، چون متغیر ثابت هنگام تعریف مقدار دهی نشده و نیز در برنامه دستوری برای تغییر مقدار آن آورده نشده است .

```
 Compiling C:\TCP\BIN\CONST1.CPP:
Error : Constant variable 'x' must be initialized
Error : Cannot modify a const object
```

**مثال :** در برنامه زیر طول آرایه توسط متغیر ثابتی تعیین می گردد و عناصر آرایه توسط حلقه **for** مقدار دهی شده و سپس توسط حلقه **for** دیگری ، مقدار عناصر آرایه چاپ می گردد .

```
#include <iostream.h>

void main()
{
   const int arraySize = 10;

   int s[ arraySize ];

   for ( int i = 0; i < arraySize; i++ )
   s[ i ] = 2 + 2 * i;

   cout << "Element Value" << endl;

   for ( int j = 0; j < arraySize; j++ )
       cout << j << "\t " << s[ j ] << endl;

}
```

خروجی برنامه فوق به صورت زیر می باشد :

```
Element  Value
0        2
1        4
2        6
3        8
4        10
5        12
6        14
7        16
8        18
9        20
```

### چند مثال از آرایه ها و کاربرد آنها

**مثال :** برنامه ای بنویسید که ۱۰ عدد صحیح را ورودی دریافت کرده و در آرایه ای قرار داده سپس مجموع عناصر آرایه را محاسبه کرده و درخروجی چاپ نماید .

```
#include <iostream.h>

void main()
{
  const int arraySize = 10;
  int total = 0,i;
  int a[ arraySize ];

  for (i = 0; i < arraySize; i++)
    {
     cout << "Enter number " << i << " : ";
     cin >> a[ i ];
    }

  for (i = 0; i < arraySize; i++ )
     total += a[ i ];

  cout << "Total of array element values is "
       << total << endl;

}
```

خروجی برنامه به صورت زیر می باشد :

```
Enter number 0 : 12
Enter number 1 : 3
Enter number 2 : 4
Enter number 3 : 654
Enter number 4 : 34
Enter number 5 : 2
Enter number 6 : 123
Enter number 7 : 794
Enter number 8 : 365
Enter number 9 : 23
Total of array element values is 2014
```

**مثال :** برنامه ای بنویسید که توسط آرایه ، نمودار میله ای افقی برای اعداد {۱ و۱۷ و۵ و۱۳ و۹ و۱۱ و۷ و۱۵ و۳ و۱۹ } رسم کند .

```
#include <iostream.h>

int main()
{
  const int arraySize = 10;
  int n[ arraySize ] = { 19, 3, 15, 7, 11,
                         9, 13, 5, 17, 1 };

  cout << "Element" << " Value" << endl;

  for ( int i = 0; i < arraySize; i++ ) {
     cout << i << "\t " << n[ i ] << "\t";

     for ( int j = 0; j < n[ i ]; j++ )
        cout << '*';
     cout << endl;
  }

return 0;

}
```

خروجی برنامه به صورت زیر می باشد :

```
Element  Value
0        19     ********************
1        3      ***
2        15     ***************
3        7      *******
4        11     ***********
5        9      *********
6        13     *************
7        5      *****
8        17     *****************
9        1      *
```

**مثال :** برنامه ای بنویسید که یک تاس را **۶۰۰۰** بار پرتاب کرده و توسط آرایه ای تعداد دفعات آمدن هر وجه را حساب کند .( تعداد دفعات آمدن هر وجه را یک عنصر آرایه ای در نظر بگیرید )