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

void main()
{
  const int arraySize = 7;
  int frequency[ arraySize ] = { 0 };

  srand( time( 0 ) );

  for ( int roll = 1; roll <= 6000; roll++ )
     ++frequency[ 1 + rand() % 6 ];

  cout << "Face Frequency" << endl;

  for ( int face = 1; face < arraySize; face++ )
     cout << face << "\t" << frequency[face] << endl;

}
```

خروجی برنامه به صورت زیر می باشد :

```
Face  Frequency
1       1023
2       990
3       1008
4       971
5       1025
6       983
```

دستور **++frequency [rand()%6 + 1];** ، مقدار عنصر مربوط به هر وجه را یک واحد اضافه می کند ، زیرا **rand()%6 + 1** عددی بین ۱ تا ۶ تولید می کند ، پس هر بار به طور تصادفی تنها مقدار عنصر مربوط به یکی از وجوه افزایش می یابد .

یکی از کار برد های آرایه ها ، استفاده از آنها برای ذخیره رشته ای از حروف می باشد . تا به حال متغیرهایی که از ورودی دریافت می کردیم و یا آرایه هایی که تا به اینجا دیدید تنها شامل اعداد می شدند در اینجا به عنوان مثال نحوه دریافت یک نام از ورودی و چاپ آن در خروجی را بررسی می کنیم .(در فصل بعد یعنی اشاره گرها و رشته ها ، به طور مفصل تر راجع به رشته ها صحبت خواهیم کرد)

یک عبارت رشته ای مانند: **"hello"** در واقع آرایه ای از حروف می باشد.

```
char string1[]="hello";
```

دستور فوق آرایه ای به نام **string1** را با کلمه **hello** مقدار دهی می کند. طول آرایه فوق برابر است با طول کلمه **hello** یعنی ۵ بعلاوه یک واحد که مربوط است به کاراکتر پوچ که انتهای رشته را مشخص می کند. لذا آرایه **string1** دارای طول ۶ می باشد. کاراکتر پوچ در زبان C++ توسط **'\0'** مشخص می گردد. انتهای کلیه عبارات رشته ای با این کاراکتر مشخص می شود.

```
char string1[]={'h','e','l','l','o','\0'}
```

دستور فوق عناصر آرایه string1 را جداگانه مقدار دهی می کند. توجه داشته باشید که عناصر آرایه در دستور فوق داخل (') قرار گرفتند و نیز انتهای رشته با **'\0'** تعیین شد. نتیجه همانند دستور **char string1[]="hello";** می باشد.

چون عبارت رشته ای، آرایه ای از حروف می باشند، لذا به هر یک از حروف رشته، توسط اندیس عنصری که شامل آن حرف می باشد، می توان دسترسی پیدا کرد. به عنوان مثال **string1[0]** شامل **'h'** و **string1[3]** شامل **'l'** و **string1[5]** شامل **'\0'** می باشد.

توسط دستور cin نیز می توان به طور مستقیم کلمه وارد شده از صفحه کلید را در آرایه ای رشته ای قار داد.

```
char string2[20];
```

دستور فوق یک آرایه رشته ای که قابلیت دریافت کلمه ای با طول ۱۹ به همراه کاراکتر پوچ را دارا می باشد.

```
cin >> string2;
```

دستور فوق رشته ای از حروف را از صفحه کلید خوانده و در **string2** قرار می دهدو کاراکتر پوچ را به انتهای رشته وارد شده توسط کاربر اضافه می کند. به طور پیش فرض دستور cin کاراکتر ها را از صفحه کلید تا رسیدن به اولین فضای خالی در یافت می کند. به عنوان مثال اگر هنگام اجرای دستور **cin >> string2;** کاربر عبارت "**hello there**" را وارد کند، تنها کلمه **hello** در **string2** قرار می گیرد. چون عبارت وارد شده شامل کاراکتر فاصله است. برنامه زیر نحوه به کار گیری آرایه های رشته ای را نشان می دهد.

```
#include <iostream.h>

void main()
{
  char name[ 20 ];

  cout << "Please Enter your name : ";
```

```
  cin >> name;

  cout << "Welcome, " << name
       << " to this program. \n" ;

  cout << "Your separated name is\n";

  for ( int i = 0; name[ i ] != '\0'; i++ )
       cout << name[ i ] << ' ';

}
```

خروجی برنامه به صورت زیر می باشد :

```
Please Enter your name : Mohammad
Welcome, Mohammad to this program.
Your separated name is
M o h a m m a d
```

در برنامه فوق حلقه for ، حروف نام وارد شده توسط کاربر را جدا جدا در خروجی چاپ می کند. ضمنا شرط حلقه **name[i] != '\0'** می باشد و تا وقتی این شرط برقرار است که حلقه به انتهای رشته نرسیده باشد.

در مبحث قوانین حوزه دیدید که اگر بخواهیم یک متغیر محلی تابع، مقدار خود را حفظ کرده و برای دفعات بعدی فراخوانی تابع نیز نگه دارد، از کلمه **static** استفاده کردیم. نوع **static** را برای آرایه ها نیز می توان به کار برد و از همان قوانین گفته شده در مبحث مذکور پیروی می کند. به برنامه زیر و خروجی آن توجه کنید.

```
#include <iostream.h>

void staticArrayInit( void );
void automaticArrayInit( void );

int main()
{
  cout << "First call to each function:\n";
  staticArrayInit();
  automaticArrayInit();

  cout << "\n\nSecond call to each function:\n";

  staticArrayInit();
  automaticArrayInit();
  cout << endl;
```

```
return 0;

}

// function to demonstrate a static local array
void staticArrayInit( void )
{
  // initializes elements to 0
  // first time function is called
  static int array1[ 3 ]={0};

  cout << "\nValues on entering staticArrayInit:\n";

  // output contents of array1
  for ( int i = 0; i < 3; i++ )
     cout << "array1[" << i << "] = "
          << array1[ i ] << " ";

  cout << "\nValues on exiting staticArrayInit:\n";

  // modify and output contents of array1
  for ( int j = 0; j < 3; j++ )
     cout << "array1[" << j << "] = "
          << ( array1[ j ] += 5 ) << " ";

} // end function staticArrayInit

// function to demonstrate an automatic local array
void automaticArrayInit( void )
{
  // initializes elements each time function is called
  int array2[ 3 ] = { 1, 2, 3 };

  cout << endl << endl;
  cout << "Values on entering automaticArrayInit:\n";

  // output contents of array2
  for ( int i = 0; i < 3; i++ )
     cout << "array2[" << i << "] = "
          << array2[ i ] << " ";

  cout << "\nValues on exiting automaticArrayInit:\n";

  // modify and output contents of array2
  for ( int j = 0; j < 3; j++ )
    cout << "array2[" << j << "] = "
```

```
          << ( array2[ j ] += 5 ) << " ";

 }
```

خروجی برنامه به صورت زیر می باشد :

```
First call to each function:

Values on entering staticArrayInit:
array1[0] = 0  array1[1] = 0  array1[2] = 0
Values on exiting staticArrayInit:
array1[0] = 5  array1[1] = 5  array1[2] = 5

Values on entering automaticArrayInit:
array2[0] = 1  array2[1] = 2  array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6  array2[1] = 7  array2[2] = 8

Second call to each function:

Values on entering staticArrayInit:
array1[0] = 5  array1[1] = 5  array1[2] = 5
Values on exiting staticArrayInit:
array1[0] = 10  array1[1] = 10  array1[2] = 10

Values on entering automaticArrayInit:
array2[0] = 1  array2[1] = 2  array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6  array2[1] = 7  array2[2] = 8
```

در برنامه فوق عناصر آرایه **array1** در اولین بار فراخوانی تابع **staticArrayInit** مقدار صفر را می گیرند ولی در دفعات بعدی فراخوانی این تابع، آخرین مقدار قبلی خود را حفظ می کنند . اما آرایه **array2** در هر بار فراخوانی تابع **automaticArrayInit** مقدار دهی اولیه می شود و با خروج از تابع مقدار خود را از دست می دهد.

ارسال آرایه ها به توابع

برای ارسال یک آرایه به عنوان آرگومان به یک تابع ، کافیست نام آرایه را بدون علامت براکت ([]) به کار ببرید . به عنوان مثال اگر آرایه ای با نام **x** به صورت زیر تعریف شده باشد :

```
int x[24];
```

برای ارسال آن به تابع **modifyArray** کافیست تابع را به صورت زیر:

```
modifyArray(x,24);
```

فراخوانی کنید . دستور فوق آرایه و طول آن را به تابع **modifyArray** ارسال می کند . معمولاً هنگامی که آرایه ای را به تابعی ارسال می کنند ، طول آرایه را نیز همراه آرایه به عنوان یک آرگومان جداگانه به تابع می فرستند.

C++ آرایه ها را با شیوه شبیه سازی شده ارسال آرگومان ها با ارجاع به تابع ارسال می نماید ، لذا تابع فراخوانی شده مقدار عناصر آرایه ارسالی را می تواند تغییر دهد . هنگامی که نام تابع را به عنوان آرگومان تابع به کار می بریم ، آدرس خانه حافظه اولین عنصر آرایه به تابع ارسال می شود لذا تابع می داند که عناصر آرایه در کجای حافظه قرار گرفته اند . بنابراین هنگامی که تابع فراخوانی شده عناصر آرایه را تغییر می دهد ، این تغییرات روی عناصر آرایه اصلی که به تابع ارسال شده است ، انجام می پذیرد .

نکته : توجه داشته باشید که عناصر آرایه را به صورت جداگانه همانند ارسال متغیرها و یا مقادیر عددی به تابع ارسال کرد . در این صورت تغییر بر روی آرگومان ارسالی ، تأثیری بر روی عنصر آرایه نخواهد داشت . در حقیقت این شیوه ، همان ارسال با مقدار می باشد .

برای اینکه تابعی قادر به دریافت یک آرایه به عنوان ورودی باشد ، هنگام تعریف تابع در لیست آرگومانهای آن ، این مطلب باید مشخص گردد . به عنوان مثال تعریف تابع **modifyArray** را به صورت زیر می توان نوشت :

```
void modifyArray (int b[] ,int array size)
```

دستور فوق مشخص می کند که تابع **modifyArray** قادر به دریافت آدرس آرایه ای از اعداد صحیح توسط آرگومان **b** و تعداد عناصر آرایه توسط آرگومان **arraySize** می باشد . ضمناً تعداد عناصر آرایه را لازم نیست بین براکت ها ([]) بنویسید ، اگر این کار نیز صورت پذیرد ، کامپایلر آن را نادیده می گیرد . توجه داشته باشید که پیش تعریف تابع فوق را به صورت زیر بنویسید :

```
void modifyArray (int [] , int);
```

برنامه زیر نحوه ارسال یک آرایه را به تابع و تفاوت ارسال یک عنصر آرایه به تابع و ارسال کل آرایه به تابع را نشان می دهد .

```
#include <iostream.h>

void modifyArray( int [], int );
```

```
void modifyElement( int );

void main()
{
  const int arraySize = 5;
  int a[ arraySize ] = { 0, 1, 2, 3, 4 };

  cout<<"Effects of passing entire array by reference:"
      <<"\n\nThe values of the original array are:\n";

  // output original array
  for ( int i = 0; i < arraySize; i++ )
    cout << "\t"<< a[ i ];

  cout << endl;

  // pass array a to modifyArray by reference
  modifyArray( a, arraySize );

  cout << "The values of the modified array are:\n";

  // output modified array
  for ( int j = 0; j < arraySize; j++ )
    cout << "\t" << a[ j ];

  // output value of a[ 3 ]
  cout<<"\n\n\n"
      <<"Effects of passing array element by value:"
      <<"\n\nThe value of a[3] is " << a[ 3 ] << '\n';

  // pass array element a[ 3 ] by value
  modifyElement( a[ 3 ] );

  // output value of a[ 3 ]
  cout << "The value of a[3] is " << a[ 3 ] << endl;

}

// in function modifyArray, "b" points to
// the original array "a" in memory
void modifyArray( int b[], int sizeOfArray )
{
  // multiply each array element by 2
  for ( int k = 0; k < sizeOfArray; k++ )
  b[ k ] *= 2;
}
```

```
// in function modifyElement, "e" is a local copy of
// array element a[ 3 ] passed from main
void modifyElement( int e )
{
  // multiply parameter by 2
  cout << "Value in modifyElement is "
       << ( e *= 2 ) << endl;
}
```

خروجی برنامه فوق به صورت زیر می باشد :

```
Effects of passing entire array by reference:

The values of the original array are:
        0       1       2       3       4
The values of the modified array are:
        0       2       4       6       8


Effects of passing array element by value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
```

در برنامه فوق ، تابع **modifyArray** مقدار عناصر آرایه **a** را که به آن فرستاده شده است دو برابر می کند . تابع **modifyElement** مقدار آرگومان دریافتی را دو برابر کرده و در خروجی چاپ می کند ولی تأثیری در نسخه اصلی عنصر آرایه نداشته و تغییری در مقدار آن ایجاد نمی کند .

بعضی مواقع ممکن است بخواهید که تابعی ، اجازه تغییر عناصر آرایه ای که به آن فرستاده شده است را نداشته باشد . برای این کار هنگام تعریف تابع کافی است از کلمه **const** قبل از آرگومان مربوط به آن آرایه استفاده کنید ، در چنین حالتی اگر داخل تابع قصد تغییر مقدار عناصر آرایه را داشته باشید با یک پیغام خطای کامپایلر مواجه می شوید و کامپایلر اجازه این کار را به شما نمی دهد . به برنامه زیر توجه کنید :

```
#include <iostream.h>

void tryToModifyArray( const int [] );

void main()
{
  int a[] = { 10, 20, 30 };
```

```
  tryToModifyArray( a );

  cout << a[0] <<' '<< a[1] <<' '<< a[2] <<'\n';

}

// In function tryToModifyArray, "b" cannot be used
// to modify the original array "a" in main.
void tryToModifyArray( const int b[] )
{
  b[ 0 ] /= 2; // error
  b[ 1 ] /= 2; // error
  b[ 2 ] /= 2; // error
}
```

هنگام کامپایل کردن برنامه فوق با پیغام های خطای زیر مواجه خواهید شد ، چون در تابع قصد تغییر عناصر آرایه ای را که به صورت ثابت به تابع ارسال شده بود ، داشتیم .

```
Error in line 19: Cannot modify a const object
Error in line 20: Cannot modify a const object
Error in line 21: Cannot modify a const object
```

مرتب کردن آرایه ها

مرتب کردن اطلاعات چه به صورت صعودی یا نزولی ، یکی از مهمترین وظایف کامپیوتر می باشد . به عنوان مثال تعیین رتبه دانش آموزان یک مدرسه بر اساس معدل ، تعیین رتبه شرکت کنندگان در کنکور ، مرتب کردن شماره تلفن ها بر اساس نام صاحب تلفن را می توان نام برد . برای آشنایی با شیوه مرتب کردن ، لیست اعداد زیر را در نظر بگیرید :

2,5,4,3,6,1

برای مرتب کردن لیست اعداد فوق از کوچک به بزرگ آنها را در آرایه ای قرار می دهیم :

```
int a[] = { 2 , 5 , 4 , 3 , 6 , 1};
```

حال کافی است آرایه **a** را به صورت صعودی مرتب کنیم . برای انجام این کار از روشی به نام مرتب کردن حبابی استفاده می کنیم . این تکنیک به دلیل اینکه مقادیر کوچکتر همانند حبابی در آب به سمت بالا حرکت می کنند ، مرتب کردن حبابی

گفته می شود . برای مرتب کردن آرایه چندین بار باید روی آرایه حرکت کنیم و در هر بار حرکت عناصر دو به دو با هم مقایسه می شوند ، و در صورتی که به صورت نزولی قرار داشته باشند مقادیرشان جابه جا می گردد و در غیر این صورت به همان ترتیب باقی می مانند .

برنامه زیر لیست اعداد ذکر شده را به شیوه مرتب کردن حبابی ، از کوچک به بزرگ مرتب می کند .

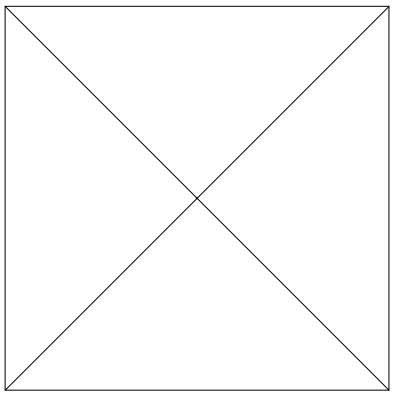
```
#include <iostream.h>

void showArray(const int [] , int);

void main()
{
  const int arraySize = 6;
  int a[ arraySize ] = { 2, 5, 4, 3, 6 ,1};
  int hold;

  cout << "Data items in original order\n";

  showArray(a,arraySize);

  for ( int i = 0; i < arraySize - 1 ; i++ )
     for ( int j = 0; j < arraySize - 1; j++ )
        if ( a[ j ] > a[ j + 1 ] ) {
           hold = a[ j ];
           a[ j ] = a[ j + 1 ];
           a[ j + 1 ] = hold;
         }

  cout << "\nData items in ascending order\n";

  showArray(a,arraySize);

}

void showArray( const int array[] ,int arraySize)
{
  for (int c=0; c<arraySize ;c++)
     cout << array[c] << " ";
     cout << endl;
}
```

خروجی برنامه فوق به صورت زیر می باشد :

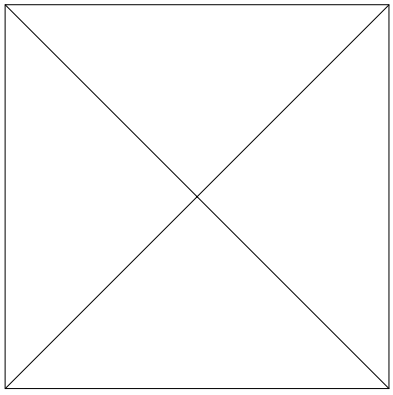
```
Data items in original order
```

```
2 5 4 3 6 1

Data items in ascending order
1 2 3 4 5 6
```

در برنامه فوق تابع **showArray** وظیفه نمایش عناصر آرایه را به عهده دارد . در اولین اجرای دستورات حلقه ها ، **i = 0** می باشد . در اولین دور اجرای حلقه داخلی ، با شمارنده **j** عناصر آرایه به صورت زیر با هم مقایسه می شوند .



پس از اولین دور حرکت روی عناصر آرایه، ترتیب اعداد به صورت فوق خواهد شد. سپس **i = 1** می گردد و دفعات بعدی مقایسه انجام گرفته و در انتهای هر بار مقایسه ترتیب عناصر به صورت زیر می شود .



که سرانجام با به انتها رسیدن حرکت روی آرایه عناصر به صورت صعودی مرتب می شوند .

جستجو در آرایه ها

عمل جستجو یکی از مهمترین وظایف برنامه های کامپیوتری می باشد . به عنوان مثال دفتر تلفنی را در نظر بگیرید که به دنبال نام فردی در آن می گردیم و یا جستجوی نام یک دانشجو در لیست دانشجویان کلاس . در این مبحث دو روش جستجو را مورد بررسی قرار می دهیم . یک روش جستجوی خطی است که معمولاً در آرایه های نا مرتب مورد استفاده قرار می گیرد و روش دیگر جستجوی دو دویی می باشد که در آرایه های مرتب از این شیوه می توانیم استفاده کنیم .

در روش جستجوی خطی ، عنصر مورد جستجو با هر یک از عناصر آرایه مقایسه می شود ، چنانچه دو عنصر برابر بودند ، عمل جستجو به پایان می رسد و اندیس عنصر برگردانده می شود و گرنه مقایسه با عنصر بعدی آرایه انجام می پذیرد . از آنجا که عناصر آرایه نا مرتب می باشند عنصر مورد جستجو در هر کجای آرایه می تواند باشد لذا عمل مقایسه تا یافتن عنصر مورد نظر و یا رسیدن به انتهای آرایه یعنی جستجو در همه عناصر آرایه ادامه می یابد . برنامه زیر نمونه ای از جستجوی خطی در آرایه می باشد :

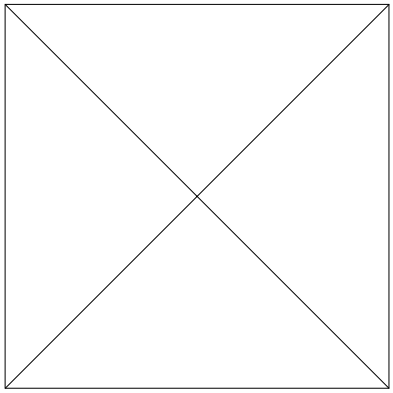
```
#include <iostream.h>

int linearSearch(const int [], int, int );

void main()
{
   const int arraySize = 7;
   int a[ arraySize ]={2,6,4,3,12,10,5};
   int searchKey;

   cout << "Enter integer search key: ";
   cin >> searchKey;

   int element=linearSearch(a, searchKey, arraySize);

   if ( element != -1 )
      cout << "Found value in element "
           << element << endl;
   else
      cout << "Value not found" << endl;

}

int linearSearch( const int array[],
                  int key, int sizeOfArray )
{
   for ( int j = 0; j < sizeOfArray; j++ )

   if ( array[ j ] == key )
      return j;

   return -1;
}
```

خروجی برنامه فوق به صورت زیر می باشد :

```
Enter integer search key: 12
```

```
Found value in element 4
```

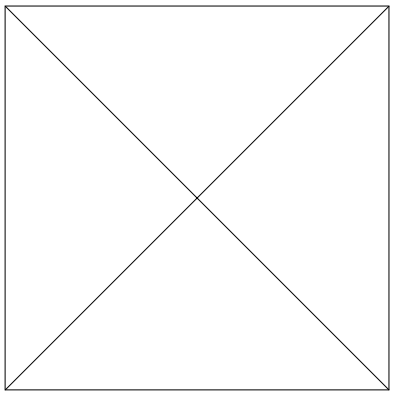
نحوه جستجوی عدد ۱۲ در آرایه به صورت زیر می باشد :



بار دیگر برنامه را برای جستجوی عدد ۲۰ در آرایه اجرا می کنیم . خروجی برنامه به صورت زیر می باشد .

```
Enter integer search key: 20
Value not found
```

نحوه جستجوی عدد ۲۰ در آرایه به صورت زیر می باشد :



روش جستجوی دو دویی در آرایه های مرتب شده قابل استفاده می باشد و از سرعت بالایی برخوردار می باشد . در این الگوریتم ، در هر بار مقایسه ، نیمی از عناصر آرایه حذف می شوند . الگوریتم عنصر میانی آرایه را می یابد و آن را با عنصر مورد جستجو، مقایسه می کند . اگر برابر بودند ، جستجو به پایان رسیده و اندیس عنصر برگردانده می شود ، در غیر این صورت عمل جستجو روی نیمی از عناصر انجام می گیرد . اگر عنصر مورد جستجو کوچکتر از عنصر میانی باشد ، جستجو روی نیمه اول آرایه صورت می پذیرد ، در غیر این صورت نیمه دوم آرایه جستجو می شود . این جستجوی جدید روی زیر آرایه طبق الگوریتم جستجو روی آرایه اصلی انجام می شود یعنی عنصر میانی زیر آرایه یافته می شود و با عنصر مورد

جستجو مقایسه می گردد ، اگر برابر نباشند زیر آرایه مجدداً نصف می شود و در هر بار جستجو زیر آرایه ها کوچکتر می گردند . عمل جستجو تا یافتن عنصر مورد نظر(یعنی برابر بودن عنصر مورد جستجو با عنصر میانی یکی از زیر آرایه ها) و یا نیافتن عنصر مورد نظر (برابر نبودن عنصر مورد جستجو با عنصر زیر آرایه ای شامل تنها یک عنصر) ادامه می یابد . برنامه زیر نمونه ای از جستجوی دو دویی در آرایه مرتب می باشد .

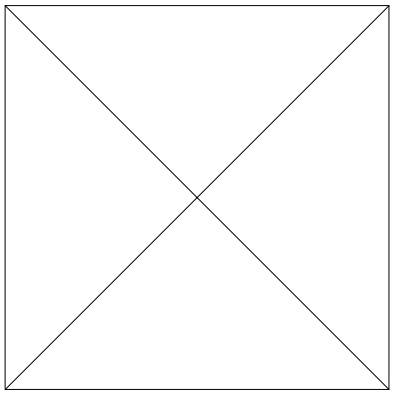
```
#include <iostream.h>

int binarySearch( const int [], int, int);

void main()
{
  const int arraySize = 15;
  int a[ arraySize ]={0,2,4,6,8,10,12,14,
                      16,18,20,22,24,26,28};
  int key;

  cout << "Enter a number between 0 and 28: ";
  cin >> key;

  int result =
      binarySearch( a, arraySize, key);

  if ( result != -1 )
    cout << '\n' << key << " found in array element "
         << result << endl;
  else
    cout << '\n' << key << " not found" << endl;
}

int binarySearch( const int b[],
                  int arraySize ,
                  int searchKey )
{
  int middle,low=0,high=arraySize - 1;

  while ( low <= high )
  {
    middle = ( low + high ) / 2;
    if ( searchKey < b[ middle ] )
      high = middle - 1;
    else
      if ( searchKey > b[ middle ] )
        low = middle + 1;
      else return middle;
  }
```

```
  return -1;
}
```

خروجی برنامه فوق به صورت زیر می باشد :

```
Enter a number between 0 and 28: 8

8 found in array element 4
```

نحوه جستجوی عدد ۸ در آرایه به صورت زیر می باشد :



بار دیگر برنامه را برای جستجوی عدد ۲۵ اجرا می کنیم . خروجی برنامه به صورت زیر می باشد :

```
Enter a number between 0 and 28: 25

25 not found
```

نحوه جستجوی عدد ۲۵ در آرایه به صورت زیر می باشد :