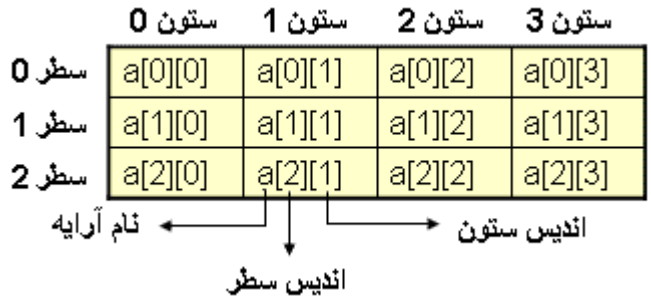
آرایه های چند بعدی

آرایه ها در **C++** می توانند بیش از یک اندیس داشته باشند . بدین صورت یک آرایه چند اندیسه یا چند بعدی خواهیم داشت . کاربردی ترین آرایه چند بعدی ، آرایه دو بعدی می باشد که توسط آن می توان جدولی حاوی مقادیر مختلف را شبیه سازی کرد . به دستور زیر توجه کنید :

```
int a[3][4];
```

دستور فوق یک آرایه دو بعدی **۳** در **۴** را به صورت زیر ایجاد می کند :



هر عنصر آرایه به صورت **a[i][j]**، که در آن **i** شماره سطر و **j** شماره ستون می باشد ، قابل دسترسی است .

برای مقدار دهی اولیه به عناصر آرایه می توانید مانند دستور زیر عمل کنید :

```
int b[2][2] = {{1,2},{3,4}};
```

دستور فوق آرایه **b** را به صورت زیر مقدار دهی می کند :

2	1
4	3

در برنامه زیر چند نمونه از مقدار دهی اولیه به آرایه دو بعدی ۲ در ۳ آورده شده است :

```
#include <iostream.h>

void printArray( int [][ 3 ] );
```

```
void main()
{
  int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
  int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
  int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };

  cout << "Values in array1 by row are:" << endl;
  printArray( array1 );

  cout << "Values in array2 by row are:" << endl;
  printArray( array2 );

  cout << "Values in array3 by row are:" << endl;
  printArray( array3 );

}

void printArray( int a[][ 3 ] )
{
  for ( int i = 0; i < 2; i++ )
  {
    for ( int j = 0; j < 3; j++ )
      cout << a[ i ][ j ] << ' ';
    cout << endl;
  }

}
```

خروجی برنامه به صورت زیر می باشد :

```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```

در برنامه فوق تابع **PrintArray** وظیفه چاپ عناصر آرایه را بر روی صفحه نمایش دارا می باشد . توجه داشته باشید که ارسال آرایه به تابع به صورت **int a[][3]** انجام گرفت . اگر بیاد داشته باشید در آرایه های یک بعدی نیازی به ذکر طول آرایه نبود اما آرایه های بیش از یک بعد تعداد عناصر بعدهای دیگر باید ذکر شود ، اما نیازی به ذکر طول بعد اول نمی باشد .

مثال : در برنامه زیر آرایه **۲** بعدی **۱۰** در **۱۰** را با مقادیر جدول ضرب ، مقدار دهی می کنیم و سپس آن را بر روی صفحه نمایش چاپ می کنیم .

```
#include <iostream.h>
void main( )
{

  int a[10][10],i,j;

  for (i=0;i<10;i++)
    for (j=0;j<10;j++)
      a[i][j]=(i+1)*(j+1);


  for (i=0;i<10;i++){
    for (j=0;j<10;j++)
      cout <<a[i][j]<<"\t";
    cout<<endl;
  }
}
```

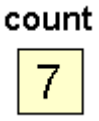
خروجی برنامه فوق به صورت زیر می باشد :

```
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
```

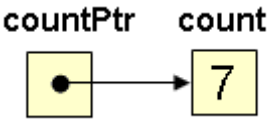
```
9   18  27  36  45  54  63  72  81  90

10  20  30  40  50  60  70  80  90  100
```

تعریف اشاره گر

متغیرهای اشاره گر، آدرس خانه های حافظه را در خود نگهداری می کنند . متغیرها معمولاً مقدار مشخصی را در خود دارند ولی اشاره گرها آدرس یک متغیر را در خود دارند . نام یک متغیر به طور مستقیم به یک مقدار ، مراجعه می کند اما یک اشاره گر به طور غیر مستقیم به یک مقدار مراجعه می کند . به شکل زیر توجه کنید :



count به طور مستقیم به مقدار **۷** مراجعه می کند .



countPtr به طور غیر مستقیم به متغیری که حاوی **۷** می باشد مراجعه می کند .

اشاره گرها نیز مانند هر متغیر دیگری ، قبل از استفاده باید تعریف شوند . به عنوان مثال دستور زیر متغیر **count** را از نوع **int** و متغیر **countPtr** را اشاره گری به متغیری از نوع **int** تعریف می کند .

```
int count , *countPtr ;
```

برای تعریف هر متغیری از نوع اشاره گر ، از علامت ستاره * قبل از نام آن استفاده می کنیم .

به دستور زیر توجه کنید :

```
double *xPtr , *yPtr ;
```

در دستور فوق ، **xPtr** و **yPtr** اشاره گرهایی به متغیرهایی از نوع **double** تعریف می شوند .

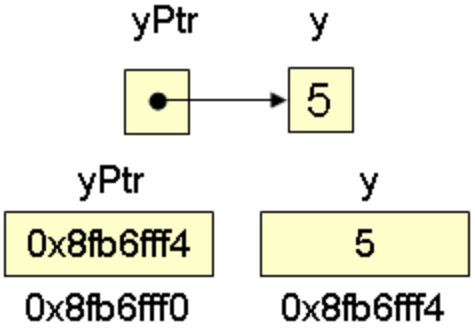
نکته: استفاده از **Ptr** در انتهای نام متغیرهای اشاره گر الزامی نمی باشد ولی برای اینکه برنامه قابل فهم تر باشد توصیه می شود از **Ptr** در انتهای نام اشاره گر استفاده کنید .

عملگرهای اشاره گر

عملگر آدرس (**&**) عملگری است که آدرس خانه حافظه عملوند خود را بر می گرداند .

```
int y=5;
int *yPtr;
yPtr = &y;
```

دستورات فوق متغیر **y** را از نوع **int** با عدد ۵ مقدار دهی کرده و سپس **yPtr** ، اشاره گری به متغیری از نوع **int** تعریف می شود و سرانجام آدرس خانه حافظه **y** در **yPtr** قرار می گیرد .



همانطور که در شکل فوق می بینید ، **yPtr** حاوی آدرس خانه حافظه **y** می باشد .

برای آشنایی با نحوه استفاده از اشاره گرها به برنامه زیر توجه کنید :

```
#include <iostream.h>

void main ()
{
  int x = 5, y = 15;
  int *xPtr, *yPtr;

  xPtr = &x;
  yPtr = &y;
```

```
  cout << "The value of x is " << x
       << "\nThe address of x is " << &x
       << "\nThe value of xPtr is " << xPtr;

  cout << "\n\nThe value of y is " << y
       << "\nThe address of y is " << &y
       << "\nThe value of yPtr is " << yPtr;

  *xPtr = 10;
  cout << "\n\nx=" << x << " and y=" << y;

  *yPtr = *xPtr;
  cout << "\nx=" << x << " and y=" << y;

  xPtr = yPtr;
  cout << "\nx=" << x << " and y=" << y;

  *xPtr = 20;
  cout << "\nx=" << x << " and y=" << y;
}
```

خروجی برنامه فوق به صورت زیر می باشد :

```
The value of x is 5
The address of x is 0x8fb4fff4
The value of xPtr is 0x8fb4fff4

The value of y is 15
The address of y is 0x8fb4fff2
The value of yPtr is 0x8fb4fff2

x=10 and y=15
x=10 and y=10
x=10 and y=10
x=10 and y=20
```

در برنامه فوق دو متغیر **x** و**y** از نوع عدد صحیح تعریف شده و **x** حاوی ۵ و **y** حاوی ۱۵ می گردد سپس **xPtr** و **yPtr** اشاره گری به عدد صحیح تعریف می شوند .

```
xPtr = &x;
yPtr = &y;
```

دو دستور فوق همانطور که در خروجی برنامه نیز می بیند ، آدرس خانه حافظه **x** را در **xPtr** و آدرس خانه حافظه **y** را در **yPtr** قرار می دهد .

دستور **xPtr = 10;*** در خانه ای از حافظه که **xPtr** اشاره می کند (یعنی متغیر **x**) عدد ۱۰ را قرار می دهد سپس**yPtr;xPtr** = ***** مقدار خانه حافظه ای که **xPtr** به آن اشاره می کند را در خانه ای از حافظه که **yPtr** به آن اشاره می کند قرار می دهد یعنی مقدار متغیر **x** در متغیر **y** قرار می گیرد .

دستور **xPtr = yPtr;** مقدار **yPtr** را که همان آدرس خانه حافظه **y** می باشد در **xPtr** قرار می دهد پس با اجرای این دستور **xPtr** دیگر به **x** اشاره نمی کند بلکه به **y** اشاره خواهد کرد ، لذا با اجرای دستور **xPtr = 20;*** همانطور که مشاهده می کنید **x** حاوی ۲۰ نمی شود بلکه این مقدار **y** است که به ۲۰ تغییر می یابد .

ارسال آرگومان به تابع توسط اشاره گر

تا به حال با دو روش ارسال آرگومانها به توابع آشنا شده اید . ارسال با مقدار و ارسال با ارجاع . در این مبحث روش دیگری را که ارسال توسط اشاره گر می باشد مورد بررسی قرار می دهیم . اشاره گرها مانند آرگومانهای ارجاع می توانند برای تغییر یک یا چند متغیر ارسال شده از داخل تابع و یا برای ارسال داده های بزرگ به توابع مورد استفاده قرار گیرند . در برنامه زیر ، شیوه ارسال آرگومان توسط اشاره گر به تابع مورد استفاده قرار گرفته است .

```
#include <iostream.h>

void callByPointer( int * );

int main()
{
  int number = 5;

  cout << "The original value of number is " << number;

  // pass address of number to callByPointer
  callByPointer( &number );

  cout << "\nThe new value of number is "
       << number << endl;

  return 0;
}

void callByPointer( int *nPtr )
{
  *nPtr = *nPtr * *nPtr * *nPtr; // cube *nPtr
}
```

خروجی برنامه فوق به صورت زیر می باشد :

```
The original value of number is 5
The new value of number is 125
```

همانطور که در برنامه فوق مشاهده می کنید ، برای ارسال آرگومان به تابع توسط اشاره گر ، در پیش تعریف تابع پس از مشخص کردن نوع آرگومان از علامت * استفاده می کنیم و در تعریف تابع نیز علامت * را قبل از نام آرگومان اشاره گر قرار می دهیم . ضمناً از آنجا که اشاره گرها آدرس متغیرها را در خود قرار می دهند برای ارسال آرگومان توسط اشاره گر به یک تابع ، هنگام فراخوانی تابع باید نام متغیر ارسالی را همراه علامت **&** به کار ببریم چون تنها در این صورت آدرس متغیر ارسال می گردد .

Constو اشاره گرها

همانطور که می دانید **const** به برنامه نویس این امکان را می دهد که مقدار یک متغیر را که از نوع **const** تعریف شده است ، در طول برنامه نتوان تغییر داد . داده هایی که توسط اشاره گرها به توابع ارسال می شوند ، داخل تابع قابل تغییر می باشند و ممکن است نا خواسته تغییر یابند . برای جلوگیری از تغییرات نا خواسته ، آنها را به صورت ثابت به تابع ارسال می کنیم . به برنامه زیر توجه کنید :

```
void f( const int * ); // prototype

int main()
{
  int y;

  f( &y ); // f attempts illegal modification

  return 0;
}

// xPtr cannot modify the value of the variable
// to which it points
void f( const int *xPtr )
{
  *xPtr = 100; // error: cannot modify a const object
}
```

برنامه فوق هنگام کامپایل شدن پیغام خطایی مبنی بر اینکه داده ثابت قابل تغییر نمی باشد را خواهد داد .

```
Compiling CONSTP1.CPP:
Error CONSTP1.CPP 16: Cannot modify a const object
```

همانطور که مشاهده می کنید برای ارسال آرگومان توسط اشاره گر به صورت ثابت به تابع از دستور **const int *xptr** استفاده کردیم . و در تابع هنگام تغییر مقدار اشاره گر ثابت با پیغام خطا مواجه شدیم .

یکی دیگر از کاربردهای **const** همراه اشاره گرها ، ایجاد اشاره گر ثابتی به داده ای غیر ثابت می باشد . چنین اشاره گری همواره به یک خانه حافظه اشاره می کند و نمی توان آن را به خانه حافظه دیگری اشاره داد ولی می توان داده داخل همان خانه حافظه را تغییر داد . به برنامه زیر توجه کنید :

```
int main()
{
  int x, y;

  // ptr is a constant pointer to an integer that can
  // be modified through ptr, but ptr always points
  // to the same memory location.
  int * const ptr = &x;

  *ptr = 7; // allowed: *ptr is not const
  ptr = &y; // error: ptr is const;
            //        cannot assign new address

  return 0;
}
```

برنامه فوق هنگام کامپایل شدن پیغام خطایی می دهد ، مبنی بر اینکه به اشاره گر ثابت آدرس جدیدی را نمی توان نسبت داد .

```
Compiling CONSTP2.CPP:
Error CONSTP2.CPP 11: Cannot modify a const object
```

در برنامه فوق توسط دستور **int * const ptr=&x;** اشاره گر **ptr** به آدرس متغیر **x** اشاره خواهد کرد و چون از نوع ثابت تعریف شده است نمی تواند به آدرس دیگری اشاره کند اما مقداری را که ptr به آن اشاره می کند قابل تغییر است لذا با اجرای دستور ***ptr=7** خطایی رخ نمی دهد و مقدار متغیر **x** برابر ۷ می شود ولی با اجرای دستور **ptr=&y** می خواهیم اشاره گر به متغیر **y** اشاره کند و از آنجایی که **ptr** تنها باید به **x** اشاره کند با پیغام خطا مواجه می شویم .

کاربرد دیگری از **const** همراه اشاره گرها ، ایجاد اشاره گر ثابتی به داده ای ثابت می باشد . چنین اشاره گری همواره به یک خانه حافظه اشاره می کند و از طریق این اشاره گر نیز نمی توان داده داخل آن خانه حافظه را تغییر داد . به برنامه زیر توجه کنید :

```
#include <iostream.h>

int main()
{
  int x = 5, y;

  // ptr is a constant pointer to a constant integer.
  // ptr always points to the same location;
  // the integer at that location cannot be modified.
  const int *const ptr = &x;

  cout << *ptr << endl;

  *ptr = 7; // error: *ptr is const;
            //        cannot assign new value
  ptr = &y; // error: ptr is const;
            //        cannot assign new address

  return 0;
}
```

هنگام کاپایل کردن برنامه فوق با پیغام خطایی مبنی بر اینکه مقدار اشاره گر و آدرس اشاره گر را نمی توان تغییر داد ، مواجه می شویم .

```
Compiling CONSTP3.CPP:
Error CONSTP3.CPP 14: Cannot modify a const object
Error CONSTP3.CPP 16: Cannot modify a const object
```

در برنامه فوق توسط دستور **const int * const ptr = &x;** آدرس و مقدار اشاره گر **ptr** از نوع ثابت تعریف شده لذا هنگام تغییر مقدار خانه ای از حافظه که **ptr** به آن اشاره می کند توسط دستور ***ptr = 7;** با پیغام خطا مواجه می شویم و نیز هنگام تغییر آدرسی که **ptr** با آن اشاره می کند توسط دستور **ptr = &y;** پیغام خطای دیگری دریافت می کنیم .

اعمال محاسباتی با اشاره گرها

اشاره گرها عملوندهایی مجاز در عبارات محاسباتی ، و رابطه ای می باشند ، البته تمامی عملگرهایی که در اینگونه عبارات به کار می روند برای اشاره گرها مجاز نمی باشند . در این مبحث به بررسی عملگرهایی که عملوندی از نوع اشاره گر می توانند داشته باشند و نحوه کاربرد آنها می پردازیم .

یک اشاره گر می تواند افزایش (++) یا کاهش (--) یابد . یک عدد صحیح می تواند به آن اضافه (+= یا +) و یا از آن کم (-= یا -) شود .

فرض کنید که آرایه **int v[5]** تعریف شده است و اولین خانه آن در آدرس **۳۰۰۰** از حافظه قرار دارد و فرض کنید که **vptr** به خانه **v[0]** از آرایه اشاره می کند . توجه داشته باشید که برای اینکه **vptr** به آرایه **v** اشاره کند کافی است از یکی از دستورات زیر استفاده کنیم :

```
vptr = v;
vptr = &v[0];
```

در ریاضیات معمول **۲+۳۰۰۰** برابر **۳۰۰۲** می شود اما در محاسبات اشاره گرها معمولاً بدین صورت نم باشد . هنگامی که عدد صحیحی به یک اشاره گر اضافه و یا از آن کم می شود ، مقدار اشاره گر معمولاً به اندازه آن عدد زیاد یا کم نمی شود ، بلکه به اندازه طول نوع داده ای که اشاره گر به آن اشاره می کند ، زیاد یا کم می شود . به عنوان مثال دستور **vptr += 2;** حاصلش برابر **3008 (3000 + 2*4)** خواهد شد البته با فرض اینکه متغیری از نوع **int** در چهار بایت از حافظه قرار می گیرد . در آرایه **v** ، اشاره گر **vptr** حالا به خانه **v[2]** اشاره می کند .

اگر متغیری از نوع **int** در دو بایت از حافظه قرار بگیرد حاصل دستور **vptr += 2;** خانه **(3000 + 2*2)3004** می بود .

اگر **vptr** به خانه **v[4]** که در آدرس **۳۰۱۶** است اشاره کند اجرای دستور زیر :

```
vptr -= 4;
```

باعث می شود که **vptr** به خانه **3000(4*4 - 3016)** یعنی **v[0]** اشاره کند .

هر یک از دستورات زیر باعث می شود که اشاره گر به خانه بعدی آرایه **v** ، اشاره کند .

```
++vptr;
vptr++;
```

و هر یک از دستورات زیر باعث می شود که اشاره گر به خانه قبلی آرایه **v** ، اشاره کند .

```
--vptr;
vptr--;
```

اشاره گرهایی که به خانه های یک آرایه اشاره می کنند می توانند از یکدیگر کم شوند . به عنوان مثال اگر **vptr** حاوی **۳۰۰۰** (یعنی به خانه**v[0]** اشاره کند) و **v2ptr** حاوی **۳۰۰۸** (یعنی به خانه**v[2]** اشاره کند) دستور زیر :

```
x=v2ptr - vptr;
```

تعداد خانه های بین **vptr** تا **v2ptr** را در **x** قرار می دهد که در اینجا **x** حاوی **۲** می گردد . محاسبات اشاره گرها تنها هنگامی که اشاره گرها به خانه های یک آرایه اشاره می کنند ، معنا دار است . همچنین مقایسه اشاره گرها نیز هنگامی مفهوم دارد که به خانه های یک آرایه اشاره کنند به عنوان مثال مقایسه دو اشاره گر می تواند نشان دهد که یکی از اشاره گرها به خانه ای با اندیس بزرگتر نسبت به اشاره گر دیگر ، اشاره می کند .

ارتباط اشاره گرها و آرایه ها

آرایه ها و اشاره گرها در زبان **C++** ارتباط نزدیکی با یکدیگر داشته و تقریباً می توان آنها را به جای یکدیگر به کار برد نام آرایه را می توان به عنوان یک اشاره گر ثابت در نظر گرفت و تمام اعمالی که توسط اندیس آرایه می توان انجام داد توسط اشاره گر نیز قابل انجام است و از طریق اشاره گر نیز می توان به تک تک عناصر آرایه دست یافت . دستور زیر را در نظر بگیرید :

```
int b[5];
int bptr;
```

از آنجا که نام آرایه (بدون اندیس) اشاره گری به عنصر اول آرایه می باشد ، می توانیم اشاره گر **bptr** را توسط دستور زیر، به اولین عنصر آرایه **b** اشاره دهیم :

```
bptr = b;
```

و یا می توانیم از دستور زیر برای اشاره دادن **bptr** به عنصر اول آرایه **b** استفاده کنیم :

```
bptr = &b[0]
```

برای دستیابی به عنصر **b[3]** توسط اشاره گر **bptr** که توسط یکی از دستورهای فوق به آرایه **b** مرتبط شد، می توان از دستور زیر استفاده کرد :

```
*(bptr + 3)
```

در مبحث قبل با مفهوم **bptr+3** و عباراتی از این قبیل آشنا شده اید. از آنجایی که **bptr** به عنصر اول آرایه اشاره می کند پس حاوی آدرس عنصر اول آرایه یعنی **b[0]** می باشد، لذا **bptr+3** آدرس خانه **b[3]** خواهد بود پس

***(bptr + 3)** به خانه **b[3]** اشاره خواهد کرد . توجه داشته باشید که استفاده از پرانتز در اینجا اجباری می باشد ، چون عملگر * اولویت بالاتری نسبت به عملگر + دارد . اگر دستور فوق را بدون پرانتز به کار ببریم یعنی از ***bptr+3** استفاده کنیم عدد ۳ به خانه **b[0]** اضافه می گردد .

توجه داشته باشید که ***(b + 3)** (در اینجا **b** نام آرایه ای می باشد که در دستورات فوق تعریف گردید) نیز به خانه **b[3]** از آرایه اشاره می کند ، چون همانطور که در ابتدای این بحث گفته شد نام آرایه همانند یک اشاره گر ثابت می باشد .

اشاره گرها را نیز می توان مانند آرایه ها اندیس دار کرد ، به عنوان مثال **bptr[1]** به عنصر **b[1]** رجوع خواهد کرد چون **bptr** اشاره گری به آرایه**b** می باشد .

نکته : همانطور که می دانید نام آرایه ، اشاره گر ثابتی می باشد لذا دستوری مانند **b+=3;** برای آرایه ای با نام **b** قابل استفاده نمی باشد ، چون اشاره گر ثابت همواره به یک خانه از حافظه اشاره می کند .

برای درک نحوه ارتباط و شباهت آرایه ها و اشاره گرها ، به برنامه زیر که در آن عناصر آرایه ای توسط چهار روش متفاوت در خروجی چاپ می شوند ، توجه کنید :

```
#include <iostream.h>

void main()
{
  int b[] = { 10, 20, 30, 40 };
  int *bPtr = b;
  int i;

  cout << "b[i]:\n";
  for ( i = 0; i < 4; i++ )
    cout << "b[" << i << "] = " << b[ i ] << '\n';

  cout << "\n*(b + i):\n";
  for ( i = 0; i < 4; i++ )
    cout << "*(b + " << i << ") = "
         << *( b + i ) << '\n';


  cout << "\nbPtr[i]:\n";
  for ( i = 0; i < 4; i++ )
    cout << "bPtr[" << i << "] = " << bPtr[ i ] <<'\n';

  cout << "\n*(bPtr + i):\n";
  for ( i = 0; i < 4; i++ )
    cout << "*(bPtr + " << i << ") = "
         << *( bPtr + i ) << '\n';
```

```
}
```

خروجی برنامه فوق به صورت زیر می باشد :

```
b[i]:
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40

*(b + i):
*(b + 0) = 10
*(b + 1) = 20
*(b + 2) = 30
*(b + 3) = 40

bPtr[i]:
bPtr[0] = 10
bPtr[1] = 20
bPtr[2] = 30
bPtr[3] = 40

*(bPtr + i):
*(bPtr + 0) = 10
*(bPtr + 1) = 20
*(bPtr + 2) = 30
*(bPtr + 3) = 40
```

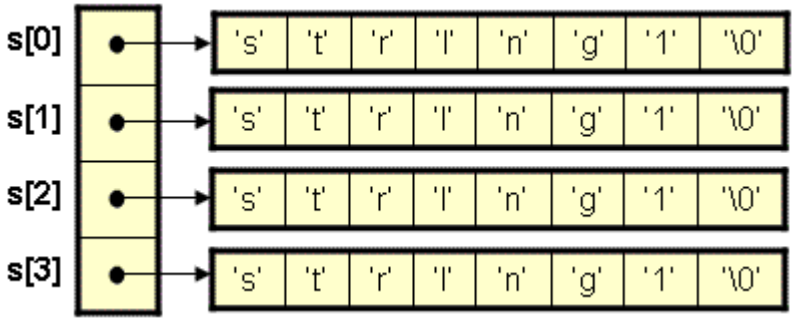
آرایه ای از اشاره گرها

عناصر یک آرایه می توانند اشاره گرها باشند . یکی از کاربردهای چنین آرایه ای ساختن آرایه ای شامل رشته هایی از حروف می باشد . هر عنصر چنین آرایه ای یک رشته از حروف می باشد که این رشته از حروف توسط اشاره گری به اولین حرف رشته مشخص می شود . به دستور زیر توجه کنید :

```
const char *s[4] =
 { "string1","string2","string3","string4" }
```

دستور فوق آرایه ای چهار عنصری شامل رشته هایی از حروف ایجاد می کند. **char*** در دستور فوق مشخص می کند که هر عنصر آرایه **s** ، اشاره گری به داده ای از نوع **char** می باشد . چهار مقداری که در آرایه قرار می گیرند **"string1"** و **"string2"** و **"string3"** و **"string4"** می باشند . انتهای هر کدام از این رشته ها که در حافظه قرار می

گیرند با کاراکتر پوچ مشخص می شود ، لذا طول این رشته ها یک واحد بیشتر از تعداد حروفی می باشد که بین " " قرار دارد . در این دستور طول هر یک از رشته ها هشت حرف می باشد . اگر چه به نظر می رسد که این رشته از حروف درآرایه قرار می گیرند ولی در واقع اشاره گرهایی به اولین حرف هر یک از این رشته ها در آرایه قرار دارد . به شکل زیر توجه کنید :



اگر چه آرایه طول ثابتی دارد ، اما این شیوه ما را قادر می سازد که به رشته هایی با طول نا مشخص دسترسی پیدا کنیم . این انعطاف پذیری مثالی از توان زبان **C++** در ایجاد ساختمان های داده ای می باشد .

نکته : توجه داشته باشید که آرایه ای از حروف را می توان توسط یک آرایه دو بعدی از نوع **char** نیز ایجاد کرد که هر سطر حاوی یک رشته و هر ستون حاوی یک حرف از یک رشته می باشد . در چنین حالتی تعداد ستون ها در هر سطر باید عددی ثابت باشد و این عدد باید برابر با طول بزرگترین رشته باشد ، لذا هنگامی که رشته های زیادی از حروف داریم و طول اکثر رشته ها از طول بزرگترین رشته کمتر می باشند ، مقدار زیادی از خانه های حافظه به هدر می رود . به شکل زیر توجه کنید :

'R'	'e'	'd'	'\0'				
'Y'	'e'	'l'	'l'	'o'	'w'	'\0'	
'H'	'o'	't'	'p'	'i'	'n'	'k'	'\0'
'G'	'r'	'a'	'y'	'\0'			

اشاره گر به تابع

یک اشاره گر به تابع حاوی آدرس آن تابع در حافظه می باشد . همانطور که می دانید نام یک آرایه در واقع آدرس اولین عنصر آرایه در حافظه می باشد. مشابهاً ، نام یک تابع ، آدرس ابتدای کدهای یک تابع ، در حافظه می باشد . اشاره گر به یک تابع می تواند به عنوان آرگومان به توابع ارسال شود ، به عنوان خروجی تابعی برگردانده شود، در آرایه قرار گیرد و یا به تابعی دیگر اشاره داده شود.