دانشگاه آزاد اسلامی واحد رودهن
دانشکده فنی و مهندسی
گروه مهندسی کامپیوتر

جزوه درس برنامه سازی پیشرفته

مریم جندقیان

Mm.Jandaghian@Gmail.com

زبان برنامه نویسی C و C++

انواع داده ای در C++

نوع داده	میزان فضای اشغالی	محدوده	توضیحات
char	۱ بایت	-128 to 127	کاراکتر
int	۴ بایت	-2147843648 to 2147843647	عدد صحیح
Short int	۲ بایت	-32768 to 32767	
long	۸ بایت	±9223372036854775808	
float	۴ بایت	دقت ۷ رقم اعشار	عدد اعشاری
double	۸ بایت	دقت ۱۵ رقم اعشار	
long double	۸ بایت	دقت ۱۹ رقم اعشار	
bool	۱ بایت	True یا false	درست یا غلط
void	۱ بایت	تهی	هیچ (پوچ)

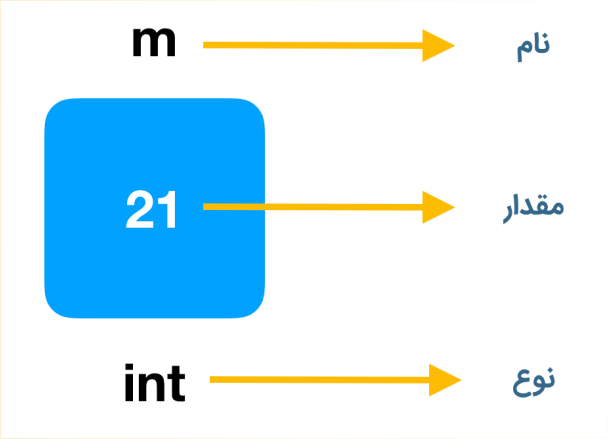
نکته مهم: میزان فضای اشغالی و محدوده بر اساس نوع کامپایلر و نوع سیستم متفاوت میباشد.

متغیرها

متغیر مکانی از حافظه را نشان میدهد که میتواند مقادیر مختلف بگیرد.

نکته ۱: در C هر متغیر قبل از استفاده باید تعریف شود.یعنی باید نوع داده ای که قرار است در متغیر قرار گیرد تعیین شود .

نکته ۲: وقتی مقدار جدید در متغیری ریخته میشود، مقدار قبلی آن حذف میشود.



نکته ۳: نام متغیر به دلخواه برنامه نویس انتخاب میشود.

نکته ۴: نام متغیر از ترکیب حروف و ارقام تشکیل میشود، ولی نمیتواند با رقم شروع شود.

نکته ۵: اگرچه نام متغیر دلخواه میباشد، ولی نمیتواند کلمه کلیدی باشد. کلمه کلیدی مانند if یا for و ..

~~1test~~
~~for~~

تعریف متغیرها

آخر تمام دستورات C سیمیکالن (;) قرار دارد مگر در مواردی که گفته خواهد شد.

فرمت کلی تعریف متغیرها:

نوع متغیر [اسامی متغیرها];

```
int i;
```

i از نوع عدد صحیح تعریف شده است.

```
int k,j;
```

k,j از نوع عدد صحیح تعریف شده است.

```
float h1;
```

h1 از نوع عدد اعشاری تعریف شده است.

```
char c,s;
```

c,s از نوع کاراکتر تعریف شده است.

نحوه مقداردهی به متغیرها

۱-در هنگام تعریف متغیر

```
int i=12;
int i=12 , j=13;
char ch='A';
```

نکته: کاراکتر داخل ' ' قرار میگیرد.

```
char ch=65;
```

۶۵ کد اسکی کاراکتر A میباشد.

۲-در خلال برنامه

```
int a,b,c;
…
a=10;
b=20;
c=a+b;
```

انواع عملگرها

۱-عملگرهای محاسباتی

+ , - , * , / , %

%باقیمانده تقسیم را محاسبه میکند.

++ , --

i++ معادل i=i+1 است و --i معادل i=i-1 است.

۲-عملگرهای رابطه‌ای

> , >= , < , <=

== , !=

i=10 و i==10 چه تفاوتی دارند؟

۳-عملگرهای منطقی

! , && , ||

! معادل not و && معادل and و || معادل or میباشد.

فرمت اصلی برنامه در C:

```
main()
{
  دستورات
}
```

تابع **main** تابع اصلی برنامه است و اولین خطی است که برنامه از آنجا شروع میشود.

دستور cout

برای نمایش اطلاعات در خروجی به کار میرود.

```
main()
{
Cout<< " hello" ;
}
```

نکته مهم: آنچه داخل " " نوشته میشود عینا در خروجی چاپ میگردد.

نکته: دستور **endl** مکان نما را به سر خط بعد منتقل میکند.

```
main()
{
Char ch='c';
Cout<<ch;
Cout<<endl;
Cout<<ch;

}
```

خروجی:

```
c
c
```

```
main()
{
Cout<<6%8<<endl<<9%8<<endl;
}
```

خروجی:

```
6
1
```

```
main()
{
int v1,v2;
v1=20;
v2=v1+10;
Cout<<"v1+10 is";
Cout<<v2<<endl;
}
```

خروجی:

```
v1+10 is 30
```

دستور cin

برای دریافت اطلاعات از ورودی استفاده میشود.

Cin>>نام متغیر ;

مثال: برنامه ای بنویسید که طول یک ضلع مربع را از کاربر دریافت کرده و مساحت آن را در خروجی نمایش دهد.

```
main()
{
float A,B;
Cout<<"toole zele moraba ra vared konid:";
Cin>>A;
B=A*A;
Cout<<"masahat="<<B;
}
```

خروجی:

```
toole zele moraba ra vared konid:6
masahat=36
```

دستورات پیش پردازنده

دستور پیش پردازنده فرمانی برای کامپایلر است. دستور پیش پردازنده به کامپایلر دستور میدهد که فایل دیگری را در فایل مبدا ضمیمه کند.

مثلا دستور پیش پردازنده زیر اجازه استفاده از دستور **cin** و **cout** را میدهد.

```
#include <iostream.h>
main()
{
float A,B;
Cout<<"toole zele moraba ra vared konid:";
Cin>>A;
B=A*A;
Cout<<"masahat="<<B;
}
```

نکته: **Conio.h** اجازه استفاده از توابع رشته ای و **math.h** اجازه استفاده از توابع ریاضی را به کامپایلر میدهد.

ساختارهای تصمیم

برای تصمیم گیری به کار میروند. در صورت وجود شرایطی خاص دستورالعملهای مشخصی را اجرا میکنند.

۱- ساختار تصمیم if

```
if  (شرط)
        دستور ;
.....
if  (شرط)
{
        دستور ۱ ;
        دستور ۲ ;
        ...
        دستور n ;
}
```

مثال

مثال: برنامه ای بنویسید که یک عدد صحیح را از کاربر دریافت کرده و اگر عدد از ۱۰۰ بزرگتر باشد پیغام مناسب بدهد.

```
main()
{
  int x;
  Cout<< "enter a number:";
  Cin>>x;
  If (x>100)
  {
     Cout<<"the number"<<x;
     Cout<<"is greater than 100";
  }
}
```

```
main()
{
  int x;
  Cout<< "enter a number:";
  Cin>>x;
  If (x>100)
     Cout<<"the number"<<x<<"is greater than 100";
}
```

۲- ساختار تصمیم if-else

```
if  (شرط)
        دستور۱ ;
Else
      دستور۲ ;
.....
if  (شرط)
{
        دستورات;
        ...
}
else
{
        دستورات;
        ...
}
```

مثال

مثال: برنامه ای بنویسید که یک عدد صحیح را از کاربر دریافت کرده و اگر عدد از ۱۰۰ بزرگتر باشد پیغام مناسبی و در غیر اینصورت پیغام مناسب دیگری بدهد.

```
main()
{
  int x;
  Cout<< "enter a number:";
  Cin>>x;
  If (x>100)
  {
    Cout<<"the number"<<x;
    Cout<<"is greater than 100";
  }
 else
 {
  Cout<<"the number"<<x;
  Cout<<"is smaller than 100";
  }
}
```

```
main()
{
  int x;
  Cout<< "enter a number:";
  Cin>>x;
  If (x>100)
     Cout<<"the number"<<x<<"is greater than 100";
else
     Cout<<"the number"<<x<<"is smaller than 100";
}
```

۳- ساختار تصمیم else-if

```
if  (شرط۱)
{
        دستور;
        ...
}
Else if  (شرط۲)
{
        دستورات;
        ...
}
...
Else
{
        دستورات;
        ...
}
```

مثال

■مثال: برنامه ای بنویسید که یک عدد صحیح را از کاربر دریافت کرده و اگر عدد صفر باشد، پیغام مناسب واگر از صفر بزرگتر باشد، پیغام مناسب دیگر و اگر از صفر کوچکتر باشد پیغام مناسب دیگری در خروجی نمایش دهد.

```
main()
{
  int x;
  Cout<< "enter a number:";
  Cin>>x;
  if (x==0)
      Cout<<"the number"<<x<<"is zero";
  else If (x<0)
     Cout<<"the number"<<x<<"is smaller than zero";
  else
     Cout<<"the number"<<x<<"is greater than zero";
}
```

ساختارهای تکرار:

۱–ساختار تکرار for

حلقه تکرار for باعث میشود بخشی از برنامه به تعداد بار معینی تکرار شود.

```
for ( مقدار دهی اولیه ; شرط حلقه ; گام حرکت )
        دستور;
```

.....

```
for ( مقدار دهی اولیه ; شرط حلقه ; گام حرکت )
{
    دستورات;
        ...
}
```

مثال:

مثال۱: چاپ اعداد ۱ تا ۱۰۰	مثال۲: چاپ میانگین اعداد ۱ تا ۱۰۰	مثال۳: چاپ ماکزیمم ۱۰۰ عدد دریافتی از کاربر

main()
{
int i;
For (i=1 ; i<=100 ; i++)
Cout<< i <<" ";
}
 	
1 2 3 4 … 100
	main()
	{
	int i,s;
	s=0;
	For (i=1 ; i<=100 ; i++)
	s=s+i;
	Cout<<"average is"<< s/100;
	}
	
	1+2+3+4+5+6+ …+100
	
	s=o
	i=1
	s=s+i=0+1=1
	i=2
	s=s+i=1+2=3
	i=3
	s=s+i=3+3=6
	…
		main()
		{
		int i,A,maxi;
		cin>>A;
		maxi=A;
		For (i=2 ; i<=100 ; i++)
		{
		Cin>>A;
		If (A>maxi)
		maxi=A;
		}
		Cout<<"maximum is"<< maxi;
		}
		12 18 13 16 50 …
		
		12
		maxi=12
		18
		maxi=18
		13
		maxi=18
		…

مثال

مثال: برنامه ای بنویسید که یک عدد صحیح را از کاربر دریافت کرده و فاکتوریل آن را در خروجی چاپ کند.
5!=1*2*3*4*5=120

```
main()
{
  int i , num , fact;
  Cout<< "enter a number:";
  Cin>>num;
  fact=1;
  for (i=1 ; i<=num ; i++)
    fact=fact * i;
  Cout<<"the factorial is"<<fact<<endl;
}
```

```
5!=1*2*3*4*5=120


fact=1
i=1
fact=fact*i=1*1=1
i=2
fact=fact*i=1*2=2
i=3
fact=fact*i=2*3=6
i=4
fact=fact*i=6*4=24
i=5
fact=fact*i=24*5=120
```

مثال

- مثال: برنامه ای بنویسید که عدد صحیح n را از کاربر دریافت کرده و n جمله اول سری فیبوناچی را در خروجی چاپ کند.
- در سری فیبوناچی هر جمله مجموع دو جمله قبل میباشد و دو جمله اول آن ۱ و ۱ میباشد.

1,1,2,3,5,8,13,...

```
main()
{
  int i,n,a,b,c;
  Cout<< "enter a number:";
  Cin>>n;
  a=b=1;
  Cout<<a<<“ , "<<b;
  for (i=3 ; i<=n ; i++)
  {
    c=a+b;
    Cout<<"  "<<c;
    a=b;
    b=c;
  }
}
```

1 , 1 , 2 , 3 , 5 , 8 , 13,...

a b c

a b c

. . .

a=1

b=1

c=a+b=1+1=2

a=1

b=2

c=a+b=1+2=3

a=2

b=3

c=a+b=2+3=5

.

.

.

حلقه for تو در تو

مثال: برنامه ای بنویسید که جدول ضرب اعداد ۱ تا ۱۰ را تولید کند.

عملگر setw(n) باعث میشود به اندازه n کاراکتر فضا برای خروجی در نظر گرفته شود و خروجی بصورت تراز شده نمایش داده شود.

سرفایل مربوط به آن iomanip.h میباشد.

```
main()
{
  Int i,j;
  For (i=1 ; i<=10 ; i++)
  {
    For (j=1 ; j<=10 ; j++)
      Cout<<setw(8)<<i*j;
    Cout<<endl;
  }
}
```

```
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
Press any key to continue
```

۲-ساختار تکرار while

حلقه تکرار while باعث میشود بخشی از برنامه تا زمانی که شرطی برقرار است تکرار شود.

```
While (شرط حلقه )
    دستور;

.....

While (شرط حلقه )
{
    دستورات;
    ...
}
```

نکته: با استفاده از دستور **while** می توان حلقه هایی نوشت که در آنها تعداد تکرار از قبل مشخص نیست. برخلاف **for** که معمولا در آن تعداد تکرار از قبل مشخص است.

مثال

مثال ۱:
تا زمانیکه کاربر ۰ وارد کند، مرتبا از کاربر ورودی میگیرد.

```
main()
{
  int n;
  n=99;
  While (n!= 0)
    Cin>>n;
}
```

مثال ۲:
چاپ اولین توان ۲ بزرگتر از ۱۰۰۰ یعنی ۱۰۲۴

```
main()
{
Int P;
P=2;
While (P<=1000)
  P=2*P;
cout<<P;
}
```

```
P=2
P=2*P=2*2=4
P=2*P=2*4=8
P=2*P=2*8=16
P=2*P=2*16=32
P=2*P=2*32=64
P=2*P=2*64=128
P=2*P=2*128=256
P=2*P=2*256=512
P=2*P=2*512=1024
```

دستور getch()

* برای خواندن یک کاراکتر از ورودی به کار میرود.
* سرفایل مربوط به ان conio.h است.

```
main()
{
  char ch;
  Cout<<"\n enter a character:";
  ch=getch();
  Cout<<"character is:"<<ch;
}
```

نکته: در بعضی از ورژنهای نرم افزار c و c++ لازم است در انتهای برنامه و بعداز دستور cout از دستور getch() استفاده شود تا فرصت دیدن جواب به کاربر داده شود. پس از فشردن یک دکمه برنامه متوقف میشود.

مثال

مثال ۱: برنامه ای بنویسید که تعداد کاراکترها ی یک جمله را تا رسیدن به انتهای جمله شمارش کند. انتهای جمله با نقطه مشخص شده است.

```
main()
{
int i;
  i=0;
  While (getch() != '.')
     i++;
  Cout<<i ;
}
```

I am a student.
14

مثال ۲: برنامه ای بنویسید که تعداد کاراکترها و اسپیس های یک جمله را تا رسیدن به انتهای جمله شمارش کند. انتهای جمله با نقطه مشخص شده است.

```
main()
{
  char ch;
   int spacecount=0 , charcount=0;
  While (ch=getch() != '.')
  {
    if (ch==' ')
      spacecount++;
    else
      charcount++;
  }
  Cout<<"spacecount="<<spacecount ;
  Cout<<"charcount="<<charcount ;
}
```

I am a student.
spacecount=3
charcount=11

۲-ساختار تکرار Do_while

حلقه تکرار Do_while مانند while عمل میکند. با این تفاوت که ابتدا حداقل یکبار دستورات حلقه اجرا میشود و بعد شرط چک میشود.

```
Do
{
    دستورات;
    ...
} While (شرط حلقه) ;
```

مثال

مثال : برنامه ای بنویسید که تا زمانیکه کاربر مایل است از کاربر دو عدد دریافت کرده و باقیمانده تقسیم آنها را در خروجی نمایش دهد.

```
main()
{
  int A,B;
  Char ch;
  Do
  {
    Cout<<"enter A and B";
    Cin>>A>>B;
    Cout<<"remainder is:"<<A%B;
    Cout<<"\n Do another?(y/n) ";
    Cin>>ch;
  }while (ch != 'n');
}
```

```
enter A and B
14
6
remainder is: 2
Do another?(y/n) y
.
.
.

enter A and B
22
3
remainder is: 1
Do another?(y/n) n
```

مثال

مثال : برنامه ای بنویسید که یک عدد را از کاربر دریافت کرده و وارون آن را در خروجی نمایش دهد. مثلا وارون ۷۶۳۱ میشود ۱۳۶۷
این برنامه باید برای صفر و هر عدد بزرگتر از آن درست عمل کند.

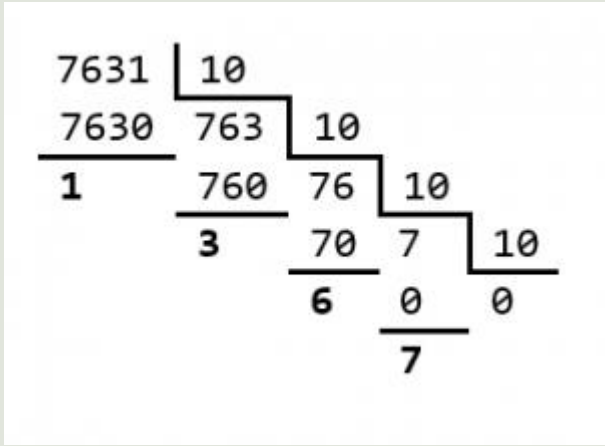
```
main()
{
  Int number , digit;
  Cout<<"enter a number:";
  Cin>>number;
  Do
  {
    digit=number%10;
    Cout<<digit;
    number=number/10;
  }while (number != 0);
}
```

۷۶۳۱/۱۰
باقیمانده= ۱
خارج قسمت= ۷۶۳

۷۶۳/۱۰
باقیمانده= ۳
خارج قسمت=۷۶

۷۶/۱۰
باقیمانده= ۶
خارج قسمت=۷

۷/۱۰
باقیمانده= ۷
خارج قسمت=۰



توابع

تابع

- برای آنکه برنامه های بزرگ قابل مدیریت باشند، برنامه نویسان این برنامه ها را به زیربخشهایی بنام تابع تقسیم میکنند که هر زیربخش یک وظیفه مشخص دارد.
- مهمترین دلیل استفاده از تابع سازماندهی برنامه و کاستن از طول برنامه است.
- مجموعه دستورات تابع تنها در یک مکان از حافظه کامپیوتر ذخیره میشوند حتی اگر تابع چند بار در برنامه اجرا شود.

انواع توابع

۱-توابع کتابخانه ای

۲-توابع ساخت کاربر

۱-توابع کتابخانه ای

کتابخانه C++ مجموعه ای است شامل توابع از پیش تعریف شده که از طریق سرفایلهای خاص قابل دسترسی میباشند.

▪مثال: تابع sqrt() جذراعداد را از ۱ تا ۶ در خروجی نمایش میدهد.

```
#include < math.h >
#include < iostream.h >
main()
{
  int i;
  for (i=1 ; i<=9 ; i++)
  {
    Cout<<"\t "<<i<<"\t "<<sqrt(i)<<endl;
  }
}
```

```
1    1
.
.
.
9    3
```

۲-توابع ساخت کاربر

ساختار تابع

```
(پارامترهای ورودی)     <اسم تابع>   <نوع خروجی تابع>
{
        بدنه تابع
}
```

نکته: زمانیکه در برنامه اصلی نام تابعی ذکر شود، به اصطلاح تابع فراخوانی یا call میشود.

نکته: اگر تابع مقداری را برنگرداند، نوع خروجی تابع void نوشته میشود.

نحوه استفاده از تابع در ساختار برنامه

مثال: تابع **starline** با هر فراخوانی ۲۸کارکتر ستاره در خروجی تولید میکند.

- نحوه استفاده از تابع در ساختار برنامه به دو فرمت میباشد که هر دو قابل قبول است.

استفاده از تابع در ساختار برنامه با فرمت ۱	استفاده از تابع در ساختار برنامه با فرمت ۲

Void starline()
{
int i;
For (i=0 ; i<28 ; i++)
cout<<'*';
Cout<<endl;
}
main()
{
starline();
Cout<<"data type:"<<endl;
Starline();
Cout<<"char"<<endl;
Cout<<"int"<<endl;
Cout<<"float"<<endl;
starline();
}
	Void starline();
	main()
	{
	starline();
	Cout<<"data type:"<<endl;
	Starline();
	Cout<<"char"<<endl;
	Cout<<"int"<<endl;
	Cout<<"float"<<endl;
	starline();
	}
	Void starline()
	{
	int i;
	For (i=0 ; i<28 ; i++)
	cout<<'*';
	Cout<<endl;
	}

****

data type:

****

char

int

float

****

آرگومانها

آرگومانها متغیرهایی هستند که از برنامه اصلی به تابع فرستاده میشود.

روشهای انتقال آرگومانها به تابع

۱–انتقال مقادیر ثابت

۲–انتقال متغیرها

انتقال آرگومانها به تابع :: انتقال مقادیر ثابت

مثال: تابع **repchar** با هر فراخوانی به تعداد مشخص شده کاراکتر مشخصی را که در آرگومان تابع به صورت مقدار ثابت آورده شده، در خروجی تولید میکند.

استفاده از تابع در ساختار برنامه با فرمت ۱	استفاده از تابع در ساختار برنامه با فرمت ۲

Void repchar (char ch, int n)
{
int i;
For (i=0 ; i<n ; i++)
cout<<ch;
Cout<<endl;
}
main()
{
repchar ('.',45);
Cout<<"data type"<<endl;
repchar ('*',20);
Cout<<"char"<<endl;
Cout<<"int"<<endl;
Cout<<"float"<<endl;
repchar ('-',10);
}
	Void repchar (char ch, int n);
	main()
	{
	repchar ('.',45);
	Cout<<"data type"<<endl;
	repchar ('*',20);
	Cout<<"char"<<endl;
	Cout<<"int"<<endl;
	Cout<<"float"<<endl;
	repchar ('-',10);
	}
	Void repchar (char ch, int n)
	{
	int i;
	For (i=0 ; i<n ; i++)
	cout<<ch;
	Cout<<endl;
	}

انتقال آرگومانها به تابع :: انتقال متغیرها

مثال: تابع **repchar** با هر فراخوانی به تعداد مشخص شده کاراکتر مشخص را که در آرگومان تابع به صورت متغیر آورده شده، در خروجی تولید میکند.

استفاده از تابع در ساختار برنامه با فرمت ۱

```
Void repchar (char ch, int n)
{
int i;
For (i=0 ; i<n ; i++)
  cout<<ch;
Cout<<endl;
}
main()
{
  char chin;
  int nin;
  Cout<<"enter a character:";
  Cin>>chin;
  Cout<<"enter number of times to repeat it:";
  Cin>>nin;
  repchar(chin,nin);
}
```

استفاده از تابع در ساختار برنامه با فرمت ۲

```
Void repchar (char ch, int n);
main()
{
  char chin;
  int nin;
  Cout<<"enter a character:";
  Cin>>chin;
  Cout<<"enter number of times to repeat it:";
  Cin>>nin;
  repchar(chin,nin);
}
Void repchar (char ch, int n)
{
int i;
For (i=0 ; i<n ; i++)
  cout<<ch;
Cout<<endl;
}
```

بازگرداندن مقدار به تابع اصلی

دستور return مقداری را به جایی که تابع فراخوانی شده برمیگرداند.

مثال: برنامه ای بنویسید که تا زمانیکه کاربر ۰ را وارد نکند، مرتبا توان ۳ عدد وارد شده توسط کاربر را با استفاده از یک تابع نمایش دهد.

```
int cuben (int x)
{
  return (x*x*x);
}
main()
{
  int n=1;
  while (n!=0)
  {
    cin>>n;
    Cout<<"cube="<<cuben (n)<<endl;
  }
}
```

مثال: برنامه ای بنویسید که میانگین n عدد اعشاری وارد شده توسط کاربر را در خروجی نمایش دهد. کاربر باید اول n یعنی تعداد اعداد را وارد کند.

```
float  average (int n)
{
   int i;
   float x,sum=0;
   for (i=0 ; i<n ; i++)
  {
     cin>>x;
     sum=sum+x;
   }
return (sum/n);
}
void main()
{
  int number;
  cin>>number;
  cout<<average(number);
}
```

مثال

مثال: برنامه ای بنویسید که دو عدد اعشاری را از کاربر دریافت کند و با استفاده از تابعی دو عدد را با هم جمع کند و حاصل را در خروجی نمایش دهد.

```
float  sum (float f1, float f2)
{
   float fsum;
   fsum= f1+f2;
   return fsum;
}
main()
{
   float num1, num2,num3;
   Cout<<'enter 2 number:";
   Cin>>num1>>num2;
   num3=sum(num1,num2);
   cout<<num3;
}
```

```
Void sum (float f1, float f2)
{
   float fsum;
   fsum= f1+f2;
   cout<<fsum;
}
main()
{
   float num1, num2;
   Cout<<'enter 2 number:";
   Cin>>num1>>num2;
   sum(num1,num2);
}
```

مثال

مثال: برنامه ای بنویسید که با استفاده از تابعی یک عدد صحیح را از کاربر دریافت کرده و فاکتوریل آن را در خروجی چاپ کند.

5!=1*2*3*4*5=120

```
int fact(int n)
{
  int i,temp;
  temp=1;
  for (i=1 ; i<=n; i++)
    temp=temp*i;
  return temp;
}
main()
{
  int num;
  Cout<< "enter a number:";
  Cin>>num;
  Cout<<"the factorial is"<<fact(num)<<endl;
}
```

مثال

▪مثال: برنامه ای بنویسید که عدد صحیح n را از کاربر دریافت کرده و با استفاده از تابعی n جمله اول سری فیبوناچی را در خروجی چاپ کند. در سری فیبوناچی هر جمله مجموع دو جمله قبل میباشد و دو جمله اول آن ۱ و ۱ میباشد.

1,1,2,3,5,8,13,...

```
Void fib(int n)
{
  int f1,f2,f3;
  f1=١;
  f2=1;
  cout<<f1<<"  "<<f2;
 for (i=3 ; i<=n ; i++)
  {
    f3=f1+f2;
    Cout<<"  "<<f3;
    f1=f2;
    f2=f3;
  }
}
main()
{
  int num;
  Cout<< "enter a number:";
  Cin>>num;
  fib(num);
}
```

مثال

مثال: برنامه ای بنویسید که با استفاده از تابعی تا زمانیکه کاربر سنی بین ۰ تا ۱۲۰ وارد نکند، مرتبا ورودی دریافت کند.

```
int age()
{
  int n;
  while (true)
  {
    cout<<"how old are you:";
    cin>>n;
    if (n<0)
      cout<<"\a \t you could not be negative";
    else
    if (n>120)
      cout<<"\a \t you could not be over 120";
    else
      return n;
    cout<<"\n \t try again.\n";
  }
}
main()
{
  int a;
  a=age();
  Cout<<"\n you are "<<a<<"years old.\n";
}
```

```
how old are you:123
        you could not be over 120
        try again.
how old are you:-6
        you could not be negative
        try again.
 .
 .
 .
how old are you:20
you are  20 years old
```

نکته: شرط کنترل حلقه **true** میباشد و این حلقه به ظاهر بی پایان به نظر میرسد ولی دستور **return** درون حلقه هم مقدار ورودی معتبر را به برنامه اصلی برمیگرداند و هم حلقه را خاتمه میدهد.

انواع متغیرها: متغیرهای محلی (local)

- متغیرهایی هستند که در داخل توابع تعریف شده اند و فقط در همان بلوکی که تعریف میشود قابل دسترسی میباشد.
- این متغیرها دارای ویژگی های زیر هستند.

۱-تنها در داخل تابعی که تعریف شده اند معتبرند.

۲-هنگام اجرای تابع ایجاد می شوند و هنگام خاتمه اجرا از بین می روند.

مثال

```
#include <iostream.h>
void func (int z)
{
  int y;
  y=z*z;
}
void main ()
{
  func(3);
  Cout<<y;
}
```

نکته: در تابع func ، y متغیر محلی است. در نتیجه تنها زمانیکه func در حال اجراست وجود دارد. و وقتی که اجرای func تمام می شود از بین می رود.
به همین علت دستور Cout<<y اشتباه است چون y ای وجود ندارد که آن را چاپ کند.

انواع متغیرها : متغیرهای سراسری (global)

به متغیرهایی که خارج از توابع تعریف می شوند متغیرهای سراسری یا global می گویند.

ویژگیهای متغیرهای global:

۱-برخلاف متغیرهای local، از نقطه ای که تعریف می شوند تا انتهای برنامه معتبرند.

۲-اگر هنگام تعریف مقداری به آنها داده نشود، مقدارشان صفر در نظر گرفته می شوند.

مثال

```
#include <iostream.h>
int num=4;
int  func1  (int x)
{
     num+ =x;
     return num;
}
int  func2  (int x)
{
      num- =x;
      return num;
}
void main ( )
{
   cout<<func1(3);
   cout<<func2(3);
}
```

نکته: متغیر num سراسری است. چون در خارج توابع تعریف شده. این متغیر در func1، func2 و main اعتبار دارد.

عملکرد برنامه:

- ابتدا num=4 می شود.
- func1(3) اجرا می شود. مقدار num برابر 7 شده و چاپ می شود.
- سپس func2(3) اجرا می شود. مقدار num برابر 4 شده و چاپ می شود.
- اگر به جای int num=4، تنها int num نوشته میشد، مقدار num صفردر نظر گرفته می شد.

توابع بازگشتی

توابع بازگشتی

تابع بازگشتی تابعی است که خودش، خودش را صدا میزند. این تابع برای حل مسئله، مسئله را به چند زیرمسئله کوچکتر تقسیم میکند و عمل تقسیم مسئله به زیرمسئله ها را تا وقتی انجام میدهد که به مقدار پایه ای برسد. سپس با ترکیب زیرمسئله ها، مسئله اصلی را حل میکند.

دو رکن اصلی هر تابع بازگشتی :

- خودش خودش را صدا کند.
- شرط خاتمه آن رسیدن به مقدار پایه است.

مثال

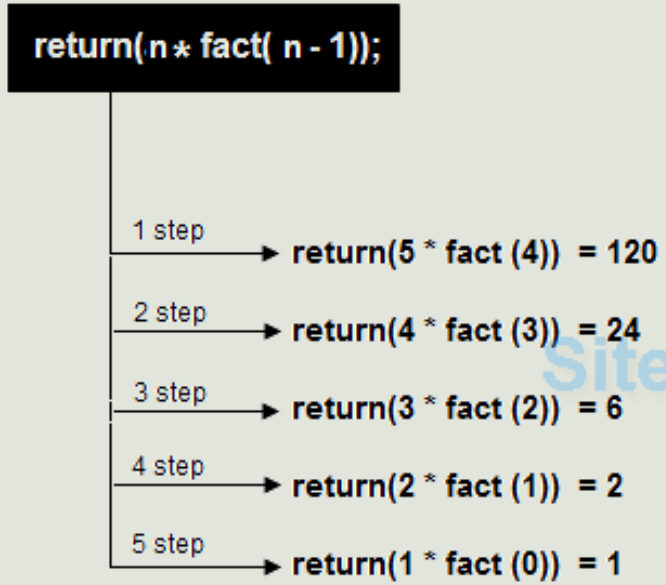
مثال: برنامه ای بنویسید که با استفاده از یک تابع بازگشتی، یک عدد صحیح را از کاربر دریافت کرده و فاکتوریل آن را در خروجی چاپ کند. 5!=1*2*3*4*5=120

```
Int fact(int n)
{
   if (n==0)
       return 1;
   else
          return (n*fact(n-1));
}
void main()
{
    int x;
    cin>>x;
    cout<<fact(x);
}
```

5!=5*4*3*2*1=5*4!
4!=4*3*2*1=4*3!
3!=3*2*1=3*2!
2!=2*1=2*1!
1!=1=1*0!
0!=1

$$n! = n*(n-1)!$$

$$fact(n) = \begin{cases} n*\ fact(n-1) & n>0 \\ 1 & n=0 \end{cases}$$



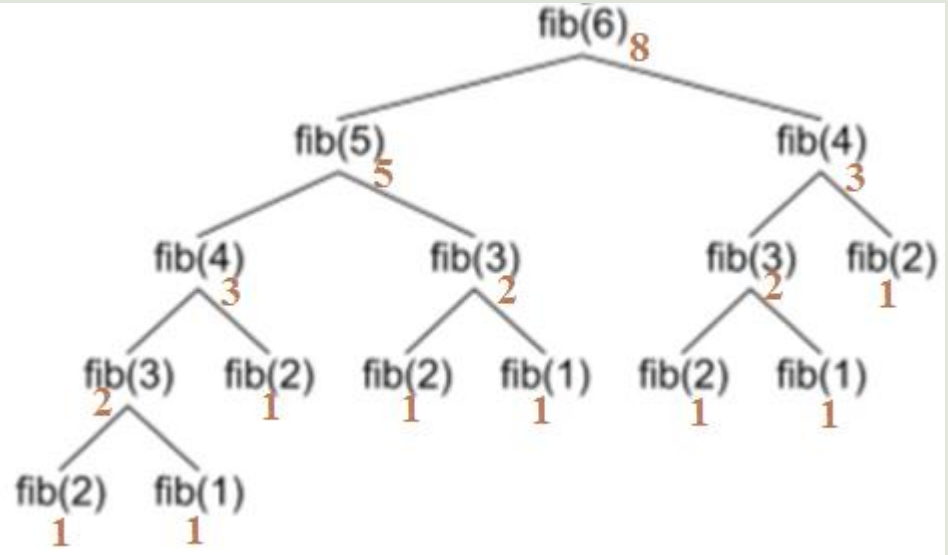
مثال

- مثال: برنامه ای بنویسید که با استفاده از یک تابع بازگشتی، جمله n ام سری فیبوناچی را در خروجی چاپ کند.
- در سری فیبوناچی هر جمله مجموع دو جمله قبل میباشد و جمله اول و دوم آن ۱ و ۱ میباشد.

1,1,2,3,5,8,13,...

```
Int fib(int n)
{
  if (n==1)
     return 1;
  else if (n==2)
     return 1;
  else
     return ( fib(n-1) + fib(n-2)  );
}
void main()
{
   int x;
   cin>>x;
   cout<<fib(x);
}
```

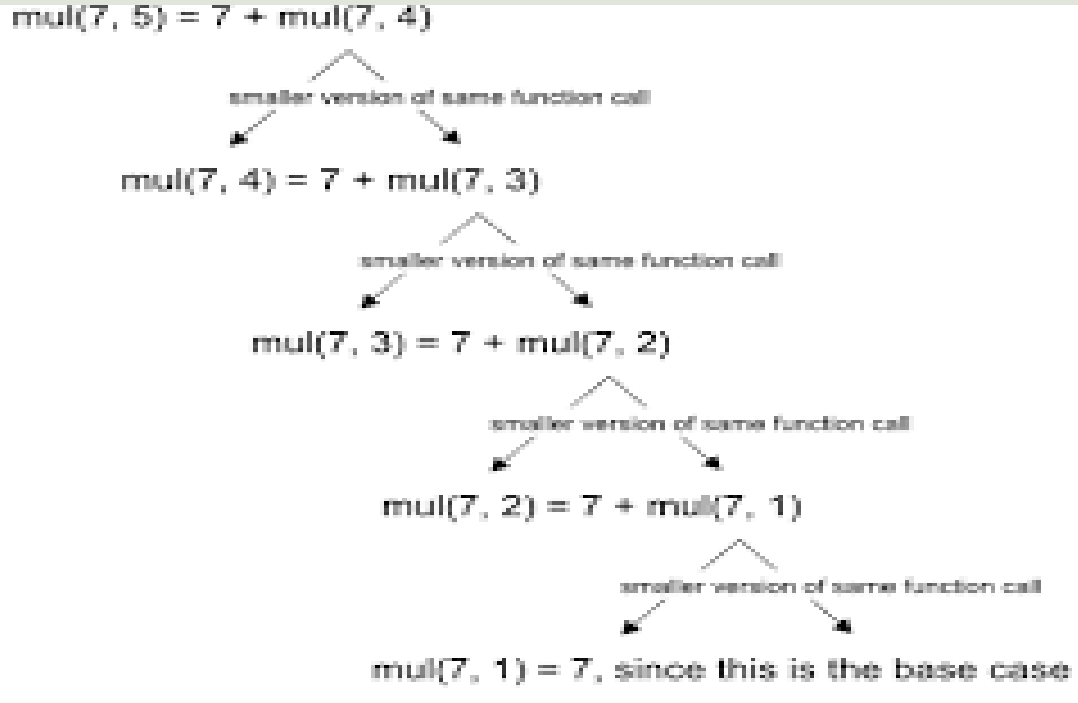
$$\text{Fib}(n) = \begin{cases} 1 & \text{if } n = 1, \\ 1 & \text{if } n = 2, \\ \text{Fib}(n-1) + \text{Fib}(n-2) & \text{otherwise} \end{cases}$$



مثال

■مثال۱: تابع بازگشتی ضرب

```
int Mul (int a , int b)
{
  if (b == 1)
    return a;
  else
    return a+Mul (a,b-1);
}
```



■مثال۲: تابع بازگشتی توان

```
int Pw(int a , int b)
{
  if (b == 1)
    return a;
  else
    return a*Pw (a,b-1);
}
```

مزایا و معایب برنامه های بازگشتی

مزایا:

- **قابل فهم تر بودن**
- **اگر قانون بازگشت به دست آورده شود نوشتن یک برنامه به صورت بازگشتی بسیار ساده تر از معادل غیر بازگشتی آن است.**

معایب:

- **مصرف حافظه بیشتری دارند و سرعت اجرای آنها هم معمولا کمتر است.** (بررسی به عنوان تحقیق)

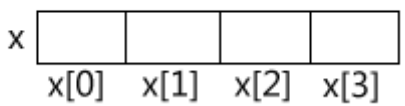
آرایه ها

آرایه ها

آرایه یک زنجیره از متغیرهای همنام است که همه از یک نوع هستند. به این متغیرها اعضای آرایه گویند. هر عضو آرایه با یک شماره مشخص میشود که این شماره اندیس آرایه است.

عناصر آرایه در خانه های پشت سر هم حافظه ذخیره میشوند.

ساختار آرایه در حافظه



نحوه تعریف آرایه:

تعریف آرایه

نوع آرایه نام آرایه [طول آرایه] ;

```
int x[4];
```

چهار مکان مجاور هم در حافظه به آرایه تخصیص داده میشود. این مکان ها به صورت x[0]، x[1]، x[2] و x[3] نامیده می شوند. یعنی آرایه ای به نام x ، از نوع int و با ۴ عنصر تعریف کرده ایم. به 0، 1، 2، 3 اصطلاحا اندیس آرایه گفته میشود.

```
float x[8];
```

1.8	3.45	1.2	3.1	10.5	12.4	13.4	23.4
0	1	2	3	4	5	6	7

دستیابی به عناصر آرایه

مثال

```
main()
{
const int size=5;
double a[size];
Cout<<"enter"<<size<<"numbers:\t";
For (int i=0 ; i<size ; i++)
   cin>>a[i];
Cout<<"reverse order:\n";
For (int i=size-1 ; i>=0 ;i - -)
   cout<<"\t"<<a[i]<<"\n";
}
```

1.8	3.45	1.2	3.1	10.5
0	1	2	3	4

reverse order:
10.5
3.1
1.2
3.45
1.8

```
main()
{
int a [3];
a[0]=12;
a[1]=22;
a[2]=17;
Cout<<"a[0]="<<a[0]<<endl;
Cout<<"a[1]="<<a[1]<<endl;
Cout<<"a[2]="<<a[2]<<endl;
}
```

Int a[3]={12,22,17};

12	22	17
0	1	2

a[0]=12
a[1]=22
a[2]=17

خطای تجاوز اندیس آرایه از محدوده تعریف شده

****

```
int a[4]={34,23,56,78};
cout<< a[4];
```

****

```
int a[4];
…
For (int i=0 ; i<7 ; i++)
   cin>>a[i];
```

****

```
int a[4]={34,23,56,78};
….
For (int i=0 ; i<10 ; i++)
   cout<<a[i];
```

****

```
int a[2]={34,23,56,78};
```

مثال

مثال۱: برنامه ای بنویسید که ۱۰۰مقدار را از کاربر دریافت کرده و آنها در یک آرایه ذخیره کند و سپس مجموع مقادیر آرایه را محاسبه کند.

```
main()
{
      int jam,i;
      int a[100];
      for (i=0 ; i<100 ; i++)
      {
          cout<<"Enter  number: ";
          cin>>a[i];
      }
      Jam=0;
      for (i=0 ; i<100 ; i++)
           Jam=a[i]+jam;
      cout<<jam;
}
```

مثال۲: برنامه ای بنویسید که نمودار میله ای افقی برای ۱۰ مقدار را رسم کند.

```
main()
{
      int a[10]={5,7,4,2,4,8,9,6,4,2};
      for (int i=0 ; i<10 ; i++)
      {
          cout<<i<<"\t"<<a[i]<<"\t";
          for (int j=0 ; j<a[i] ; j++)
               Cout<<'*';
          Cout<<endl;
      }
}
```

```
0     5     *****
1     7     *******
2     4     ****
3     2     **
4     4     ****
5     8     ********
6     9     *********
7     6     ******
8     4     ****
9     2     **
```

مثال

مثال: برنامه ای بنویسید که ۴۰ مقدار را از کاربر دریافت کرده و آنها در یک آرایه بریزد و سپس ماکزیمم مقادیر و اندیس عنصر ماکزیمم را محاسبه کند.

```
main()
{
      float x[40];
      float max;
      int max_index,i;
      for (i=0 ; i<40 ; i++)
      {
           cout<<"Enter  number: ";
          cin>>x[i];
      }
      max=x[0];
      max_index=0;
      for (i=1 ; i<40 ; i++)
       {
            if (x[i]>max)
            {
                  max=x[i];
                  max_index=i;
            }
       }
      cout<<"\nThe maximum="<<max;
      cout<<"\nmax_index="<<max_index;
}
```

1.8	3.45	1.2	. . .	13.4	23.4
0	1	2	. . .	38	39

ارسال آرایه به تابع

مثال: ارسال آرایه به تابعی که مجموع عناصر آرایه را برمی گرداند.

```
int sum(int[],int);

main()
{
   int a[] = { 11, 33, 55, 77, 44 };
   int size = sizeof(a)/sizeof(int);
   cout << "sum(a,size) = " << sum(a,size);
}

int sum(int a[], int n)
{
   int sum=0;
   for (int i=0 ; i<n ; i++)
         sum += a[i];
   return sum;
}
```

```
int sum(int a[], int n)
{
  int sum=0;
  for (int i=0; i<n; i++)
       sum += a[i];
  return sum;
}

main()
{
   int a[] = { 11, 33, 55, 77, 44 };
   int size = sizeof(a)/sizeof(int);
   cout << "sum(a,size) = " << sum(a,size);
}
```

تابع **sizeof** اندازه آرگومان ارسالی به آن را بر اساس بایت برمیگرداند.

مثال

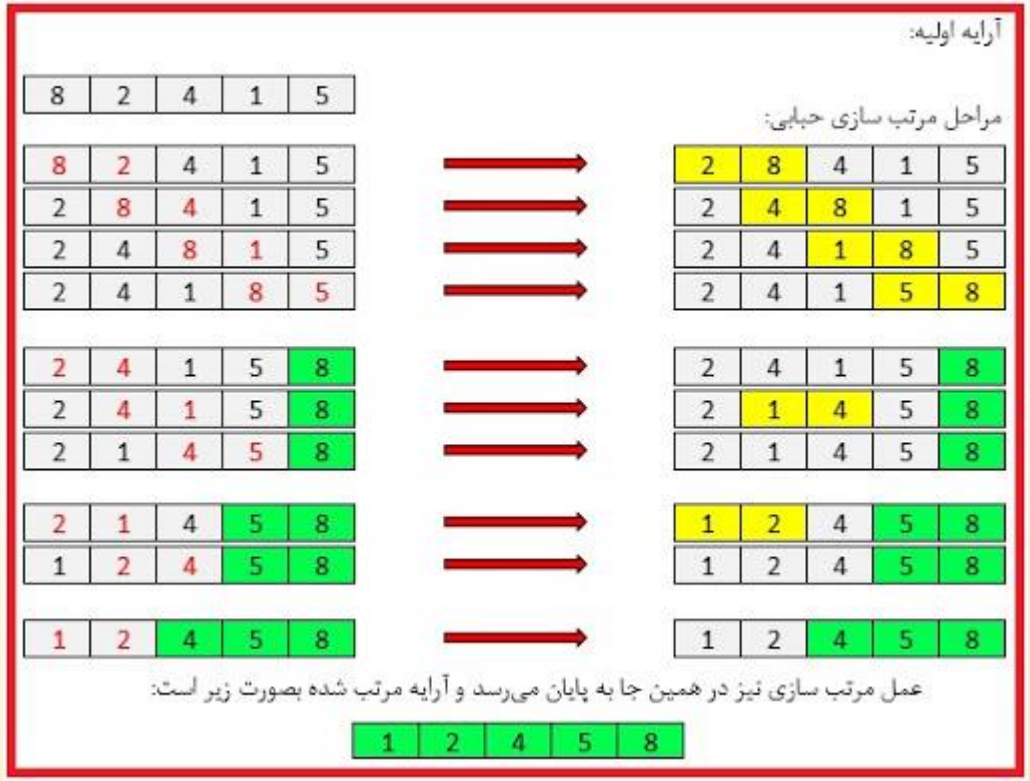
مثال: برنامهٔ زیر دارای یک تابع است که مشخص می‌نماید که آیا آرایهٔ داده شده غیر نزولی است یا خیر.

```
bool isNondecreasing(int a[], int n);

main()
{
   int a[] = { 22, 44, 66, 88, 77, 66, 55 };
   cout<<"isNondecreasing(a,4) = " << isNondecreasing(a,4)<< endl;
   cout<<"isNondecreasing(a,7) = " << isNondecreasing(a,7) << endl;
}


bool isNondecreasing(int a[], int n)
{
   for (int i=1; i<=n-1 ; i++)
      if (a[i-1]>a[i])
           return false;
   return true;
}
```

```
isNondecreasing(a,4) = 1
isNondecreasing(a,7) = 0
```

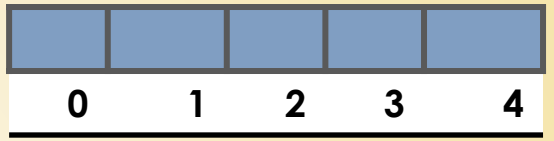
مرتب سازی حبابی

"مرتب‌سازی حبابی" یکی از ساده‌ترین الگوریتم‌های مرتب‌سازی است. در این روش، آرایه چندین مرتبه پویش می‌شود و در هر مرتبه بزرگ‌ترین عنصر موجود به سمت بالا هدایت می‌شود و سپس محدودهٔ مرتب‌سازی برای مرتبهٔ بعدی یکی کاسته می‌شود. در پایان همهٔ پویش‌ها، آرایه مرتب شده است. طریقهٔ یافتن بزرگ‌ترین عنصر و انتقال آن به بالای عناصر دیگر به این شکل است که اولین عنصر آرایه با عنصر دوم مقایسه می‌شود. اگر عنصر اول بزرگ‌تر بود، جای این دو با هم عوض می‌شود. سپس عنصر دوم با عنصر سوم مقایسه می‌شود. اگر عنصر دوم بزرگ‌تر بود، جای این دو با هم عوض می‌شود و به همین ترتیب مقایسه و جابجایی زوج‌های همسایه ادامه می‌یابد تا وقتی به انتهای آرایه رسیدیم، بزرگ‌ترین عضو آرایه در خانهٔ انتهایی قرار خواهد گرفت. حالامحدودهٔ جستجو یکی کاسته می‌شود و دوباره زوج‌های کناری یکی یکی مقایسه می‌شوند تا عدد بزرگ‌تر بعدی به مکان بالای محدوده منتقل شود. این پویش ادامه می‌یابد تا این که وقتی محدوده جستجو به عنصر اول محدود شد، آرایه مرتب شده است.



برنامه مرتب سازی حبابی

```
void sort(int[],int);
main()
{
  int a[]={8,2,4,1,5};
  sort(a,5);
 }

void sort(int a[], int n)
{
  for (int i=1; i<n ; i++)
  {
      for (int j=0; j<n-i ; j++)
      {
          if (a[j] > a[j+1])
              swap (a[j],a[j+1]);
       }
  }
}
```



i=1
j=0 j<4
i=2
j=0 j<3
i=3
j=0 j<2
i=4
j=0 j<1

تابع sort() از دو حلقهٔ تودرتو استفاده می‌کند.

۱- حلقه for داخلی زوج‌های همسایه را با هم مقایسه می‌کند و اگر آن‌ها خارج از ترتیب باشند، جای آن دو را با هم عوض می‌کند.وقتی for داخلی به پایان رسید، بزرگ‌ترین عنصر موجود در محدودهٔ فعلی به انتهای آن هدایت شده است.

۲-سپس حلقهٔ for بیرونی محدودهٔ جستجو را یکی کم می‌کند و دوباره for داخلی را راه می‌اندازد تا بزرگترین عنصر بعدی به سمت بالای آرایه هدایت شود.

تابع swap جای دو مقدار را جابجا میکند. بجای تابع swap میتوان از قطعه کد زیر استفاده کرد:

```
temp=a[j];
a[j]=a[j+1];
a[j+1]=temp;
```

برنامه جستجوی خطی

اغلب لازم است که بررسی شود آیا یک مقدار خاص درون یک آرایه موجود است یا خیر. ساده‌ترین راه این است که از اولین عنصر آرایه شروع کنیم و یکی یکی همهٔ عناصر آرایه را جستجو نماییم تا بفهمیم که مقدار مورد نظر در کدام عنصر قرار گرفته است. به این روش "جستجوی خطی" می گویند.

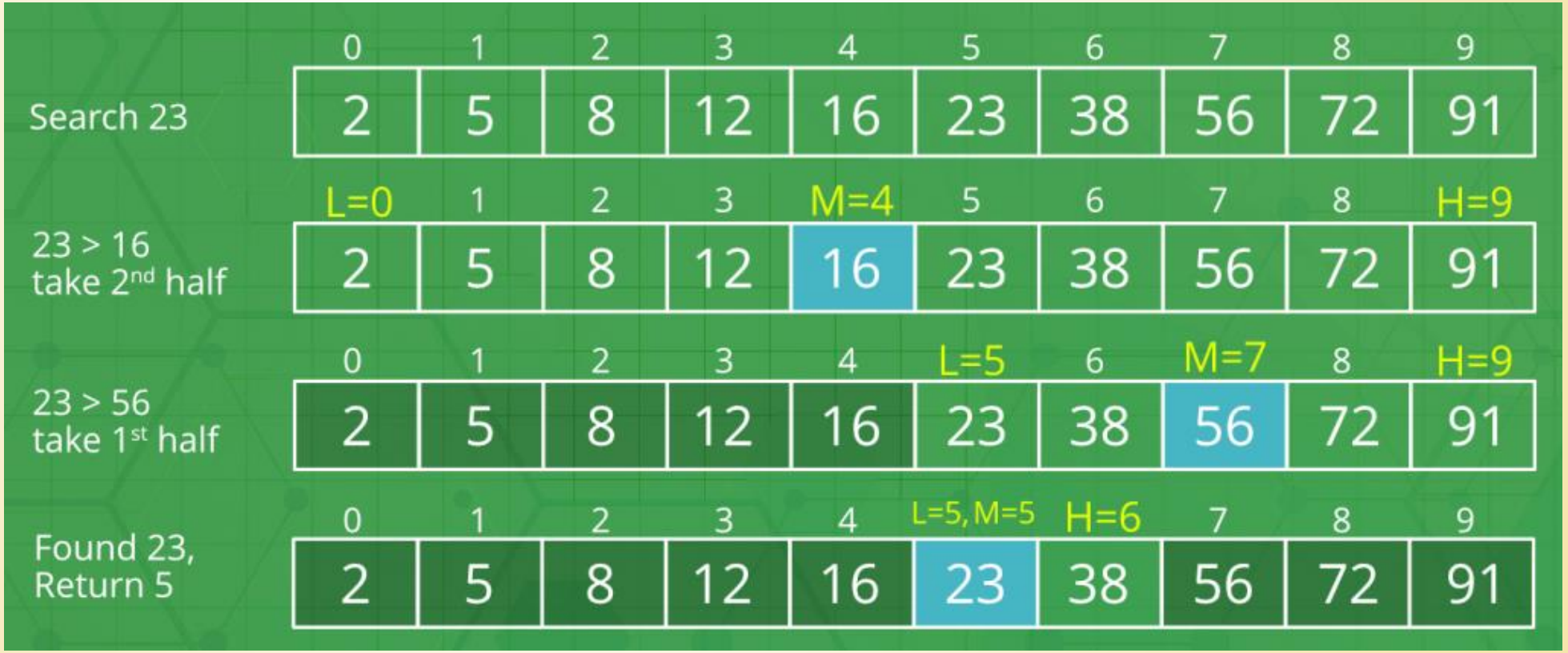
```
int index(int,int[],int);
main()
{
   int a[] = { 22, 44, 66, 88, 44, 66, 55};
   cout << "index(44,a,7) = " << index(44,a,7) << endl;
   cout << "index(50,a,7) = " << index(50,a,7) << endl;
}
int index(int x, int a[], int n)
{
   for (int i=0; i<n;i++)
        if (a[i] == x)  return i;
   return n; // x not found
}
```

تابع index() سه پارامتر دارد: پارامتر x مقداری است که قرار است جستجو شود، پارامتر a آرایه‌ای است که باید در آن جستجو صورت گیرد و پارامتر n ایندکس عنصری است که مقدار مورد نظر در آن پیدا شده است.

```
index(44,a,7) = 1
index(50,a,7) = 7
```

جستجوی دودویی

در روش *جستجوی دودویی* به یک آرایهٔ مرتب نیاز است. هنگام جستجو آرایه از وسط به دو بخش بالایی و پایینی تقسیم می‌شود. مقدار مورد جستجو با آخرین عنصر بخش پایینی مقایسه می‌شود. اگر این عنصر کوچک‌تر از مقدار جستجو بود، مورد جستجو در بخش پایینی وجود ندارد و باید در بخش بالایی به دنبال آن گشت. دوباره بخش بالایی به دو بخش تقسیم می‌گردد و گام‌های بالا تکرار می‌شود. سرانجام محدودهٔ جستجو به یک عنصر محدود می‌شود که یا آن عنصر با مورد جستجو برابر است و عنصر مذکور یافت شده و یا این که آن عنصر با مورد جستجو برابر نیست و لذا مورد جستجو در آرایه وجود ندارد. این روش پیچیده‌تر از روش جستجوی خطی است اما در عوض بسیار سریع‌تر به جواب می‌رسیم.



برنامه جستجوی دودویی

```
int index(int, int[],int);
main()
{  int a[] = { 22, 33, 44, 55, 66, 77, 88 };
   cout << "index(44,a,7) = " << index(44,a,7) << endl;
   cout << "index(60,a,7) = " << index(60,a,7) << endl;
}
int index(int x, int a[], int n)
{
   int lo=0, hi=n-1, M;
   while (lo <= hi)
   {
      M = (lo + hi)/2;
      if (a[M] == x)
         return M;
      else if (a[M] < x)
          lo= M+1;
      else hi = M-1;
   }
   return n;
}
```

برای این که بفهمیم تابع چطور کار می‌کند، فراخوانی index(44,a,7) را دنبال می‌کنیم:
وقتی حلقه شروع میشود، x=44 و n=7 و lo=0 و hi=6 است. ابتدا M مقدار ۳ را می‌گیرد. پس عنصر a[M] عنصر وسط آرایهٔ a[0..6] است. مقدار a[3] برابر با ۵۵ است که از مقدار x بزرگ‌تر است. پس x در نیمهٔ بالایی نیست و جستجو در نیمهٔ پایینی ادامه می‌یابد. لذا hi با M-1 یعنی ۲ مقداردهی می‌شود و حلقه تکرار می‌گردد.
حالا hi=2 و lo=0 است و دوباره عنصر وسط آرایهٔ a[0..2] یعنی a[1] با x مقایسه می‌شود. a[1] برابر با ۳۳ است که کوچک‌تر از x می‌باشد. پس این دفعه lo برابر با M+1 یعنی ۲ می‌شود.
در سومین دور حلقه، hi=2 و lo=2 است. پس عنصر وسط آرایهٔ a[2..2] که همان a[2] است با x مقایسه می‌شود. a[2] برابر با ۴۴ است که با x برابر است. پس مقدار ۲ بازگشت داده می‌شود؛ یعنی x مورد نظر در a[2] وجود دارد.

تفاوتهای جستجوی دودویی و خطی

۱-جستجوی دودویی فقط روی آرایه‌های مرتب کارایی دارد و اگر آرایه‌ای مرتب نباشد، جستجوی دودویی پاسخ غلط می‌دهد ولی جستجوی خطی همیشه پاسخ صحیح خواهد داد. بنابراین برای اعمال جستجوی دودویی باید یک بار آن را مرتب کرد.

۲-جستجوی دودویی در حالت کلی سریع‌تر از جستجوی خطی است. چون در هر بار جستجو نصف محدوده مورد جستجو را حذف میکند.

آرایه‌های چندبعدی

می‌توانیم آرایه‌ای تعریف کنیم که از نوع آرایه باشد، یعنی هر خانه از آن آرایه، خود یک آرایه باشد. به این قبیل آرایه‌ها، آرایه‌های چندبعدی می گوییم.

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

آرایه‌ای با ۳ عنصر از نوع intتعریف می کند. این یک آرایهٔ یک بعدی است.

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

یک آرایهٔ دو بعدی است با ۱۲عضو تعریف میکند.

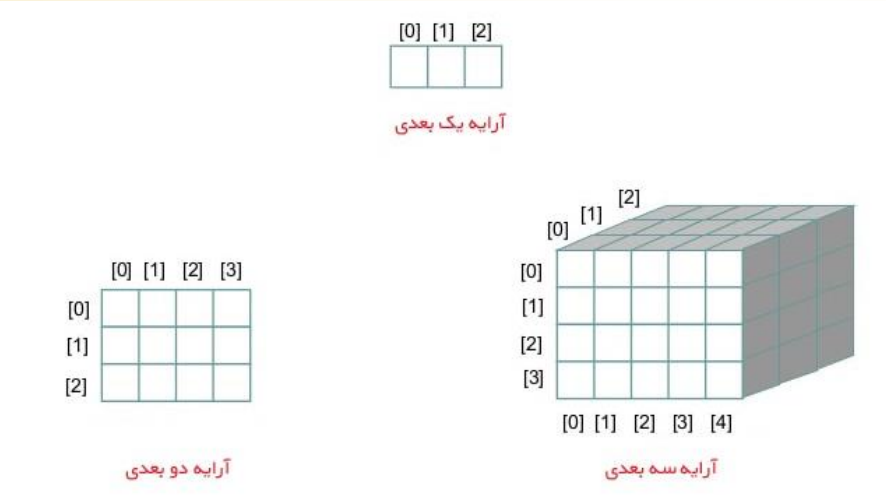
```
int a[4][5][3];
```

یک آرایهٔ سه بعدی است با ۶۰ عضو تعریف میکند.

شکل دستیابی به عناصر در آرایه‌های چند بعدی مانند آرایه‌های یک بعدی است. مثلا دستور

```
a[1][2][1] = 99;
```

مقدار ۹۹را در عنصری قرار می‌دهد که ایندکس آن عنصر(۱،۲،۱)است.



رشته ها

رشته (string)

یک رشته کاراکتری، آرایه ای از کاراکترهای کنار هم در حافظه است که با کاراکتر **NUL** خاتمه می یابد. کاراکتر **NUL** به معنی کاراکتر تهی (\0) است. آخرین خانه هر رشته با این کاراکتر پر می شود و وظیفه آن مشخص کردن انتهای رشته است.

```
Char name[7]="George";
Char name[]="George";
```

G	e	o	r	g	e	\0
0	1	2	3	4	5	6

****

```
int main()
{
    char s[] = "ABCD";
    cout << s << endl;
    for(int i = 0 ; i < 5 ; i++)
       cout << "s[" << i << "] = '"<< s[i] << "'\n";
}
```

```
ABCD
s[0] = 'A'
s[1] = 'B'
s[2] = 'C'
s[3] = 'D'
s[4] = ''
```

دستور Cin.get

برای خواندن رشته ها میتوان از دستور Cin استفاده کرد، اما ممکن است در این مورد دچار مشکل شویم و آن مشکل این است که اگر نامی را بعنوان یک متغیر رشته ای مثل Alireza Rad در نظر بگیریم، وقتی در میان رشته با کاراکتر space مواجه میشود، خواندن داده ورودی را متوقف میکنند و کلمات بعد از آن را نادیده میگیرد. در اینصورت تابع cin بخش Rad را دریافت نمیکند. برای رفع مشکل، از تابع cin.get() استفاده میکنیم که در سرفایل conio.h میباشد.

Cin.get(نام متغیر,حداکثر طول);

مثال: برنامه ای که نام و نام خانوادگی یک نفر را دریافت میکند.

```
main()
{
   char strName[20];
   cin.get(strName,20);
   Cout<<strName;
}
```

تابع **strlen()** تعداد کاراکترهای موجود در یک رشته کاراکتری را محاسبه می کند (بدون در نظر گرفتن NUL). به عبارت دیگر کار این تابع محاسبه طول یک رشته کاراکتری است.

```
main()
{
    char s[] = "ABCD";
    cout << s << endl;
    cout << strlen(s);
}
```

```
ABCD
4
```

****

تابع **strcmp()** دو رشته کاراکتری را با هم مقایسه می کند. اگر رشته اول از رشته دوم کوچکتر باشد (به لحاظ ترتیب الفبایی)، یک مقدار منفی برگشت داده می شود. چنانچه دو رشته با هم برابر باشند، مقدار برگشتی صفر است. و در نهایت چنانچه رشته اول از رشته دوم بزرگتر باشد یک عدد مثبت برگردانده می شود.

```
main()
{
    char s1[] = "xyz";
   char s2[] = "abcdef";
    cout << "s1: " << s1 << endl;
   cout << "s2: " << s2 << endl;
    int result = strcmp(s1, s2);
    if(result < 0)
      cout << "s1 < s2";
    if(result == 0)
      cout << "s1 = s2";
    if(result > 0)
      cout << "s1 > s2";
}
```

```
s1: xyz
s2: abcdef
s1 > s2
```

توابع کتابخانه ای کار با رشته های کاراکتری در C++

تابع **strcpy()** جهت کپی کردن یک رشته بر روی رشته دیگر استفاده می شود. این تابع پارامتر دوم را روی پارامتر اول کپی می کند.

```
main()
{
    char s1[20] = "abcdef";
   char s2[20] = "xyz";
   cout << "s1: " << s1 << endl;
   cout << "s2: " << s2 << endl;
   strcpy(s1, s2);
   cout << "s1: " << s1 << endl;
   cout << "s2: " << s2 << endl;
}
```

```
s1: abcdef
s2: xyz
s1: xyz
s2: xyz
```

تابع **strcat()** جهت اضافه کردن (چسباندن) یک رشته به انتهای یک رشته دیگر استفاده می شود. این تابع پارامتر دوم را به انتهای پارامتر اول اضافه می کند.

```
main()
{
   char s1[20] = "abcdef";
   char s2[20] = "xyz";
   cout << "s1: " << s1 << endl;
   cout << "s2: " << s2 << endl;
   strcat(s1, s2);
   cout << "s1: " << s1 << endl;
   cout << "s2: " << s2 << endl;
}
```

```
s1: abcdef
s2: xyz
s1: abcdefxyz
s2: xyz
```

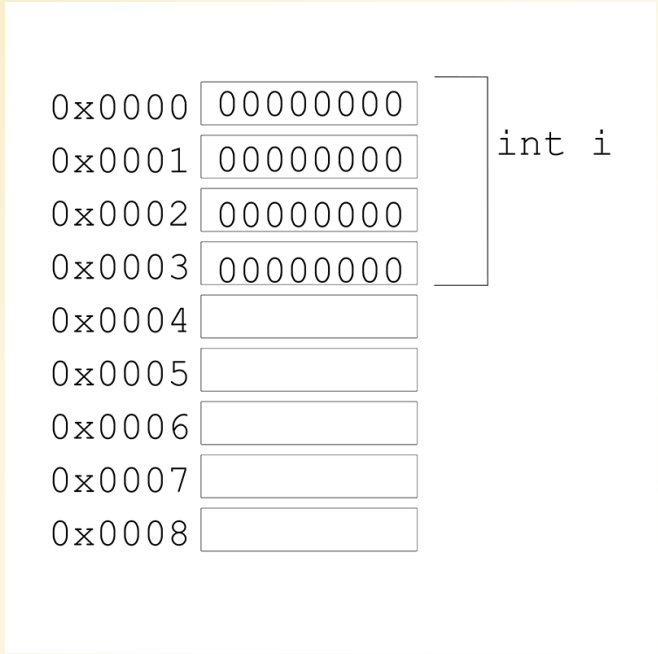
اشاره گرها و ارجاع ها

عملگر ارجاع

حافظهٔ رایانه را می توان به صورت یک آرایهٔ بزرگ در نظر گرفت.

هر متغیر دارای مقداری است که داخل خانه آرایه قرار دارد.

هر متغیر در حافظه دارای یک آدرس است.



برای بدست آوردن آدرس یک متغیر میتوان از عملگر ارجاع & استفاده کرد.

```
main()
{
   int n = 44;
   cout << "n= " << n<< endl;
   cout << "&n = " << &n << endl;
}
```

```
n  = 44
&n = 0x7fff5e9b5aac
```

ارجاعها

"ارجاع" یک اسم مستعار یا واژهٔ مترادف برای متغیر دیگر است. برای اعلان یک ارجاع به شکل زیر عمل میکنیم:

نوع& نام مستعار=نام متغیر;

```
int& rn=n;
```

rnیک ارجاع یا نام مستعار برای n است. البته n باید قبلا تعریف شده باشد.

مثال استفاده از ارجاعها:

```
main()
{
  int n=44;
  int& rn=n;
  cout << "n = " << n << ", rn = " << rn << endl;
  n=n-1;
  cout << "n = " << n << ", rn = " << rn << endl;
  rn= rn *2;
  cout << "n = " << n << ", rn = " << rn << endl;
}
```

```
n = 44, rn = 44
n = 43, rn = 43
n = 86, rn = 86
```

n و rn نامهای متفاوتی برای یک متغیر است. این دو همیشه مقدار یکسانی دارند.
در حقیقت یک متغیر n داریم که نام مستعار ان rn است. به عبارتی n و rn یک متغیر هستند.

کاربرد اول عملگر ارجاع: ارسال به طریق ارجاع

مثال: ارسال به طریق مقدار

```
void f(int,int);
main()
{
  int a = 22, b = 44;
  cout << "a = " << a << ", b = " << b <<endl;
  f(a,b);
  cout << "a = " << a << ", b = " << b <<endl;
  f(2*a-3,b);
  cout << "a = " << a << ", b = " << b << endl;
}
void f(int x , int y)
{
  x = 88;
  y = 99;
}
```

```
a=22,b=44
a=22,b=44
a=22,b=44
```

مثال: ارسال به طریق ارجاع

```
void f(int,int&);
main()
{
  int a = 22, b = 44;
  cout << "a = " << a << ", b = " << b <<endl;
  f(a,b);
  cout << "a = " << a << ", b = " << b <<endl;
  f(2*a-3,b);
  cout << "a = " << a << ", b = " << b << endl;
}
void f(int x , int& y)
{
  x = 88;
  y = 99;
}
```

```
a=22,b=44
a=22,b=99
a=22,b=99
```

نکته: استفاده از عملگر & در جلوی نام پارامتر تابع، باعث میشود که تابع به جای آنکه یک کپی محلی از آن آرگومان ایجاد کند، خود آرگومان محلی را به کار گیرد. بنابراین تابع هم میتواند مقدار آرگومان ارسال شده را بخواند و هم تغییر دهد.

کاربرد دوم عملگر ارجاع: بازگشت بیش از یک مقدار

مثال:

```
void ComputeCircle(double& moh, double& mas, double r)
{
   const double PI = 3.141592653589793;
   moh= PI*r*r;
   mas= 2*PI*r;
}
main()
{
   double r, a, c;
   cout << "Enter radius: ";
   cin >> r;
   ComputeCircle(a, c, r);
    cout << ""mohit= " << a << ", masahat= "<< c << endl;
}
```

تابع با استفاده از عملگر ارجاع، دو مقدار را برگردانده است. در حقیقت عملگر & باعث میشود تمامی تغییرات انجام شده توسط تابع فرعی روی متغیرها، توسط تابع main پذیرفته شود.

کاربرد سوم عملگر ارجاع: ارسال از طریق ارجاع ثابت

این روش مانند ارسال از طریق ارجاع است با این فرق که تابع نمی‌تواند محتویات پارامتر ارجاع را دست‌کاری نماید و فقط اجازهٔ خواندن آن را دارد. برای این که پارامتری را از نوع ارجاع ثابت اعلان کنیم باید عبارت **const** را به ابتدای اعلان آن اضافه نماییم.

```
void f(int x, int& y, const int& z)
{
  x= x+z;
  y= y+z;
  cout << "x = " << x << ", y = " << y << ", z = "<< z << endl;
}
main()
{
  int a = 22, b = 33, c = 44;
  cout << "a = " << a << ", b = " << b << ", c = " << c << endl;
  f(a,b,c);
  cout << "a = " << a << ", b = " << b << ", c = " << c << endl;
  f(2*a-3,b,c);
  cout << "a = " << a << ", b = " << b << ", c = " << c << endl;
}
```

```
a=22,b=33,c=44
x=66,y=77,z=44
a=22,b=77,c=44
x=85,y=121,z=44
a=22,b=121,c=44
```

تابع فوق پارامترهای **x**و **y**را می‌تواند تغییر دهد ولی قادر نیست پارامتر **z** را تغییر دهد. تغییراتی که روی **x** صورت می گیرد اثری روی آرگومان **a** نخواهد داشت. زیرا **a** از طریق مقدار به تابع ارسال شده. تغییراتی که روی **y** صورت می گیرد روی آرگومان **b** هم تاثیر می گذارد. زیرا **b** از طریق ارجاع به تابع فرستاده شده.

کاربرد ارجاعها

ارجاعها بیشتر برای ساختن پارامترهای ارجاع در توابع به کار میروند. تابع میتواند مقدار یک آرگومان را که به طریق ارجاع ارسال شده تغییر دهد. زیرا آرگومان اصلی و پارامتر ارجاع هر دو یک شی هستند.

ارسال به طریق مقدار باعث میشود که متغیرهای برنامهٔ اصلی از تغییرات ناخواسته در توابع مصون بمانند. اما گاهی اوقات میخواهیم این اتفاق رخ دهد. یعنی میخواهیم که تابع بتواند محتویات متغیر فرستاده شده به آن را دستکاری کند. در این حالت از ارسال به طریق ارجاع یا ارسال اشاره گرها استفاده میکنیم. برای این که مشخص کنیم یک پارامتر به طریق ارجاع ارسال میشود، علامت & را به نوع پارامتر در فهرست پارامترهای تابع اضافه میکنیم. این باعث میشود که تابع به جای این که یک کپی محلی از آن آرگومان ایجاد کند، خود آرگومان محلی را به کار بگیرد. به این ترتیب تابع هم میتواند مقدار آرگومان فرستاده شده را بخواند و هم میتواند مقدار آن را تغییر دهد.

ارسال پارامترها به طریق ارجاع دو خاصیت مهم دارد:

اول این که تابع میتواند روی آرگومان واقعی تغییراتی بدهد

دوم این که از اشغال بی مورد حافظه جلوگیری میشود

اشاره گر pointer

عملگر ارجاع & آدرس حافظه یک متغیر موجود را بدست میدهد. میتوان این آدرس را در متغیر دیگری ذخیره کرد. متغیری که یک آدرس در آن ذخیره میشوداشاره گر نامیده میشود. برای اینکه یک اشاره گر تعریف کنیم باید مشخص کنیم که چه نوع داده ای قرار است در آن ذخیره شود.

تعریف متغیراشاره گردر ++ cبه صورت زیر عمل می کنیم:

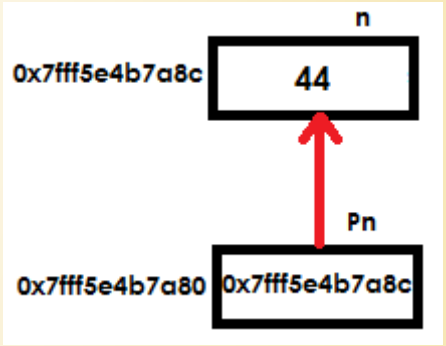
نام اشاره گر * نوع;

```
Float* Px;
```

Px اشاره گری است که آدرس متغیری از جنس float را در خود نگه میدارد.

```
main()
{
    int n = 44;
   cout << "n = " << n << " , &n = " << &n << endl;
   int* pn = &n;
   cout << "pn  = " << pn  << endl;
   cout << "&pn = " << &pn << endl;
   cout << "*pn = " << *pn << endl;
}
```

pn اشاره گری تعریف میشود که به یک عدد صحیح اشاره میکند و آدرس n را در اشاره گر pn قرار میدهد.




```
n   = 44 , &n = 0x7fff5e4b7a8c
pn  = 0x7fff5e4b7a8c
&pn = 0x7fff5e4b7a80
*pn = 44
```

عملگر new

```
float* p;
```

این دستور یک فضای چهاربایتی به p تخصیص داده میشود. اما p به هیچ جایی اشاره نمیکند. زیرا هنوز آدرسی درون آن قرار نگرفته است. به چنین اشاره گری اشاره گر سرگردان میگویند. اگر سعی کنیم یک اشاره گر سرگردان را مقداریابی یا ارجاع کنیم با خطا مواجه میشویم. مثلا دستور:

```
*p = 3.14159;
```

خطاست. زیرا p به هیچ آدرسی اشاره نمیکند و سیستم عامل نمیداند که مقدار 3.14159 را کجا ذخیره کند.
برای رفع این مشکل میتوان اشاره گرها را هنگام اعلان، مقداردهی کرد:

```
float x = 0;
float* p = &x ;
*p = 3.14159;
```

راه حل دیگر این است که یک آدرس اختصاصی ایجاد شود و درون p قرار بگیرد. بدین ترتیب p از سرگردانی خارج میشود. این کار با استفاده از عملگر new صورت میپذیرد:

```
float* p;
p = new float;
*p = 3.14159;
```

یا

```
float* p = new float(3.141459);
```

با این دستور، اشاره گر p از نوع *float تعریف میشود و سپس یک بلوک خالی از نوع float منظور شده و آدرس آن به p تخصیص مییابد و همچنین مقدار 3.141459 در آن آدرس قرار میگیرد. اگر عملگر new نتواند خانهٔ خالی در حافظه پیدا کند، مقدار صفر را برمیگرداند. اشاره گری که این چنین باشد، "اشاره گر تهی" یا NULL مینامند.

روشهای ایجاد و مقداردهی متغیر

روش اول:

```
float x = 3.14159;
```

روش دوم:

```
float* p = new float(3.14159);
```

نکته: در حالت اول، حافظهٔ مورد نیاز برای x هنگام کامپایل تخصیص میابد. در حالت دوم حافظهٔ مورد نیاز در زمان اجرا و به یک شیء بینام تخصیص میابد که با استفاده از *p قابل دستیابی است.

عملگر delete

عملگر **delete** عملی برخلاف عملگر **new** دارد. کارش این است که حافظهٔ اشغال شده را آزاد کند. وقتی حافظه ای آزاد شود، سیستم عامل میتواند از آن برای کارهای دیگر یا حتی تخصیصهای جدید استفاده کند. عملگر **delete** را تنها روی اشاره گرهایی میتوان به کار برد که با دستور **new** ایجاد شده اند. وقتی حافظه یک اشاره گر آزاد شد، دیگر نمیتوان به آن دستیابی نمود مگر این که دوباره این حافظه تخصیص یابد:

```
float* p = new float(3.14159);
delete p;
```

وقتی اشاره گر **p** در کد بالا آزاد شود، حافظه ای که توسط **new** به آن تخصیص یافته بود، آزاد میشود. وقتی اشاره گری آزاد شد، به هیچ چیزی اشاره نمیکند، مثل متغیری که مقداردهی نشده است. به این اشاره گر، **اشاره گر سرگردان** میگویند. بنابراین دستور زیر پیغام خطا میدهد.

```
*p = 2.71828;
```

انواع آرایه

ا-آرایه ایستا

```
float a[20];
```

بلوکی حاوی ۲۰ متغیر float که حافظهٔ مورد نیاز برای aدر زمان کامپایل تخصیص داده شود. وقتی برنامه اجرا شود، به هر حال حافظهٔ مربوطه تخصیص خواهد یافت حتی اگر از آن هیچ استفاده‌ای نشود.

۲-آرایه‌های پویا

می‌توانیم با استفاده از اشاره‌گر، آرایهٔ را طوری تعریف کنیم که حافظه مورد نیاز آن فقط در زمان اجرا تخصیص یابد:

```
float* p = new float[20];
```

دستور بالا، ۲۰ خانهٔ خالی حافظه از نوع float را در اختیار گذاشته و اشاره‌گر p را به خانهٔ اول آن نسبت می‌دهد. به این آرایه، "آرایهٔ پویا" می گویند.

مقایسه آرایهٔ ایستا و آرایهٔ پویا:

آرایهٔ ایستا a در زمان کامپایل ایجاد می‌شود و تا پایان اجرای برنامه، حافظهٔ تخصیصی به آن مشغول می‌ماند. ولی آرایهٔ پویا p در زمان اجرا و هر جا که لازم شد ایجاد می‌شود و پس از اتمام کار نیز می‌توان با عملگر delete حافظهٔ تخصیصی به آن را آزاد کرد:

```
delete [] p;
```

برای آزاد کردن آرایهٔ پویای p براکت‌ها [] قبل از نام p باید حتما قید شوند زیرا p به یک آرایه اشاره دارد.

مثال استفاده از آرایه‌های پویا :

```
void get(double*& a, int& n)
{
   cout << "Enter number of items: ";
   cin >> n;
   a = new double[n];
   cout << "Enter " << n << " items, one per line:\n";
   for (int i = 0; i < n; i++)
   {
      cout << "\t" << i+1 << ": ";
      cin >> a[i];
   }
}
void print(double* a, int n)
{
   for (int i = 0; i < n; i++)
      cout << a[i] << " " ;
   cout << endl;
}
main()
{
   double* a;// a is simply an unallocated pointer
   int n;
   get(a,n); // now a is an array of n doubles
   print(a,n);
   delete [] a;// now a is simply an unallocated pointer again
   get(a,n); // now a is an array of n doubles
   print(a,n);
}
```

```
Enter number of items: 4
Enter 4 items, one per line:
1: 44.4
2: 77.7
3: 22.2
4: 88.8
44.4 77.7 22.2 88.8
Enter number of items: 2
Enter 2 items, one per line:
1: 3.33
2: 9.99
3.33 9.99
```

شی گرایی

برنامه نویسی شی گرا Object-Oriented Programming

هدف این است که داده ها و توابعی را که روی آنها کار میکنند در یک عنصر خاص یا موجودیت یا Object قرار دهیم.
یک شی موجودیتی است که شامل داده ها و توابعی است که روی این داده ها اعمال میشوند.

داده های یک آسانسور و توابع آن:

کلاس(Class): اشیا نمونه هایی از کلاس هستند.
هر دانشجو یک شی از کلاس دانشجو میباشد.

Student Farhadi;
فرهادی از کلاس دانشجو است.

int i,j;
i,j نمونه هایی از جنس کلاس int میباشند.

ویژگیهای برنامه نویسی شی گرا

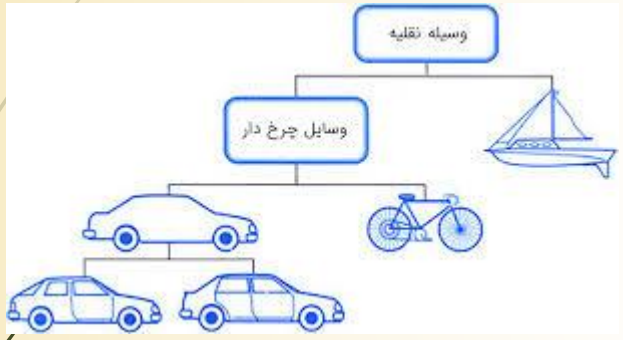
۱- کپسوله کردن Encapsulation

یعنی داده ها و توابعی که روی داده ها اثر میگذارند را در جایی محدود کنیم تا از استفاده نادرست و تداخل خارجی جلوگیری شود.

۲-مخفی سازی اطلاعات information hiding

یعنی داده ها و توابع یک شی میتواند از شی دیگر مخفی بماند. بنابراین توابع یک شی نمیتواند داده های شی دیگر را تغییر دهد.

۳-ارث بری inheritance



در شی گرایی هر قدر در سلسله مراتب پایین رویم، اشیا خصوصیات شی بالایی را دارا میباشند.

۴-تعریف نوع داده جدید Data Type Definition

```
int i,j;
```

i,j از جنس کلاس int میباشند.

```
Student Farhadi,Rezaei;
```

فرهادی و رضایی از جنس کلاس student هستند. ولی چون کامپایلر نوع داده Student را نمیشناسد، بر خلاف int باید آن را تعریف کنیم.

۵-چند ریختی توابع Polymorphism

چندریختی در C++ به معنای یک چیز بودن و چند شکل داشتن است.

۶-قابلیت استفاده مجدد

چون داده ها و توابع را در شی محصور میکنیم، به راحتی میتوان آنها را به جای دیگر منتقل کرد.

چند ریختی توابع

اگر توابع همنام با فهرست پارامترهای گوناگون داشته باشیم که نوع پارامترها یا تعدادشان با هم فرق دارد، کامپایلر C++ فهرست آرگومانها را در تابع اصلی بررسی میکند و میفهمد باید از کدام تابع استفاده کند.

```
int max(int x, int y)
{
  if x>y return x;
 else return y;
}
int max(int x, int y, int z)
{
   int m;
   if x>y
     m=x;
  else
     m=y;
  If m>z
     return m;
  else
     return z;
 }
float max(float x, float y)
{
   if x>y return x;
   else return y;
}
main()
{
  cout << max(99,77) << " " << max(55,66,33) << " " << max(44.4,88.8);
}
```

در این برنامه سه تابع با نام max() تعریف شده است.

وقتی تابع max() در جایی از برنامه فراخوانی می‌شود، کامپایلر فهرست آرگومان آن را بررسی می‌کند تا بفهمد که کدام نسخه از max باید احضار شود.

ساختار کلاس

```
Class نام کلاس
{
   Private:
        داده ها
   Public:
        داده ها و توابع
};
```

کلید واژه های private باعث می شود داده ها و توابع خصوصی شوند. داده ها و توابع خصوصی تنها در داخل کلاس خود قابل دسترسی هستند.

کلید واژه public باعث می شود که داده ها و توابع عمومی شوند. داده ها و توابع عمومی در خارج از کلاس هم قابل دسترسی هستند.

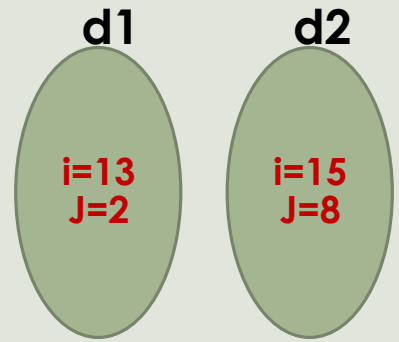
فرمت اول: تعریف بدنه توابع در خارج از کلاس

```
نام تابع :: نام کلاس   نوع تابع
{
   بدنه تابع
}
```

فرمت دوم: تعریف توابع در داخل کلاس

فرمت اول

```
Class divide
{
      int i,j,r,s;
   Public:
      Void init();
      Void division();
      Void show();
};
Void divide :: init()
{
   Cout<<"enter two number:";
   Cin>>i>>j;
}
Void divide :: division()
{
   r=i/j;
   s=i%j;
}
Void divide :: show()
{
   cout<<r<<" "<<s<<endl;
}
Main()
{
   divide d1,d2;
   d1.init();
   d2.init();
   d1. division();
   d2. division();
   d1. show();
   d2. show();
}
```



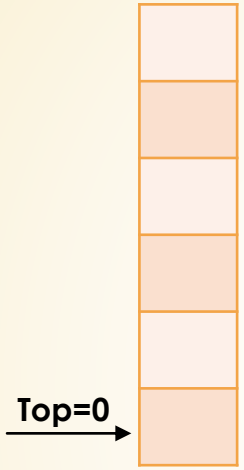
فرمت دوم

```
Class divide
{
      int i,j,r,s;
   Public:
      Void init()
      {
        Cout<<"enter two number:";
        Cin>>i>>j;
       };
      Void division()
      {
         r=i/j;
         S=i%j;
       };
      Void show()
      {
         cout<<r<<" "<<s;
      };
}
Main()
{
   divide d1,d2;
   d1.init();
   d2.init();
   d1. division();
   d2. division();
   d1. show();
   d2. show();
}
```

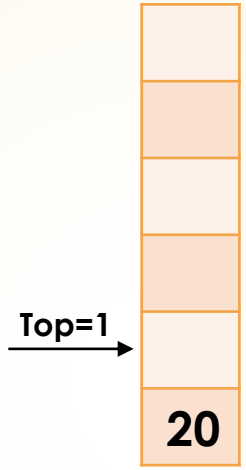
پشته یا Stack یا Last in First Out

پشته ساختاری است که درآن عناصر روی یکدیگر قرار میگیرند. بدین صورت که هر عنصر جدید بالای پشته قرار میگیرد و برای برداشتن عنصر از پشته فقط مجازیم عنصر بالای پشته را برداریم. برای اضافه کردن عنصر به پشته باید دقت کرد که پشته پر نباشد. برای برداشتن عنصر از پشته باید دقت کرد که پشته خالی نباشد.

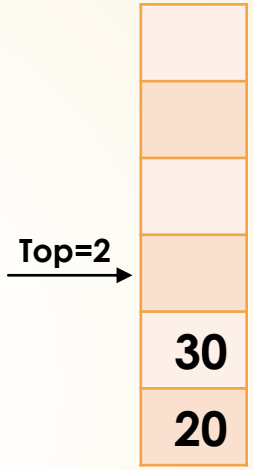
Top به روی پشته(مکانی که باید عنصر جدید اضافه شود) اشاره میکند.



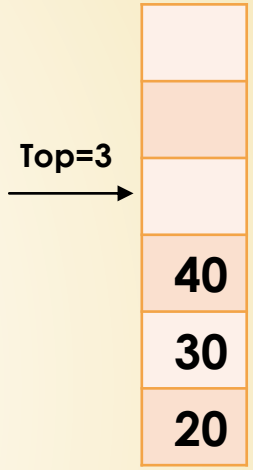
مرحله۱



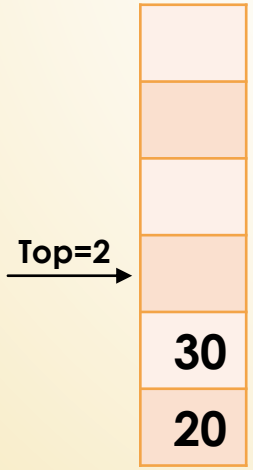
مرحله۲–اضافه کردن ۲۰



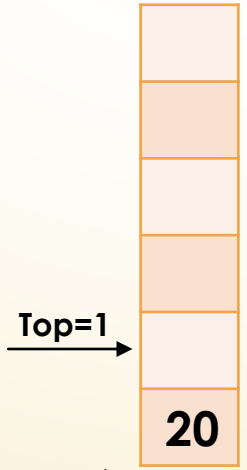
مرحله۳–اضافه کردن ۳۰



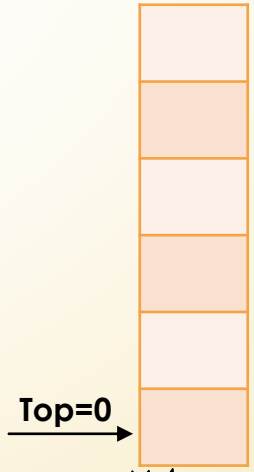
مرحله۴–اضافه کردن ۴۰



مرحله۵–حذف



مرحله۶–حذف



مرحله۷

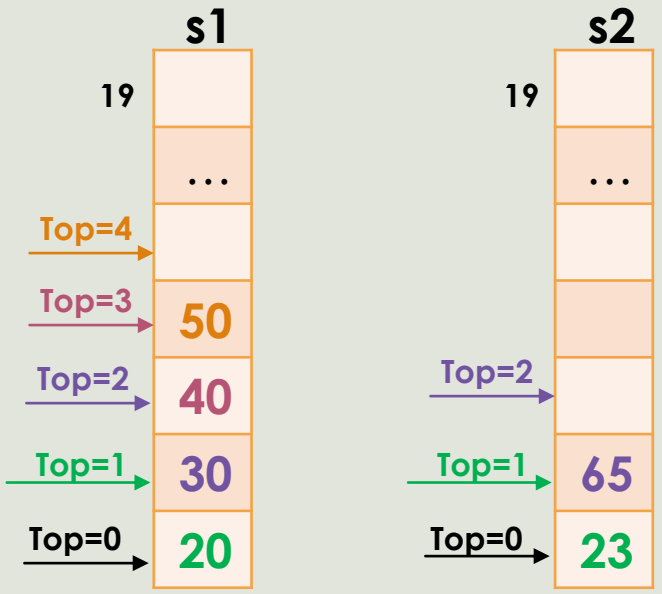
پیاده سازی پشته

پیاده سازی پشته

```
class Stack
{
   int stck[20];
    int top;
Public:
   void init();
   void add(int x);
   int remove();
};
Void Stack :: init()
{
   top=0;
}
Void Stack :: add(int x)
{
   if (top>19)
      cout<<"Stack is full';
   else
   {
      stck[top]=x;
      top++;
   }
}
```

```
int Stack :: remove()
{
   If (top==0)
     Cout<<'Stack is empty";
    Else
    {
      Top--;
      Return(stck[top]);
    }
}
Main()
{
    Stack s1,s2;
    int i;
    s1.init();
    s2.init();
    s1.add(20);
    s1.add(30);
    s1.add(40);
    s1.add(50);
    s2.add(23);
    s2.add(65);
```

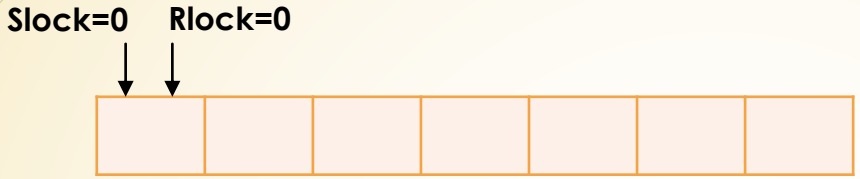
```
For (i=0;i<4;i++)
     Cout<<s1.remove()<<" ";
  Cout<<endl;
  For (i=0;i<2;i++)
     Cout<<s2.remove()<<" ";
}
```



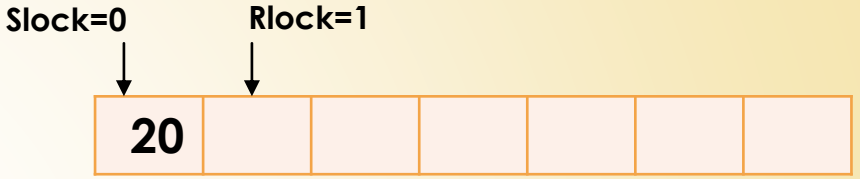
صف Queue یا First in First Out

صف ساختاری است که در آن عناصر پشت سر یکدیگر قرار میگیرند. بدین صورت که هر عنصر جدید انتهای صف قرار میگیرد و برای برداشتن عنصر از صف فقط مجازیم عنصر ابتدای صف را برداریم. برای اضافه کردن عنصر به صف باید دقت کرد که صف پر نباشد. برای برداشتن عنصر از صف باید دقت کرد که صف خالی نباشد.

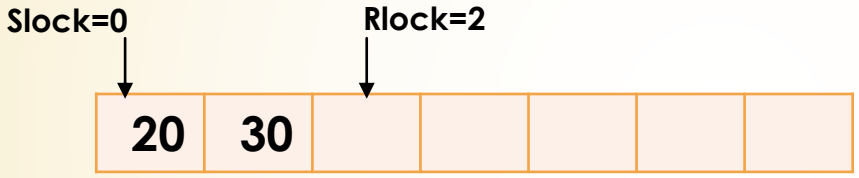
Sloc به عنصر سر صف اشاره میکند. Rloc پشت آخرین عنصر است.



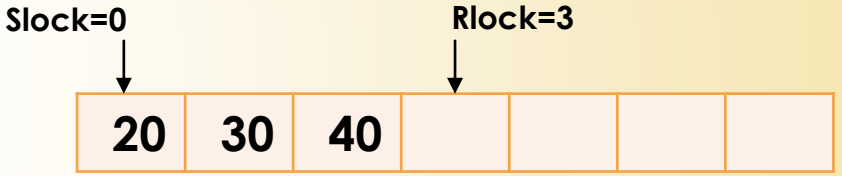
مرحله۱



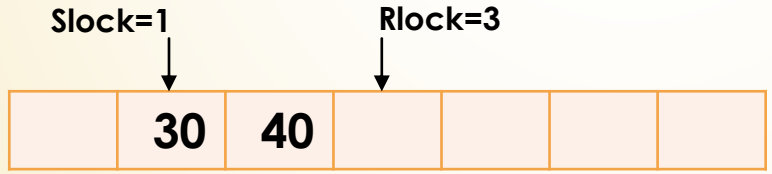
مرحله۲–اضافه کردن ۲۰



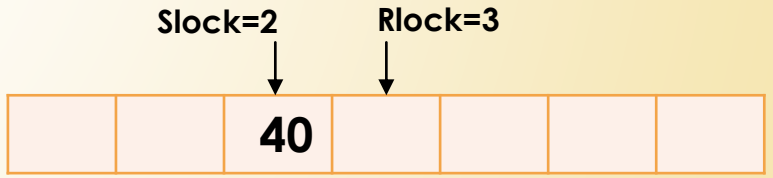
مرحله۳–اضافه کردن ۳۰



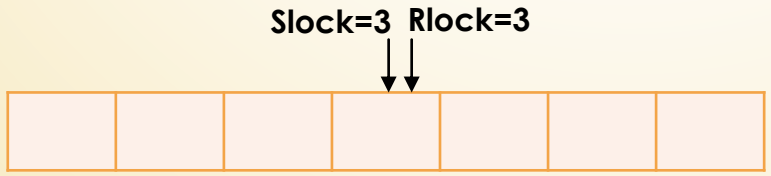
مرحله۴–اضافه کردن ۴۰



مرحله۵–حذف



مرحله۶–حذف



مرحله۷–حذف

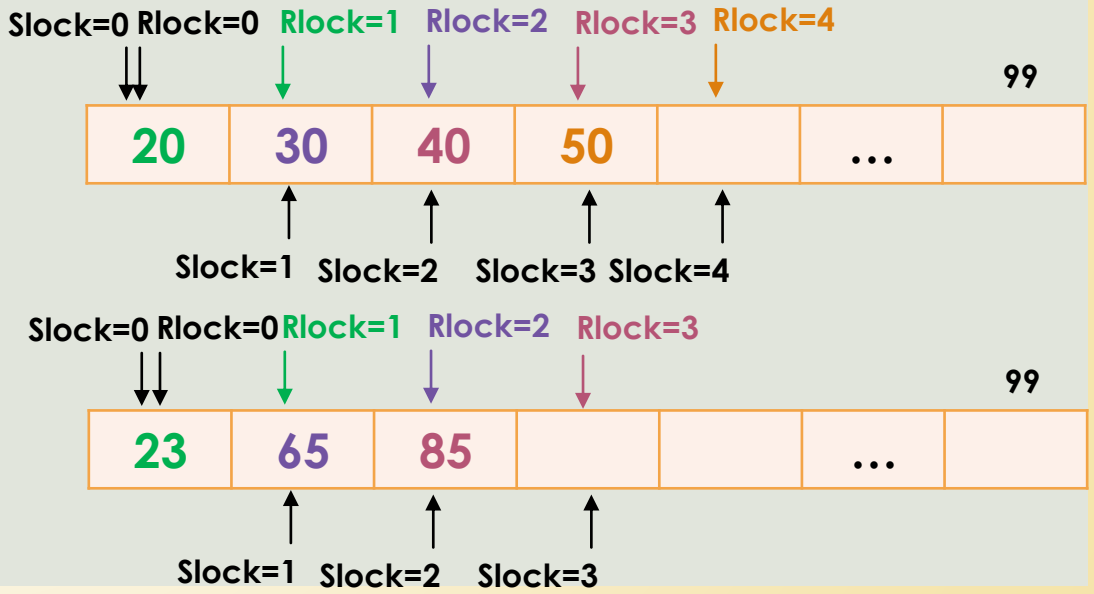
پیاده سازی صف

پیاده سازی صف

```
class queue
{
   int Sloc,Rloc;
   int q[100];
Public:
   Void init();
   Void add(int x);
   Int remove();
};
Void queue :: init()
{
   Sloc=Rloc=0;
}
Void queue :: add(int x)
{
   if (Rloc>=100)
      cout<<“queue is full';
  else
  {
     q[Rloc]=x;
     Rloc++;
   }
}
```

```
int queue :: remove()
{
   If (Rloc==Sloc)
      Cout<<queue is empty";
   Else
      Return(q[Sloc++]);
}

Main()
{
   queue q1,q2;
   int i;
   q1.init();
   q2.init();
   q1.add(20);
   q1.add(30);
   q1.add(40);
   q1.add(50);
   q2.add(23);
   q2.add(65);
   q2.add(85);
```

```
  For (i=0;i<4;i++)
      Cout<<q1.remove()<<" ";
For (i=0;i<3;i++)
      Cout<<q2.remove()<<" ";
}
```



پایان

"موفق باشید"