حلقـه‌ها:

۱- تعداد تکرار حلقه‌ها نامحدود و نامعین است.

۲- تعداد تکرار حلقه‌ها محدود و معین است.

توجه: اگر برنامه‌ای بنویسید که هیچگاه از حلقه بیرون نیاید «دارای حلقه بی‌نهایت باشد» برنامه *Run* نمی‌شود چراکه حلقه بی‌نهایت یک ایراد برنامه‌نویسی محسوب می‌شود.

نکته: فلـش‌گذاری در حلقه‌ها بسیار مهم است.

حلقـه‌ها در زبان C++ :

```
for ( 1مقداردهـی اولیـه  ;  2 شـــرط  ; 4 گام‌حرکت )
{
        3دستـورات
}
```

مثال: برنامه‌ای بنویسید که اعداد بین ۱ تا ۱۰ را چاپ کند.

```
#include “stdafx.h”
#include ”iostream”
using namespace std;
int main(void)
{
for(int i=1;i<=10;i++)
{
cout<<”i:”<<i;
}
system(“pause”);
return 0;
} //end of main
```

نکته: معمولاً اندیس حلقه نوع « **Data type**» داخل *for* تعریف می‌شود به عنوان مثال برای مثال قبل می‌توانیم برنامه را به صورت زیر تعریف کنیم:

$$for\ (\ \text{int}\quad i=1\ ;\ i\prec=10\ ;\ i=i+1\ )$$

مثال: برنامه‌ای بنویسید که اعداد فرد کوچکتر از 20 را چاپ کند.

```
#include”stdafx.h”
#include”iostream”
using namespace std;
int main()
{
for(int i=1;i<20;i++)
{
cout<<”i:”<<i;
}
system(“pause”);
return 0;
}
```

مثال: برنامه‌ای بنویسید که دوعدد از ورودی دریافت و عدد اول را به توان عدد دوم برساند.

```
#include “stdafx.h”
#include”iostream”
using namespace std;
int main()
{
int x,y,s=1;
cin>>x,y;
for(int n=1;n<=y;n++)
{
s=s*x;
}
cout<<”s:”<<s;
}
```

for (مقداردهی اولیه ; شرط ; گام‌حرکت)

{

دستورات

}

نکته: اگرداخل دستور *for* تنها یک دستورداشته باشیم دیگر نیازی نیست دوآکلاد ابتدایی و نهایی پیام *for* را بگذاریم یعنی می‌توانیم بنویسیم:

$$for \quad (\text{ int } \quad i = 0 \quad ; \; i \prec 10 \quad ; \; i \; + + \;)$$

$$Cout \quad << i \quad ; \rightarrow \text{۰/۱/۲/ ..... /۹}$$

نکته: اگردرانتهای *for* « ; » بگذاریم حلقه درخودش اجرا می‌شود و عملیات آن دیگر قابل استفاده نیست و تنها آخرین عملیات انجام شده را می‌توان دردستورات بعدی استفاده‌کرد.

$$for \quad (\text{ int } \quad i = 0 \quad ; \; i \prec 10 \quad ; \; i \; + + \;) \quad ;$$

cout<<i;

توجه: بعضی از برنامه‌نویسان ازاین اشکال برای تأخیر در برنامه استفاده می‌کنند که البته روش درستی برای ایجاد تأخیر نیست. مثلاً این کار را انجام می‌دهند.

```
cout<<"A";
for(int i=0;i<10000000;i++)
cout<<"B";
```

این روش به این علت خوب نیست که وابسته به سرعت کامپیوتر است و بسته به هرکامپیوتر میزان تأخیر ایجاد شده فرق می‌کند و این موضوع اصلاً مطلوب نیست.

نکته: دردستور *for* گذاشتن هریک از سه قسمت؛ گام‌حرکت ،مقداردهی‌اولیه ۳/ شرط اختیاری است ولی توجه داشته باشیم هرقسمت را که نمی‌گذاریم «; » را باید بگذاریم و برای قسمت شرط دو «; » نیازاست. البته منظور از حذف‌کردن هریک از این سه قسمت آن است که درجای‌دیگر خارج از *for* قرارگیرند چراکه درغیراین صورت دیگر حلقه‌ای نخواهیم داشت. مثلاً برای دستورکامل زیرداریم.

```
for(int i=0;i<10;i++)
cout<<i;
```

مقداردهي اوليه قبل از for قرار مي گيرد :

```
int i=0;
for( ;i<10;i++)
cout<<i;
```

گام حرکت در در دستورات for قرار گرفته است :

```
for(int i=0;i<10 ; )
{
cout<<i;
i++;
}
```

شرط داخل دستور forمي تواند قرار بگيرد ولي بايد توجه داشته باشيم كه براي آنكه حلقه بي نهايت نشود بايد از ifاستفاده كنيم:

```
for(int i=0; ;i++)
{
cout<<i;
 if( )
}
```

نکته: روش صحیح و خوب برنامه‌نویسی آن است که دستورات *for* کامل باشد چراکه خواناتر و بهتـراست.

for (int $i=1$ و int $j=2$; $i+j \prec 10$; i ++ و j ++)

{

cout<<i*j;

}

i	j	$i+j$	$i+j \prec 10$	$i*j$
1	2	3	✓	2
2	3	5	✓	6
3	4	7	✓	12
4	5	9	✓	20
5	6	11	×	خروجی نداریم

برنامه فوق را به شکل زیر نیز می‌توان نوشت:

int $j=2$;

for (int $i=1$; $i+j \prec 10$; i ++)

{

Cout $<< i*j <<"\n"$;

}

توجه: این‌که کدام یک از این دو برنامه بهتراست بسته به برنامه‌ای که می‌نویسم فرق دارد ولی درحال حاضر با برنامه‌ای که ما نوشتیم شکل اول بهتراست.

نکته: ما می‌توانیم از چندگام حرکت، مقداردهی اولیه و شرط استفاده‌کنیم ولی توجه به این نکته ضرروی است که درتئوری گفتن n تا گام‌حرکت یا مقداردهی اولیه و یا شرط ساده است ولی در عمل انجام این‌کار از خوانا بودن برنامه کسر می‌کند. برای جداسازی چندگام‌حرکت یا مقداردهی اولیه از ویلگول (،) و برای جداسازی چند شرط از 88 یـا ۱۱ استفاده می‌کنیم.

```
for ( int i=5 و int j=20 ; i≺50  88j≺100 ; i=i×2 و j=j +۱۰ )
{
        cout<<i*j<<"\n";
}
```

i	j	$i \prec 5088 j \prec 100$	$i+j$
5	20	✓	25
10	30	✓	40
20	40	✓	60
40	50	✓	90
80	60	×	خروجی نداریم

نکته: حلقه روبه‌رو یک حلقه بی‌نهایت است زیرا شرط همواره برقرار است.

```
for ( int  i=10  ;  i≻2  ;  i + + )
  {
      دستورات
  }
```

توجه: باید توجه داشته باشیم که باید شرط بالاخره نقض شود.

شرط زیر هنگامی نقض می‌شود که:

$(i \prec = 105 \;\&\&\; j \succ 50) \;||\; K!= 42$

$(i \succ 105 \text{ or } j \prec = 50) \;\&\&\; K = 42 \;\rightarrow$ شرط نقض می‌شود

نکته: الزامی وجود ندارد که گام‌حرکت و مقداردهی اولیه از نوع int باشد و می‌تواند اعشاری هم باشد ولی چون کامپایلر به int راحتر از *float* دسترسی دارد و فرمان *for* فرمانی است که چندین‌بار در برنامه تکرار می‌شود بهتراست داده‌هایم درون این فرمان از نوع int باشد « برنامه صفحه ۱۶۲ مثال ۵- ۴ را مطالعه‌کن»

کدهای اسکیASCII :

کدهایی هستند که معادل دکمه‌هایی که روی صفحه *Keybord* خود مشاهده می‌کنیم .

نکته:

۱/ *for* (int $i=0$ [1] ; i !=100 [2] ; i + + [4])

Cout << i [3] ;

۲/ *for* (int $i=0$; i <=100 ; i ++)

cout<<i;

چون در روش اول تنها نقطه‌ای که محل نقض شرط است عدد ۱۰۰ است پس ریسک این روش بالا است چراکه به هردلیلی اگر اعداد از ۱۰۰ بالا بزند حلقه بی‌نهایت خواهد شد و این موضوع در برنامه‌های چندین هزارخطی بسیار مهم است از این دو روش اول را روش ضعیف شرطنویسی گویند و بهتراست از روش دوم استفاده شود.

حلقه های تو در تو:

```
for (گام حرکت؛ شرط؛ مقدار دهی اولیه)
{
دستورات for اول
for(گام حرکت؛ شرط؛ مقدار دهی اولیه)
{
دستورات for دوم
}
دستورات for اول
}
```

برنامه جدول ضرب را بنویسید:

```
for( int   i=۱؛i <۱۰; i++)
{
for(int j=۱; j <۱۰; j++)
cout << i*j;
}
```

نحوه چاپ:

1 2 3 4 5 6 7 8 9 10

1 1 2 3 4 5 6 7 8 9 10

2 2 4 6 8 10 12 14 16 18 20

3 ……

4 ……

5 ……

6 ……..

7 ……

8 ……

9 ……..

10 ……..

نکته :

باید توجه داشته باشید که اگر دستورات بیش از یکي باشد حتما باید از بلاك استفاده کنید.

مثال: برنامه ای بنویسید که حاصل ضرب اعداد فرد بین ۰-۲۰ را در اعداد زوج بین ۲۰-۴۰ چاپ کند.

```
# include "stdafx.h"
# include "iostream"
using namespace std;
int main (void)
{
for (int n-0=1 ; n-o<20 ; n-o +=2)
{
for (int n-e=20 ; n-e<40; n-e +=2)
cout << " n-o * n-e : " << n-o * n-e ;
cout << "\n" ;
}
}//end of int main
```

Blouck

به هر دستوری که بین دو آکلاد نوشته می شود یک blouck می گوییم و باید توجه داشته باشیم که خارج از هر blouck دیگر متغیرهای تعریف شده در آن blouck غیرقابل استفاده و مفهوم است.

توجه: برنامه زیر ۶۰۰۰ بار اجرا شده و از دید کامپایلر کل این قسمت از برنامه ۱ دستور است.

```
for (i=1 ; i<10 ; i++)

for (j=1 ; j<20 ; j++)

for (k=1 ; k<30; k++)

{

int X=i+j+k;

cout << x;

}
```

حلقه while:

این حلقه بیشتر برای حلقه هایی قابل استفاده است که تعداد تکرار حلقه هایش معلوم نیست، ولی این بدین معنا نیست که نمی توان برای حلقه هایی که تعداد تکرارشان معلوم است از این دستور استفاده کرد ولی روش خوبی برنامه نویسی نیست.

```
مقداردهی اولیه

while (شرط)

{

دستورات گام حرکت

}
```

نکته: برای حلقه while همانند for هنگامی که تنها یک دستور داشته باشیم نیازی به گذاشتن آکلاد نیست.

مثال: برنامه ای بنویسید که اعداد بین ۱ تا ۱۰ را با دستور while چاپ کند.

```
# include "Stdaf x.h"
# include "iostreeam"
using namespace std;
int main (void)
{
int i=1;
while (i<=10)
{
cout << i;
i++
}
system ("pause");
return o;
}
```

نکته: چون پیغام while برای حلقه هایی است که تعداد تکرار آن نامشخص است و باید شرط پایان خودمان تعریف کنیم مثلاً بگوییم که برنامه با وارد کردن کاراکتر "*" به پایان می رسد و این شرط همان شرطی است که رو به روی while قرار می گیرد.

```
int main (void)
{
int a,b;
Cout << "Enter 2 number:" ;
Cin >> a>>b ;
int s=a+b;
cout << S ;
return 0 ;
}
```

*برای آنکه بتوانیم عملیات مورد نظر را با یک بار Run کردن برنامه چند بار اجرا کنیم می توانیم:

```
int main (void)
{
cout << " Do you want to continue ? (Y/N)"
cin >> k;
while (k= 'Y' || k='y')
{
Cin >>a>>b ;
Int S= a+b ;
Cout S ;
Cout <<"if you want repeat this calculation Enter 'Y' << "\n" ;
Cin>>k;
}
Return 0;
}
```

مقایسه while با for :

for (مقدار دهی اولیه ; شرط ; گام حرکت)

{

دستورات

}

حال اگر بخواهیم for را با while بنویسیم به شکل زیر عمل می کنیم.

مقدار دهی اولیه

while (شرط)

{

دستورات

گام حرکت

}

حلقه do – while :

مقدار دهی اولیه

do

{

دستورات

گام حرکت

}

while (شرط) ;

در do – while ابتدا عملیات انجام شده بعد شرط while چک می شود ولی در while در همان ابتدا شرط while چک شده سپس عملیات انجام می شود.

مثال: برنامه ای بنویسید اعداد بین ۱ تا ۱۰ را به توان ۲ برساند.

*با for، while و do-while می نویسم:

1- for:

```
for (int i=1 ; I < = 10 ; i++)
cout << "i2:" << i*i;
```

2- while:

```
int i=1;
while (i<=10)
{
cout << i*i;
i++;
}
```

3- do…while:

```
int i=1;
do
{
cout <<" i2 :"<<i*i;
}
while(i<=10);
```

*ULONG-MAX : بزرگترین عددی است که در یک متغیری علامت می تواند ذخیره شود.

```
Sum=0+1;        sum=1+1;        sum=1+2;
Next=1;         next=1;         next=2;
Last=1;         last=2;         last=3;
```

- برنامه ای که تعداد نامشخص عدد را از ورودی دریافت مقلوب آنها را حساب کند و به خروجی بدهد.

```
# include "stdafx.h"
# include "ioStream"
using namespace std;
int main(void);
{
int number , digit;
while (true)
```

این حلقه بی نهایت است باید چاره ای بیندیشم ولی اگر این کار را هم نکردیم و کامپایلر احیاناً error نداد، cout1+G را می گیریم تا از برنامه خارج شویم.

```
{
cout << "\n Enter an integer number for convert:";
cin>>number;
cout << "Inverse is:" ; cout=3 2 1
do
{                                    number= 1 2 3
                                      digit= 123 %10=3
digit=number %10;                    number= 123 %10=12
cout<<digit;                          digit=12%10=2
Number /=10;                    number=12%10=1
}                                      Digit=1%10=1
while(number !=0);                   number1/10=0
```

```
}
system ("pause")
return 0;
}
```

دستور break:

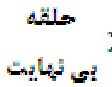
خارج شدن از حلقه توسط break انجام می شود.

```
int i=1;
while (i<10)
{
cout<<i;
Break;
i++;
}
cout<<"hello";
```

دستور Continue:

این دستور، دستورهای بعد از خودش را اجرا نکرده و به ابتدای حلقه می رود:

```
int i=1;
while(i<10)
{
cout<<i;
Continue;
i++;
}
```

خروجی: 1,1,1,…….. 

دستورgo to: «معروف به دستور اسباگتی«ماکارونی»»

استفاده از این دستور اصلاً توصیه نمی شود چونکه موجب گیج کردن مخاطب و کاربر و در بعضی موارد هم خود برنامه نویس خواهد شد. روش استفاده به شکل زیر است:

goto labale;

مثال → int i=1;

Ali: cout <<i;

labale: i=i+1;

goto Ali;

***برخی از دستورات صفحه نمایش برنامه:**

System: دستوری است که به برنامه نویس اجازه می دهد دستورات صفحه duste را در برنامه خود اجرا کند

برای ایجاد تأخیر است تا کاربر فرصت دیدن خروجی را پیدا کند:

1- System("pause");

صفحه را کامل پاک می کند و دستورات جدید را چاپ می کند :

2- System("CLS");

. برای رنگی کردن صفحه نمایش استفاده می شود :

3- System("color ");

*در قسمت start , ← run ← پنجره ای باز می شود در آن پنجرهcmd را چاپ می کنیم ← inter ← صفحه duste باز می شود← ? ☐ color ← شماره رنگها را می دهد.

دستور if :

```
if (شرط)
{
 دستورات(true)
}
else
{
دستورات(false)
}
```

نکته: قسمت else را هیچ گاه برای خودمان تغییر نکنیم یعنی شرط درست را برای خود معنا می کنیم و بدانیم شرط else نقیض قسمت true شرط است و اگر else را برای خود تفسیر کنیم در بعضی از موارد ممکن است حالت های خاص را در نظر بگیریم و برنامه اشتباه اجرا شود. «حالت های غیر قابل پیش بینی را در نظر نمی گیریم»

```
int x=10;
if (x%21==0)
{
cout<<"even";
}
else
	{
	cout<<"odd";
	}
```

نکته: در دستور if هم اگر قسمت دستورات تک دستوری باشد می تواند دیگر آکلاد قرار ندا.

نکته: if، else کلاً برای برنامه C++ یک دستور است و کامپایلر فقط یک از این دو را اجرا می شود همزمان غیر قابل اجرا هستند.

نکته: قسمت else اختیاری است یعنی if در حالت true تنها عملیات انجام دهد ولی در قسمت false چنین حالتی نداری

نکته: در هر برنامه تنها یك ;0 return اجرا می شود حتی اگر چندین ;0 return در برنامه باشد.

* ;0 return به این معنا است که برنامه تمام شد و صحیح اجرا شد و اگر این دستور در وسط برنامه باشد دستورات بعد از خودش را دیگر اجرا نمی کند.

برنامه (password) :

```
Const int password;
cout<<inter password;
int p;
cin >> p;
if (p==password)
{
cout<<"welcome to my program";
}
else
{
cout<<"sorry you can't continue";
}
return 0;
```

: getch() , getche()

برای آنکه دو تابع را تعریف کنیم از "#include "conio.h استفاده می کنیم.

نکته: ( ) getche کار cin را می کند با این تفاوت که getche تنها ۱ کاراکتر را خوانده و در متغیر مربوطه قرار می دهند ولی cin چندین داده را هم می تواند دریافت کند.

```
getche ( ) ;
cin>> x;
```

getche ( )
getch ( ) با هم تفاوت دارند ولی هردو آنها باید از نوع char باشند.

*اگر از ( ) getch استفاده کنیم چیزی در صفحه نمایش دیده نمی شود ولی در ( ) getche چیزی که تایپ می کنیم روی صفحه نمایش هم دیده می شود و از ( ) getch می توان برای password نویسی نیز استفاده کرد.

نکته: در صفحه ۱۷۳ علت اینکه از while از cin استفاده نمی کنیم و ( ) getche را جایگزین می کنیم این است که cin تمام پاراگراف را پشت سر هم می خواند و دیگر نمی تواند محاسبه کند سپس از ( ) getche استفاده می کنیم تا کاراکتر به کاراکتر محاسبه کنیم تا فرصت محاسبه را پیدا کند.

if های تو در تو:

```
if (شرط اول)
{

}
else if (نقیض شرط اول)
{

}
else
{

}
```

مثال:

```
float  x;
if (x<=20 && x> = 15)
cout << "علی" ;
```

بهتر است این شکلی بنویسیم:

```
else
        If (x <15 && x>=10)
        cout << "خوب";
else
        cout << " رد";
```

*مهم است if و else های تو در تو را خوب بنویسیم تا کامپایلر محاسبات بی مورد انجام ندهد.

شکل بهتر:

```
if (x> =0 && x< =20)
{
        if (x > 15)
        cout << "علی" ;
        else
                if (x> = 10)
                cout << "خوب" ;
        else
                cout << "رد" ;
}
else
        cout << " عدد را اشتباه وارد کردید" ;
```

نکته:

```
if (شرط ۱)
{
…..
}
else
if (شرط ۲)
{
……
……..
}
else
{
……
……..
}
```

نکته مهم: همواره کامپایلر else را برای نزدیک ترین if به خود در نظر می گیرد.

```
if (x>10)
if (y>10)
cout << "A" ;
else
cout<<"A";
 else
Cout<<"B" ;
Cout<<"C";
```

برای داده های زیر داریم:

X=5 ; y=8 ; → cout → C

X=20 ; y=5 ; → cout → 1- B 2- C

X=2 ; y=30 ; → cout → 1-A 2- C

```
If(x>10)
        If(y>10)
                If(z>10)
                cout << "A" ;
cout <<"B" ;
```

→

```
if(x>10  && y>10 &&  z>10)
cout << "A" ;
cout << "B" ;
```

X=5; →B

X=12 , y=7 → B

X=17 , y=15 , Z=4 → B

X=20 , y=20 , Z=20 → A,B

روش خوبی نیست چون محاسبات اضافه انجام می دهد و این موضوع به این دلیل است که از else استفاده نشده است فرم صحیح را در زیر آکلاد اول مشاهده کنید.

```
cin>x;
if (x==1)
        cout <<"A";
if (x==2)
        cout  <<"B";
if (x==3)
        cout << "C";
```

این برنامه صحیح تر است چون هم سریع تر اجرا می شود چرا که کامپایلر الکی معطل محاسبات اضافی نمی شود و هم برنامه خواناتر است:

```
cin>> x;
if (x==1)
        cout<<"A";
else
        if(x==2)
                cout<<"B";
else                    → else
        cout <<"C";       if(x==3)
                                Cout << "C";
```

← این برنامه میگه اگر x=1 بود A را چاپ کن.

```
cin >> x;
if(x==1)
        cout <<"A" ;
```

اگر x=2 بود B را چاپ کن و اگر x=3 بود c را چاپ کن.

```
if(x==2)
        cout << "B" ;
```

و هر چه جز این بود d را چاپ کن.

```
if(x==3)
cout << "C" ;
if(x!=1  &&   x!=2  &&   x!=3)
        cout << "d" ;
```

دستور Switch:

این دستور می تواند کار if های تو در تو را انجام دهد ولی این دستور ۲ محدودیت بزرگ دارد و آن هم :

۱- فقط برای اعداد صحیح قابل استفاده است.

۲- فقط شرط برابری را چک می کند.

```
        switch(x)
{
case 1:
        cout << "A" ;
           break;
case 2:
        cout << "B" ;
          break ;
case 3:
        cout << "C" ;
         break;
default :
        cout << "D" ;
}
```

نکته مهم: در دستور Switch تنها راه خارج شدن از caseاستفاده از دستور break است و در صورتی که دستور break را فراموش کنیم دستورات case بعدی هم در همین Case اجرا و محاسبه خواهد کرد.

دو محدودیت بزرگ Switch

برنامه زیر را با Switch نمی توان نوشت چون Switch قدرت محاسبه و پردازش بازه ای از اعداد را ندارد.

```
if(scare > = 15)
        cout << "خوب" ;
if (scare > = 10)
cout << "متوسط" ;
else
        cout << "رد" ;
```

فرض می کنیم x بین ۱ تا ۶ است و این برای حالتی که ما بخواهیم برای چند عدد حالت و کار مشابهی انجام دهیم که دیگر نیاز نیست تک تک برای همه انها دستور case و break را انجام دهیم و می توانیم به فرم زیر بنویسیم.

```
cin >> x;
Switch (x)
{
        Case 1:
        Case 3:
        Case  5:
                Cout <<"odd number"
                Break;
Case 2:
Case  4:
Case  6:
        Cout << "even number" ;
        Break;
}
```

توجه داشته باشیم break آخر را ننویسیم هم مهم نیست.

کاراکترها (char) فشرده شده int هستند پس می توان همچین چیزی را هم نوشت:

```
char  ch;
cin >> ch;
swich (ch) → ch=getch ( )
{
case (y):
        cout << "yes" ;
        break;
case (n):
        cout << "No" ;
        break;
default:
        cout << "error" ;
  }//end of switch
```

نکته: این برنامه به درداین می خورد که هنگامی while استفاده می کنیم فقط در صورتی که yes یا No وارد کند ادامه برنامه را می بینید.

نکته: if و else های خیلی ساده را با دستور زیر هم می توان نوشت:

اگر غلط باشد چه شود:اگردرست باشد چه شود؟(شرط)

```
if (x>10)
        cout << "A" ;
else
        cout << "B" ;


(x>10) ? cout << "A" : Cout << "B" ;
```

توجه: برای if و else های تک دستوری روش فوق قابل استفاده است ولی کلاً روش خوبی نیست.

تابع gotoxy(x,y) :

جزء توابع تعریف شده C++ نیست حال برای آنکه شناخته شود باید دستور زیر را عیناً قبل in main ( ) و بعد از ; using namespace std بنویسیم.

```
void    gotoxy (int  x ,  int  y)
{
HANDLH hConcole Output ;
COOR D dw Cursor Position ;
Cout. Flush ( ) ;
Dw Cursar Position . X=x;
Dw Cursar Position . Y=y;
hConsole Output = GetStd Handle (STD –OUTPUT-HANDLE);
Set Console Cursar Position (hConsole Output, dwCursar Position );
}
```

در قسمت # include "windows.h" را بنویسیم.

مثال : برنامه ای که با فشردن کلمه R,L,D,V یک علامت ☺ را در صفحه نمایش به ترتیب به سمت بالا ، پایین، چپ، راست می برد.

```
# include "Stdafx.h"
# include "iostream"
# include "Conio.h"
# include "windows.h"
usingname space std;
Void gotoxy (int x , int y)
{
HANDLE hConsole Output;
```

```
COORD dwCursar Position;
Cout. Flush ( );
Dw Cursar Positin .X=x ;
dwCursar Position .Y=y ;
h ConsoleOutput=GetStdHandle(STD-OUTPUT-HANDL);
Set ConsoleCursorPosition(hConsoleOutput,dwCursorPosition);
}
Int main(void)
{
System ("CLS");
gotoxy(0 , 25);
```

کدهای اسکی ↑ و ↓ و ← و → است.

```
char V=24 , D=25 , R=26 , L=27 ;
cout << "Use these keys for more:"<< "VI" <<"v": <<u<< "D:" <<D<< "R:" << "L:" << l;
Char   sg=1 , ch=0 ;
```

Sg: کد اسکی ☺است.

```
Int x:39   ,   y=11;
Go to  xy(x,y) ;
Cout. Put(sg);
```

چون بعد چاپ (۳۹/۱۲) می شود و ما می خواهیم روی آدمک باشد تا برابر بالا شود باید دوباره (۳۹/۱۱) شود.

```
gotoxy(x,y);
While ((ch=getch ( )) !=27)→
```

۲۷:کداسکیspaceاست.

{

if (ch == 'V' || ch=='v') && y>1)

چون y نقش سطر را بازی می کند و ما از سطر ۱ بالاتر نمی توانیم برویم.

{

cout= put(' '); → می شود.☺باعث پاک شدن

gotoxy (x , --y);

cout.Put(sg);

gotoxy(x , y); →

مکان نما یکی بیاد عقب تا قبل ت قرار بگیرد تا بتوانیم☺ را حرکت بدهیم.

}

else

if ((ch=='D' || ch == 'd') && y<24)

{

cout. Put (' ');

gotoxy (x , ++y);

cout . Put (sg);

gotoxy (x , y);

}

else

if ((ch == R)|| ch == r)&& x<79)

{

cout.Put (' ') ;

gotoxy (++x , y);

```
cout. Put (sg);
gotoxy (x , y);
}
else
            if ((ch== 'L' || ch='l') && x>0)
           {
            cout.Put ( '  ' );
            gotoxy ( -- x, y);
            cout. put(sg);
            gotoxy( x , y);
            }
}
system (" cls");
system ("pause");
return 0;
}
```

تولید اعداد تصادفی:

در C++ تابعی به نام ؛( ) rand داریم که نحوه استفاده از آن به صورت زیر است:

```
int x;
x= rand ( );
```
⇒
```
int x=rand ( )
```

حال اگر بخواهیم اعداد تصادفی داده شده در بین بازه خاصی باشد از «%» استفاده می کنیم مثلاً اگر بخواهیم اعداد تصادفی بین (۱۰-۰) باشد می نویسیم:

```
int x= rand( ) % 10;
```

اعداد تصادفی بین ۱۵ تا ۲۵:

```
int x=(rand%9)+ 16;
```

نکته: به چند برنامه زیر که همگی اعداد تصادفی تولید می کند و ما برنامه را چندین بار run می کنیم دقت کنید:

```
int   x,y,z;
x=rand( ) %10 ;
y=rand ( ) %20 ;
z=x+y ;
cout << "z" ;
```

```
int  x,y,z ;
for(int i=1 ;  i<4  ; i++)
{
    x=rand(  ) %10 ;
    y=rand(  ) %20 ;
    z= x+y ;
    cout << "z" << z ;
}
```

همان طور که مشاهده می شود هر دفعه که برنامه دوباره از اول run می شود همان اعداد تصادفی قبلی را می دهد این در حالی است که هنگامی که داخل برنامه هستیم همواره اعداد جدیدی تولید می کند علت این موضوع این است که الگوریتمی که برای تابع ( ) rand تعریف شده الگوریتم واحدی است و معیار تولید عدد تصادفی از اولین عدد پیش فرض شده در الگوریتم مثلاً صفر است و چون هر دفعه با یک صفر یک الگوریتم واحد اجرا می شود اعداد همواره یکی است حال برای حل این مشکل از تابع ( ) Srand استفاده می کنیم که در این حالت مبنای تولید عدد تصادفی هر بار تغییر کرده و در نتیجه بعد از هر بار run شدن اعداد جدیدی دریافت خواهیم کرد.

نحوه استفاده:

```
srand (10);
int x= rand (  ) ;
```

حال مثال نکته قبل را با srand() ببینید:

```
int   x,y,z ;
srand (20);
for(int i=1 ; i< 4; i++)
      {
            x=rand (  )% 10;
            y=rand (  )% 20;
            z=x+y ;
            cout << z ;
       }
```

با کمی دقت متوجه می شویم که هنوز هم مشکل قبل وجود دارد چرا که ما تنها برای ۱ بار معنای الگوریتم را تغییر دادیم و از این به بعد مبنای الگوریتم عدد مشخص شده ماست ولی همواره بعد از هر دفعه Run شدن اعداد تکراری خواهد بود!

```
int   x,y,z,b;
cout << "Enter base" ;
cin >> b ;
srand (b) ;
x=rand ( ) ;
y=rand ( ) ;
z=x+y ;
cout << z ;
```

	باراول	باردوم	بارسوم
	b=2	b=4	b=2
	x=5	x=3	x=5
	y=7	y=4	y=7
	z=12	z=7	z=12

همان طور که مشاهده می شود به نظر می رسد مشکل حل شده است ولی اگر کمی دقت کنیم متوجه می شویم ممکن است b تکراری تعریف شود و در نتیجه چون مبنا الگوریتم تکراری می شود اعداد یکسان هم تعریف کند.

نکته: در کامپیوتر زمان همراه تاریخ ذخیره می شود پس هیچگاه تکراری نخواهد بود.

توجه: برای آنکه شکل صفحه قبل پیش نیاید به تغییری نیازمند هستیم که همواره تغییر کند و هیچ گاه تکراری نباشد تا در نتیجه مبنای الگوریتم تکراری نشود و اعداد تکراری تولید نکنیم.

سپس می توانیم در مثال قبل مبنا را time بگذاریم تا همواره تغییر کند در مبنای تابع ()srand را time قرار دهیم.

```
Srand (time(0));
```

این تابع باید در ابتدای برنامه تعریف شود و نحوه تعریف آن استفاده از دو include زیر است:

```
# include ("time .h")
# include ("Ctime")
```

دونکته:

۱ ← به معنای آن است که شرط درست است(True)

۰ ← به معنای آن است که شرط غلط است(False)