آرایه ها:

برای تعریف متغیرهایی که چند مقدار هم نوع در آنها جا شود می توان از آرایه ها استفاده کرد.

تعریف آرایه:

مجموعه مرتب و محدودی از عناصر هم نوع هستند.

محدودیت: تعداد عناصر آرایه از قبل باید معلوم باشد.

مرتب:عناصر آرایه در حافظه پشت سر هم قرار می گیرند.

در حالت عادی :

int x_1 , x_2 , x_3 , x_k ;

double y;

char z;

کاملاً تصادفی،هر داده در قسمتی از RAM ذخیره می شود.

اما هنگام استفاده از آرایه ها همه به ترتیب و پشت سرهم قرار مس گیرند :

int x[4];

double y;

char z;

نحوه استفاده از آرایه ها:

نوع داده نام آرایه [طول آرایه];

مثلاً:

```
int     A[10];
```

A آرایه صحیح و ده عنصری است.

نکته: اندیس آرایه ها از صفر شروع می شود مثلاً ۱ آرایه k عنصری ۰ – ۳ تا خانه دارد.

0	1	2	3
2.5	183.75	100.1	4.3

```
B[0]=2.5
B[1}=183.75
B[2]=100.1
B[3]=4.3
```

نکته: نمی توانیم یک دفعه به تمام (آرایه) داده های آرایه دسترسی پیدا کنیم و اگر بخواهیم عملیاتی انجام دهیم تک تک باید عملیات را انجام دهیم مثلاً برای آرایه بالا داریم.

```
B[0]=B[0]*2;
```

***نکته:B نام اولین خانه آرایه است و مثلاً وقتی می گوییم B[2] یعنی اینکه کامپایلر از خونه B[0] دو تا خونه به جلو حرکت کند.**

از لحاظ مدیریت برنامه استفاده از آرایه بهتر است مثلاً گاهی هم تعداد برنامه ها زیاد است و هم اینکه باید همه این ها در برنامه حفظ شود.

روش online: داده همان لحظه استفاده شده و بعد از آن دیگر در حافظه ذخیره نمی شود.

روش offline: داده ها را گرفته نگهداری می کند و بعداً استفاده می کند.

int A[10]; 0 1 2 3 4 5 6 7 8 9

```
for < int i=0 ; i<10 ; i++)
    cin >> A[i];
{int   S=0 ;
for  (int j=0 ; j<10 ; j++)
s= s+A[10];
{A[5]=A[5]+A[1]+A[9]
for (int k=0  ;  k<10  ; k++)
cout << A[k] ;
```

داده ها (اعداد) فقط هنگام استفاده از برنامه «اجرا شدن برنامه» ذخیره می شود و دیگر بعد از اجرا شدن برنامه و تمام شدن آن دیگر save **نیست.**

مقدار دهی اولیه آرایه ها:

روش کار:

int A[4] ={3,7,10,2}

0	1	2	3
3	7	10	2

int A[4]={3,7}

0	1	2	3
3	7	0	0

int a[0]={0};

0	1	2	3
0	0	0	0

همواره وقتی متغیری تعریف می کنیم مقدار دهی اولیه آن را صفر قرار می دهیم تا هنگام انجام عملیات، عبارات محاسبه شده و درست محاسبه شود.

```
int   A[8] = {10,20,30,40}
```

0	1	2	3	4	5	6	7	8
10	20	30	40	0	0	0	0	

```
for (int  i=2  ;  i<5  ;i++)

cin >> A[i] ;
```

0	1	2	3	4	5	6	7	8
10	20	30	3	9	0	0	0	

```
srand (time(0)) ;

for (int i=2 ; i<9 ; i++)

    A[i]= rand (  ) %20  ;
```

0	1	2	3	4	5	6	7	8
10	20	30	11	3	12	10	16	

نکته: علت اینکه باید srand(time(0)) را قبل از for یا حلقه های دیگر قرار دهیم این است که اگر داخل حلقه قرار گیرد در یک لحظه مبنای محاسبه عدد تصادفی را تغییر داده و در نتیجه محاسبات اشتباه می شود.

نکته: در آرایه ها همواره باید در ابتدا طول آرایه مشخص باشد.

```
int    A[4]={10,20,30,40}
cout << A[2] ; → 30
cout << A[0] ; → 10

int   A[  ] = {4,5,7}
```

به طور خودکار A را به طول ۳ قرار می دهد.

توجه به این نکته ضروری است که در صورتی طول آرایه را می توانیم معین بکنیم که خودمان مقدار دهی اولیه بدهیم در غیر این صورت خطای برنامه نویسی رخ داده است.

هر دو، مقدار اولیه را صفر قرار می دهند ولی اولی بهتر است.

```
Int    A[10]= {0};
Int    A[10]= { };
```

ایجاد تاخیر در برنامه:

یکی از روش های بسیار بد ، استفاده از for به شکل زیر است.

For(in i=0 ; i<1000,000 , i++) ; علت بد بودن این روش این است که میزانی تأخیر می اندازد. و البته به سرعت کامپیوتری دارد که کامپایلر بر رویش نصب شده است به همین دلیل اصلاً مناسب نیست.

تابع Sleep :

این تابع بر اساس میلی ثانیه کار می کند و برای شناسایی آن به هدر فایل :

```
#include "windows.h"
```

Sleep (**یک عدد صحیح**) ;

مثلاً : Sleep (1000) ; → **۱۰۰۰ میلی ثانیه**

```
cout << "B" ;
```

نکته: برای محاسبه میزان فضای اشغال شده توسط آرایه ها می توان به شکل زیر عمل کرد:

int x[10]; ⇒ 10 ×4 =40 byte

char y[20]; ⇒ 20 ×1 =20 byte

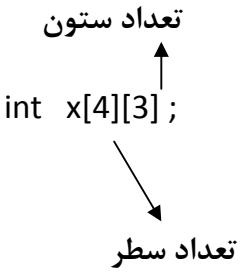
float z[50]; ⇒ 50 ×4 =200 byte

double h[100]; ⇒ 100 ×8 =800 byte

می شود از دستور sizeof **هم استفاده کرد:**

```
cout << sizeof (x) ; ⇒ 40
```

آرایه های دو بعدی:



مقدار دهی اولیه به آرایه های دو بعدی:

تک بعدی:

⇐ int x[10] ;

ورودی از کاربر: ←

```
For (int i=0 ; i<10 ; i++)
Cin >> x[i] ;
```

```
Srand (time (0);
For (int i=0  ; i<10  ;  i++)
X[i]=rand (  )%100;
```

دو بعدی:

Int x[10][20]; **ورودی از کاربر** → for(int i=0 ; i<10 ; i++)

```
 for(int j=0  ; j<20 ; j++)
cin >> x[i][j];
srand (tim (0)) ;
for(int i=0  ; i<10 ; i++)
for(int j=0  ; j<20 ; j++)
x[i][j]=rand( ) %20 ;
```

مقدار دهی اولیه توسط برنامه نویس در آرایه ۲ بعدی:

```
int x[3][4]={4,5,6,7,8,9,10,12,18}
```

نحوه پر شدن :

4	5	6	7
8	9	10	12
18	0	0	0

نکته: همان طور که مشاهده می کنید اگر اندازه آرایه بزرگتر از مقدار دهی اولیه توسط برنامه نویس باشد مشکلی پیش نمی آید و جای خانه های اضافی صفر قرار می دهد ولی عکس این حالت یعنی مقدار دهی اولیه توسط کاربر بیشتر از طول آرایه تعریف شده باشد غلط است و برنامه error **خواهد بود.**

حال اگر بخواهیم در هر سطر ۱ مقدار اولیه بدهیم:

برای این منظور مقدار دهی را به صورت زیر می نویسیم « هر آکلاد داخلی بیانگر ۱ سطر است» استفاده از این روش مناسب تر است زیرا برنامه خواناتر خواهد شد.

int A[3][3]= {{4},{3},{7,8}};

4	0	0
3	0	0
7	8	0

اگر آکلادهای داخلی را نگذاریم به شکل زیر چاپ خواهد شد:

4	3	7
8	0	0
0	0	0

یافتن میانگین با استفاده از ۲ for :

```
Int   x[10][5] ;
Float  A[10];
Srand (time (0) );
for(int i=0  ; i<10 ; i++)
{
Int   S=0  ;
for(int j=0  ; j<5 ; j++)
S+=x[i][j]
```

Int average= float (s) / 5;

A[i] = average ;

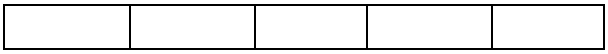
}

رشته ها «مبحث بسیار مهم »

رشته ها:

نوعی از آرایه ها هستند نوع آنها کاراکتر است.

Char S[5];



Char S[5]= {‘A’ , ‘B’ , ‘C’ , ‘D’ , ‘E’}

A	B	C	D	E

Char S=[5]={‘A’,’B’}

A	B	?	?	?

یکی از راههایی که کاراکتر عددی را به خود عدد تبدیل کنیم تا قابل محاسبه باشد روش زیر است:

```
switch  (ch) ;
{
case (0) ;
X=0 ;
Break;
case (1);
X=1 ;
break;
……
……..
}
```

تولید کاراکتر تصادفی:

برای این منظور می توانیم کد اسکی کاراکترهای مورد نظر را در بیاوریم مثلاً برای a-z کد اسکی ۹۸-۱۲۲ است حال با استفاده از ((srand(time(0 عددی تصادفی بین ۸۹-۱۲۲ تولید کرده و بعد آن را در متغیر (آرایه) کاراکتری ذخیره می کنیم.

نحوه استفاده از رشته:

```
char S[5]="Ali" ;
cout << S;  → Ali
```

یک آرایه کاراکتری است که در ۳ کاراکتر اول حروف Ali قرار گرفته و سپس کامپایلر به طور خودکار "10" null را در انتهای حروف قرار می دهد که نشانگر این است : کلمه تمام شده است.

```
char    S[5] = {‘A’,’L’,’I’}
for (int  i=0  ; i<5  ; i++)
cout << S[i] ; ⇒ Ali+*
```

همان طور که می بینید اگر از آرایه استفاده کنیم برای چاپ آرایه حتماً نیاز به یک for داریم و بعد از چاپ شدن علاوه بر Ali دو کاراکتر بی مورد هم چاپ خواهد کرد.

null «\ 0» :

«0 \ » بیانگر استفاده از رشته است چون باید همواره در رشته «o \ » وجود داشته باشد حواسمان باید باشد که طول رشته همواره باید یکی بشه از حروف کلمه باشد مثلاً برای کلمه reza باید یک رشته به طول۵ تعریف کنیم تا در خانه پنجم"\0" قرار گیرد و یا مثلاً برای یک رشته به طول ۲۱ حداکثر کلمه ای با حروف ۲۰ حرف جای خواهد گرفت.

```
char S1 [10] , S2[10];
for (int i=0 ; i<10 ; i++)
cin>> S1[i];
cin >> S2 ;
char S3[10] ={ (c),(+),(+)};
char S4[10]="C++";
for(int j=0 ; j<10 ; j++)
cout << S1[j];
cout << S2;
for (int k=0 ; k<10 ; k++)
cout << S3 [k] ;
cout << S4 ;
```

کاراکترهای فضای سفید:

کاراکترهایی که بین دو مقدار فاصله می اندازند مثل:

Space , tab , enter و .....

1- cin. get (**طول آرایه** و **نام آرایه**)

مورد استفاده دستور بالا برای این است که اسم هایی همانند علی رضا که فاصله برای خود کلمه است فاصله را نیز کاراکتر حساب کند تا علی و رضا کنار هم با یک نیم فاصله چاپ شود نه اینکه علی جدا و رضا هم جدا طوری که انگار ۱ اسم هستندچاپ شوند.

```
char S[20];
cin. get (S,15);
```

همان طور که در بالا ذکر کردیم در این روش کامپایلر آنقدر از کاربر حرف می گیرد تا به طول آرایه تعریف شده برسد حال آنکه ممکن است کلمه مورد نظر کاربر ۳ حرفی باشد برای حل این مشکل می توانیم از روش زیر استفاده کنیم:

2- cin. get (**طول آرایه** و **نام آرایه**) ;

```
cin . get (S,15,'*');
```

کاراکتر جدا کننده در واقع شرط پایان است که خودمان تعیین می کنیم و یادمان باشد همواره ۱ کاراکتر کم می شود از کاراکترهای داده شده چون از آنجا شرط تمام است.

نکته : طول رشته را تنها هنگامی که برنامه نویس مقدار دهی اولیه می کند می توان ننوشت در این حالت کامپایلر طول رشته را از فرمول زیر محاسبه خواهند کرد.

تعداد حروف کلمه از پیش تعیین شده توسط برنامه نویس+۱

نکته: در یک آرایه اگر نحوه تعریف به شکل زیر باشد که"\0" خودمان دستی اضافه کنیم آرایه به رشته تبدیل خواهد شد.

```
char  S4 [  ]  = {‘p’,’r’,’a’,’y’,….,’\0’}
```

1- `char  A[10]={‘A’,’B’,’C’,’D’}⇒`
   `for (int i=0 ; i<10 ; i++)`
   `cout << A[i]; ⇒A B C D (2  3  7  *  +`

2- `char  B[b]="ABCD"`
   `cout << B;` → ABCD

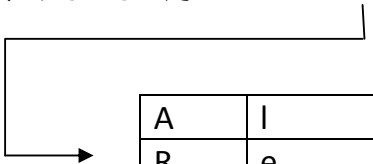
3- `char  C[10] ={‘A’,’B’,’\0’};`
   `cout << C;` → AB

1-`Char  y[3][5] = {{‘A’,’L’,’I’},{‘R’,’C’,’Z’,’a’},{‘M’,’o’,’N’,’a’}};`
2-`Char  y[3][5] = {"Ali";"Reza","Mona"};`

فرم ۱ رضا را چاپ کند.: `for (int i=0 ; i<5 ; i++)`

`Cout << y[1][i] ;` →Reza2

فرم ۲ رضا را چاپ کند.: `cout <<(y 1;)` → Reza



A	l	i		
R	e	z	a	
M	o	n	a	

String:

یک نوع داده است همانند سایر Data taype ها ولی واقعاً داده نیست و به آن class می گویند چرا که Data taype هایی که قبلاً هم تعریف کرده بودیم به طور مستقل تعریف می شدند اما String یک نوع مستقل نیست و از روی سایر داده ها درست می شود و در واقع انشعابی از یک آرایه ی کاراکتری است.

علت استفاده از دستور "string" آن است که دیگر برنامه نویس درگیر مدیریت حافظه داده ها نشود . در رشته ها چون مقداری مدیریت داده ها سخت تر است از string استفاده می کنیم تا کار آسانتر شود.

برای String سر فایل های رو به رو موجود است:

1- # include "string.h" **یا** 2- # include "cstring"

```
Char  A[10] = "Ali" ;          |    String  A; "Ali";
Char  B[10];                   |    String  B;
B=A ;          غلط             |         B=A;   صحیح
```

در صورت انجام این کار باید از حلقه While استفاده کنیم:

```
int  i=0 ;
while (A[i]! = '\0')
{
B [i]=A[i];
i++ ;
}
B[i] = '\0' ;
```

مثالی از عملگر الحاق در String :

```
String  A; "Ali" ;
String  B; "Reza" ;
String  C= A+B ;
Cout << C; → Ali Reza
```

تابع های String:

۱- **Strcpy : یک آرایه رشته ای را به یک آرایه رشته ای دیگر Copy می کند.**

```
Char B[10]="Ali" ;
Char A[10] ;
```
فقط برای آرایه ها (رشته ها) کاراکتری قابل استفاده است. Strcpy (A,B);

۲- Stremp: مقایسه دو رشته

یک عدد صحیح = Strcmp (A,B);

مثال:

```
int     x=strcmp (A, B)
        if (x<0)
        cout << "A<B";
        if (x=0)
        cout << "A=B";
        if (x>0)
        cout << "A>B";
```

نکته: نحوه مقایسه دو کلمه و عدم اینکه یکی از آن دو از دیگری بزرگتر است. مثلاً هنگامی که بخواهیم کلمات را بر طبق حروف الفبا طبقه بندی کنیم مورد استفاده قرار گیرد همان نحوه قرار گیری حروف الفبا است z-a

۳- Strcat: کار متصل کردن دو رشته را دارد.

Strcat (B,A);

نحوه عمل کردن تابع به این شکل است که محتویات B را به انتهای رشته A از مکانی که علامت Null **قرار گرفته اضافه می کند به عنوان مثال اگر** A:Computer **و** B:Scince **باشد بعد از استفاده از این تابع کلمه:** Computer Since **را خواهیم داشت.**

۴- Strncpy: همان کار تابع Strncpy **را می کند و تنها** n **موجب می شود که** n **حرف اول رشته دوّم را به** n **حرف اول رشته اول کپی می کند.**

Strncpy (B,A,n);

۵- **Strncmp**: همان کار Strncmp را می کند فقط n موجب می شود تعداد کاراکتر مقایسه ای را خود انتخاب می کنیم.

Strncmp(A,B,n)

۶- **Strnicmp**: این تابع همانند تابع Strncmp عمل کرده با این تفاوت که مقایسه دو رشته را بدون توجه به کوچکی و بزرگی حروف انجام می دهد به همین دلیل دو رشته: "ali" , "Ali" برای این تابع یکسان است.

۷- **Stncat**: این تابع نیز تعداد مشخصی کاراکتر را از ابتدای رشته مبدأ به انتهای رشته مقصد متصل می کند و به شکل کلی زیر کار می کند.

Strncat (**تعداد کاراکتری که باید متصل شود** و **رشته مقصد** و **رشته مبدأ**)

توابع:

- طراحی شده توسط کاربر
- طراحی شده توسط کامپایلر

می دانیم که برنامه C++ دارای تعداد بسیاری تابع است پس این سؤال مطرح می شود که چرا با این وجود در بعضی از موارد خود برنامه نویس تابع را طراحی می کند؟

برای پاسخ گویی به این سؤال ۲ مورد مطرح می شود:

۱- **توابع آماده c++ اکثراً تنها نیازهای پایه ای و بنیادی برنامه نویس را حل می کنند حال آنکه در بیشتر موارد برنامه نویس نیاز به توسعه این تابع دارد پس مجبور می شود برای رفع نیاز خود، خود تابع را طراحی کند.**

۲- **استفاده از تابع های آماده C++ سرعت اجرا برنامه را پایین می آورد چرا که برای توابع آماده را کامپایلر خود برنامه های از پیش تعیین شده که عموماً حجم زیادی هم در خود اشغال می کنند را به ابتدای برنامه می چسباند و این موجب طول کشیدن عمل اجرا می شود.**

تابع main: تابع اصلی است که حتماً باید در برنامه وجود داشته باشد چرا که در غیر این صورت اصلاً برنامه اجرا نخواهد شد.

ساختار یک تابع:

نوع خروجی نام تابع (پارامترهای ورودی)

{

بدنه تابع

}

نکته: تابع Strcpy هیچ مقداری را بر نمی گرداند .

نکته: قواعد نام گذاری توابع همانند قواعد نام گذاری نام متغیرها است فقط استفاده از:

۱- حروف کوچک

۲- حروف بزرگ

۳- اعداد ۰-۹

۴- اندر لا ین(_)

توجه: باید توجه داشت که هنگامی که از تابع استفاده می کنیم در واقع تابعی که برنامه نویس تعریف کرده است، باید تابع توسط main صدا زده شود «Call شود» این در حالی است که در برنامه نویسی معمولی چنین چیزی مطرح نیست.

مثال: برنامه ای بنویسید که دو عدد را جمع و حاصل را چاپ کند.

← بدون استفاده از تابع:

```
int  main (void)
{
int   x,y,z;
cin>> x>>y ;
z=x+y ;
cout << z;
return 0;
}
```

← با استفاده از تابع:

```
int    Sum (int   x,   int  y)
}
 int z;
z=x+y;
return z;
}
int      main (void)
{
int   a,b,c ;
cin >> a >> b ;
```

```
فراخوانی تابع ← c=Sum (a,b);
cout << c ;
return  0;
}
```

نکته: متغیرهای مورد استفاده در تابع متغیرهایی سوری هستند چرا که در واقعیت وجود ندارد ولی متغیرهایی که در main قرار دارند متغیرهای واقعی هستند به همین دلیل دوگانگی نام در تابع و main مشکلی به وجود نمی آورند و تضادی برای برنامه به وجود نمی آورد.

ما به ۴ روش می توانیم تابع تعریف کنیم:

۱- تابع هم ورودی داشته باشد و هم خروجی

۲- تابع ورودی داشته باشد ولی خروجی نداشته باشد.

۳- تابع خروجی داشته باشد ولی ورودی نداشته باشد.

۴- تابع ورودی و خروجی نداشته باشد.

- **ورودی ندارد ، خروجی دارد:**

```
int      Sum(void)
{
int  x,y,z,
cin >> x >> y  ;
z= x+y ;
return Z;
}

int main (void)
{
int a;
//calling
a=Sum ( );
cout << a;
return 0;
}
```

- **خروجی ندارد ولی ورودی دارد:**

```
void  Sum (int x   , int y)
{
int z;
z=x+y ;
 cout << z;
}
int     main(void)
{
int  a, b;
cin >> a >> b;
//calling
Sum (a,b);
return 0;
}
```

- **خروجی و ورودی ندارد:**

```
        void  Sum (void)
        {
        int  x,y,z ;
        cin >> x >> y ;
        z=x+y;
        cout << z;
       }
int    main  (void)
{
Sum (  ) ;
return 0;
}
```

برای مکان تعریف تابع ۲ حالت داریم:

1- int Sum ( )
{
……
……
}
int main (void)
{
فراخوانی x= Sum ( ) ;
تابع }

تعریف تابع

2- int Sum ( ) ; اعلان تابع
int main (void)
{
…….
فراخوانی تابعx=sum ( ) ;
}
int Sum ( )
}
…….
…….
}

تعریف تابع

مثال: استفاده ساده از کاربرد تابع:

```
double  fact (int  x)
{
double   f, 1;
for (int i=1     ;   i< x  ; i++)
f  *=i ;
return  f ;
}
bool   odd (int y)
{
      if (y %2  ==0)
return    false ;
else
            return true ;
        }

        int main (void)
        {
        int a;
        cin >> a;
        double d= fact (a) ;
        cout<< d;
        bool   K;
        K= odd (a) ;
        if (K= true)
        cout <<"odd" ;
        else
        cout<<"even";
```

```
int  h;
cin >> h;
d= fact (h) ;
cout<<”d”;
return 0;
}
```

فراخوانی تابع با مقدار(call by value) :

```
void  change (int x)
{
x++ ;
}

int  main (void)
{
int a= 10 ;
cout << a ; → 10
a++ ;
cout << a; → 11
change(a)→12
cout << a; → 11
return 0;
}
```

یعنی مقدار a فقط در تابع change قابل تغییر است یعنی تغییر کرده است و این تغییرات در main بروز نکرده است و مقدرر a در main همان است که قبلاً هم بود.

این روش نکته مهمی است در برنامه نویسی و کارش این است که مقداری که داخل متغیر است را به تابعی که خودمان نوشتیم می دهد «توجه داشته باشید که این تابع تعریف شده باید خروجی اش Void باشد» چون این تابع void است و خروجی به main بر نمی گرداند متغیر در تابع تغییر کرده ولی دیگر تغییراتش در main تأثیر نخواهد گذاشت.

- اگر بخواهیم تغییرات در main نیز وارد شود:

```
int change (int  x)
{
x++ ;
cout << x;
return x;
}
int   main  (void)
{
int  a=10 ;
cout << a; → 10
a++ ;
cout << a; → 11
int   b=change (a)
cout << a; → 11
cout << b; → 12  ;
return  0;
}
```

نکته: تابعی که Call by value نباشد call br reference گوییم.

نکته: آرایه ها call by reference **هستند. یعنی تنها به این شکل ذخیره می شوند.**

```
int   f1  (int  x)
{
 return  x++ ;
}
int   f2  (int  x)
{
 return  x-- ;
}
int   main  (void)
{
int   x=10;
cout << f1 (x) ; → 11
cout << f2 (x) ; → 9
int   y = f3 (x) ; → 20
return 0;
}
```

اگر بخواهیم تابعی را call **کنیم که خروجی دارد یا باید تابع را جلوی متغییری از جنس تابع بگذاریم و یا باید تابع را در داخل** cout **قرار دهیم.**

ارتباط آرایه ها و توابع:

یک عنصر آرایه وارد تابع شود :

```
int   A[4] = {2,7,10,15}
vaid  change  (int  x1 ,  int  x2  , int  x3  , int  x4}
{
x1 ++ ;
x2 ++ ;
x3 ++ ;
x4 ++ ;
cout << x1<< x2 << x3 << x4 ;
}
int main()
{
int A[4]  = { 2,4,6,8}
change (A[0] , A[1] , A[2] , A[3]);
int   a=10  , b=20  , c=30  , d=40  ;
Change  (a,b,c,d);
return a;
}
```

این روش ، روش خوبی نیست علتش هم این است که ما ۴ متغیر داریم و این موضوع سبب می شود که برنامه حالت عمومی پیدا نکند و این یک اشکال بزرگ است.

انتقال تک تک عناصر به به آرایه :

```
void  change  (int x)
{
cout  << x++ ;
}
int  main (void)
{
int A[100];
……..
…….
for  (int i=0  ; i<100  ; i++)
change (A[i]);
……
……
}
```

انتقال کل آرایه در تابع:

```
void  change  (int  x[100])
{
for (int i=0  ; i<100  ; i++)
X[i]++ ;
}
int   main (void)
{
int   A[100];
…….
……
Change (A) ;
```

در این جا خود آرایه وارد می شود ⟵

ارسال آرایه های چند بعدی به به تابع :

```
Void  change  (int  x[  ]  , int n)
{
for (int  i=0 ; i<n  ; i++)
     x[i] ++ ;
}
int   main ( )
{
int   A[5] ={ 10,20,30,50};
int   B[2]={100 ; 200} ;
change  (A,5);
change  (B,2);
return 0;
}
```

نکته : در برنامه بالا باید به این نکته توجه کرد که چون برنامه از main **شروع به کامپایل کردن می کند ، پس طول آرایه ی تابع** chang **مشخص است.**

***برای یک آرایه دو بعدی**

```
int A[10][20];

Void change (int A[10][20])

{

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

for (int  j=0  ; j<10  ;j++)

A[i][j]++;

}

Void  change (int  A[  ] [20] ; int m)

{

……..

……..

}
```

علت اینکه تنها سطرها نامعین است این است که C++ تنها به ما اجازه می دهد تعداد سطرها را نامعین کنیم ولی حتماً باید تعداد ستون ها مشخص باشد.

نکته: کلاً در یک آرایه n بعدی تنها سطر اول می تواند نامشخص باشد ولی باید بقیه سطرها کاملاً معین باشد که دارای چه طولی است.

نحوه فراخوانی تابع توسط کامپایلر:

```
int sum(int x,int y)
{
……
……..
……..
}
int main()
{
……..
d=sum(a,b);
……….
f=sum(c,d);
……..
}
```

استفاده از توابع که خود برنامه نویس می نویسد علی رغم مزایای فراوان یک عیب هم دارد که خود را هنگامی نشان می دهد که :

۱- تعداد توابع مورد استفاده زیاد باشد. ۲- سرعت اجرای برنامه برایمان مهم باشد.

همان طور که در برنامه بالا مشاهد می شود کامپایل برنامه در حد نانوثانیه ممکن است طول بکشد ولی همین مقدار هنگامی که از چند تابع استفاده کنیم به چند ثانیه می رسد و اگر سرعت اجرا برایمان مهم باشد همین مقدار هم برایمان بسیار قابل توجه خواهد بود حال برای آنکه هم بتوانیم از مزایای توابع استفاده کنیم و هم مشکل سرعت برایمان پیش نیاید می توان از تابع های « in line » استفاده کرد.

توابع in line :

این توابع برای سرعت بخشی به کامپایلر شدن توابع معمولی مورد استفاده قرار می گیرند و نحوه کارشان به شرح زیر است:

کلمه کلیدی «in line» را قبل از خروجی تابع می نویسم. همیشه کامپایلر قبل از اینکه به سراغ mainبرود ؛اول چک می کند که برنامه inlineهست یا نه . این کار موجب می شود که هنگام کامپایل کردن ، تمام خطوط تابع را در برنامه کپی کند و بعد برنامه را از main اجرا می کند و این مشکل را برطرف خواهد کرد چرا که دیگر نیاز نیست همانند صفحه قبل کامپایلر به قبل یا بعد main برای اجرای تابع برود.

```
inline  int  Sum (int x  , int y)
{
……
……
}
int  main (void)
{
……..
…..
d= Sum (a,b);
f= Sum (c,d);
……
……
}
```

آیا می توان از توابع همنام استفاده کرد(Fountion over lading)

به طور کلی استفاده از توابع همنام خوانایی برنامه را پایین می آورد ولی اگر چند تابع باشند که کار یکسان انجام می دهند ولی ورودی های متفاوت دارند ؛ تفاوت در «Data type » ،بهتر است از توابع هم نام استفاده کنیم.

```
void  Sum  (int  x  , int y)

vout << x+y ;

void  Sum  (float  x  , float y)

cout << x+y ;

vaid  Sum  (double  x  , double y)

cout << x+y ;

int   main (void)

{

int  a=10 , b=20;

 float C=45 , d= 7.5;

double  e=7.12 ; f=8.15;
```

تابعی که می خواند که ورودی هایش از نوع a وb باشد. (int)

```
Sum (a,b);
```

تابعی که می خواند که ورودی هایش از نوع c وd باشد. (float)

Sum (c,d);

تابعی که می خواند که ورودی هایش از نوع e وf باشد. (double)

sum(e,f);

Sum (a,c);→ Errore

errorبه خاطر این است که a (int) و c(float)می باشد.

return 0;

}

علت خوانایی بیشتر هنگام استفاده از توابع مشابه ولی با ورودی های متفاوت این است که برنامه نویس برای فهمیدن اینکه تابع چه کار انجام می دهد مجبور نباشد مدام به source تابع رجوع کند.

نکته مهم:

خروجی توابع هیچ تأثیری در تعریف توابع همنام ندارد و فقط ورودی در این جا مؤثر است و ایجاد تفاوت می کند مثلاً شکل رو به رو اشتباه است:

int Sum (int a, int b)

{

……..

……

}

```
void  Sum (int a, int b)
{
……..
……
}
```

ورودی می تواند در موارد زیر متفاوت باشد:

۱- ترتیب

۲- نوع

۳- تعداد

```
void  Sum (int x);
void  Sum (int x, int y); →  تفاوت در تعداد
void  Sum (float  x); →  تفاوت در نوع
void  Sum (int x, float y); →تفاوت تعداد و نوع
void  Sum (float x, int y); → تفاوت ترتیب
```

توابع با مقدار پیش فرض :

```
vaid  f(int  x, int y)
{
cout << x+y;
}
int  main (void)
{
int a=10;
f (a , 2);
int   b=20 ;
f (b,2);
int c=30;
Int (c , 4);
}
```

در این مواقع بهتر است به تابع پیش فرض داده شود.

همان برنامه با پیش فرض:

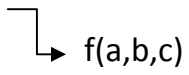
```
Void  f(int  x  , int y=2)
                {
cout << x+y;
                }


int   main (void)
{
int   a=10;
f(a); → f(a,2);
int  b=20;
f(b , 2);
int  C=30
F(C,4);
```

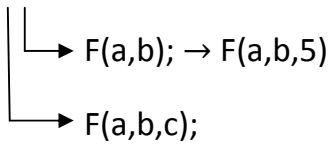
توجه داشته باشید که هر جای برنامه به yمقدار داده نشود به طور خودکار عدد ۲ قرار می گیرد .

نکته: مقدار پیش فرض تابع حتماً باید سمت راست تعیین شود چرا که کامپایلر کلاً از سمت راست داده ها را می خواند.

void f(int x , int y , int z=10); f(a,b)→f(a,b,10)



void f(int x , int y=2 , int z=5); f(a); →f(a,2,5)



void f(int x=5 , int y , int z=10) ⟶ Error

تابع بازگشتی:

این توابع هرگاه برای آنکه اجرا شود و به جواب برسد مجبور باشد چندین وچند بار خود را صدا زند تا به جواب نهایی و درست برسد.

نکته: می شود هنگام تعریف یک تابع، تابع دیگری را که قبلاً تعریف کردیم Call کنیم و از عملیات داخل آن استفاده کنیم.

```
void  f1(int x)
{
…….
……..
}
void  f2(int y)
{
…….
…….
f1(a);
}
int   main (void)
{
…….
……
f2 (b);
```

```
…….

……

void  f2(int y)

{

…….

…….

f2 (a);

}

int   main (  )

{

……..

……..

f2(10);

……..

……

}
```

توجه: روش کار توابع بازگشتی همانند حلقه ها است از این رو برای خارج شدن از توابع بازگشتی نیز نیاز به شرط داریم .

مثال: تابع فاکتوریل را با استفاده از توابع بازگشتی و بدون استفاده از آن بنویسید.

$n! = 1 \times 2 \times 3 \times \ldots\ldots \times n$

```
int f (int  n)
{
int  S= 1;
for  (int i=1 ; i<=n ; i++)
S*=I;
return S;
}
```

نکته: شاه کلید توانستن نوشتن یک تابع بازگشتی این است که هم بتوانی برایش فرمول پیدا کنی و هم شرط پایان!

تابع فاکتوریل با استفاده از تابع برگشتی:

شرط پایان: if

فرمول:

$$fact(n)\begin{cases} n=0 \text{ یا } n=1 \Rightarrow return\ 1; \\ n>1 \Rightarrow return < n * fact(n-1) \end{cases}$$

```
int   fact  (int n)
{
if (n==0  , n==1)
return  1;
else
return (n* fact (n-1));
}
```

نکته: توابع بازگشتی یک روال را طی می کند تا به شرط پایان خود برسند و بعد به شکل معکوس عمل می کنند تا جواب را بیابند در واقع کامپایلر این توابع به صورت نمادین به شکل زیر است.

نکته: reurnt به معنای خارج شدن از قسمتی (بلاکی)است که در آن هستیم.

نکته: مفهوم in line برای توابع بازگشتی معنایی ندارد.

مثال: ضرب دو عدد را به شکل بازگشتی بنویسید.

$$\text{mult}(m,n)\begin{cases} \text{if } m=0 \Rightarrow 0 \\ \text{if } m>0 \Rightarrow n+\text{mult}(m-1\ ,\ n) \end{cases}$$

$4\times 7= 7+(3\times 7)$

$7+ (2\times 7)$

$7+ (7\times 1)$

$7+ (7\times 0)\Rightarrow 0$

```
int   mult  (int x,  int y)
{
if (x==0)
return 0;
else
return (y+mult (x-1) , y))
}
```

مثال: تابع توان را به شکل بازگشتی بنویسید.

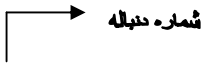
$X^n = x * x^{n-1}$

```
int  Tavan(int  x, int y)
{
if (x==0)
return 1;
else
return (y * Tavan (x-1 , y))
}
```

مثال: دنباله فیبلوناچی را به شکل تابع بازگشتی بنویسید:

0	1	2	3	5	8	13

شماره دنباله ←




```
int  fibo  (int  x)
{
if (x ==0  ||  x==1)
return 1;
else
return (fibo (x-1)+ fibo (x-2));
```

مثال: دو دنباله زیر را به شکل توابع بازگشتی بنویسید:

0	1	2	3	4	5
1	1	3	5	11	21

حل :دوبرابر دو تا قبلی+قبلی

```
int  tasaod (int x)
{
if (x==0 || x==1)
return 1;
else
return (2*Tasaod (x-2)+ T(x-1));
}
```

0	1	2	3	4	5	6	**شماره دنباله**
1	1	1	5	13	29	73	

حل:سه برابر ۳ تا قبلی +۲برابر قبلی

```
int   Tasaod (int x)
{
if (x==0 || x==1 || x==2)
return 1;
else
return (3* Tasaod (x-3)+2* Tasaod (x-1))
}
```