

نام خدا



سری های آموزشی آشنایی با زبان برنامه نویسی ++C

قسمت پنجم : کار با توابع 2

تابع نویسی

یک مرور کلی :

تابع : دستوری یا مجموعه ای از دستورات است که می تواند بصورت پیش فرض و یا بصورت نوشته شده توسط خود کاربر باشند ، که عملیات خاصی را بر روی متغیری که دریافت می کنند ، انجام می دهند .

همانطور که گفته شد توابع دو گروهند :

- 1- توابعی پیش ساخته و از قبل به همراه کمپایلر ++C ارائه شده اند که به آنها **توابع کتابخانه ای** می گویند . مثل تابع cos
- 2- توابعی که کاربر بر حسب ضرورت و نیاز آنها را می نویسد و در برنامه ی خود استفاده می کند .

به طور کلی تابع از سه قسمت تشکیل شده است :

- 1- مقدار دهی اولیه برای ورود به تابع
- 2- مجموعه ای از دستورات عمل ها که روی مقدار ورودی انجام می شوند .
- 3- مقدار خروجی بعد از انجام عملیات روی مقدار ورودی

با توجه به سه قسمت بالا ، هر تابع را به صورت زیر ، در خارج از تابع `main()` تعریف خواهیم کرد :

```
Return-type function-name ( parameter-type parameter-Name , ... )
{
    statements;
    return value;
}
```

به این مثال ساده توجه کنید :

می خواهیم برنامه ای بنویسیم که در آن تابعی را تعریف کنیم که عددی را به عنوان ورودی بگیرد و مربع آن را به ما بازگرداند :
1- مثل روال گذشته ابتدا می نویسیم :

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

2- برای مثال اسم تابعمان را `sqr` در نظر می گیریم و کار آن را تعریف می کنیم :

```
int sqr(int x)
{
    return x*x;
}
```

3- متن اصلی برنامه :

```
int main()
{
    int a;

    cin>>a;
    cout << sqr(a) << endl;

    return 0;
}
```

برنامه ی ما حالا کامل است :

```
#include <iostream.h>

int sqr(int x)
{
    return x*x;
}

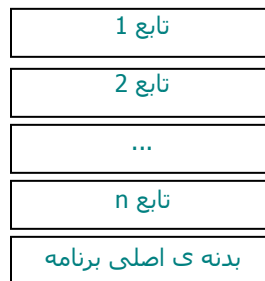
int main()
{
    int a;

    cin>>a;
    cout << sqr(a) << endl;

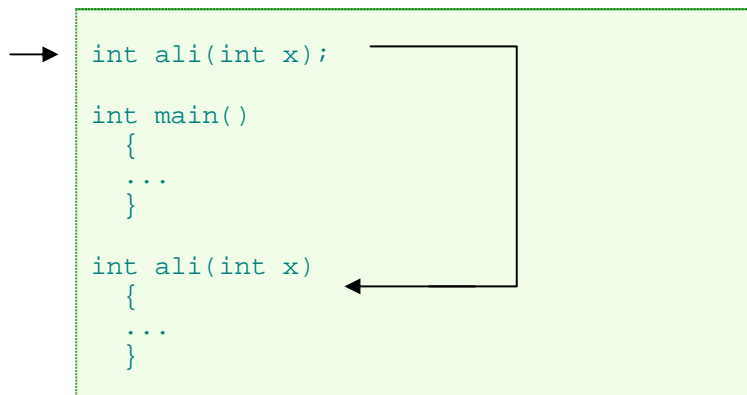
    return 0;
}
```

یکبار دیگر به برنامه ی بالا برگردید ! می خواهیم نکات و ریزه کاری های آن را بررسی کنیم ! (پس خوب بخوانید !):

-کامپایلر برنامه را به این صورت می خواند :
 ... ابتدا در قسمت بالا تابع را شناسایی می کند و به ترتیب به پایین می آید ، تا اینکه به تابع اصلی می رسد و در نهایت آنرا اجرا می کند .
 و چون در بالا ، شناخته است که تابعی با نام sqr معرفی کرده ایم ، هر جا که در متن برنامه نام تابع را ببیند ، سریعاً برگشته و از روی دستورات تابع ، عملیات را انجام می دهد . در واقع ساختار استفاده ی ما ، از تابع ، در حالت کلی بدین صورت است :



اما در صورتی که بخواهیم ، ابتدا تابع اصلی برنامه را بنویسیم و بعد سایر توابع را ، ابتدا باید قبل از تابع `main()` یک اشاره ی کوچک به آن تابع داشته باشیم . بدین صورت :



یعنی قبل از اینکه کامپایلر به تابع `main()` برسد ، به آن اطلاع می دهیم که تابعی با این نام و مشخصات داریم ، اگر فراخوانی شد ، دستورات آن در زیر تابع اصلی است . توجه کنید که همیشه بعد از اشاره به تابعی حتما ؛ فراموش نشود .

- به ساختار تعریف تابع `sqz` نگاه کنید . در اولین قسمت قبل از نام تابع نوع مقدار خروجی نوشته می شود . یعنی اینکه ما که در بالا `int` نوشته ایم ، بدین معنا است که مقداری که تابع آن را بازگشت خواهد داد ، یک عدد و از نوع `int` می باشد. در واقع می توان این قسمت را بر حسب نیاز به مقدار بازگشتی از انواع دیگری مثل `char` , `long int` و یا ... تعیین کنید .

- در ادامه خواهیم دید یک نوع از تابع ها هستند که هیچ مقداری را بر نخواهند گرداند . این نوع از تابع ها از نوع `void` هستند . در صورتی که تابعی را از نوع `void` تعریف کنیم دیگر نیازی به نوشتن `return` نیست . البته می توان آنرا نوشت و جلوی آن را خالی گذاشت :

```
void ali(int x)
{
    Statement;
}
```

یا :

```
void ali(int x)
{
    Statement;
    Return ;
}
```

در مورد توابعی که هیچ مقداری بر نمی گردانند ، بیشتر بررسی خواهیم کرد .

- در انتخاب نام تابع (مثلاً `sqz`) سعی کنید نام تابع با کاری که می کند ، همخوانی داشته باشد . چون در برنامه های چند صد خطی ، دیگر فرصتی برای اتلاف وقت برای بازخوانی کد ها یا حفظ کردن نام ها و ... نخواهید داشت . همچنین دقت کنید نامی که در قسمت معرفی تابع معرفی می کنیم ، در بدنه ی اصلی برنامه ، تابع را با همان نام فراخوانی می کنیم .

- در هنگام معرفی تابع در داخل پرانتز ، نوع و تعداد مقدار هایی که تابع به عنوان ورودی خواهد گرفت را معرفی می کنیم . مثلاً در این تابع که نوشتیم ، تابع یک مقدار عددی از نوع `int` می گیرد .

- ممکن است در بعضی برنامه ها ، نیاز باشد که یک تابع ، چندین مقدار را دریافت کند . در اینصورت به عنوان مثال می توانیم بنویسیم :

```
int ali(int x, int y , int z)
{
    ...
}
```

که در اینصورت در برنامه ی اصلی می توانستیم تابع را به این صورت فراخوانی کنیم :

```
Cin>>a>>b>>c;
Cout<< ali(a,b,c)<<endl;
```

- یک نکته ی جالب : (default value) : اگر برای مثال ، من ، در معرفی تابعی بدین صورت بنویسم :

```
int ali(int x, int y , int z=2)
{
    ...
}
```

من در این حالت به `z` یک مقدار پیش فرض نسبت داده ام . یعنی اینکه ، در صورتی که در هنگام فراخوانی تابع در متن اصلی برنامه ، مقداری را به عنوان عدد سوم به تابع نفرستیم ، که مثل حالت معمولی در `z` قرار خواهد گرفت . وگرنه اگر هیچ مقداری به آن نفرستیم ، مقدار `z` برابر با 2 خواهد بود . به یاد داشته باشید که همیشه مقدار اولیه متغیر ها در هنگام معرفی آنها همیشه در سمت راست ترین قسمت نوشته می شوند . یعنی اینکه حالت معرفی زیر اشتباه است :

```
int ali(int x, , int z=2, int y)
{
    ...
}
```

مقدار نتیجه ای که تابع به عنوان خروجی پس خواهد داد ، با استفاده از دستور `return` باز گردانده می شود . مثلا در مثال قبلی داریم :

```
return x*x;
```

یعنی اینکه پس از فراخوانی و اجرای دستورات تابع مقدار توان 2 ی `x` را به عنوان خروجی باز میگرداند . اگر می خواستیم که خروجی یک تابع همواره 2 باشد :

```
return 2;
```

و یا...

مثال 2 : برنامه ای که مربع اعداد 1 تا 10 را در خروجی چاپ کند :
این برنامه هم مشابه مثال قبلیست :
جواب:

```
#include <iostream.h>

int sqr(int x)
{
    return x*x;
}

int main()
{
    for (int i=1; i<=10; i++)
    {
        cout << sqr(i) << endl;
    }
    return 0;
}
```

یا به این صورت :

```
#include <iostream.h>

int sqr(int x);

int main()
{
    for (int i=1; i<=10; i++)
    {
        cout << sqr(i) << endl;
    }
    return 0;
}

int sqr(int x)
{
    return x*x;
}
```

مثال 3 : با استفاده از یک تابع ، برنامه ای بنویسید که اعداد 1 تا 100 را در خروجی چاپ کند : راهنمایی : تابعی خواهیم نوشت که عدد را بگیرد و توسط مقدار برگردانده شده نشان دهد که اول یا نه . اگر اول بود 1 برگرداند اگر نه 0 را برگرداند .

```
#include <iostream.h>
int aval(int x)
{
    int w=0;
    for(int i=1;i<=x;i++)
    {
        if (x%i==0)
            w++;
    }
    if(w==2)
        return 1;
    else
        return 0;
}

int main()
{
    int a;
    for(int i=1;i<=100; i++)
    {
        if(aval(i)==1)
            cout<<" adad e aval = " << i<<endl;
    }
    return 0;
}
```

به این شکل هم می توانستیم بنویسیم :

```
#include <iostream.h>
int aval(int x);
int main()
{
    int a;
    for(int i=1;i<=100; i++)
    {
        if(aval(i)==1)
            cout<<" adad e aval = " << i<<endl;
    }
    return 0;
}

int aval(int x)
{
    int w=0;
    for(int i=1;i<=x;i++)
    {
        if (x%i==0)
            w++;
    }
    if(w==2)
        return 1;
    else
        return 0;
}
```

مثال : برنامه ای بنویسید که سه عدد را از کاربر گرفته و آنها را به تابعی فرستاده و بزرگترین آنها را پیدا و چاپ کند .

```
#include <iostream.h>

int max ( int x, int y, int z)
{
    int max=x;
    if(max<y) max=y;
    if(max<z) max=z;
    return max;
}

int main()
{
    int a,b,c;
    cout<<"Enter three numbers: " <<endl;
    cin>>a>>b>>c;
    cout<<"The max number is = " <<max(a,b,c)<<endl;
    return 0;
}
```

یا به این صورت :

```
#include <iostream.h>

int max ( int x, int y, int z);

int main()
{
    int a,b,c;
    cout<<"Enter three numbers: " <<endl;
    cin>>a>>b>>c;
    cout<<"The max number is = " <<max(a,b,c)<<endl;
    return 0;
}

int max ( int x, int y, int z)
{
    int max=x;
    if(max<y) max=y;
    if(max<z) max=z;
    return max;
}
```

یک نمونه استفاده از توابعی که هیچ مقداری را برنمی گردانند :
 برنامه ای که مقداری را می گیرد و مشخص می کند در چه محدوده ای از صفر قرار دارد .

```
#include <iostream.h>
void fun (int x)
{
    if(x<0)
        cout<<"The number is lower then zero ! " <<endl;
    else
        cout<<"The number is higher then zero ! (Or equal with ) " <<endl;
    return ;
}
int main()
{
    int a;
    cout<<"Enter a number: " ;
    cin>>a;
    fun(a);
    return 0;
}
```

خوب به ساختار استفاده ی تابع بالا توجه کرده و آن را به خاطر بسپارید . با فراخوانی تابع و اجرای آن مانند این است که عینا دستورات آن را در همان مکان اجرا می کنیم .

ممکن است در جایی حتی لازم نباشد تابع مقداری را بگیرد ! پس :
 مثال : برنامه ای که با استفاده از تابعی در صفحه چاپ کند : MajidOnline.com First Persian Graphic and Web design Resource

```
#include <iostream.h>
void fun ()
{
    cout<<"***MajidOnline.com " <<endl
        <<"First Persian Graphic and Web design Resource "
<<endl;
    return ;
}
int main()
{
    fun();
    return 0;
}
```


یک نکته ی مهم :
انواع شیوه های عمومی فراخوانی داده ها توسط یک تابع :
 pass by refrence -1
 pass by value -2

برای متوجه شدن مفهوم ایندو شیوه به برنامه ی زیر توجه فرمایید :

```
#include <iostream.h>

int f1( int a )
{
    return a *= a;
}

void f2( int &b )
{
    b*= b;
}

int main()
{
    int x = 2, z = 4;

    cout << "x = " << x << " before passByValue\n"
         << "Value returned by passByValue: "
         << f1( x ) << endl
         << "x = " << x << " after passByValue\n" << endl;
    //*****
    cout << "z = " << z << " before passByReference" << endl;
    f2( z );
    cout << "z = " << z << " after passByReference" << endl;

    return 0;
}
```

در شیوه ی pass by value می خواهیم با استفاده از تابع f1 مقداری را مربع کنیم . در نهایت خواهید دید که بعد از اجرای تابع ، هیچ تاثیری در مقدار اولیه ی تابع نخواهد داشت .
 اما اگر به قسمت pass by reference و تابع f2 توجه کنید ، خواهید دید از یک علامت & (آمبر سند) بین اسم متغیر ارسالی و نوع آن استفاده کرده ایم . این یعنی اینکه هر تغییر که روی مقدار ارسالی ما در داخل تابع صورت گرفت ، آنرا بر روی آن ذخیره کن . در واقع در حالت pass by reference مقدار متغیر اولیه ارسالی به تابع تغییر کرده و در آن ذخیره خواهد شد .

درواقع در خروجی خواهیم داشت :

```
x = 2 before passByValue
Value returned by passByValue: 4
x = 2 after passByValue

z = 4 before passByReference
z = 16 after passByReference
```

می توانید تغییرات زیادی روی برنامه بالا انجام دهید تا به شیوه ی کار هر کدام پی ببرید . مثلا :

- 1 یکبار علامت آمبر سند را بردارید .
- 2 یکبار سعی کنید نوع توابع را تغییر دهید .
- 3 ...

برای آشنایی بیشتر با کاربرد علامت & به برنامه ی ساده ی زیر توجه کنید :
 مثال : برنامه ای بنویسید که عددی را گرفته و مقدار همان را 10 برابر کند :

```
#include <iostream.h>
void jj(int &a)
{
    a*=10;
}
int main()
{
    int x;
    cout<<"Enter a number plz ! : ";
    cin>>x;
    jj(x);
    cout<<"result is : " <<x;
    return 0;
}
```

با قرار دادن علامت آمپرسند ، با تغییر مقدار a ، این تغییرات در همانجا ذخیره می شود.
 می توانید یکبار هم & را بردارید و نتیجه آنرا ببینید . در اینصورت ، مقدار ورودی هیچ تغییری نخواهد کرد.
 فقط یادتان نرود که بین نام متغیر و نوع آن (که & در میان آنها قرار خواهد گرفت) فاصله را رعایت کنید .

آشنایی با یک نوع داده : static int :
 Static int یک نوع داده است که تقریباً کار آن ، شبیه عمل نوع داده ی int به همراه & است .
 برای درک این مطلب به برنامه ی زیر توجه بفرمایید:

```
#include <iostream.h>
void printme()
{
    static int i=1;
    cout<<i++<<" ";
}
int main()
{
    for(int k=0; k<10; k++)
        printme();
    return 0;
}
```

حالا خودتان با توجه به خروجی می توانید نحوه ی کار این نوع داده را بگویید ؟
 در خروجی این برنامه خواهیم داشت :

1 2 3 4 5 6 7 8 9 10

حال در برنامه static int را تبدیل کنید به int تبدیل کنید . در خروجی خواهیم داشت :

1 1 1 1 1 1 1 1 1 1

حال int را دوباره به حالت اولیه برگردانید و این بار مقدار دهی اولیه ی آن را پاک کنید . یعنی به این صورت : static int i;
 در خروجی خواهیم داشت :

0 1 2 3 4 5 6 7 8 9

از این آخری می توانیم نتیجه بگیریم که متغیر static int برعکس متغیرهای دیگر (که اگر به آنها مقدار اولیه ندهیم ، مقدار آنها نامعلوم خواهد بود) دارای مقدار اولیه ی صفر است .

آشنایی به توابع بازگشتی :

یک روش استفاده از تابع هستند که در واقع می توانیم با این سبک تابع نویسی ، بعضی از مسائل خاص را به روش ساده تری حل بکنیم .

*** مثلا دنباله ی زیر را در نظر بگیرید :

1 , 3 , 6 , 10 , 15 , 21 , ...

اگر شماره ی جمله را n در نظر گرفته و اولین عدد را 1 بگیریم ، خواهیم داشت : $f(1)=1$ ، $f(n) = f(n-1) + n$ ، می خواهیم برنامه ای به روش بازگشتی بنویسیم که شماره ی عدد از دنباله ی بالا را گرفته و خود عدد را چاپ کند : (خوب به طرز حل این مسئله به روش بازگشتی توجه بکنید)

- در روش بازگشتی از یک فرمول عمومی به صورت $f(n)$ که بر حسب $f(n-1)$ و یا ... خواهد بود استفاده خواهیم کرد . بصورتی که اگر نتیجه ی تابع $f(n)$ مورد نظر باشد ، کامپایلر ابتدا شروع به محاسبه ی $f(n-1)$ می کند . بعد چون داریم : $f(n-1) = f(n-2) + n-1$ برای محاسبه ی $f(n-1)$ وارد محاسبه ی $f(n-2)$ می شود و ... و این اعمال تا جایی ادامه پیدا می کند که کامپایلر به $f(1)=1$ برسد که مقدارسیت ثابت و مشخص . بعد از رسیدن به این مقدار معلوم دوباره همان را آمده را برمی گردد تا جایی که به $f(n)$ برسد . پس از این داستان بالا ، دو مورد زیر را حتما به خاطر بسپارید :

- 1- نحوه ی کار توابع بازگشتی
- 2- حتما به خاطر داشته باشید که حد معین و مجازی برای تابع بازگشتی خود تعیین کنید . مثلا ما در مثال بالا $f(1)=1$ را تعریف کردیم . اگر تابع بازگشتی خود را محدود نکنید ، کامپایلر تا منفی بینهایت خواهد رفت (و در نتیجه هیچ بازگشتی در کار خواهد بود)

اما خود برنامه :

```
#include <iostream.h>
int f ( int x)
{
    if (x==1)
        return 1;
    else
        return f(x-1)+x;
}
int main ()
{
    int a;
    cout<<"enter Number of figure which u want : " ;
    cin>>a;
    cout<<"f ("<<a<<" )="<<f(a)<<endl;
    return 0;
}
```

*** بزرگترین فایده ی توابع بازگشتی سادگی کار آن ، و بزرگترین اشکال آن ، سرعت پایین آن است .

مثال : برنامه ای بنویسید که دنباله ی فیبوناچی را با استفاده از تابع بازگشتی محاسبه کند.

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

```
#include <iostream.h>
int f ( int x)
{
    if ((x==1) || (x==2))
        return 1;
    else
        return f(x-1)+f(x-2);
}
int main ()
{
    int a;
    cout<<"enter Number of figure which u want : " ;
    cin>>a;
    cout<<"f ("<<a<<" )="<<f(a)<<endl;
    return 0;
}
```

یک نکته : به برنامه ی زیر توجه کنید :

```
#include <iostream.h>
int main()
{
    int a=3;
    cout<<a++<<" "<<a++<<" "<<a++<<" "<<a++;
    return 0;
}
```

فکر می کنید نتیجه ی آن چه باشد؟!

6 5 4 3

اگر به خورده به نتیجه ی آن نگاه کنید شاید عجیب و تازه به نظرتان برسد . پس این مورد را به خاطر داشته باشید زبان ++C بر عکس زبان های دیگه ای مثل پاسکال و ... محتوای داخل دستور خروجی را از راست به چپ ، پردازش می کند . مشابه این مورد را می توان در جاهای دیگه ای یافت :

```
#include <iostream.h>
void vahid(int a , int b)
{
    cout<<"a="<<a<<endl;
    cout<<"b="<<b<<endl;
    return ;
}
int main()
{
    int a=3;
    vahid(a,a++);
    return 0;
}
```

اگر توجه کرده باشید ، مقادیر از راست به چپ به تابع ارسال می شوند .

تمرینات:

- 1- برنامه ای که فاکتوریل عددی را با استفاده از تابع بازگشتی محاسبه کند .
- 2- برنامه ای که مقدار n راز ورودی گرفته . s(n) را بصورت بازگشتی ، حساب کند :

$$S_n = \sum_{i=1}^n i^2 \quad \begin{cases} S_n = S_{n-1} + n^2 \\ S_1 = 1 \end{cases}$$

- 3- تابعی را بنویسید که یک مقدار ورودی (آرگومان) بگیرد و مقدار مقلوب آن را به عنوان خروجی ، به برنامه ی اصلی را برگرداند .

پایان قسمت پنجم!