توابع

دلیل استفاده از تابع

- اجتناب از نوشتن تکراری خطوط یکسان در یک برنامه
- ساختار برنامه بهتر و قابل فهم میشود
- استقلال زیر روالها از تابع main که باعث میشود بتوانیم از افراد مختلف برای پیاده سازی بخشهای مختلف استفاده کنیم

تابع چیست؟(ادامه)

- مفهوم تابع در زبان **C** مشابهت زیادی با مفهوم تابع در ریاضیات دارد و در واقع مدل کننده تابع در ریاضیات است.

مثال

نوع بازگشتی

اسم تابع

پارامترها

```
int summation(int a, int b)
{
  return(a+b);
}
```

بدنه تابع

مثال:

تابعی بنویسید که یک عدد را گرفته و قدرمطلق آنرا برگرداند

```
float Absolute(float num)
{
    float result;
    if (num < 0)
        result = -num;
    else
        result = num;
    return result ;
}
```

محل قرار گرفتن توابع

- هرتابع قبل از استفاده شدن باید توسط کامپایلر شناخته شده باشد
- بنابراین باید قبل از فراخوانی توابع آنها را به سیستم معرفی نمود
- از آنجا که در برنامه نویسی معمول است که پیاده سازی توابع بعد از تابع main انجام شود لذا قبل از تابع main تنها یک معرفی از توابع انجام میشود
- در معرفی تابع تنها نوع خروجی، نام تابع و نوع آرگومانها معرفی میشوند که به آن نمونه اولیه (prototype) میگویند

نمونه اولیه توابع
Function Prototypes

- نمونه اولیه تابع به کامپایلر می گوید که شکل تابع به چه صورت است.
  - لذا می توان تابع را استفاده کرد حتی اگر کامپایلر هنوز پیاده سازی تابع را ندیده باشد.
- نمونه اولیه تابع موارد زیر مشخص می کند:
  - اسم تابع
  - تعداد و نوع آرگومانها
  - نوع مقدار بازگشتی

مثال:

```
float Absolute(float num)
{
    if (num < 0)
        return -num;
    else
        return num;
}
void main()
{
    float x1,x2;
    printf( "\nEnter a number");
    scanf( "%f" , &x1);
    x2 = Absolute(x1);
    printf( "\nAbsolute=%f", x2 );
}
```

```
float Absolute(float num);
void main()
{
    float x1,x2;
    printf( "\nEnter a number");
    scanf( "%f" , &x1);
    x2 = Absolute(x1);
    printf( "\nAbsolute=%f", x2 );
}
float Absolute(float num)
{
    if (num < 0)
        return -num;
    else
        return num;
}
```

مثال:

با استفاده از تابع برنامه‌ای بنویسید که دو عدد گرفته و ماکسیمم آنرا نمایش دهد

```
int Max( int a , int b )
{
    if (a > b )
        return a;
    else
        return b;
}
void main()
{
    int x1,x2;
    printf( "\nEnter two numbers ");
    scanf( "%d %d" , &x1 , &x2);
    printf( "\nMax(%d,%d)=%d", x1, x2, Max(x1, x2) );
}
```

```
Enter two numbers 5 8

Max(5,8)=8
```

پارامترهای تابع

- پارامترهای تابع در داخل بدنه آن متغیر محلی محسوب میشوند.
- وقتی که تابع صدا زده می شود، فراخواننده تابع مقدار پارامترها را به تابع می فرستد.
- تابع یک کپی از مقدار پارامترها را دریافت می کند.
- لذا تغییراتی که در مقدار متغیرها، درون تابع اعمال میشود تاثیری در مقدار آنها در تابع فراخواننده ندارد.

مثال:

تابعی بنویسید که دو عدد گرفته و حاصل جمع آنها را برگرداند

```
int add2nums( int firstnum, int secondnum )
{
  int sum;

  sum = firstnum + secondnum;

  // just to make a point
  firstnum = 0;
  secondnum = 0;

  return sum;
}
```

مثال:

تست برنامه جمع

```
void main()
{
    int y , a , b ;
    printf( "\nEnter two numbers ");
    scanf( "%d %d" , &a , &b);
    y = add2nums( a , b );
    printf( "\na is: %d " , a );
    printf( "\nb is: %d " , b );
    printf( "\ny is: %d " , y );
}
```

```
Enter two numbers 5 9

a is: 5
b is: 9
y is: 14
```

```
int add2nums( int a, int b)
{
  int sum;

  sum = a+ b;
  // just to make a point
  a= 0;
  b= 0;
  return sum;
}

void main()
{
    int y , a , b ;
    printf( "\nEnter two numbers ");
    scanf( "%d %d" , &a , &b);
    y = add2nums( a , b );
    printf( "\na is: %d " , a );
    printf( "\nb is: %d " , b );
    printf( "\ny is: %d " , y );
}
```

متغیرهای محلی
(Local Variables)

- پارامترها و متغیرهایی که داخل بدنه تابع و در قسمت تعریف تابع اعلان می شوند از نوع محلی هستند.
- متغیرهای محلی فقط در محدوده تابع با معنی هستند.
- بعد از پایان تابع، این متغیرها دیگر وجود ندارند.
- پس از بازگشت از تابع تمامی فضای اختصاص یافته به متغیرهای محلی آن، به سیستم بازپس داده میشود.

متغیرهای بلاکی
(Block Variables)

- اگر متغیری در داخل یک بلاک از کد (بین } و {) تعریف شده باشد، این متغیر از نوع بلاکی است و فقط در محدوده آن بلاک معتبر است و خارج از آن قابل شناسایی و استفاده نیست.

```
{
    int var;
        ...
        ...
}
```

متغیرهای عمومی
(Global Variables)

- شما می توانید متغیرها را خارج از محدوده توابع و بلاکها تعریف کنید این متغیرها از نوع عمومی هستند.
- متغیرهای عمومی از همه جای برنامه قابل استفاده هستند و همه بلاکها و توابع می توانند از آنها استفاده کنند.

محدوده یک متغیر (Scope)

- محدوده یک متغیر **قسمتی از برنامه است که متغیر در آن دارای معنی و قابل استفاده است** (در واقع، در این محدوده متغیر وجود دارد).

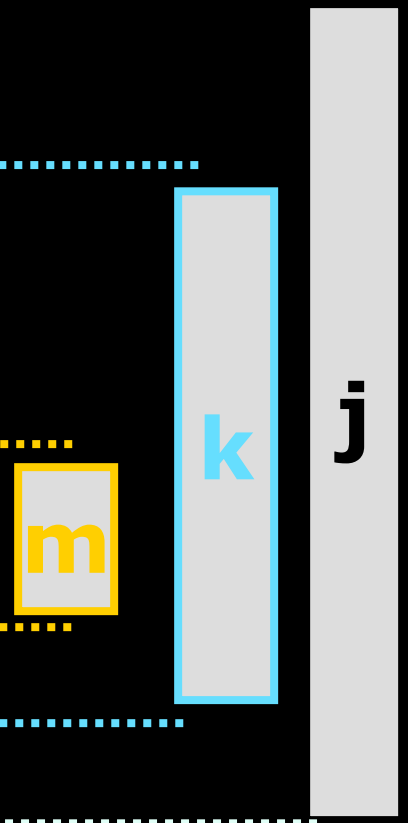
مثال:

```
void main()
{
   int y;
   {
        int a = y;
        printf( “%d\n” , a);
   }
    printf( “%d\n” , a);
}
```

خطای کامپایل!! a خارج از محدوده بلاک معنی ندارد.

متغیرهای بلوکی تودرتو

```
void func1()
{
  for (int j=0;j<10;j++)
  {
      int k = j*10;
      printf("%d,%d\n", j , k );
      {
          int m = j+k;
          printf("%d,%d\n", m,j);
      }
  }
}
```



هنگام فراخوانی تابع چه اتفاقی می افتد؟

- با فراخوانی تابع کنترل برنامه (program counter) بعد از پاس کردن آرگومانهای تابع به بدنه تابع منتقل میشود
- دستورات در بدنه تابع اجرا میشود تا
  - به دستور return برسیم
  - در توابعی که مقدار بازگشتی ندارند به { برسیم
- بعد از اجرای تابع مقدار خروجی تابع تولید شده و کنترل برنامه به محل فراخوانی تابع بازگردانده میشود.

توابع بازگشتی (چرخشی)
Recursion

- توابع می توانند خودشان را صدا بزنند!! این موضوع بازگشت نام دارد.

- از آنجا که خیلی از مسائل ماهیت بازگشتی دارند، لذا با استفاده از این ویژگی بسیاری از مسائل را میتوان حل نمود.

- اغلب می توان مسائلی که صورت پیچیده دارند را با بازگشت به راحتی حل نمود.

مثالی از توابع بازگشتی

- تابع فاکتوریل

$$n! = n * (n-1) * \ldots * 2 * 1$$

- ویا

$$n! = n * (n-1)! \quad \text{For} \quad n > 0$$

$$0! = 1$$

پیاده سازی تابع فاکتوریل بصورت بازگشتی

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

اصول طراحی توابع بازگشتی

- ابتدا حالات توقف (stop case) را تعریف کنید.
  - یعنی شرایطی که دیگر لازم نیست تابع خودش را دوباره صدا کند.
- سپس مرحله بازگشت را مشخص کنید.
  - یعنی مساله را به مساله یا مسائل کوچکتر ولی مشابه بشکنید و دوباره تابع را صدا کنید.

اصول طراحی توابع بازگشتی
(ادامه)

- اغلب با استفاده از یک دستور if مشخص میشود که کدامیک از حالتها باید انجام شود. (حالت تکرار یا توقف)

if (به حالت توقف رسیدی)

مساله را حل کن

else

تابع را باردیگر با پارامترهای جدید فراخوانی کن

مثال

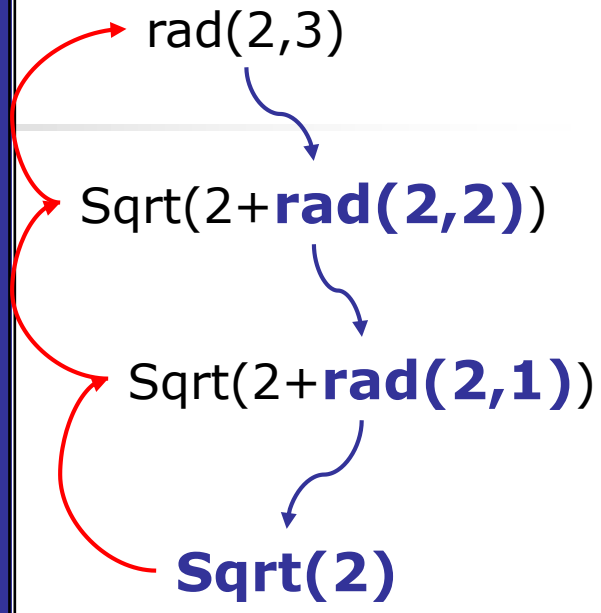
- برنامه بازگشتی بنویسید که حاصل عبارت زیر را محاسبه کند:

$$\sqrt{a+\sqrt{a+\sqrt{a+.....}}}$$

- (تعداد n تا a داریم)
- تابعی بازگشتی به نام rad(a,n) مینویسیم که این کار را انجام دهد.
- قانون بازگشت و شرط خاتمه :

$$rad(a,n)=\begin{cases}\sqrt{a+rad(a,n-1)} & n>1\\ \sqrt{a} & n=1\end{cases}$$

```
float rad(float a , int n)
{
   if (n==1)
        return sqrt(a);
   else
        return sqrt(a+rad(a,n-1));
}
void main()
{
   printf("%f",rad(2,3));
}
```



rad(2,3) = sqrt(2+sqrt(2+sqrt(2)))=1.961571

تمرین: محاسبه ب.م.م دو عدد

- برنامه ای بازگشتی بنویسید که ب.م.م دو عدد را محاسبه کند.
- ب. م .م: Greatest Common Divisor (GCD)
- می خواهیم تابعی به نام gcd(a,b) بنویسیم که ب.م.م دو عدد a و b را محاسبه کند.
- روش معمول محاسبه ب.م.م، روش پلکانی است.

	3	1	2	
۳۳	۹	۶	۳	۰
27	6	3		

- با دقت در رابطه صفحه قبل می توان گفت:

gcd(33,9)=gcd(9,33%9)

(٪: باقی مانده)

به این ترتیب می توان قانون بازگشت و شرط خاتمه را به صورت زیر به دست آورد:

$$\gcd(a,b)=\begin{cases}\gcd(b,a\%b) & b\neq 0\\ a & b=0\end{cases}$$

تمرین

- یک تابع بازگشتی بنویسید که یک مثلث به صورت زیر چاپ کند.

triangle(4);

```
*
***
*****
*******
```

triangle(5);

```
*
***
*****
*******
*********
```

فراخوانی با مقدار در مقایسه با فراخوانی با مرجع
(Call-by-value vs. Call-by-reference)

- تا کنون، توابع ما فقط یک کپی از متغیر را دریافت می کردند
  - این روش صدا کردن با مقدار نامیده می گردد. چون مقدار متغیر به تابع فرستاده می شود نه خود متغیر.
- ما می توانیم توابعی را تعریف کنیم که آدرس متغیر (خود متغیر) را دریافت کنند.
  - این روش صدا کردن با مرجع نامیده می گردد. در این روش، تابع می تواند متغیر را مستقیما تغییر دهد.

مراجع

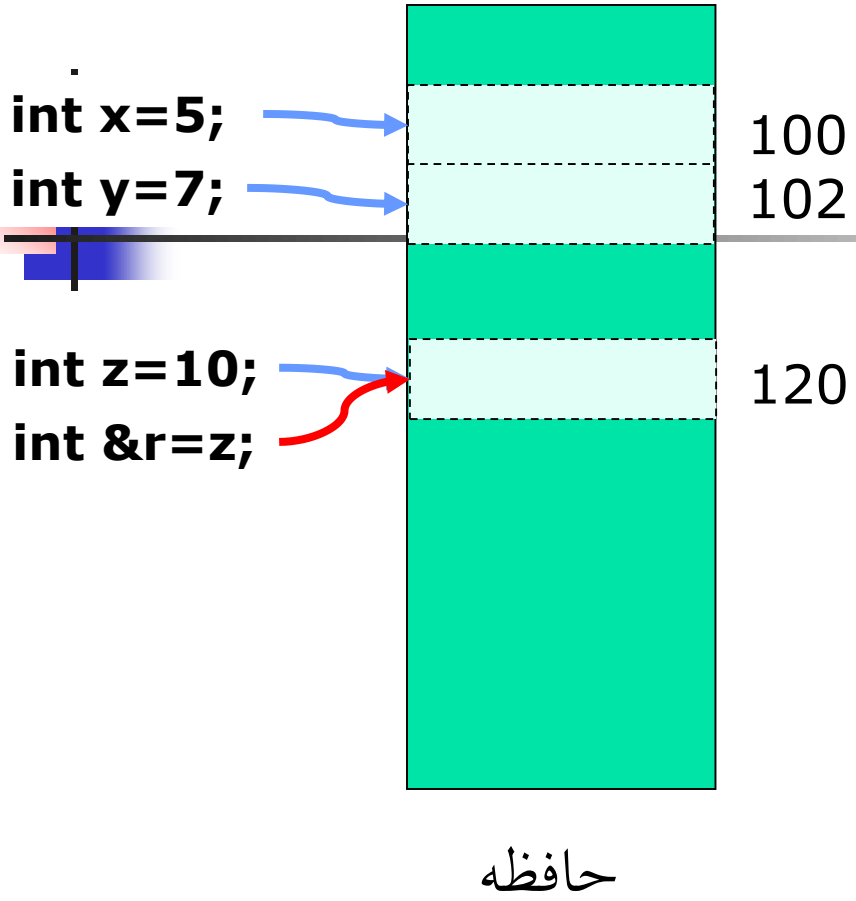
- متغیر مرجع در واقع یک اسم جایگزین برای یک متغیر واقعی است.
- هر متغیر مرجع باید مقدار دهی اولیه شود تا ما را به یک متغیر دیگر (واقعی) ارجاع دهد.
- به محض اینکه این نوع متغیر ارجاع داده شد (یعنی مرجع آن مشخص شد) می توان مانند بقیه متغیرها از آن استفاده کرد.

Reference Variable Example

```
int count;
int &rcnt = count;
// rcnt is the same variable as count

count = 1;
printf( "rcnt is %d\n" , rcnt );
rcnt++;
printf( "count is %d\n" , count );
```

```
rcnt is 1
count is 2
```



متغیر	آدرس	محتویات
x	100	5
y	102	7
z	120	10
r	120	10

پارامترهای مرجع در تعریف تابع

- در تعریف یک تابع، می توانید از متغیرهای مرجع استفاده کنید. در این صورت، هنگام صدا زدن تابع مرجع متغیر مشخص خواهد شد.

```
void add10( int &x)
 {
   x = x+10;
 }
 ...
 add10(counter);
```

The parameter is a reference

تابعی بنویسید که دو متغیر بگیرد و محتویات آنها را جابجا کند

```
void swap( int &x, int &y)
{
   int tmp;
   tmp = x;
   x = y;
   y = tmp;
}
```

استفاده از متغیرهای مرجع

- اگر تابع بخواهد چند مقدار خروجی داشته باشد از متغیرهای مرجع استفاده میشود.
- از آنجا که برای متغیرهای مرجع حافظه جدید اخذ نمیشود، برای بالا بردن سرعت پردازش و اجتناب از عملیات کپی کردن مقدار متغیرها از متغیرهای مرجع استفاده میشود.
- اگر بخواهیم متغیری درون تابع قابل تغییر نباشد از کلمه کلیدی const استفاده میشود.