تابع

توابع در C++ در این بخش توضیح داده می شود. توابع بلاک های مستقلی از کد هستند که کار خاصی را انجام می دهند. هنگامی که برنامه نیاز به انجام آن کار دارد تابع را صدا می زند. تابع اساس برنامه نویسی ساختیافته است که باعث بالارفتن کارائی برنامه ها و آسانتر شدن برنامه نویسی می شود. نحوه ایجاد و فراخوانی تابع به علاوه متغیر های محلی و نحوه ارسال و دریافت پارامترهای تابع، توابع پیش ساخته و بازگشتی نیز شرح داده می شود.

تابع (function) بلاکی ازکد است که تحت یک نام گروه بندی می شود. بدین ترتیب بلاک کد می تواند بعد با استفاده از نامش اجرا شود. این عمل فراخوانی تابع (function call) نامیده می شود. وقتی تابعی فراخوانی می شود دستورات درون تابع اجرا می شود. در انتهای تابع اجرا به همان محلی که تابع فراخوانی شده بود برمی گردد. ممکن است پارامترهایی به تابع ارسال شود. معمولا تابع معمولا مقداری را به برنامه فراخوان برمی گرداند.

برنامه ای که از تابع استفاده می کند به صورت ساختیافته (structured) است. یعنی وظایف توسط بخش های مستقل انجام می پذیرد. برنامه نویسی ساختیافته روی طراحی به روش top-down تاکید دارد. در این روش مسئله به وظایف مختلف تقسیم می شود و هر وظیفه توسط یک تابع انجام می پذیرد. بدنه تابع اصلی کوچک می شود و جزئیات درون توابع قرار می گیرند. به این ترتیب نوشتن و اشکال زدائی برنامه های پیچیده آسانتر می شود. علاوه بر این استفاده مجدد توابع در برنامه های دیگر میسر می شود.

هر تابع باید تنها یک کار را انجام دهد و از انجام چند کار در یک تابع باید پرهیز شود.

تعریف تابع

تعریف تابع (function definition) شامل کدهائی است که هنگام فراخوانی تابع باید اجرا شود. تعریف تابع شامل دو قمت است: اعلان و بدنه. اولین خط تعریف تابع اعلان یا هدر است که اسم تابع، نوع مقدار برگشتی و آرگومان های تابع را مشخص می کند. بعد از هدر بدنه تابع قرار می گیرد. بدنه تابع بلاکی از دستورات است که درون یک جفت آکولاد محصور می شود و هر زمان که تابع فراخوانی می شود اجرا می شود.

فرم کلی تعریف تابع به صورت زیر است:

```
return_type function_name( arg-type name-1,...,arg-type name-n)
{
    /* statements; */
}
```

مثال. تابع مربع یک عدد.

```
double square(double n)       /* function header */
{
    return n * n;             /* function body   */
}
```

اعلان تابع

در تابع مثال فوق خط double square(double n) اعلان تابع است. خط اعلان شامل ۳ قسمت است:

۱.نوع برگشتی
نوع برگشتی تابع نوع داده مقداری است که توسط تابع محاسبه و به برنامه فراخوان برگردانده می شود. نوع برگشتی می تواند از هر نوع استانداردی باشد. اگر نوع برگشتی تابع قید نشده باشد از نوع int درنظر گرفته می شود. اگر تابع مقداری را بر نمی گرداند نوع برگشتی آن void باید تعریف می شود.

۲.نام تابع
تابع اولیه که برنامه با آن شروع می شود main نام دارد. بقیه توابع هر نام دلخواهی می توانند بگیرند. نام تابع از قوانین نامگذاری متغیرها تبعیت می کند. بهتر است نام تابع بیان کننده کاری که تابع انجام می دهد باشد. مثلا اگر تابع برای محاسبه مربع یک عدد است نام هایی مثل square_num یا number_square یا num_2 مناسب است.

۳.لیست پارامترها
تابع می تواند هیچ، یک یا چند پارامتر داشته باشد. اگر تابعی پارامتر بگیرد به این معنی است که به مقادیری نیاز دارد تا روی آنها کار کند. در مثال تابع مربع، عددی که می خواهید مربع کنید باید به تابع داده شود. این عدد پارامتر تابع است. پارامترها از هر نوع داده ای می تواند باشد.. اگر تابع بیش از یک پارامتر بگیرد پارامترها توسط کاما جدا می شوند و نوع و نام هر پارامتر باید جداگانه مشخص شود. برای اعلان تابعی بدون پارامتر پرانتز باید خالی بماند.
مثال. تابع زیر دارای سه آرگومان x، y و z است.

```
void func1(int x, float y, char z)
```

توجه کنید که در انتهای اعلان تابع علامت سمیکولن وجود ندارد واگر گذاشته شود کامپایلر پیغام خطا صادر می شود.

گاهی بین پارامتر و آرگومان اشتباه پیش می آید. یک پارامتر (parameter) یک محلی در هدر تابع است که برای ذخیره مقدار آرگومان عمل می کند. پارامترهای تابع ثابت هستند و در طی اجرای برنامه تغییر نمی کنند. یک آرگومان مقدار واقعی ارسال شده به تابع توسط دستور فراخوانی است. هر زمان که تابعی فراخوانی می شود آرگومان های مختلف می تواند به آن ارسال شود. درون تابع به آرگومان ها توسط نام پارامترهای متناظر دسترسی می شوند.

بدنه تابع

قسمت دوم تعریف تابع بدنه تابع (function body) است که با آکولاد {} محصور می شود و مشابه بلاکی از کد است. بدنه تابع بلافاصله بعد از اعلان تابع قرار می گیرد. دستورات لازم که برای انجام وظیفه تابع در اینجا نوشته می شود. وقتی تابع فراخوانی می شود اجرای آن از ابتدای بدنه تابع شروع می شود و وقتی به انتهای تابع یا دستور return برسد پایان می پذیرد.

تابع ممکن است مقداری را به فراخواننده خود برگرداند. مقدار برگشتی تابع باید از همان نوعی باشد که در اعلان تابع ذکر شده است در غیر اینصورت با خطای mistake erroe مواجه می شوید.

مثال. در برنامه زیر تابع square برای محاسبه مربع عدد صحیح توسط تابع اصلی فراخوانی می شود.

```
#include <iostream.h>
double square(double n);
int main() {
    double num;
    do {
        cout << "What do you want to know the square "
             << "of (enter 0 to quit): ";
        cin >> num;
        cout << "The square of " << num << " is "
             << square(num) << ".\n";
    } while (num != 0);
}
double square(double n) {
    return n * n;
}
```

فراخوانی تابع

هر زمان که به تابع نیاز باشد نام آن صدا زده می شود. این عمل فراخوانی تابع نامیده می شود. برنامه یا تابعی ک تابع را فراخوانی می کند فراخواننده تابع (function caller) نامید می شود. هنگام فراخوانی تابع، مقدار آرگومان های تابع باید مشخص شود.

اگر تابع مقدار برگشتی دارد هنگام فراخوانی مقدار برگشتی در یک متغیر باید ذخیره شود. یعنی سمت راست تابع علامت مساوی و یک متغیر همنوع مقدار برگشتی تابع قرار می دهیم.

مثال. تابع square به صورت زیر در برنامه اصلی صدا زده می شود.

```
double sq = square(num);
cout << sq;

cout << square(num);
```

مثال. فراخوانی تابعی که آرگومانی ندارد. تابع مربع اعداد ۱ تا ۱۰ را نمایش می دهد.

```
#include <iostream.h>
void squares();
main() {
   squares();
}
void squares() {
   int loop;
   for (loop=1;loop<10;loop++);
   cout<<loop*loop;
}
```

پروتوتایپ تابع

کامپایلر C++ نمی تواند تابعی را صدا بزند که چیزی درباره آن نمی داند. باید قبل از فراخوانی تابع به کامپایلر اجازه دهید بداند تابع چیست و چه کار می کند. یک راه برای انجام این کار اطمینان از نوشتن بدنه تابع قبل از فراخوانی آن است. را ه دیگر استفاده از پروتوتایپ تابع (prototype) است. پروتوتایپ تابع شرحی از نام، نوع برگشتی و نوع آرگومانهای تابع به کامپایلر ارائه می دهد. به این ترتیب هنگام فراخوانی تابع کامپایلر صحت فراخوانی را بررسی می کند که آیا نوع و تعداد آرگومانهای ارسال شده به تابع و استفاده از مقدار برگشتی صحیح است یا خیر.

مثال. پروتوتایپ تابع square.

```
double square(double n);
```

پروتوتایپ تابع مشابه اعلان تابع است منتها همیشه به سمیکولن(;) ختم می شود. بعد از تکمیل تابع با استفاده از امکان copy-and-paste ادیتور می توانید هدر تابع را کپی کرده و پروتوتایپ آنرا بسازید. فقط دقت کنید در انتهای آن علامت سمیکولن را اضافه کنید.

پروتوتایپ توابع بهتر است در ابتدای برنامه و قبل از تابع اصلی باشند.
در پروتوتایپ اسم متغیرهای آرگومان هم می تواند ذکر شود ولی اختیاری است.

دستور return

برای برگرداندن مقداری از تابع به فراخواننده دستور return استفاده می شود. مقدار برگشتی تابع که به دنبال دستور return نوشته می شود باید از همان نوعی باشد که در اعلان تابع قبل از اسم تابع معین شده است.

وقتی اجرا به دستور return می رسد از تابع خارج شده مقدار برگشتی را به فراخواننده تابع برمی گرداند. اگر تابعی مقداری را بر نمی گرداند از کلمه void استفاده کنید. در این صورت تابع به دستور return نیاز ندارد.

تابع ممکن است دارای چند دستور return باشد تا بتواند در شرایط متفاوت مقادیر مختلف را برگرداند. در این صورت اولین دستورreturn که اجرا می شود موثر است.

مثال. برنامه زیر دو عدد را از ورودی گرفته عدد بزرگتر را نمایش می دهد.

```
#include <iostream.h>
int larger_of( int , int );
int main() {
   int x, y, z;
   cout<<"Enter two different integer values: ";
   cin>>x>> y;
   z = larger_of(x,y);

   cout<<"\nThe larger value is "<< z;

   return 0;
}

int larger_of( int a, int b) {
   if (a > b)
       return a;
   else
       return b;
}
```

همان طور که قبلا گفته شده است main هم یک تابع است البته با کمی تفاوت. تابع main نیازی به پروتوتایپ ندارد و اتوماتیک هنگام اجرای برنامه فراخوانی می شود و هرگز در کد برنامه صدا زده نمی شود تابع main یک عدد صحیح را می تواند برگرداند که صفر یا غیرصفر است. اگر اجرای برنامه بطور موفق به انتها رسیده باشد مقدار صفر و در غیراینصورت یک مقدار غیرصفر در مقابل دستور return نوشته می شود.

ارسال مقدار به تابع

وقتی مقداری به تابع ارسال می شود یک کپی از محتویات آرگومان به پارامتر نسبت داده می شود یعنی در اصل پارامتر یک کپی از متغیری است که به تابع ارسال می شود و مقدار آن خارج از تابع تغییر نمی کند. این روش ارسال یک متغیر با مقدار (passing variable by value) نامیده می شود. که روش معمول است. روش دیگر ارسال یک متغیر به تابع به صورت مرجع (called by reference) است. در این حالت به جای یک کپی از مقدار متغیر آدرس آن به تابع داده می شود بنابراین نام متغیر و نام پارامتربه یک مکان حافظه ارجاع می کنند. یعنی پارامتر متغیر جدیدی نیست بلکه همان متغیر قبلی با نام جدید است. در این حالت وقتی متغیر درون تابع تغییر می کند متغیر خارج از تابع هم تغییر می کند.

برای تعیین پارامتری به صورت مرجع کافی است علامت & (عملگر آدرس) قبل از پارامتر تابع در اعلان اضافه شود.

مثال. متغیر m به صورت مقداری به تابع ارسال شده است. خروجی تابع عدد ۱ است.

```
#include <iostream.h>
void f(int n) {
    n = 4;
}

int main() {
    int m = 1;
    cout << m << "\n";
    f(m);
    cout << m << "\n";
}
```

مثال. پارامتر number مرجع است.

```
#include <iostream.h>
void demo(float &number);
int main () {
    float num1;
    cout << "Please enter a number. \n";
    cin >> num1;
    cout << "Before the demo function your number is " << num1 << "\n";

    demo(num1);
    cout << "After the demo function your number is still " << num1 << "\n";

   return 0;
}

void demo(float &number) {
   number = number * 3;
   cout << "Inside the demo function the number is now " << number << "\n";
}
```

متغیرهای محلی

متغیرهائی که درون تابع تعریف می شوند متغیرهای محلی (local variables) نامیده می شوند. محلی بر این دلالت دارد که متغیرها تنها خاص تابع هستند و از متغیرهای هم نام در هر جای دیگر برنامه مجزا می باشند. تابع می تواند هر تغییری روی آنها بدهد بدون اینکه روی قسمت های دیگر برنامه اثر داشته باشد. متغیرهائی که خارج از هر بلاکی تعریف می شوند متغیر های سراسری (global variables) نامیده می شوند و در کلیه توابع قابل دسترسی هستند.

مثال. در برنامه زیر متغیر m سراسری است و توسط کلیه توابع قابل دسترسی و تغییر است. درحالیکه دو متغیر n از هم مستقل هستند و هرکدام تنها درون تابعی که اعلان شده اند تغییر می کنند.

```
#include <iostream.h>
int m;                  // M is a global variable

int f(int n) {
    n++;
    return n;
}
int g() {
    m++;
    return m;
}
int main() {
    int n = 5;
    cout << "n = " << n << "\n";
    cout << "f(n) = " << f(n) << "\n";
    cout << "n = " << n << "\n";

    m = 5;
    cout << "m = " << m << "\n";
    cout << "g() = " << g() << "\n";
    cout << "m = " << m << "\n";
}
```

سربارگذاری توابع

اکثر زبان های برنامه نویسی برنامه نویس را ملزم می کنند برای هر تابع نام منحصر بفردی را تعریف کنند. ویژگی چندریختی (polymorphism) در زبان C++ این امکان را فراهم کرده که یک نام برای بیش از یک تابع به طور مشترک استفاده شود. این

عمل سربارگذاری توابع (function overloading) هم نامیده می شود. برای اینکه کامپایلر توابع هم نام را از هم تشخیص دهد نوع و تعداد پارامترهای توابع باید با هم متفاوت باشد. کامپایلر از نوع آرگومان متوجه می شود کدام تابع فراخوانی شده است.

مثال.

```
#include <iostream.h>

float cube_number(float num);
int cube_number(int num);

int main() {
   float number;
   float number3;
   cout << "Please enter a number \n";
   cin >> number;
   number3 = cube_number(number);
   cout << number << " cubed is " << number3;
   return 0;
}

int cube_number(int num) {
   return num * num * num;
}

float cube_number(float num) {
   return num * num * num;
}
```

توابع پیش ساخته

علاوه بر توابع متعددی که می توانید بسازید C++ توابع آماده مفیدی را در اختیار می گذارد. توابع پیش ساخته (built-in functios) یا کتابخانه ای توابعی هستند که می توانید از آنها در برنامه استفاده کنید. برای استفاده از یک تابع کتابخانه ای ابتدا تابع و شرح آنرا را از مراجع برنامه نویسی پیدا کنید. هر تابع کتابخانه ای دارای یک هدر فایل است که پروتوتایپ تابع را دربردارد و توسط دستور #include باید به برنامه ضمیمه شود. تابع را طبق گرامر آن فراخوانی کنید. اگر در فراخوانی تابع اشتباهی وجود داشته باشد کامپایلر پیغام خطا می دهد.

توابع ریاضی

تابع	شرح	هدرفایل
double cos(double);	یک زاویه را گرفته کسینوس آنرا می دهد	<math.h>
double sin(double);	یک زاویه را گرفته سینوس آنرا می دهد	<math.h>
double tan(double);	یک زاویه را گرفته تانژانت آنرا می دهد	<math.h>
double log(double);	یک عددراگرفته لگاریتم طبیعی آنرا برمی‌گرداند	<math.h>
double pow(double,double);	عدد اولی را به توان دومی می رساند	<math.h>
double hypot(double,double);	دوضلع مثلث قائم الزاویه راگرفته و طول وتر را می‌دهد	<math.h>
double sqrt(double);	ریشه دوم عدد صحیح را برمی گرداند	<math.h>
int abs(int);	قدرمطلق یک عدد صحیح را برمی گرداند	<math.h>
double fabs(double);	قدرمطلق یک عدد اعشاری را برمی گرداند	<math.h>
double floor(double);	بزرگترین عدد صحیحی که کمتر یا مساوی آرگومان باشد را برمی گرداند	<math.h>
double ceil(double);	کوچکترین عدد صحیحی که بیشتر یا مساوی آرگومان باشد را برمی گرداند	<math.h>

تابع	شرح	هدرفایل
char *itoa(int value, char *buffer, int radix)	یک عدد صحیح را به معادل کاراکتری تبدیل می کند. Radix مبنای عدد صحیح است	<stdlib.h>
int atoi(const char *string);	یک کاراکتر را به یک عدد صحیح معادل تبدیل می کند	<stdlib.h>
int tolower(int c);	کاراکتر را به حالت کوچک تبدیل می کند	<ctype.h>
int toupper(int c);	کاراکتر را به حالت بزرگ تبدیل می کند	<ctype.h>
void *memcpy(*destination, *source, size)	هر چیزی که در source باشد در مقصد کپی می کند. Size تعداد کپی را معین می کند	<string.h>
void *memset(*destination, replacement, size)	distination را با کارکتری که در replacement قرار دارد تنظیم می کند	<string.h>

عملگر & با تابع itoa بکار می رود. ;x=itoa(&a)

مثال. memcpy و memset اکثرا برای داده هائی مانند ساختمان و آرایه بکار می روند. برنامه زیر آرایه x را در y کپی می کند.

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

int main() {
   int x[10]={1,2,4,6,6,9,0,11,7,2};
   int y[10];
   memcpy(x,y,10);
   return 0;
}
```

توابع زمان

با اضافه کردن فایل هدر time می توان به توابع زمان دسترسی پیدا کرد. برخی توابع پر استفاده در جدول زیر آمده است:

تابع	شرح	هدرفایل
time_t time(time_t *time);	تعداد ثانیه های سپری شده از ساعت 00:00 اول ژانویه ۱۹۷۰ را می دهد	<time.h>
struct tm *localtime(const time_t *time);	زمان را به ساختمان tm با ساعت محلی تبدیل می کند	<time.h>
double difftime(time_t time2,time_1 time1);	تفاوت دو زمان را می دهد	<time.h>
struct tm *gmttime(const time_t *time);	تایمر را به ساختمان tm با ساعت GMT تبدیل می‌کند	<time.h>
Char *ctime(const time_t *time);	زمان را به رشته ای شامل ساعت و تاریخ متناسب با زمان محلی در فرمت خوانا تبدیل می کند	<time.h>

مثال. برنامه زیر ساعت و تاریخ جاری سیستم را می دهد.

```
#include <ctime.h>
#include <iostream.h>
int main () {
  time_t rawtime;
  time ( &rawtime );
  cout << "Current date and time is: "<< ctime (&rawtime);
  return 0;}
```

توابع اعداد تصادفی

برای تولید کلید رمز یا بازی ها نیاز به تابعی برای تولید یک عدد تصادفی است. توابع زیر برای این منظور می تواند بکار رود.

تابع	شرح	هدرفایل
int rand (void);	یک عدد تصادفی بین 0 و RAND_MAX را بر می گرداند.	<stdlib.h>
int random(int n);	یک عدد تصادفی بین 0 و n-1 را بر می گرداند.	<stdlib.h>
srand ((time(NULL));	تولید اعداد تصادفی در هر اجرا	<stdlib.h> , <time.h>

مثال. برنامه زیر ۱۰ عدد تصادفی را نمایش می دهد.

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

int main() (
   int i,j;
   srand(time(NULL));
   for( i = 0;i < 10;i++ ){
      j= rand();
      cout << j << endl; }
   return 0;
}
```

فایل های ضمیمه

همانطور که در قسمت قبل مشاهده کردید C++ تعداد زیادی تابع کتابخانه ای دارد. شما هم می توانید توابع خود را درون یک فایل کتابخانه ای جمع کنید. اگر قصد دارید یک فایل کتابخانه بسازید ابتدا یک فایل هدر برای پروتوتایپ های کلیه توابع کتابخانه خود بسازید و با پسوند h. ذخیره کنید. اگر این فایل در فولدر جاری باشد توسط دستور #include "header.h" یا اگر در فولدر INCLUDE کامپایلر باشد توسط #include <header.h> پیدا می شود. سپس کلیه توابع را به یک فایل کتابخانه منتقل کنید. توابع کتابخانه ای معمولا انشعاب lib. یا a. را دارند.

توابع بازگشتی

این امکان وجود دارد که یک تابع خودش را صدا بزند. در این حالت تابع را بازگشتی (recursive) می نامند. یک تابع بازگشتی باید دارای شرطی باشد که به ازای آن خود را فراخوانی نمی کند. روش بازگشتی برای مسائلی استفاده می شود که مسئله اصلی می تواند به نسخه های کوچک تری از نسخه اصلی شکسته شود.

مثال. تابع بازگشتی زیر اعداد مابین آرگومانهایش را نمایش می دهد.

```
#include <iostream.h>

void recursive_function(int start, int end) {
    if (start < end) {
        cout << start << " ";
        recursive_function(start + 1, end);     }
}

int main() {
    recursive_function(1, 10);
    return 0;
}
```

البته توابع بازگشتی استفاده وسیعی ندارند زیرا خیلی کارآمد نیستند و اغلب زمانی کاربرد دارند که راه حل مسئله ساده تر می شود. خصوصا وقتی سرعت زیاد مهم نباشد.

مثال معروف بازگشتی تابع فاکتوریل است. تابع فاکتوریل در ریاضی به صورت حاصل ضرب اعداد صحیح غیر منفی به صورت زیر تعریف می شود:

```
0! = 1
n! = n * (n-1) * (n-2) * ... * 2 * 1    for n > 0
```

این تعریف براحتی به تابع زیر منتهی می شود:

```
long factorial(int n) {
   int k;
   long result = 1;

   for (k = 2; k <= n; k++)
      result = result * k;

   return result;
}
```

اما تابع فاکتوریل را به صورت زیر نیز می تواند تعریف کرد:

```
0! = 1
n! = n * (n-1)!    for n > 0
```

که با توجه ضابطه فوق تابع بازگشتی آن به صورت زیر نوشته می شود:

```
long factorial(int n)  {
   if ((n == 0) || (n == 1))    // stopping cases
      return 1;
   else                         // recursive case
      return n * factorial(n - 1);
}
```

تابع بازگشتی معروف دیگر فیبوناچی است که ضابطه آن صورت زیر است:

```
fib(0) = 1
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2)    for n > 1
```

تابع بازگشتی فیبوناچی به صورت زیر است:

```
long fib(int n)   {
   if ((n == 0) || (n == 1))      // stopping cases
      return 1;
   else                           // recursive case
      return fib(n - 1) + fib(n - 2);
}
```