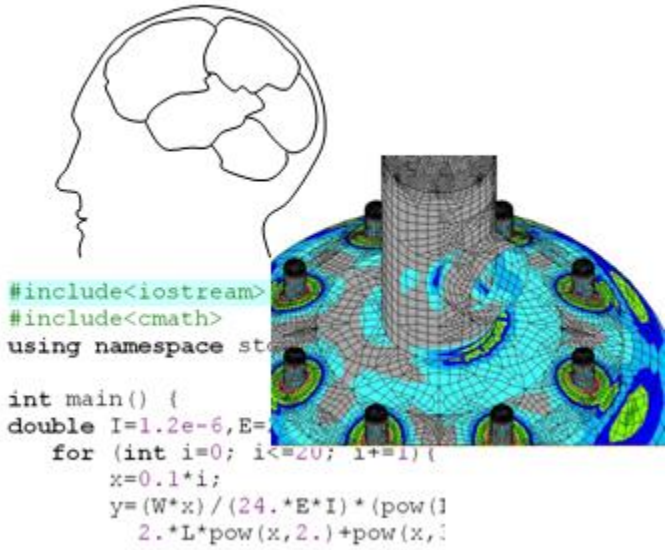


ME 110 Computation for Mechanical Engineering

Functions I:

Introduction and Basics (Part 1)

Contents

Effective use of functions is key to good programming.
Although we will cover only the basics in this course, we will spend two weeks on this important topic.

This week's content:

- The function concept
- General form of a function
- Using functions
- Function prototypes
- Return values
- **`void`** functions

The Function Concept

Functions allow the programmer to *divide* a programming task into parts, each divided task being performed by a function.

This *modular* concept allows complex tasks to be programmed piece-by-piece.

Also any *piece* that is used many times only needs to be defined once – simply call the function each time it is required.

We have seen *intrinsic* functions such as the **`sin()`**, **`fabs()`** and **`sqrt()`** functions defined in **`<cmath>`**.

In addition to intrinsic functions, other *programmer-defined* functions may be created.

A function contains a group of statements that is executed when it is called from some point in the program.

Generally, a function can be represented by the following figure:

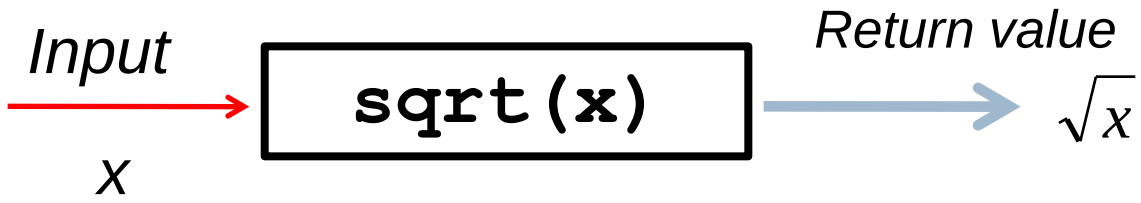
A function may:

- have zero, one, or more inputs
- have zero or one **`return`** value (but not more than one)
- perform tasks just as the **`main()`** function does:

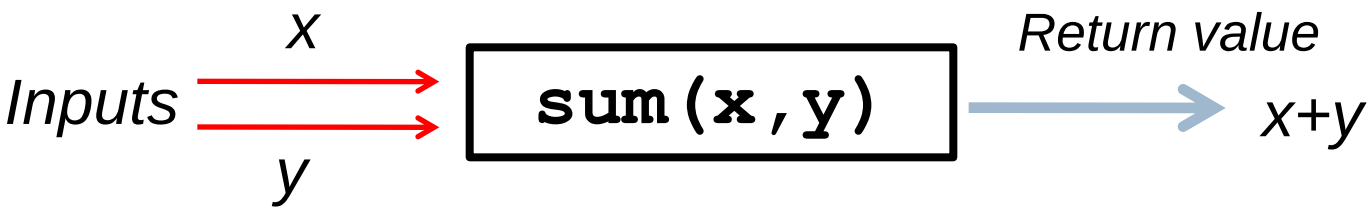
  input from the keyboard, output to the screen, set/process variables …

Examples

A function for the square-root of a variable;



A function for the summation of two variables;



General form of a function

```
type name(parameter1, parameter2, ...) {
   ...
   statements
   ...
   return value;
}
```

type is the data type of the function (i.e. type of the return value) such as float.

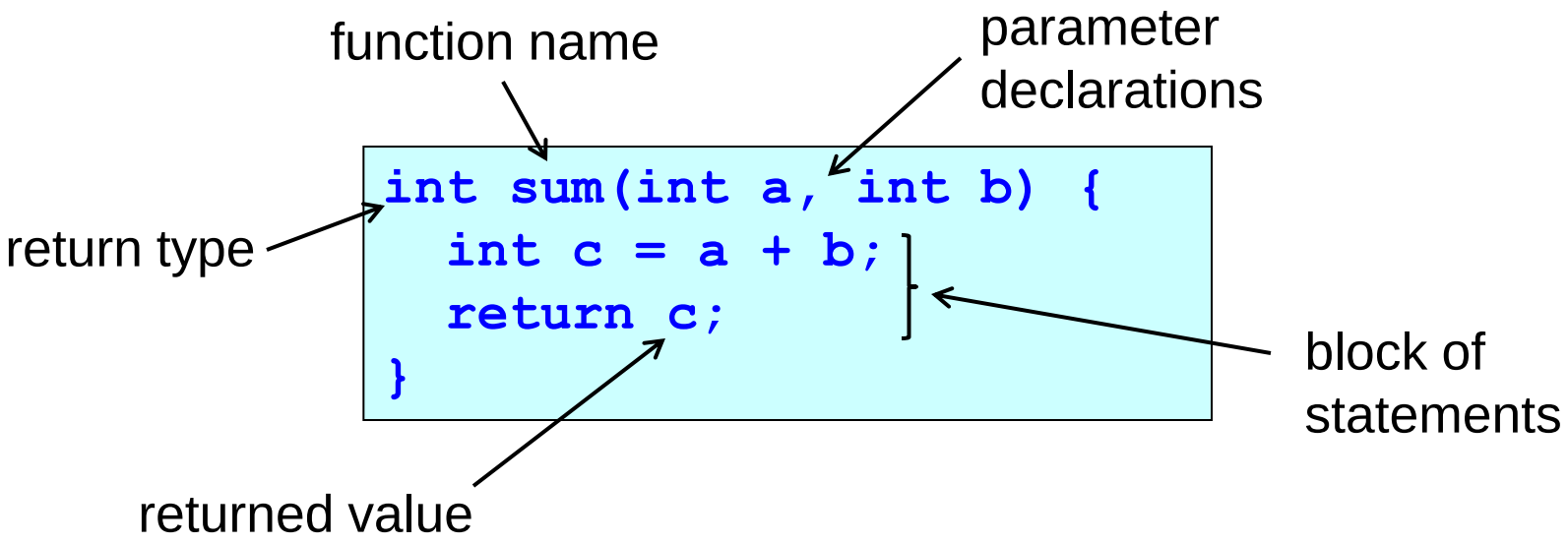
parameter1.. is a list of (input) arguments. Each consists of a data type followed by an identifier.

name is a valid C++ identifier representing the name of the function.

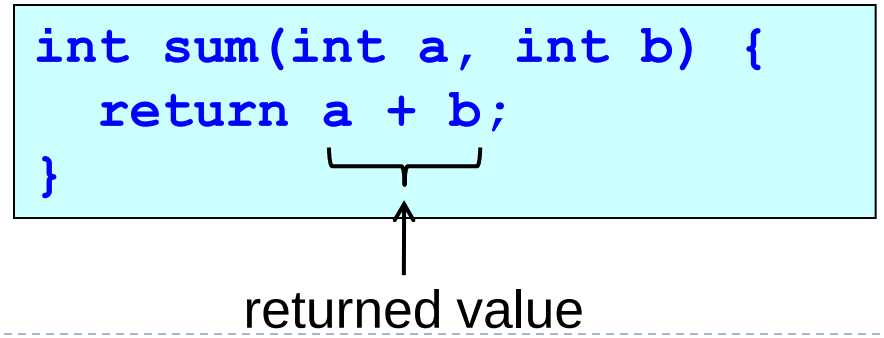
statements is the body of the function. These are executed when the function is called.

Example

A programmer-defined function that returns the sum of two integers.



This form is also valid



Using Functions

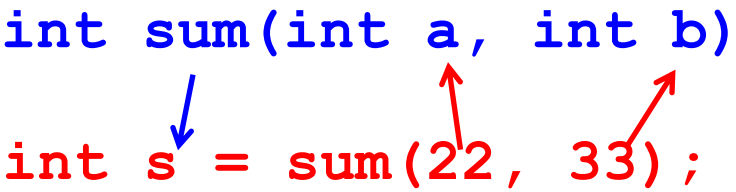
```
#include <iostream>
using namespace std;

int sum(int a, int b) {
    int c = a + b;
    return c;
}

int main () {
  int s = sum(22, 33);
  cout << "The sum is "
       << s << endl;
  return 0;
}
```

The function **sum()** is defined before the **main()** function so that the compiler recognises it when it is used.

Output

```
The sum is 55
```

Note the flow of information:



Example:The distance between two points

```
#include <iostream>
#include <cmath>
using namespace std;
double distance(double x1,double y1,double x2,double y2){
   return sqrt(pow(x2-x1,2.)+pow(y2-y1,2.));
}
int main () {
    double px1,py1,px2,py2;
    cout<<"input coordinates of first point\n";
    cin>>px1>>py1;
    cout<<"input coordinates of second point\n";
    cin>>px2>>py2;
    cout<<"distance is:"<<distance(px1,py1,px2,py2)<<endl;
  system("pause");
  return 0;
}
```

Function Prototypes

We sometimes wish to define a function after the **`main()`** function. In this case, a function prototype has to be given before **`main()`**. The function prototype declares the function without fully defining it. It consists of the function's return type, name, and parameter list.

```
#include <iostream>
using namespace std;

int sum(int a, int b);

int main() {
  int s = sum(22, 33);
  cout << "The sum is " << s << endl;
  return 0;
}

int sum(int a, int b) {
  int c = a + b;
  return c;
}
```

function prototype (parameter *names* are optional)

function definition

Return Values

The keyword **`return`** in a function has two jobs:

- to return a value to the calling statement
- to return execution to the calling statement

Execution is sent to **`sum()`**.

Execution is returned to **`main()`**.

Execution is returned to the OS.

```
#include <iostream>
using namespace std;

int sum(int a, int b) {
    int c = a + b;
    return c;
}

int main() {
   int s = sum(22, 33);
   cout << "The sum is "
        << s << endl;
   return 0;
}
```

The return statement may contain a variable, a constant, an expression or a function. Examples:

```
return x;
```
A variable.

```
return 10;
```
A literal constant.

```
return (a+b/c);
return a+b/c;
```
An expression.
The use of the parenthesis is optional.

```
return sqrt(2.0*x);
```
A function. First `sqrt()` is evaluated.

```
return;
```
No value (type `void` function).

The `return` statement in `main()`

As the **`main()`** function is called by the operating system (OS), **`return 0`** terminates the program returning the integer value 0 to the OS.
The value 0 indicates that the program was executed successfully.

Values other than **0** can indicate the occurrence of an error.

For example we can *catch* an error and terminate, passing **1** to the OS:

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {

   double x;
   cout << "input x: ";
   cin >> x;
   if (x<0) {
      cout << "Negative Value!" << endl;
      return 1;
   }
   cout << "Square root of x = " << sqrt(x);
   return 0;

}
```

Output

```
input x: -2
Negative Value!
```

A programmer-defined function may also contain more than one **`return`** statement. Again, the function terminates at a **`return`** statement passing execution back to the calling statement.

For example, the following function has five **`return`** statements; the function returns one of the characters **`A`**, **`B`**, **`C`**, **`D`** or **`F`**.

```
char grade(float average) {
  if ( average >=  0. && average < 50. ) return 'F';
  if ( average >= 50. && average < 70. ) return 'D';
  if ( average >= 70. && average < 80. ) return 'C';
  if ( average >= 80. && average < 90. ) return 'B';
  if ( average >= 90. )                  return 'A';
}
```

For example **`grade(54.2)`** returns the character **`'D'`** and **`grade(93.2)`** returns **`'A'`**.

Void Functions

Consider that we want to create a function that only outputs a message on the screen; i.e we do not want the function to return a value.

In this case, we use the **`void`** specifier in the function type declaration:

type **`void`** function.

`return` statement without a value.

Here, both **`Return`** statements are optional.

```
#include <iostream>
#include <string>
using namespace std;

void showMessage(string mess) {
   cout << mess << endl;
   return;
}

int main() {
   showMessage("This is a message.");
   return 0;
}
```

Output

```
This is a message.
```

Example: Calculating the area and perimeter of a circle

The area and perimeter of the circle are displayed inside user defined functions without returning value

```
#include <iostream>
#include <cmath>
using namespace std;
void circle(double rad){
   cout<< "area is:"<<M_PI*rad*rad<<endl;
   cout<< "perimeter is:"<<2.*M_PI*rad<<endl;
   return;
}
int main() {
   double r;
   cout<<"input the radius:";
   cin>>r;
   circle(r);
   system("pause");
   return 0;
}
```

Using Multiple Functions

The area and perimeter of the circle are displayed inside the **main function** by returning values from functions

```
#include <iostream>
#include <cmath>
using namespace std;
double area(double rad){
  return M_PI*rad*rad;
}
double perimeter(double rad){
  return 2.*M_PI*rad;
}
int main() {
  double r;
  cout<<"input the radius:";
  cin>>r;
  cout<<"area is:"<<area(r)<<endl;
  cout<<"perimeter is:"<<perimeter(r)<<endl;
  system("pause");
  return 0;
}
```