MA144: Problem Solving and Computer Programming

Lecture-17

Functions-1

Function (also referred as **module**)

A program segment that carries out some specific, well-defined task

Example

A function to add two numbers

A function to find the maximum of n numbers

A function will carry out its intended task whenever it is called

Can be called multiple times

Why Functions?

- Split a large problem into smaller pieces
- Easy to understand
- Easy to code
- Re–use of code

  (Functions can be called several times in the same program, allowing the code to be reused - avoids code repetition)

How functions are different from iterations (loops)?

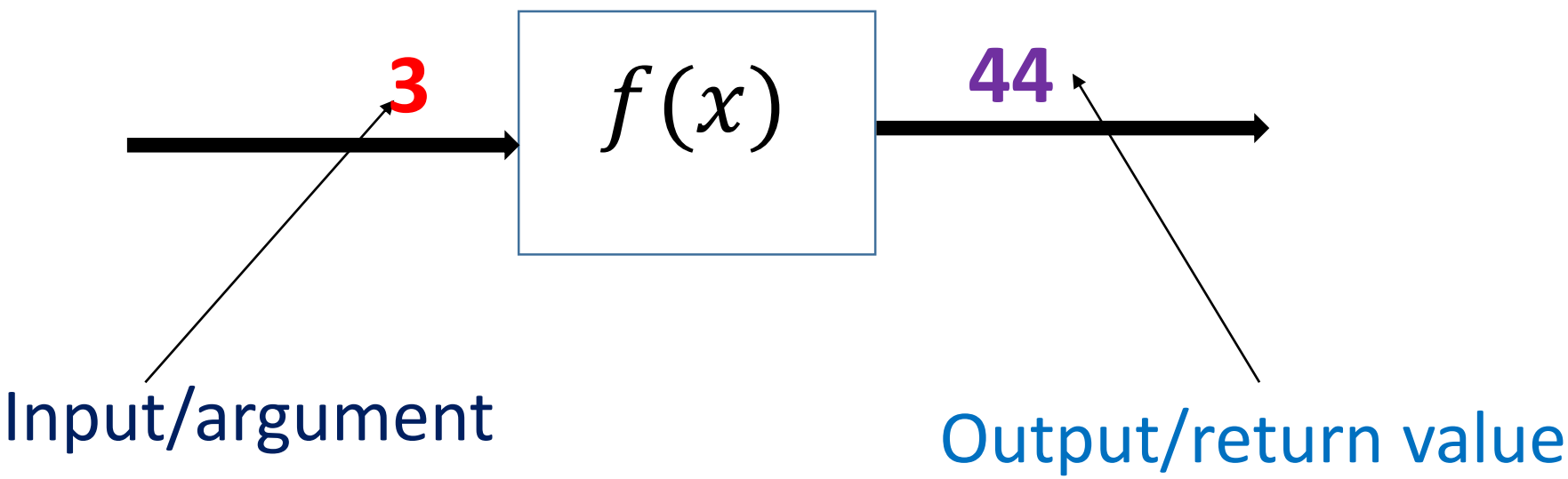
- Iterations are used when the same code is repeated at the same place again and again.
- By using functions, the same code can be used at different parts of the program.

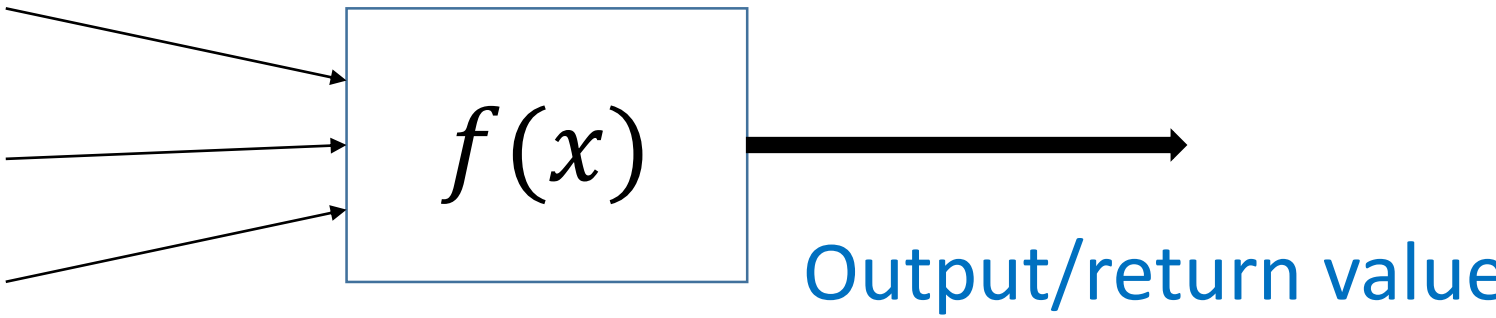
Function (mathematical perspective)

Let $f(x) = 2x^2 + 7x + 5$ be a function.

If $x = 3$, then what is $f(3)$?

$$f(3) = 2 * 3 * 3 + 7 * 3 + 5 = 44$$





Input/arguments

Function may take more than one input,
but it produces only one output

Classification w.r.t. communication between calling function and called function

- Function with arguments and return value
- Function with arguments and NO return value
- Function with NO arguments and return value
- Function with NO arguments and NO return value

Function components and their location

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

function prototype

int main()
{
  ------
  ------
function call
  ------
  ------
}

function definition
```

Calling function (caller)

Called function (callee)

Function Prototype

Function Prototype

Return data type

Arguments data type list

```
type  function_name(type, type, type);
```

Function name

Like a variable declaration, the function prototype tells the compiler

- the name of the function
- the type of arguments

  (not necessary the name of the arguments)
- the type of return type

Function Prototype

- **Function with arguments and return value**

```
int add(int, int);
int calc(int,int,char);
```

- **Function with arguments and NO return value**

```
void add(int, int);
void calc(int,int,char);
```

- **Function with NO arguments and return value**

```
int add();
int calc();
```

- **Function with NO arguments and NO return value**

```
void add();
void calc();
```

Function Call

Function Call

Actual arguments/ parameters list

```
function_name(arg1, arg2, arg3);
```

Function name

Function Call

- **Function with arguments and return value**

```
add(x, y);
calc(x, y, c);
```

- **Function with arguments and NO return value**

```
add(x, y);
calc(x, y, c);
```

- **Function with NO arguments and return value**

```
add();
calc();
```

- **Function with NO arguments and NO return value**

```
add();
calc();
```

- Control transferred to the definition of the function
- Code in the function definition is executed
- Control returns to the calling function

Function Definition

Function Definition

Formal arguments/ parameters list

Return data type

Function name

```
type function_name (type arg1, type arg2, type arg3)
{

   statements;

  }
```

A function definition specifies

1. the name of the function
2. the types and number of parameters it expects to receive
3. its return type
4. function body with the statements to perform the specific task assigned to the function

Function Definition

- **Function with arguments and return value**

```
type add (int a, int b)
 {
   statements;
   }
```

- **Function with arguments and NO return value**

```
void add (int a, int b)
 {
   statements;
   }
```

- **Function with NO arguments and return value**

```
int add ()
{
    statements;
  }
```

- **Function with NO arguments and NO return value**

```
void add ()
{
    statements;
  }
```

Return Statement

- Return statement terminates execution of the current function
- Control returns to the calling function
- If **return** **is an *expression then***
  - The value of the **expression** is returned as the value of the function call
  - Only one value can be returned (Exception for arrays)
- In case of no return type, the return type must be void (keyword)- at the function prototype and also at the function definition

Communication between calling function and called function

- The communication between calling function (or caller) and called function (or callee) done by sending arguments to called function.

- The calling function sends the arguments in two ways –
  call by value
  call by reference

- Formal parameters must match with actual parameters in order, number and data type.

First we consider call by value

```
#include<iostream>
using namespace std;
int add2(int,int);
int main()
{   int a,b, sum;
    cout<<"enter two numbers: ";
    cin>>a>>b;

    sum=add2(a,b);
    cout<<sum;
    return 0;
}
int add2(int x,int y)
{   int z;
    z=x+y;
    return z;
}
```

Function prototype

Function call

Function definition

Parameter Passing: call by value

Call by value

Passes the value of the argument to the function

Execution of the function does not change the actual parameters

All changes to a parameter done inside the function are done on a copy of the actual parameter

The value of the actual parameter in the caller is not affected

Avoids accidental changes

Some points to remember

A function cannot be defined within another function

All function definitions must be disjoint

Nested function calls are allowed

(i) A calls B, B calls C, C calls D, etc.

(ii) The function called last will be the first to return

A function can also call itself, either directly or in a cycle

(i) A calls B, B calls C, C calls back A (in a cycle)

(ii) Called recursive call or recursion (direct call)

Scope of a Variable

Scope of a variable - Within the block in which the variable is defined

Block = group of statements enclosed within { }

Local variable – scope is usually the function in which it is defined

So two local variables of two functions can have the same name, but they are different variables

Global variables – declared outside all functions (even main)

scope is entire program by default, but can be hidden in a block if local variable of same name defined

```
#include<iostream>
using namespace std;
int M=20;
int main()
{
  int M=40;
  cout<<"value of M= "<<M;


    return 0;
}
```

Global variable

local variable

```
value of M= 40
```

Scope resolution operator ::

```
#include<iostream>
using namespace std;
int M=20;
int main()
{
   int M=40;
   cout<<"local value of M= "<<M<<endl;

   cout<<"global value of M= "<<::M<<endl;


    return 0;
}
```

```
local value of M= 40
global value of M= 20
```

```
#include<iostream>
using namespace std;
int M=20;
int main()
{ int x=4;
  int M=40;
   M=M+9;
  ::M=::M+5;
   ::M=::M%x;
  cout<<"local value of M= "<<M<<endl;

  cout<<"global value of M= "<<::M<<endl;
  return 0;
}
```

```
#include<iostream>
using namespace std;
int M=20;
int main()
{ int x=4;
  int M=40;
   M=M+9;
  ::M=::M+5;
   ::M=::M%x;
  cout<<"local value of M= "<<M<<endl;

  cout<<"global value of M= "<<::M<<endl;
  return 0;
}
```

```
local value of M= 49
global value of M= 1
```

Find out the output

```
#include<iostream>
using namespace std;
int M=20;
int main()
{ int x=4;
  int M=40;
   M=M+9;
  ::M=M+5;
   ::M=::M%x;
  cout<<"local value of M= "<<M<<endl;

  cout<<"global value of M= "<<::M<<endl;
  return 0;
}
```

One Programming Example:
addition of two numbers
(in four different ways)

Function with arguments and return value

```
#include<iostream>
using namespace std;
int add2(int,int);
int main()
{  int a,b, sum;
    cout<<"enter two numbers: ";
    cin>>a>>b;
    sum=add2(a,b);
    cout<<"sum of "<<a<<" and "<<b<<" is "<<sum;
    return 0;
}
int add2(int x,int y)
{
    return x+y;
}
```

```
enter two numbers: 4 9
sum of 4 and 9 is 13
--------------------------
```

Function with arguments and NO return value

```
#include<iostream>
using namespace std;
void add2(int,int);
int main()
{  int a,b, sum;
    cout<<"enter two numbers: ";
    cin>>a>>b;
    add2(a,b);
    return 0;
}
void add2(int x,int y)
{
    cout<<"sum of "<<x<<" and "<<y<<" is "<<x+y;
}
```

```
enter two numbers: 4 9
sum of 4 and 9 is 13
```

Function with NO arguments and return value

```
#include<iostream>
using namespace std;
int add2();
int main()
{    int sum;
     sum=add2();
     cout<<"sum is "<<sum;
     return 0;
}
int add2()
{
     int x,y;
     cout<<"enter two numbers: ";
     cin>>x>>y;
     return x+y;
}
```

```
enter two numbers: 4 9
sum is 13
```

Function with NO arguments and NO return value

```
#include<iostream>
using namespace std;
void add2();
int main()
{
    add2();
    return 0;
}
void add2()
{
    int x,y;
    cout<<"enter two numbers: ";
    cin>>x>>y;
    cout<<"sum of "<<x<<" and "<<y<<" is "<<x+y;
}
```

```
enter two numbers: 4 9
sum of 4 and 9 is 13
```

Another Programming Example:
math calculator

```
#include<iostream>
using namespace std;
double calc(double, double, char);
int main()
{  double a, b, res;
   char op;
   cout<<" enter two integers: ";
   cin>>a>>b;
   cout<<"\n enter an operator (+,-,*,/): ";
   cin>>op;
   res=calc(a,b,op);
   cout<<"the result is "<<res;
   return 0;
}
double calc(double x,double y, char op1)
{
  switch(op1)
  {  case '+': return x+y; break;
     case '-': return x-y; break;
     case '*': return x*y; break;
     case '/': return x/y; break;
     default: return -1;
      }
 }
```

```
 enter two integers: 8 6

 enter an operator (+,-,*,/): /
the result is 1.33333
--------------------------------
```

```
 enter two integers: 7 9

 enter an operator (+,-,*,/): %
the result is -1
--------------------------------
```

Another Programming Example:
finding primes between 1 and n

```
#include<iostream>
using namespace std;
void isprime(int);
int main()
{  int i,n;
    cout<<"enter a number: ";
    cin>>n;
    for(i=2;i<=n;i++)
       isprime(i);
    return 0;
}
void isprime(int m)
{  int j,flag=0;
     for(j=2;j<=m/2;j++)
      {  if(m%j==0)
            flag+=1;
         }
         if(flag==0)
           cout<<m<<" ";
 }
```

```
enter a number: 1000
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 1
63 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 3
37 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 5
21 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 7
19 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 9
29 937 941 947 953 967 971 977 983 991 997
```