MA517M-Basic Programming Laboratory

Laboratory 6: Functions

Panchatcharam Mariappan[1]

[1]Associate Professor
Department of Mathematics and Statistics
IIT Tirupati, Tirupati

September 22, 2025

Functions

Functions

1. A collection of statements grouped to perform a specific task
2. Allows you to modularize the program
3. All variables in the function are local variables
4. Divide code into separate functions logically
5. Declaration: tells the compiler about its name, parameter (input) and return type
6. Definition: Provides the actual body
7. Method, sub-routine, procedure
8. Recursion: A function that calls itself is known as a recursive function

Functions

1. When your code contains plenty of lines, the developer gets more confused; therefore, it is recommended to break the code into pieces for better understanding and maintenance.
2. The portion of code may be used many times. For example, matrix-vector multiplication Matrix-matrix addition operations may be required many times in the same problem.
3. Instead of rewriting the code many times, it is better to use functions or sub-programs for matrix-vector multiplication and matrix-matrix addition, and then call that function whenever and wherever required.

Functions

The following are important concepts in function

- Function Definition
- Inputs of the function or arguments
- Outputs of the function or return values or output arguments

Functions Syntax

```
returnType functionName( input parameters )
{
  statement(s);
}
```

Functions

1. A collection of statements grouped to perform a specific task
2. Allows you to modularize the program
3. All variables in the function are local variables
4. Divide code into separate functions logically
5. Declaration: tells the compiler about its name, parameter (input) and return type
6. Definition: Provides actual body
7. Method, sub-routine, procedure
8. Recursion: A function that calls itself is known as a recursive function

Functions

- returnType is the data type of the value that the function returns, which can be void
- input parameters: can be empty, if non-empty, the same number of arguments should be specified while calling.

Functions

Area of a Rectangle

```
#include<iostream>
using namespace std;
//Function Definition  l and b are input arguments that receive
float CalculateArea(float l,float b)
{
  return l*b;
}
int main()
{
  float length, breadth;
  cout<<"Enter the Length of the Rectangle";
  cin>>length;
  cout<<"Enter the Breadth of the Rectangle";
  cin>>breadth;
  float area=CalculateArea(length,breadth); //Call the function or Caller
  cout<<"Area of the Rectangle is"<<area<<endl;
}
```

Functions

- Here, CalculateArea is a function which is called by the line `CalculateArea(length, breadth)`
- The main function invokes the `CalculateArea` function with its parameters
- Values of length and breadth are passed to the function `CalculateArea`.
- Product of length and breadth is computed and returned by the function `CalculateArea`
- Returned values are stored in the variable area

Note: Datatype in the parameter and return should match. Before calling the function, the function should be either defined or declared.

Functions

Calculate Area

```
#include<iostream>
using namespace std;
float CalculateArea(float l,float b);//Function Prototype or Function Declaration
int main()
{
  float length, breadth;
  float area;
  cout<<"Enter the Length of the Rectangle";
  cin>>length;
  cout<<"Enter the Breadth of the Rectangle";
  cin>>breadth;
  area=CalculateArea(length,breadth); //Call the function or Caller
  cout<<"Area of the Rectangle is"<<area<<endl;
}
//Function Definition, receives l and b and outputs the area
float CalculateArea(float l,float b)
{
  return l*b;
}
```

Functions

- Here, instead of defining the function at the beginning, we declare the function and then define the function after the main function.
- For function declaration, a semicolon at the end is a must. Do not put any semicolon at the end of the function definition.
- You can also define the function in a separate header file and include the same in the main file
- The returnType specifies the type of the return value, like int, float, or double.

Functions

- When there are no values to be returned, you can use void
- In C++, a function is allowed to return one value or no value (void).
- It cannot return multiple values. **Note:** [C++ does not allow you to return an array!]
- When defining the function, appropriately give the function name.
- Spacing between function names is not allowed.

Functions

- `Calculate Area(float a, float b)` is not a valid name
- `Print Number(float a)` is not a valid name.
- The number of arguments should be consistent when you define, declare, and call the function. However, C++ introduces default arguments, which are explained later.

Functions from C++ Libraries

Functions from C++ Libraries

Function	Description	Example
`sqrt(x)`	Square root of x	`sqrt(9.0)`
`cbrt(x)`	Cube root of	`xcbrt(8.0)`
`exp(x)`	Exponential	`exp(3.0)`
`log(x)`	Natural logarithm of x	`log(3.0)`
`pow(x,y)`	x raised to power y	`pow(2,5)`
`fmod(x,y)`	Remainder of x/y as a floating point	`fmod(13.657,2.333)`
`sin(x)`	trigonometric functions in radians	`sin(0)`
`cos(x)`	trigonometric functions in radians	`cos(0)`
`tan(x)`	trigonometric functions in radians	`tan(45.0/180.0*3.1415926)`
`log10(x)`	logarithm of x base 10	`log10(1000.0)`
`fabs(x)`	absolute value of x	`fabs(-8.0)`
`ceil(x)`	smallest integer not less than x	`ceil(9.2)`
`floor(x)`	smallest integer not greater than x	`floor(x)`

Functions

Given two integers, find which one is smaller than the other

```
#include<iostream>
using namespace std;
int smallest(int x, int y)
{
  if(x<y)
  {
    return x;
  }
  else
  {
    return y;
  }
}
int main(void)
{
  int a,b;
  cout<<"Enter two integers, let me tell you their relations\n";
  cout<<"Enter the first integer: "<<endl;
  cin>>a;
  cout<<"Enter the second integer: "<<endl;
  cin>>b;
  int c=smallest(a,b);
  cout<<c<<"is the smallest between "<<a<<" and "<<b<<endl;

  return 0;
}
```

Functions

Write a C++ program to find all prime numbers between 1 to n using functions

```
#include <iostream>
using namespace std;
void PrimeNumber(int x) {
  int Prime = 1;
  for(int j=2; j<=x/2; j++)   {
    if(x%j==0)      {
      Prime = 0;
      break;
    }
  }
  if(Prime==1)
    cout<<x<<endl;
}
int main()  {
  int n;
  int Prime;

  cout<<"Enter an integer to find prime numbers between 1 to n: "<<endl;
  cin>>n;
  cout<<"Prime numbers between 1 to "<<n<<" are:"<<endl;
  for(int i=2; i<=n; i++)
    PrimeNumber(i);
  cout<<"\n";
  return 0;
}
```

Functions

Write a C++ program to find all prime numbers between 1 to n using functions

```
#include <iostream>
using namespace std;
int factorial(int x)  {
  if(x>1)
    return x*factorial(x-1);
  else
    return 1;
}
int main()  {
  int n;
  cout<<"Enter an integer to find the factorial: "<<endl;
  cin>>n;
  int f;
  if(n<0)
  {
    cout<<"Factorial of a Negative number is not possible\n";
    return 0;
  }
  else if(n==0)
    f=1;
  else
    f=factorial(n);
  cout<<n<<"! = "<<f<<endl;
  return 0;
}
```

Static, Local, Global

Function Overloading

```
#include<iostream>
using namespace std;
int x=100;
void useLocal()
{
  int x=75;
  cout<<"Entry Local x = "<<x<<endl;
  x/=5;
  cout<<" Exit Local x = "<<x<<endl;
}
void useStaticLocal()
{
  static int x=64;
  cout<<"Entry Static Local x = "<<x<<endl;
  x/=2;
  cout<<" Exit Static Local x = "<<x<<endl;
}
void useGlobal()
{
  cout<<"Entry Global x = "<<x<<endl;
  x+=8;
  cout<<" Exit Global x = "<<x<<endl;
}
```

```
int main()
{
  int x=10;
  cout<<"Entry Main Local x = "<<x<<endl;
  {
    int x=6;
    cout<<"Entry Main Inner Scope Local x = "<<x<<
        endl;
  }
  cout<<"Entry Main Outer Scope Local x = "<<x<<
      endl;
  useLocal();
  useStaticLocal();
  useGlobal();
  x*=10;
  useStaticLocal();
  useGlobal();
  useLocal();
  cout<<" Exit Main Local x = "<<x<<endl;
  return 0;
}
```

Static, Global, Local

Types	Scope	Life time
Global	The entire life	The lifetime of the application
Static	The function where it is declared within	The lifetime of the application
Automatic or local	The function where it is declared within	While the function is executing
Dynamic	Determined by pointers that reference this memory	Until the memory is freed

Static, Global, Local

1. C++ has another feature called default argument functions. These default values would be used if the caller omits the corresponding actual argument in calling the function.
2. However, default arguments must be specified in the function prototype, and that should not be repeated in the function definition.
3. The default arguments are resolved in their positions.
4. Default arguments can only be used to substitute the trailing arguments to avoid ambiguity.

Function Overloading

Function Overloading

- C++ also introduces function overloading or function polymorphism.
- Same function name can be used for different versions, if their arguments differ
- Upon execution, the respective version will be selected.

Function Overloading

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

template <typename T>
T bigger(T a, T b)
{
  return (a>b)? a:b;
}

float maximum(float a,float b, float c)
{
  return bigger(a,bigger(b,c));
}
int maximum(int a, int b, int c)
{
  return bigger(a,bigger(b,c));
}

double maximum(double a, double b, double c)
{
  return bigger(a,bigger(b,c));
}
```

```cpp
int main()
{
    int a=5,b=7,c=8;
    float x=7.5, y=-8.5, z=6.5;
    double u=3.14, v=2.8, w=9.8;
    cout << "Max = " << maximum(a,b,c) << endl;
    cout << "Max = " << maximum(x,y,z) << endl;
    cout << "Max = " << maximum(u,v,w) << endl;
    return 0;
}
```

Function with Arrays and Pointers

Function with Arrays and Pointers

1. Arrays can also be passed to a function
2. Caution: Pass the size of the array for computation purposes, if required

Functions with Arrays

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

float DotProduct(float x[], float y[], int
      length)
{
  float dot=0.0;
  for(int i=0;i<=length;i++)
  {
    dot+=x[i]*y[i];
  }
  return dot;
}
```

```

int main()
{

  float x[100],y[100],z,xl,xu,yl,yu;
  int length=40;

  //cin codes here
  float hx=(xu-xl)/(float)length;
  float hy=(yu-yl)/(float)length;
  for(int i=0;i<=length;i++)
  {
    x[i]=xl+hx*i;
    y[i]=yu+hy*i;
  }
  cout<<DotProduct(x,y,length)<<endl;
  return 0;
}
```

Pass By Value

Function with Arrays and Pointers

1. Copies the actual value of an argument into the formal parameter of the function
2. Parameter modified inside the function, but does not affect the argument

Functions with Arrays

```
#include<iostream>
using namespace std;
void swap(int x, int y)
{
  int temp=x;
  x=y;
  y=temp;
}
int main(void)
{
  int a,b;
  cin>>a>>b;
  cout<<"a = "<<a<<" b= "<<b<<endl;
  swap(a,b);
  cout<<"a = "<<a<<" b= "<<b<<endl;
  return 0;
}
```

Pass By Reference

Pass By Reference

1. Copies the address of an argument into the formal parameter of the function
2. Parameter modified inside the function, affects the value of the argument

Pass By Reference

```
#include<iostream>
using namespace std;
void swap(int *x, int *y)
{
  int temp=*x;
  *x=*y;
  *y=temp;
}
int main(void)
{
  int a,b;
  cin>>a>>b;
  cout<<"a = "<<a<<" b= "<<b<<endl;
  swap(&a,&b);
  cout<<"a = "<<a<<" b= "<<b<<endl;
  return 0;
}
```

Functions with Pointers

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

float DotProduct(float *x, float *y, int
      length)
{
  float dot=0.0;
  for(int i=0;i<=length;i++)
  {
    dot+=x[i]*y[i];
  }
  return dot;
}
```

```

int main()
{
  float *x,*y,z,xl,xu,yl,yu;
  int length=40;
  x=new  float[length];
    y=new  float[length];
  //cin codes here
  float hx=(xu-xl)/(float)length;
  float hy=(yu-yl)/(float)length;
  for(int i=0;i<=length;i++)
  {
    x[i]=xl+hx*i;
    y[i]=yu+hy*i;
  }
  cout<<DotProduct(x,y,length)<<endl;
  return 0;
}
```

Recursion

Recursion

- It is a process in which a function calls itself
- The function is called a recursive function.
- Example: Factorial Function

$$f(n) = n * f(n-1), f(1) = 1$$

- There should be at least two cases, one is the base case, and another one is the recurrence relation
- Write the base case in the if part and write the non-base case in the else part
- Most of the problems could be solved using if and else conditions; there are a few cases where you may need if..else if..else

Recursion

Factorial

```
#include<iostream>
using namespace std;
unsigned long int factorial(unsigned long int n)
{
  if(n==1)
  return 1;
  else
  return n*factorial(n-1);
}
int main()
{
  unsigned long int n;
  cin>>n;
  cout<<n<<"! = "<<factorial(n)<<endl;
  return 0;
}
```

Direct and Indirect Recursion

1. When the body of a function calls itself, it is called **direct recursion**
2. When the body of a function A calls another function B, B calls another function C, and C calls function A, then it is called **indirect recursion** or mutual recursion.

Indirect Recursion: Write a C++ program to print numbers 1 to 10 in such a way that after dividing the number by 3,

- if the reminder is 0, add 2,
- if the reminder is 1, add 3,
- if the reminder is 2, subtract 1

Indirect Recursion

```
#include<iostream>
using namespace std;
int n=1;

void reminder0();
void reminder1();
void reminder2();

void reminder0()
{
  if(n<=10)
  {
    cout<<n+2<<endl;
    n++;
    reminder1();
  }
}

void reminder1()
{
  if(n<=10)
  {
    cout<<n+3<<endl;
    n++;
    reminder2();
  }
}
```

```

void reminder2()
{
  if(n<=10)
  {
    cout<<n-1<<endl;
    n++;
    reminder0();
  }
}


int main()
{
  reminder1();
  return 0;
}
```

Direct and Indirect Recursion

Direct Recursion	Indirect Recursion
Only one function called by itself	More than one function are called by the other function and number of time
Called by the same function	Called by other function
When function called next time, value of local variable will be stored	value will automatically lost when any other function is using the local variable
Engaged with memory location	For local variables, it is not
Dynamic	Determined by pointers that reference this memory Until the memory is freed

Direct and Indirect Recursion

- Generally slower than non-recursive programs
- Need to make a function call
- Program must save all its current state and retrieve it again later.
- Time Consuming
- Requires more memory to hold intermediate states in a stack.

`main` **function**

`main` function

- Remember that main is also a function.
- Can we pass arguments to main? Yes

`main` **function**

Let us reconsider the factorial problem

```
    #include<iostream>
using namespace std;
unsigned long int factorial(unsigned long int n)
{
  if(n==1)
  return 1;
  else
  return n*factorial(n-1);
}
int main(int argc, char **argv)
{
  cout<<argv[1]<<"! = "<<factorial(atoi(argv[1]))<<endl;
  return 0;
}
```

```
./a.out 5
```

`main` **function**

Let us reconsider the area of the rectangle problem

```
  #include<iostream>
using namespace std;
int main(int argc, char **argv)
{
  cout<<"Number of Arguments Passed = "<<argc<<endl;
  for(int i=0;i<argc;i++)
  cout<<"arguments["<<i<<"] = "<<argv[i]<<endl;
  float a=stof(argv[1]);
  float b=stof(argv[2]);
  cout<<"Area of the Rectangle is "<<a<<"x"<<b<<" = "<<a*b<<endl;
  return 0;
}
```

```
./a.out 5.2 6.5
```

Functions

- Beware of existing C++ functions. Use them appropriately to reduce your development time
- Give the name of the function appropriately
- Break functions into smaller tasks
- Distinguish between functions that return values and that don't
- Do not use the same name for function arguments and the parameters
- Collection of functions would be a good practice to debug and modify
- Try to fit the function in one line (don't use too many parameters)
- Number of arguments and parameters should agree
- function prototype with parameter name is valid. That is, both int `sum(int x , int y )` and int `sum(int , int )` are valid

Functions: Common Mistakes

- `double x,y` instead of `double x, double y` in the function parameter produces a compilation error
- Placing a semicolon after the function definition is a syntax error
- `double x` in the function parameters and using `double x;` (declaring again) inside the function would lead to a compilation error
- Defining a function inside another function
- Forgetting to return a value from a recursive function when it is needed
- Forgetting to write the base case in the recursive function

Thanks

Doubts and Suggestions

panch.m@iittp.ac.in