Functions

Marwadi University

Department of Diploma Engineering

UNIT 3

Object Oriented Programming with C++ – 09CE2301

Prof. Meghnesh Jayswal

Objectives

- Functions
- String function
- Call by value and Call by reference
- Return by reference
- Inline Function
- Difference between inline and macro
- Default argument
- Const argument
- Function Overloading

Function

- A function is a code that performs a specific task and optionally returns a value to the calling program or/and receives values(s) from the calling program.
- A function is a block of code that performs a specific task.

- **Types of function**
- There are two types of function
- Standard library functions(Predefined / Built-in Function)
- User-defined functions

Built-in Function

- In C++, the functions of related functionalities are grouped together in a single standard library, such as the cin and cout functions, that are grouped together in the iostream library. Other examples are iostream.h, math.h, string.h etc.

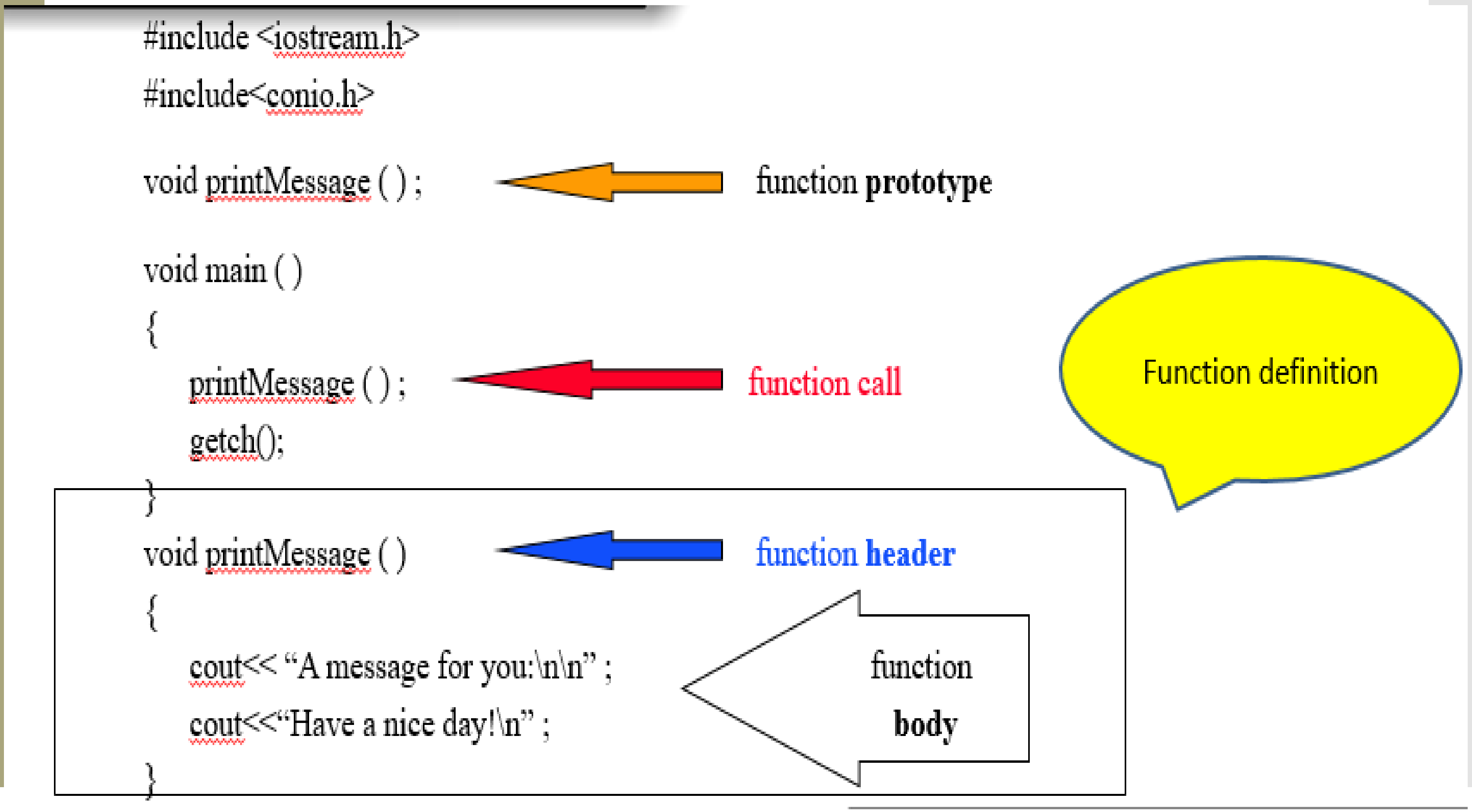
Using a Standard Library Function

```
#include <iostream.h>
#include <string.h>
int main()
{
char string1[] = "Hello";
cout<< "The length of string1 is "<< strlen(string1)<<”characters."<<endl;
return 0;
}
```

Output: The length of string1 is 5 characters

User-defined functions

- Functions that programmers create for specialized tasks.

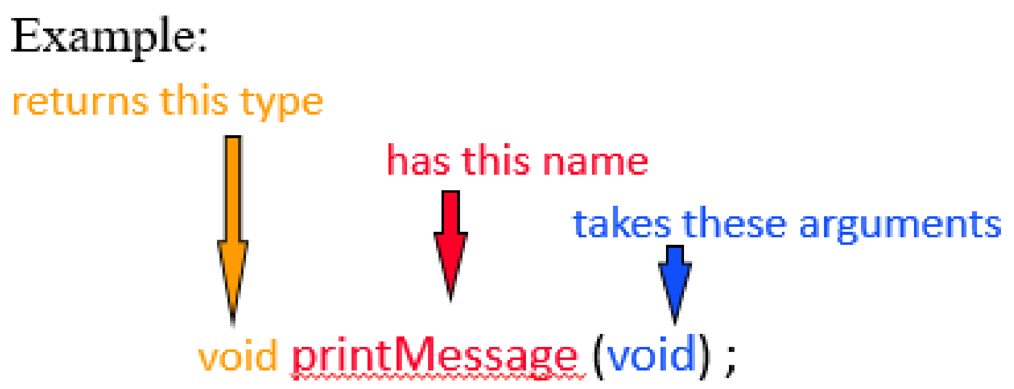


```
#include <iostream.h>
#include<conio.h>

void printMessage ( ) ;

void main ( )
{
    printMessage ( ) ;
    getch();
}
void printMessage ( )
{
    cout<< "A message for you:\n\n" ;
    cout<<"Have a nice day!\n" ;
}
```

Parts of Functions

Format of Function

```
return-type name(param list)
{
    var and const declarations
    executable statements
    return statement(s)
}
```



String Functions

Sr.No	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strrev(s1);** Returns to reverse the given string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

Task

- Task to do: Find more Examples and programs to other string functions

Call by Value

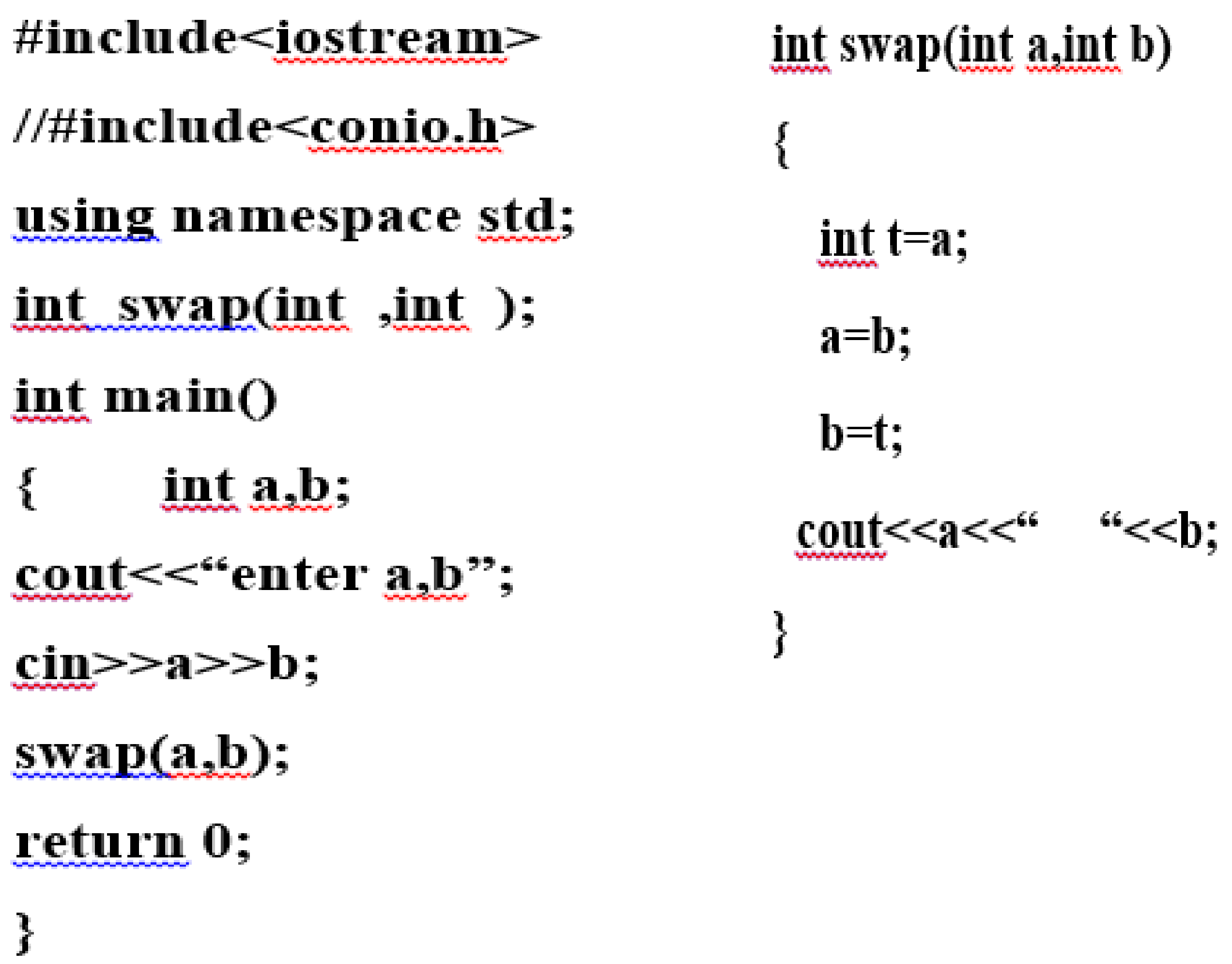
A function call passes arguments by value.

- The called function creates a new set of variables and copies the values of arguments into them.

- The call by value method of passing arguments to a function copies the actual values(Function call) of an argument into a formal parameter(function definition/declaration) of the function.

- The function does not have access to the actual variables in the calling program and can only work on the copies of values.

Example



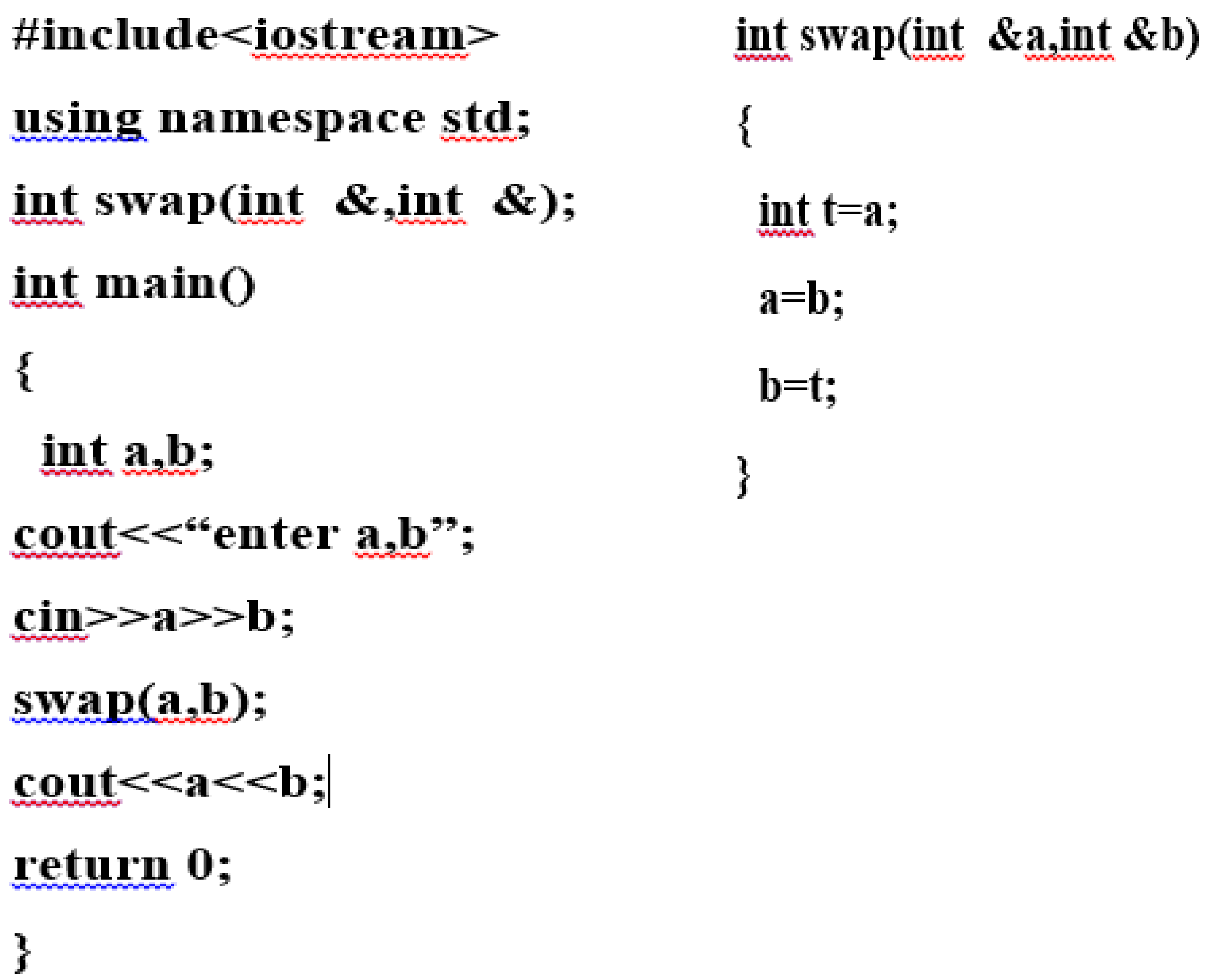
```
#include<iostream>
//#include<conio.h>
using namespace std;
int  swap(int  ,int  );
int main()
{        int a,b;
cout<<“enter a,b”;
cin>>a>>b;
swap(a,b);
return 0;
}
```

```
int swap(int a,int b)
{
   int t=a;
   a=b;
   b=t;
  cout<<a<<“    “<<b;
}
```

Call by Reference

- The **call by reference** method of passing arguments to a function copies the reference of an argument into the formal parameter(**parameters** as they appear in function declarations). Inside the function, the reference is used to access the actual argument (**parameters** as they appear in function call) used in the call. This means that changes made to the parameter affect the passed argument.

Example



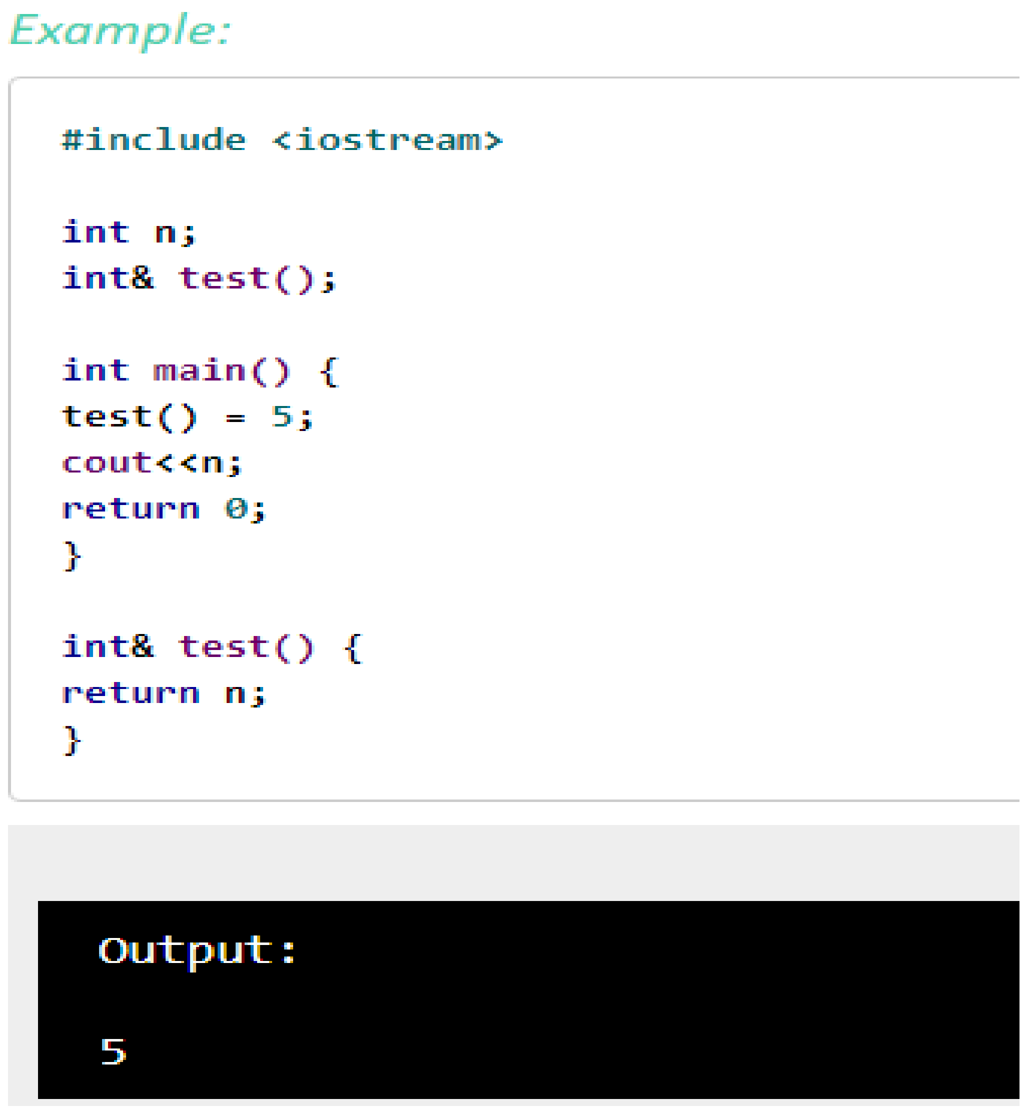
```
#include<iostream>
using namespace std;
int swap(int  &,int  &);
int main()
{
  int a,b;
cout<<“enter a,b”;
cin>>a>>b;
swap(a,b);
cout<<a<<b;
return 0;
}
```

```
int swap(int  &a,int &b)
{
  int t=a;
  a=b;
  b=t;
}
```

Return by Reference

- A C++ program can be made easier to read and maintain by using references rather than pointers.
- A C++ function can return a reference in a similar way as it returns a pointer.
- When a function returns a reference, it returns an implicit pointer to its return value.
- This way, a function can be used on the left side of an assignment statement.

Return by Reference



Example:

```cpp
#include <iostream>

int n;
int& test();

int main() {
test() = 5;
cout<<n;
return 0;
}

int& test() {
return n;
}
```

```
Output:

5
```

Return by Reference

```
#include<iostream.h>
#include<conio.h>
int & large (int & p, int & q);
int main()
{ clrscr();
   int l,k;
   cout<<"\n Enter values of l and k: ";
   cin>>l>>k;
   large (l,k)=120;
   cout<<" l= "<<l << " k="<<k;
   return 0;
}
int & large (int & p, int & q)
{
   if (p>q) return p;
   else return q;
}
```

OUTPUT

```
Enter values of l and k : 4 8
l= 4 k=120
Enter values of l and k : 9 2
l= 120 k=2
```

Inline function

- C++ inline function is powerful concept that is commonly used with classes.
- If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.
- To inline a function, place the keyword inline before the function name and define the function before any calls are made to the function.
- The compiler can ignore the inline qualifier in case defined function is more than a line.
- Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality.
- A function definition in a class definition is an inline function definition, even without the use of the inline specifier.

Inline function

Syntax:- inline function-header

```
        {
              function body
        }
```

Eg: inline double cube(double a)

```
{
  return (a*a*a);
}
```

Inline function

Max. from 3 no. using Inline Function

```
#include<iostream.h>
#include<conio.h>
inline int max(int a,int b,int c)
{      return (a>b ?((a>c)?a:c) : ((b>c? b: c)));    }
int main()
{   int x,y,z;
cout<<"Enter x,y,z";
cin>>x>>y>>z;
cout<< "Max no is"<<max(x,y,z);
   getch();       }
```

```
Enter x,y,z
 1   2   3

Max no is3
```

Multiple inline function

```
#include<iostream.h>
#include<conio.h>
inline float mul(float x,float y)
{     return (x*y);        }
inline double div(double p,double q)
{       return (p/q);      }
int main()
{   float a=12.345;
    flaot b=9.82;
cout<<mul(a,b)<<"\n";
cout<<div(a,b)<<"\n";
getch();       }
```

Output:-

```
121.228
1.25713
```

- **Inlining is only a request to the compiler, not a command. Compiler can ignore the request for inlining.**
- **Compiler may not perform inlining in such circumstances like:**

1. For functions returning values, like as loop, switch or goto exist.
2. If returns statement exist
3. Function contain static variable
4. Inline functions are recursive.

- **Inline function makes program run faster because overhead(optimize or reduce) of function call and return is eliminated**

Difference between Macro and Inline Function

- A macro is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro.
  - Inline function is a function that is expanded in line when the function is called. That is the compiler replaces the function call with the function code (similar to macros).
  - The disadvantage of using macros is that the usual error checking does not occur during compilation.

Difference between Macro and Inline Function

- Macros are expanded by the pre-processor, while inline functions are parsed by the compiler. There are several important differences: ·
- Inline functions follow all the protocols of type safety enforced on normal functions. ·
- Inline functions are specified using the same syntax as any other function except that they include the inline keyword in the function declaration.
- Syntax of Pre-processor : #define pi 3.14

Example of Macro

```
#include<iostream.h>
#include<conio.h>
#define square(s) s*s
main()
{
cout<<"Square of 5 is.."<<square(5)<<endl;
getch();
}
```

Output:

```
The square of 5 is 25
```

- Above code is right but it can not work as square (2+3)

Default arguments

- Definition: A default argument is a value provided in function declaration that is automatically assigned by the compiler if caller of the function doesn’t provide a value for the argument.
- Use: The default arguments are useful while making a function call if we do not want to take effort for passing arguments that are always same.
- Rule: It is not allowed to assign default value to any variable, which is in between the variable list. We must add defaults from right to left. We can’t provide argument in middle of an argument list.

Default arguments

- **Syntax**

```
int sum(int a, int b=10, int c=5);
```

Default arguments

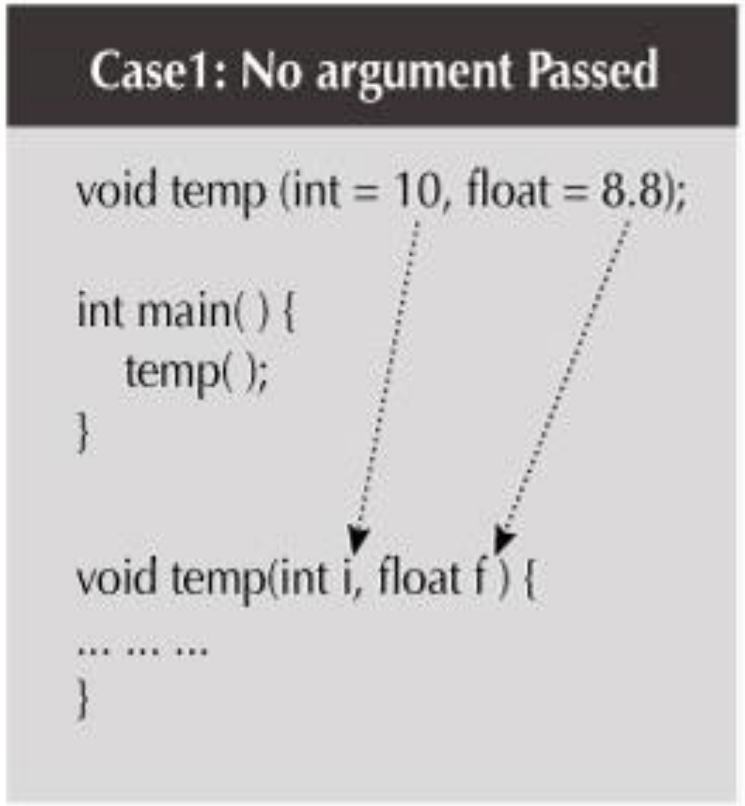
- **Valid: Following function declarations are valid –**

```
int sum(int a=10, int b=20, int c=30);
int sum(int a, int b=20, int c=30);
int sum(int a, int b, int c=30);
```

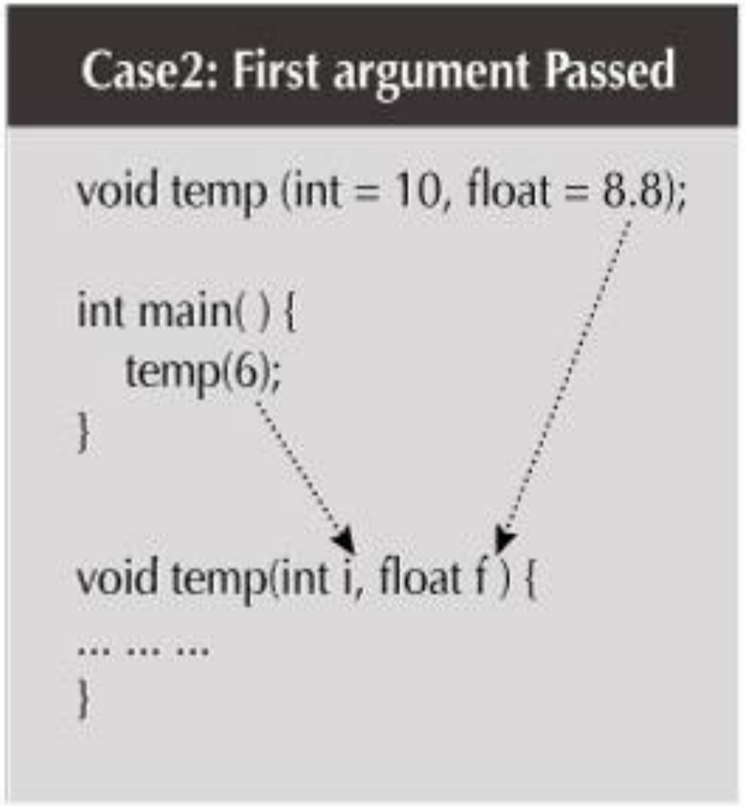
- **Invalid: Following function declarations are invalid –**

```
int sum(int a=10, int b, int c=30);
int sum(int a, int b=20, int c);
int sum(int a=10, int b=20, int c);
```

Default Arguments

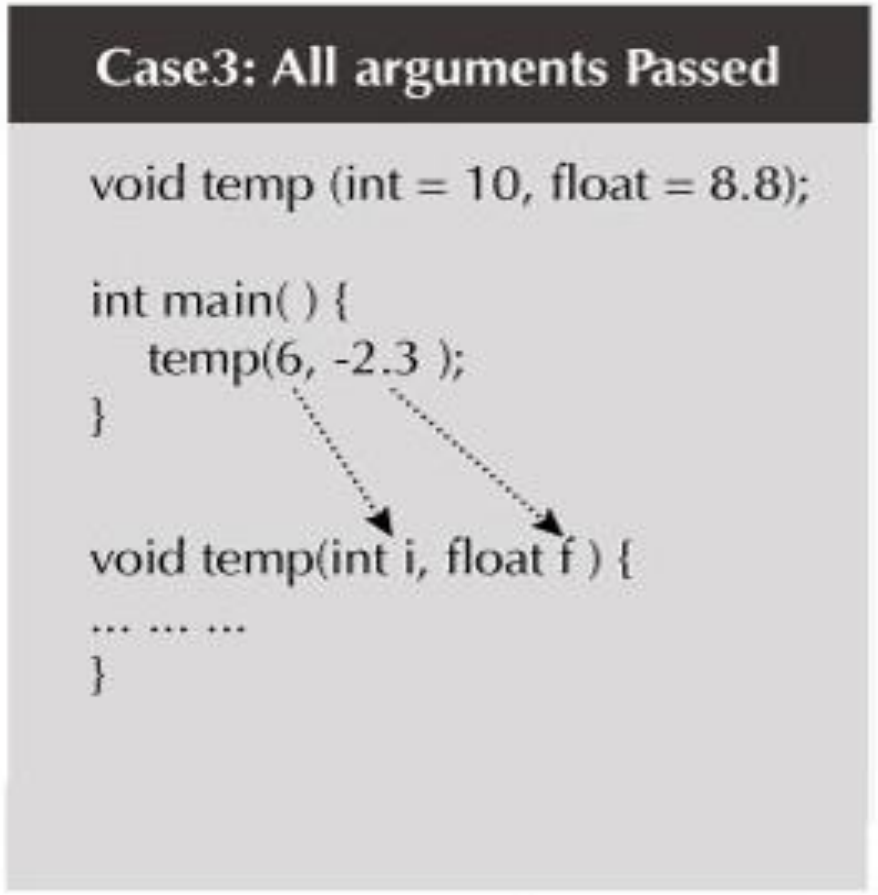

Case1: No argument Passed

```
void temp (int = 10, float = 8.8);

int main( ) {
   temp( );
}

void temp(int i, float f ) {
... ... ...
}
```


Case2: First argument Passed

```
void temp (int = 10, float = 8.8);

int main( ) {
   temp(6);
}

void temp(int i, float f ) {
... ... ...
}
```

Default Arguments


Case3: All arguments Passed

```
void temp (int = 10, float = 8.8);

int main( ) {
    temp(6, -2.3 );
}


void temp(int i, float f ) {
... ... ...
}
```

Example – Default Arguments

```
#include <iostream>
using namespace std;
int sum(int a, int b=10, int c=20);
int main()
{            cout<<sum(1)<<endl;
             cout<<sum(1, 2)<<endl;
             cout<<sum(1, 2, 3)<<endl;
  return 0;
}
int sum(int a, int b, int c)
{
  int z;
  z = a+b+c;
  return z;
}
```

In this case **a** value is passed as **1** and **b** and **c** values are taken from default arguments.

In this case **a** value is passed as **1** and **b** value as **2**, value of **c** is taken from default arguments.

In this case all the three values are passed during function call, hence no default arguments have been used

Const Arguments

In C++, an argument to a function can be declared as const

Syntax:

```
type function_name(const data_type variable_name=value);
```

Example:

```
int max(const int a=3, int b);
```

```
int strlen(const char *p);
```

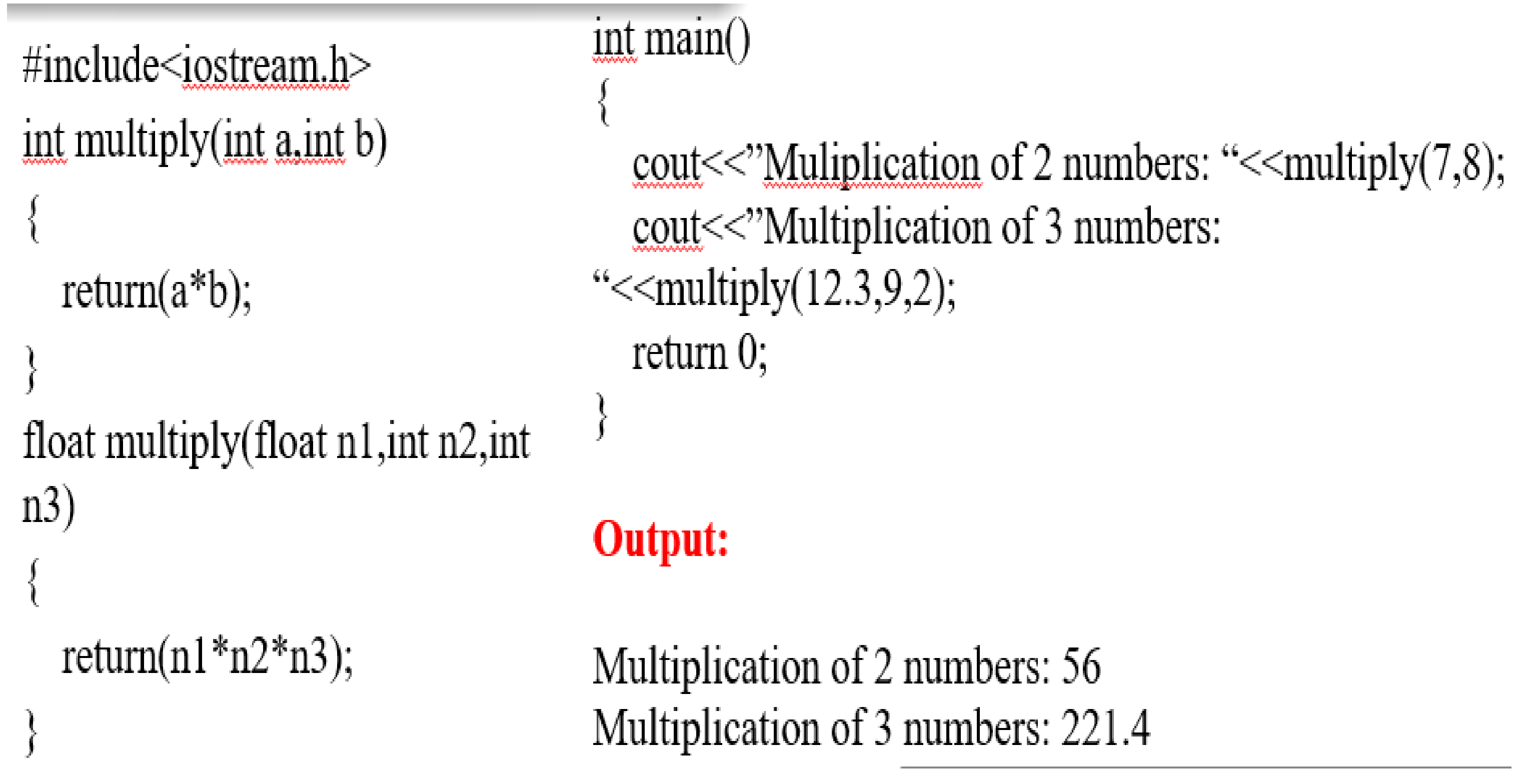
```
int length(const string &s);
```

The qualifier const tells compiler that function should not modify argument. Compiler will generate error when condition is violated.

Function Overloading

- Function overloading is a feature in C++ where two or more functions can have the same name but different parameters.
- Concept of Polymorphism
- The overloaded function must be **different in their argument** list and with **different data types.**

Example



```
#include<iostream.h>
int multiply(int a,int b)
{
   return(a*b);
}
float multiply(float n1,int n2,int
n3)
{
   return(n1*n2*n3);
}
```

```
int main()
{
   cout<<”Muliplication of 2 numbers: “<<multiply(7,8);
   cout<<”Multiplication of 3 numbers:
“<<multiply(12.3,9,2);
   return 0;
}
```

Output:

```
Multiplication of 2 numbers: 56
Multiplication of 3 numbers: 221.4
```

Thank You