BSCS-Spring-2021

CL118-Programming Fundamentals

Lab Manual 11

LAB 11 Manual

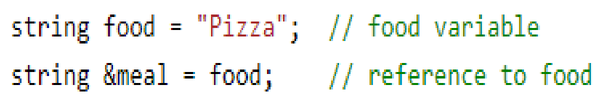
Programming Fundamentals

Topic: C++ Functions

Parameter passing by reference:

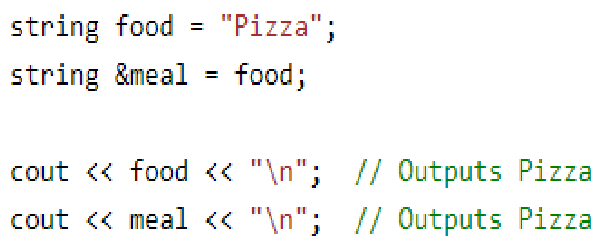
Previously we used normal variables when we passed parameters to a function. You can also pass a reference to the function. This can be useful when you need to change the value of the arguments. A reference variable is a "reference" to an existing variable, and it is created with the & operator.

When a variable is passed *by reference*, what is passed is no longer a copy, but the variable itself, the variable identified by the function parameter, becomes somehow associated with the argument passed to the function, and any modification on their corresponding local variables within the function are reflected in the variables passed as arguments in the call. Arguments by reference do not require a copy. The function operates directly on aliases passed as arguments.



```
string food = "Pizza";  // food variable
string &meal = food;    // reference to food
```

Now, we can use either the variable name food or the reference name meal to refer to the food variable:

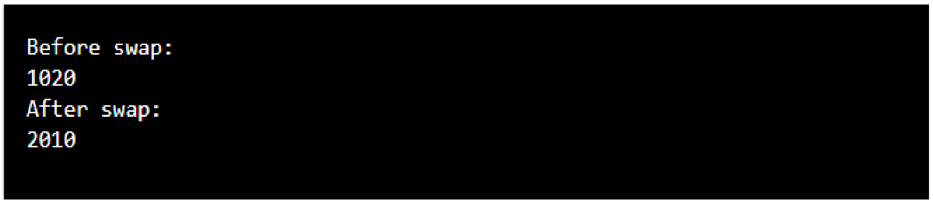


```
string food = "Pizza";
string &meal = food;

cout << food << "\n";  // Outputs Pizza
cout << meal << "\n";  // Outputs Pizza
```

Example:

```cpp
void swapNums(int &x, int &y) {
  int z = x;
  x = y;
  y = z;
}

int main() {
  int firstNum = 10;
  int secondNum = 20;

  cout << "Before swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  // Call the function, which will change the values of firstNum and secondNum
  swapNums(firstNum, secondNum);

  cout << "After swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  return 0;
}
```

Output:



```
Before swap:
1020
After swap:
2010
```

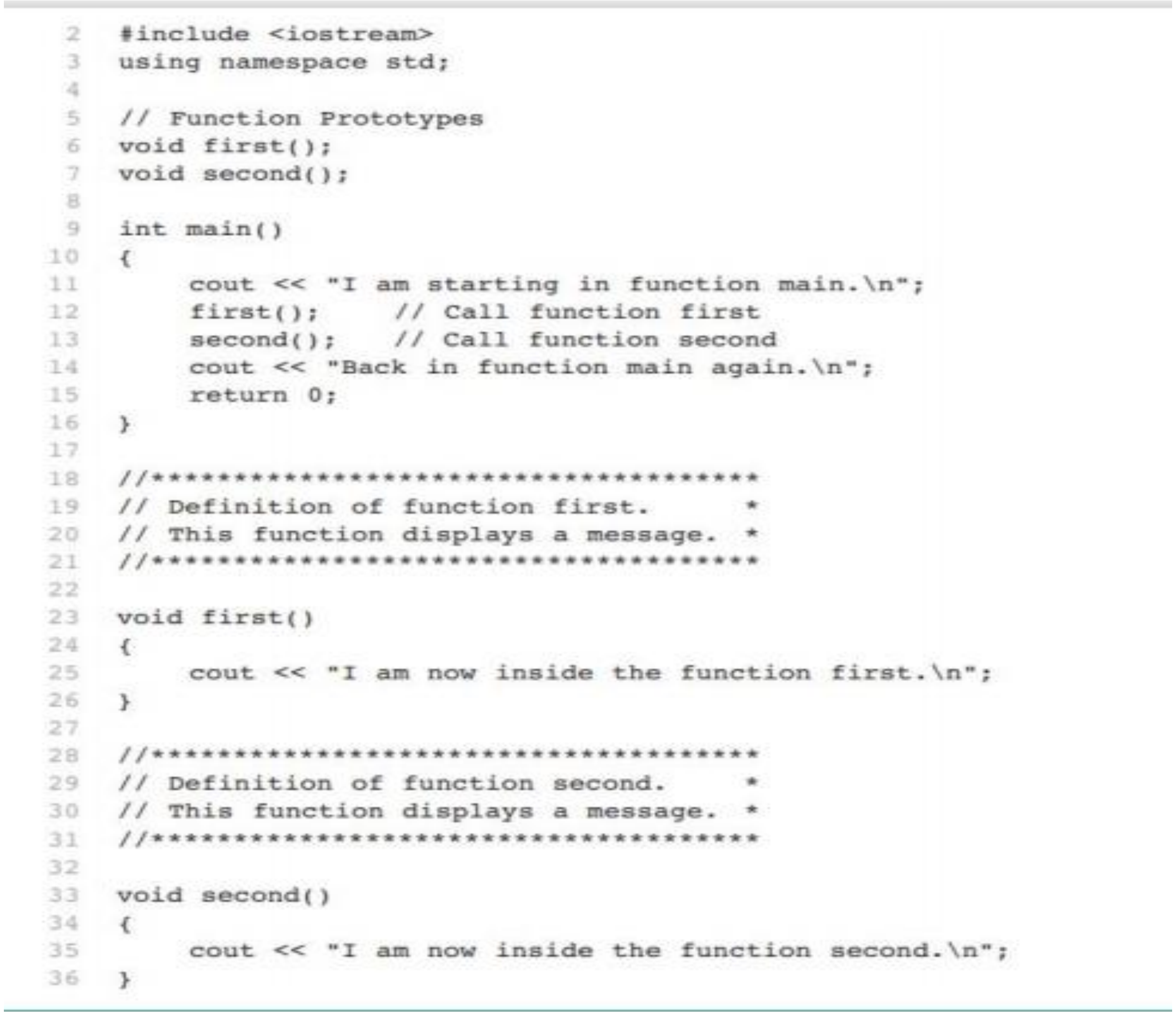
Function prototype

The prototype looks similar to the function header, except there is a semicolon at the end.

Function prototypes are also known as function declarations.

You must place either the function definition or either/the function prototype ahead of all calls to the function. Otherwise the program will not compile.

Example:



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

// Function Prototypes
void first();
void second();

int main()
{
    cout << "I am starting in function main.\n";
    first();     // Call function first
    second();    // Call function second
    cout << "Back in function main again.\n";
    return 0;
}

//*************************************
// Definition of function first.      *
// This function displays a message.  *
//*************************************

void first()
{
    cout << "I am now inside the function first.\n";
}

//*************************************
// Definition of function second.     *
// This function displays a message.  *
//*************************************

void second()
{
    cout << "I am now inside the function second.\n";
}
```

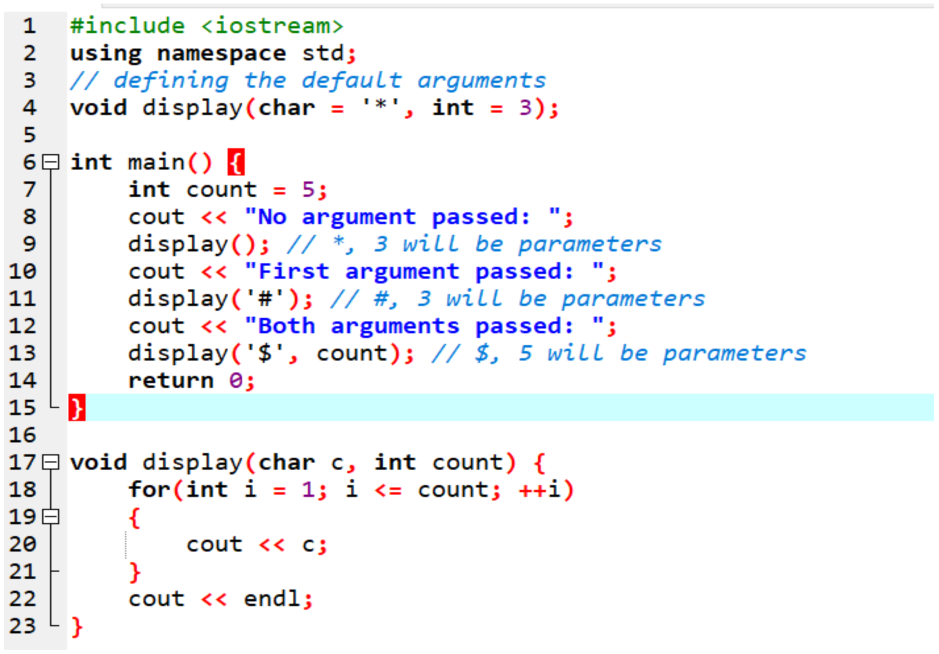
Default Arguments:

In C++ programming, we can provide default values for function parameters. For example, a function with three parameters may be called with only two. For this, the function shall include a default value for its last parameter, which is used by the function when called with fewer arguments.

If a function with default arguments is called without passing arguments, then the default parameters are used.

However, if arguments are passed while calling the function, the default arguments are ignored.

Example:



```cpp
#include <iostream>
using namespace std;
// defining the default arguments
void display(char = '*', int = 3);

int main() {
    int count = 5;
    cout << "No argument passed: ";
    display(); // *, 3 will be parameters
    cout << "First argument passed: ";
    display('#'); // #, 3 will be parameters
    cout << "Both arguments passed: ";
    display('$', count); // $, 5 will be parameters
    return 0;
}

void display(char c, int count) {
    for(int i = 1; i <= count; ++i)
    {
        cout << c;
    }
    cout << endl;
}
```

Output:

```
No argument passed: ***
First argument passed: ###
Both arguments passed: $$$$$
```

Return Values:

The void keyword indicates that the function should not return a value. If you want the function to return a value, you can use a data type (such as int, string, etc.) instead of void, and use the return keyword inside the function. You can also store the result in a variable.

Example:

```cpp
int myFunction(int x, int y) {
  return x + y;
}

int main() {
  int z = myFunction(5, 3);
  cout << z;
  return 0;
}
// Outputs 8 (5 + 3)
```

Lab Tasks

Q1) Write a complete C++ program with the two alternate functions specified below, of which each simply triples the variable count defined in main. Then compare and contrast the two approaches by printing the value of variable before and after function calls of the following two functions. Also use the concept of function prototype n this question.

a) Function **tripleCallByValue( )** that passes a copy of count call-by-value, triples the copy and returns the new value. The function prototype is: **int tripleCallByValue(int);**

b) Function **tripleByReference( )** that passes count with true call-by-reference via a reference parameter and triples the original copy of count through its alias (i.e., the reference parameter). The function prototype is: **void tripleByReference(int &);**

Q2) Write a program that simulates coin tossing. For each toss of the coin, the program should print Heads or Tails. Let the program toss the coin 100 times and count the number of times each side of the coin appears. Print the results. The program should call a separate function **flip( )** that takes no arguments and returns 0 for tails and 1 for heads.

Q3) Write a function to divide variable 'a' with 'b'. Store the result in a separate variable and return it from the function. Practice default parameters by calling your function with 1 argument and with 2 arguments. Use the concept of function prototyping for this question. The signature of function is as follows: **int divide (int a, int b=2)**

Q4) Write a function **qualityPoints( )** that inputs a student's average and returns 4 if a student's average is 90–100, returns 3 if the average is 80–89, returns 2 if the average is 70–79, returns 1 if the average is 60–69 and returns 0 if the average is lower than 60.

Q5) An integer number is said to be a perfect number if the sum of its factors, including 1 (but not the number itself), is equal to the number. For example, 6 is a perfect number, because 6 = 1 + 2 + 3. Write a function **perfect( )** that determines whether parameter number is a perfect number. Use this function in a program that determines and prints all the perfect numbers between 1 and 1000.

Submission Instructions:

1. Save all .cpp files with your roll no and task number e.g. i20XXXX_Task01.cpp

2. Now create a new folder with name ROLLNO_LAB11 e.g. i20XXXX_LAB11

3. Move all of your .cpp files to this newly created directory and compress it into .zip file.

4. Now you have to submit this zipped file on Google Classroom.