Lab 5: Functions with Parameters

Variable Scope

The scope of a variable is a region of the program where a variable 'exists'. We have seen that a variable created inside a loop will no longer exist once the loop ends. The same can be said about functions. A variable created in function A will not exist in function B. Consider the following program. As you can see, the compiler will complain "Symbol 'number' could not be resolved." This means that the variable 'number' created in main does not exist in print_function.

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

void print_function();

int main()
{
   int number = 100;

   print_function();

   return 0;
}

void print_function()
{
   cout << number;
}
```

Parameters and Arguments

As seen in the previous examples, data will return back from a function by returning it with the 'return' statement. It is possible to send data to funtions by sending variables as parameters through the function call. Here is a basic example of a program using parameters. Notice the syntax.

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

// This is a prototype using parameters. There are 2 acceptable syntaxes:
// 1. returnType functionName ( dataType1, dataType2, ...) ;
// 2. returnType functionName ( dataType1 parameter1_name, dataType2
parameter2_name, ...) ;
void print(int, const double);

int main()
{
   const double PI = 3.14159;

   // Function call arguments can be variables or literals
   print(10, PI);

   return 0;
}

// Function definition
// returnType functionName ( dataType1 parameter1_name, dataType2 parameter2_name, ...)
void print (int param1, const double param2){
   cout << "parameter 1 is " << param1 << endl
        << "parameter 2 is " << param2 << endl;


}
```

How Parameters Work

IMPORTANT! A variable passed to a function through parameters creates a new variable local to the function. If that variable is changed in the callee function, the value does not change in the caller function!

```
#include <iostream>
using namespace std;
void add_10(double);
int main()
{
   double var = 5.0;
   cout << "in main before call : "<< var << endl;
   add_10(var);
   cout << "in main after call : "<< var << endl;
   return 0;
}
void add_10(double foo)
{
   foo += 10;
   cout << "in function : "<< foo << endl;
}
```

Example Program 1

One of the advantages of functions is the ability to be called multiple times and given different input. This saves the need to write the same code multiple times.

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

void print_sum(int, int);

int main()
{
    int a = 10;
    int b = 20;
    int c = 30;
    int total;

    print_sum(b, c);
    print_sum(a, c);
    print_sum(a, b);

    return 0;
}

void print_sum(int first, int second)
{
    cout << "The sum of " << first << " + " << second << " = "
         << first + second << endl;
}
```

Example Program 2

Good style dictates a function should only preform a single specific task. This way a function is easy to understand and debug. It also makes functions more modular and able to be reused.

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

// prototype
int sum(int, int);

int main()
{
    int num1,
        num2,
        total;

    cout << "Welcome to my super calculator 2000." << endl
         << "Please enter two numbers to find their sum!" << endl;

    cin >> num1 >> num2;

    // function call
    total = sum(num1, num2);

    cout << num1 << " + " << num2 << " = " << total << endl;

    return 0;
}

// function definition
// This function returns the sum of the two numbers passed to it.
int sum(int A, int B)
{
    return (A + B);
}
```

Example Program 3

The following program uses the Pythagorean theorem to find the third side of a right triangle. Notice the difference between a local and passed variable, otherwise known as an argument.

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


double P_thm(double, double);

int main()
{
    double num1, num2, answer;

    cout << "Please enter two sides of your triangle to find the third\n";
    cin >> num1 >> num2;

    answer = P_thm(num1, num2);

    cout << num1 << "^2 + " << num2 << "^2 = " << answer << "^2";

    return 0;
}

double P_thm(double A, double B) //
{
    // this is a local variable to p_thm
    double C;

    C = sqrt((A * A) + (B * B));

    return C;
}
```

Example Program 4

Here is another example. This is a simple calculator to add and subtract.

```
#include <iostream>
using namespace std;
// prototypes
int sum (int, int);
int subtract (int, int);
void print(char, int, int, int);

int main(){
    int num1,
        num2,
        total;
    char op;
    bool flag = false;

    cout << "Welcome to a simple calculator." << endl
         << "Please enter the first number, "
         << "operation (+ or -), then second number." << endl;

    cin >> num1 >> op >> num2;
    switch(op){
        case '+':
           total = sum(num1, num2);
            break;
        case '-':
            total = subtract(num1, num2);
            break;
        default:
            cout << "Invalid operator" << endl;
            flag = true;
            break;
    }
    if(!flag)
        print (op, num1, num2, total);

    return 0;
}
void print (char op, int A, int B, int result)
{
    cout << A << " " << op << " " << B << " = " << result;
}

int sum (int A, int B)
{
    return (A + B);
}

int subtract (int A, int B)
{
    return (A - B);
}
```