Programming 1: Tutorial 6

Introduction

For this tutorial you are going to be advancing the concept of controlling the flow of the program, this time using the final method of remote statements may be obeyed (subroutines).
You can either over write previous question solutions with new ones or you can have a project for each of the exercises – don't forget to make them blank applications.

Hopefully you remember the basic layout of a C++ program?
If not here is a refresher, basic C++ programs like the ones you have been asked to do here have the following layout:

```
#include <iostream>
// include statements up here

int main()
{
  // block of code between the { and } brackets

  return 0;
}
```

As we are dealing with functions are layout may look something like this:

```
#include <iostream>
// include statements up here

int main()
{
  // block of code between the { and } brackets

  return 0;
}


int My_Function(int x, int y)
{
  // block of code between the { and } brackets

  return value;

}
```

It may also help if you go through a design stage for each exercise where you plan and puzzle through the logic before actually coding.

To complete these exercises you may need to reference the notes, either online or use the printed ones provided. Doing this isn't a bad thing; most programmers have to look up the syntax to make sure there code is correct.

There is sample code at the end of the tutorials, typing and running this code may help you when it comes to solving the exercises.

It is also important to understand what is happening with the questions, rather than just typing it into the compiler. If you look at it, run it and don't understand then just ask.

Errors

Again after the actual tutorial is a look at another type of error and what results it can cause.

Code Example

Input and run the following as an example of a basic program containing a function.

```
// header files first
#include <iostream>
// namespaces next
using namespace std;
// function prototypes
int GetAge();

int main () // then main
{
	// stuff in here

	return 0; // at the very end just before...
} // .. the terminating parentheses

// function definitions down here
int GetAge()
{
	int userAge = 0;

	cout << "Your age please ";
	cin >> userAge;

	return userAge;
}
```

Examples

Don't forget to look at the example code supplied at the end of this tutorial. Compile and run the examples and make sure you understand what is occurring.

Questions

1) What is the difference between an argument and a parameter?

2) What is a local variable?

3) Why not make all variables global?

4) By looking at the follow function prototype what data type does it return:
```
float getPlayerSpeed();
```

Exercises

1) Write a function which draws a line of n asterisks, n being passed as a parameter to the function. Write a driver program (a program that calls and tests the function) which uses the function to output a row of n asterisks, n being a value entered by the user.

2) Write a function named toSeconds which takes three integer parameters, a number of hours, number of minutes, and number of seconds and returns the total number of seconds. The prototype should look like.

```
int toSeconds(int hours, int minutes, int seconds);
```

3) Write a function named promptYN which takes one string parameter. It prints the string (which is a question the caller supplies) on cout, then prompts the user to enter Y or N. It then reads one character from cin. If the character is a 'Y', the function returns true, if the character is a 'N', it returns false, if the character is neither of the above, your function should prompt the user again for a 'Y' or 'N' and read another character, continuing with this until a Y or N is entered. The prototype should be the following.

```
bool promptYN(string question);
```

Function in a Loop

This builds on the first exercise.
Write a function which draws a m * n block of asterisks, n being passed as a parameter to the function.
We should have this function already coded in the first exercise.

We need to write a program that calls the function m times, therefore we call our function within a loop.
The values m and n should be entered by the user.

Functions with Loops

Write a program very similar to the one above. It should contain a function that has one int value passed to it. This functions should then use nested loops to build a m * n block of asterisks.

Logic Errors

To start of with this week we are going to look at the fourth and final type of error – Logical Errors

Logic Errors

- Program produces incorrect results, does not necessarily cause the program to crash
- Generally down to down to incorrect algorithm
- Program source code compiles and links without errors
- Errors are detected by checking output of program
- The cause(s) of the error must be determined by a logical analysis of the error and the source code

The Most Common Logic Errors

Are likely to result from using incorrect operators:
Using the assignment operator (=) instead of the comparative operator (==)
Using OR (||) instead of AND (&&)
Using NOT (!) when it isn't necessary or by getting arithmetic operators confused.

Try the following code out:

```
#include <iostream>

using namespace std;

int main()
{
	int num = 0;
	int guess = 0;
	bool bTest = true;
```

```
	while(bTest = true)
	{
		cout << "What number am i thinking of ";
		cin >> guess;
		if(guess == num)
		{
			cout << "correct\n";
			bTest = false;
		}
	}

	return 0;
}
```

This is an example of a logical error that will cause your program to run infinitely.

Sample Programs

Basic Examples in One

```
// function prototypes
int GetAge();
string GetName();
void TypeCast(const int value, char letter);

// constant - gobal in scope
const int DOG_YEARS = 7;

int main()
{
	int userAge = GetAge();
	cout << "Your age in dog years is " << (userAge*DOG_YEARS) <<
endl;

	string name = GetName();
	cout << "Your name is " << name << endl;

	TypeCast(userAge, 'a');

	return 0;
}

void TypeCast(const int value, char letter)
{	// example of conversion
	cout << (int)letter << '\n' << (char)(letter+value) << endl;
}

int GetAge()
{	// variable local only to this function
	int userAge = 0;

	cout << "Your age please ";
	cin >> userAge;

	return userAge;
}

string GetName()
{	// variable local only to this function
	string userName;

	cout << "Your name please ";
	cin >> userName;

	return userName;
}
```

Example of `static` variables

```
#include <iostream>     // For cout.
using namespace std;

void Static_Example();  // Function Prototype

int main()
{
      Static_Example();
      Static_Example();
      Static_Example();

      return      0;
}

void Static_Example()
{
      static int Count_Val = 0;     // Static, initial value = 0.
      cout << "My Count = " << Count_Val << endl;
      ++Count_Val;                  // Increment Count_Val.
}
```

Pass-by-Reference and Value Example

```
#include <iostream>     // For cout.
using namespace std;

// Function Prototypes
void WillNotAlterNumbers(int number1, int number2);
void AlterNumbers(int& number1, int& number2);

int main()
{
      int x = 0;
      int y = 0;
      // call function that uses
      // pass-by-value
      WillNotAlterNumbers(x,y);
      cout << x << '\t' << y << endl;

      // call function that uses
      // pass-by-reference
      AlterNumbers(x, y);
      cout << x << '\t' << y << endl;

      return      0;
}

// Pass-by-value
void WillNotAlterNumbers(int number1, int number2)
{     // value changes are not reflected
      // back in calling program
      number1++;
      number2--;
}
```

```
// Pass-by-reference
void AlterNumbers(int& number1, int& number2)
{     // value changes are reflected
      // back in calling program
      number1++;
      number2--;
}
```

Function with Comments

```
/* Include the relevant header files */
#include <iostream>           // for cin, cout
#include <string>       // for string

using namespace std;    // so don't have to write std:: all the time...


/*****************************************************************
**
Purpose -   This function prompts the user for an integer, gets their
                  input, and ensures that a valid integer was entered.
The
                  return value of the function contains the validated
integer.

Receives -  teamName : Standard class string containing user defined
input
                  typeChar : The single character used to print the
horizontal bar
                  barSize : The number of 'typeChar' to be printed to
screen

Remarks -   Function loops to print out bar

Example Use -     Horizontal_Bar('red', '*',20);

*****************************************************************
*/
void Horizontal_Bar(string teamName, char typeChar, int barSize);


/***** declare main *****/
int main()
{

      /* Declare varaibles for user input */
      int   userSize;
      char  userChar;
      string teamName;  // std class string


      /*******************************/

      cout << "Input team names, characters and sizes" << endl;
```

```
// tell user what inputs we expect


      /*********************************/

      cout << "Team1 Name:" << endl;
// prompt for user input - team name 1

      cin >> teamName;
// get team1 name

      cout << "Team1 Character:" << endl;
// prompt for user input - team 1 character

      cin >> userChar;
// get team1 character type for horizontal bar

      cout << "Team1 Score:" << endl;
// prompt user for input - team 1 bar size

      cin >> userSize;
// get team1 bar size

// call Horizontal_Bar function - pass variables containing user input
as parameters
      Horizontal_Bar(teamName, userChar, userSize);

      /*********************************/

      cout << "Team2 Name:" << endl;
// prompt for user input - team name 2

      cin >> teamName;
 // get team2 name

      cout << "Team2 Character:" << endl;
// prompt for user input - team 2 character

      cin >> userChar;
// get team2 character type for horizontal bar

      cout << "Team2 Score:" << endl;
// prompt user for input - team 2 bar size

      cin >> userSize;
// get team2 bar size

// call Horizontal_Bar function - pass variables containing user input
as parameters
      Horizontal_Bar(teamName, userChar, userSize);


      /*********************************/

      cout << "Team3 Name:" << endl;
// prompt for user input - team name 3
```

```
	cin >> teamName;
// get team3 name

	cout << "Team3 Character:" << endl;
// prompt for user input - team 3 character

	cin >> userChar;
// get team3 character type for horizontal bar

	cout << "Team3 Score:" << endl;
// prompt user for input - team 3 bar size

	cin >> userSize;
// get team3 bar size

// call Horizontal_Bar function - pass variables containing user input
as parameters
	Horizontal_Bar(teamName, userChar, userSize);


	return 0; // end program
}


// see header for full definition
void Horizontal_Bar(string teamName, char typeChar, int barSize)
{
	int   i;    // for counter

	cout << teamName << "\t";     // print out teamName

	cout << barSize << "\t";      // print out barSize


	/* loop to print out barSize numer of typeChar */
	for(i = 0; i < (barSize+1); i++)    // initialise loop
	{
		cout << typeChar; // print out typeChar for every loop

	}

	cout << "\n\n";   // create space to make screen look tidy

	/* function ends here and returns to main - execution continues
from point after function was called */
}
```