**ICS 103: Computer Programming in C**

**Lab #6: Functions without and with input arguments**

**Objectives:**

* Learn how to use C standard mathematical functions.
* Learn how to write user defined functions:
  1. Learn **void** functions without and with input arguments.
  2. Learn functions with input arguments and returning a single result.
* Learn the scope of variables.

In C, a function is a block of one or more statements that does a specific task. A pair of curly braces { and } define a block. Functions are used to organize a program into smaller building blocks and make it easy to understand and follow its behavior.

Why functions?

* Modularity (a program is divided into a number of blocks)
* Maintainability (Easy to find and correct errors)
* Code reusability (A function can be used in different programs)
* They facilitate information hiding (We can use a function without knowing how it is implemented).

1. Standard Mathematical Functions (some additional functions)

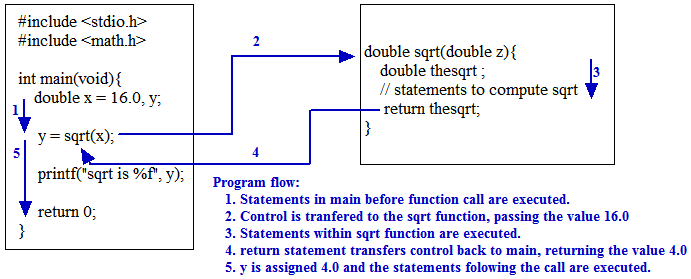
Here are some additional math functions not covered before. The prototypes of all mathematical functions below, except abs(x), are defined in **math.h**.

|  |  |  |  |
| --- | --- | --- | --- |
| C function | Mathematical Notation | Example | Comment |
| abs(x) | | x | | abs(-4) = 4 | Returns the absolute value of an integer expression. The prototype is defined in **stdlib.h** |
| ceil(x) | ⎡ x ⎤ | ceil(45.1) = 46.0  ceil(-7.9) = -7.0 | Returns the smallest integral value (of type double) greater or equal to x |
| floor(x) | ⎣ x ⎦ | floor(12.99) = 12.0  floor (-5.1) = -6.0 | Returns the largest integral value (of type double) less than or equal to x |

Note:

* Except for abs(x), the return type of each of the above functions is **double**.

**Note:** A function call transfers control to the called function, and when the statements in the function have been executed, control is transferred back to the calling function at the point of the call. Example:



1. User defined functions

* The general structure of a C function is:

**returnDataType functionName(listOfParameters){**

**declarations if any;**

**executable statements;**

**return expression;**

**}**

* Where **listOfParameters** is a list of variable declarations separated by comma; the list may be empty, in which case the function header is written as:

returnDataType **functionName( )**

**or**

returnDataType **functionName(void)**

* A function returning no value has return data type **void**.In such a function it is optional to use a **return** statement that does not return a value:

**return;**

This optional **return** may be omitted.

* The value or values passed to a function when it is invoked are expressions called **actual arguments**.
* If no value is passed to a function when it is invoked, the argument list is empty.
* The call to a **void** function is a **statement** of the form:

**functionName(listOfExpressions);**

Example: For the void function:

**void instruct(void){**

**printf("This program computes the area\n"**

**"and circumference of a circle.\n\n"**

**"To use this program, enter the radius of\n"**

**"the circle after the prompt: Enter radius: \n");**

**}**

The call is:

**instruct();**

* The call to a function that returns a single value is an **expression** that can appear anywhere an expression is allowed. To use the value returned by such a function later on in the program, that value is assigned to a variable that has the same data type as the function’s return data type:

**variable = functionName(listOfExpressions);**

Example: Consider the following function that returns the sum of squares of two values:

**double sumSquares(double x, double y){**

**return x \* x + y \* y;**

**}**

A sample call to that function is highlighted in blue below:

**double result, num = 5.0;**

**result = sumSquares( num, 3.0);**

**printf("The sum of squares is %f", result);**

* If a function does not physically appear before the function calling it, it must be declared before any call to the function. That declaration is called the **function prototype**.
* The syntax of a function prototype is:

**returnDataType functionName(listOfParameters);**

Where **listOfParameters** is a list of variable declarations separated by comma; the list may be empty, in which case the function prototype is written as:

returnDataType **functionName( );**

**or**

returnDataType **functionName(void);**

* A function prototype can be global (i.e., it can appear before and outside all functions) or it can be local (i.e., it can appear inside a calling function). If the prototype is global, the corresponding function can be invoked from all other functions. If the prototype is local, the corresponding function can only be invoked from the function containing the prototype.
* **Example of complete programs that use functions**:

**#include <stdio.h>**

**double sumSquares(double x, double y);**

**int main(void){**

**double w, z;**

**printf("Enter two numbers: ");**

**scanf("%lf%lf", &w, &z);**

**printf("\nw = %.2f , z = %.2f\n", w, z);**

**printf("\nThe sum of squares of 3.0\*w + 2.0 and z is %.2f\n",**

**sumSquares(3.0\*w + 2.0, z));**

**return 0;**

**}**

**double sumSquares(double x, double y){**

**return x \* x + y \* y;**

**}**

**Note:**

* There must be a one-to-one correspondence between actual arguments and formal parameters in

terms of number, order and upward-compatible type.

[Upward compatibility means a variable or a value of type that is on the left side of the following

chain:

**char 🡪 int 🡪 long 🡪 float 🡪 double**

can be assigned to a variable of any type on its right without data loss. An example of an

assignment that results in data loss is:

int x = 5.9999;

In this case the value assigned to x is **5**; the **0.9999** is lost]

* A variable actual argument may or may not have the same name as the corresponding formal

parameter (Note that even if an argument and its corresponding parameter have the same names,

they are different variables because their scopes are different).

* The scope of an identifier is the program area where the identifier can be used:
* The scope of a formal parameter is its function block.
* The scope of a local variable (i.e., a variable declared within a method, is from its point

of declaration to the end of the block.

* The expression in the return statement should have a type that is upward-compatible with the function's return type.

**Note:** An argument that is used to send a value to a function is called an **input** argument. All arguments that we have studied in this lab are input arguments. In the coming labs we will also study **output**, and **inputOutput** arguments.

**Note:** Before writing a C program for each of the problems in the Laboratory tasks below, solve the problem by writing the problem analysis, then a pseudo-code algorithm for the problem. Finally translate the pseudo-code algorithm into a complete C program. A sample problem and its analysis and pseudo-code algorithm is:

Write a C program that prompts for and reads the radius of a circle [in cm]. It then computes and displays the area and the circumference of the circle in square cm and cm respectively. Your program must use functions **getArea** and **getCircumference** to do the computations.

**Analysis:**

**Input:** radius [cm]

**Input range**: radius ≥ 0

Program constant: π

Relevant formulas: area = π radius2 , circumference = 2 \* π \* radius

**Output**: area [cm2] and circumference [cm]

**Pseudocode Algorithm:**

main( )

1. Prompt for radius [cm]

2. Input: radius

3. area = getArea(InputArgument: radius)

4. circumference = getCircumference(InputArgument: radius)

5. Output: “Radius = ”, radius, “cm, Area = ”, area, “sqr cm, Circumference = ”, circumference, “ cm ”

endmain

getArea(InputParameter: radius)

1. area = π \* radius2

2. return area

endgetArea

getCircumference(InputParameter: radius)

1. circum = 2 \* π \* radius

2. return circum

endgetCircumference

**C program:**

**#include <stdio.h>**

**#define PI 3.14159**

**double getArea(double radius);**

**double getCircumference(double radius);**

**int main(void){**

**double radius, area, circum;**

**printf("Enter radius >= 0 [cm]: ");**

**scanf("%lf", &radius);**

**area = getArea(radius);**

**circum = getCircumference(radius);**

**printf("\nRadius = %.2f cm, Area = %.2f sqr cm, Circumference = %.2f cm\n", radius, area, circum);**

**return 0;**

**}**

**double getArea(double radius){**

**double area;**

**area = PI \* radius \* radius;**

**return area;**

**}**

**double getCircumference(double radius){**

**double circum = 2 \* PI \* radius;**

**return circum;**

**}**

**Lab tasks**

**Task 01:**

If we know the lengths of two sides (*b* and *c*) of a triangle and the angle between them in degrees (α), we can compute the length of the third side ( *a* ) using the following formula:

*a*2 = *b*2 + *c*2 - 2*bc* cos α

Heron’s formula to calculate the area of a triangle with sides a, b, and c is:

***Please do the following steps:***

1. Write a function named **deg\_to\_radians** that receives the angle in degrees and returns the equivalent angle in radians. [180 degrees = π radians], π = 3.14159
2. Write a function named **length\_third\_side** that receives the length of two sides of a triangle and the angle between them (**in radians**), it then returns the length of the third side based on the formula shown above.
3. Write a function **calculateArea** that receives the lengths of the three sides of a triangle, it then returns the area of that triangle
4. In the main function, read the length of two sides and the angle between them (in degrees), then call the three functions mentioned above.
5. Display the triangle area in the main as shown below. The printing must be done in the **main** function and not in any of the other functions.

Sample program runs:

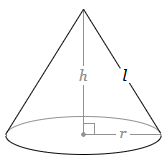
|  |  |
| --- | --- |
|  |  |

**Task 02:**

Write a complete C program that prompts for and reads the height [in cm] and the surface area of a cone [in cm2], it then calculates and displays the volume [in cm3] and the length of the slant side **l** [in cm].

Note: Your program must:

* Define a constant π with the value 3.141592
* Use three functions **getRadius**, **getVolume** and **getSlantSide** . There must be no input or output statements in any of these three functions



Note:

Sample program runs:

|  |
| --- |
|  |
|  |

**Task 03:**

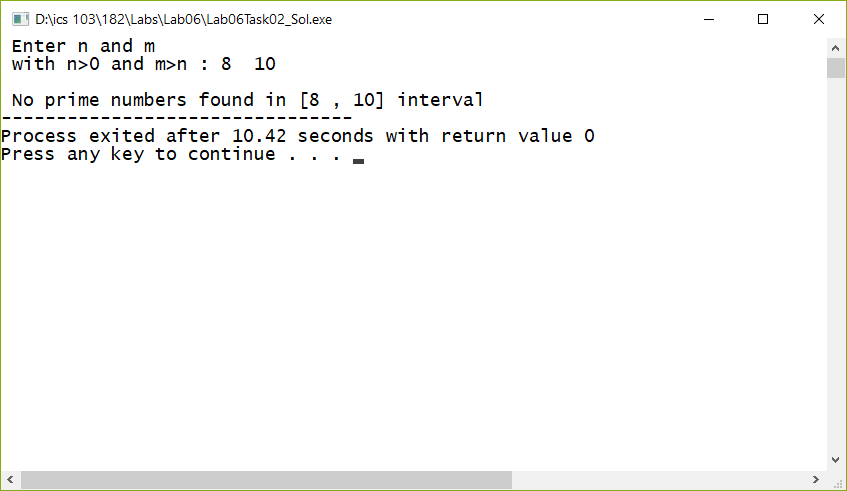
Write a C program to print all Prime numbers between n and m using loop.

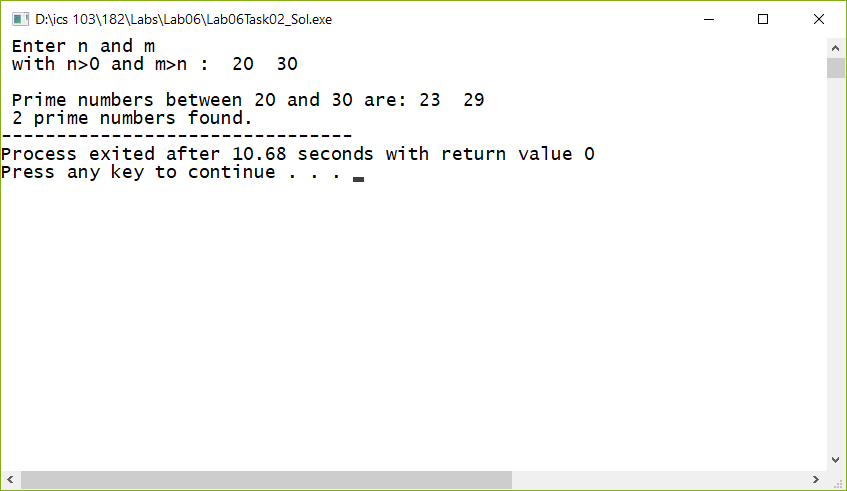
Your program contains a logical function **isPrime** which receives a positive integer number and returns **1** if it is prime; **0** otherwise.

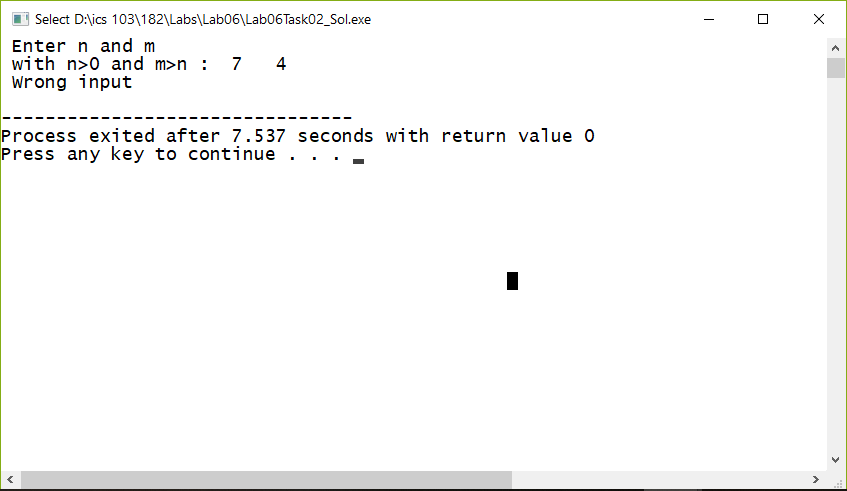
Your program should ask the user to enter the values of n and m and make sure both are positive and m is greater than n. If the input is wrong, your program displays an error message and stops. If the input is valid, it will find and display the prime numbers between n and m as shown below. Each value between m and n is checked if it’s prime by calling the function is prime.

Note that the program counts also how many prime numbers found. Also it prints “No prime numbers found inside the given interval” in case it does not find any prime number.

Note that a prime number is a **positive** integer **greater than 1** that is **only divisible by 1 and itself.** For example: 2, 3 , 5, 7, 11 are the first five prime numbers.







**Task 04 :**

Printing The rectangle of stars solved last week using functions. We will use 3 functions as follows:

Function **lineType1** : will receive n as parameter. It will print n \*s, then it will print a new line

Function **lineType2** : will receive n as parameter (n>=2). It will print one \*, then (n-2) ' ', then one \* followed by new line.

Function **rectangle**: will receive width and height as parameters (width>=2 and height>=2). It will use the 2 functions **lineType1** and **lineType2** functions to print the rectangle of **\***’s.

Below is the solution based on the previous lab. Your role is to rewrite this solution using the 3 functions discussed above.

# include <stdio.h>

int main(){ int width,height,i,j;

do {

printf("Enter number of \* in width and height (must be > 1): ");

scanf("%d%d",&width,&height);

if(width<=1 || height <=1)

printf("Wrong input both values must be > 1\n");

} while(width<=1 || height <=1);

printf("\n");

for(i=1;i<=width;i++)

printf("\*");

printf("\n");

for(i=1;i<=height-2;i++){

printf("\*");

for(j=1;j<=width-2;j++)

printf(" ");

printf("\*\n");

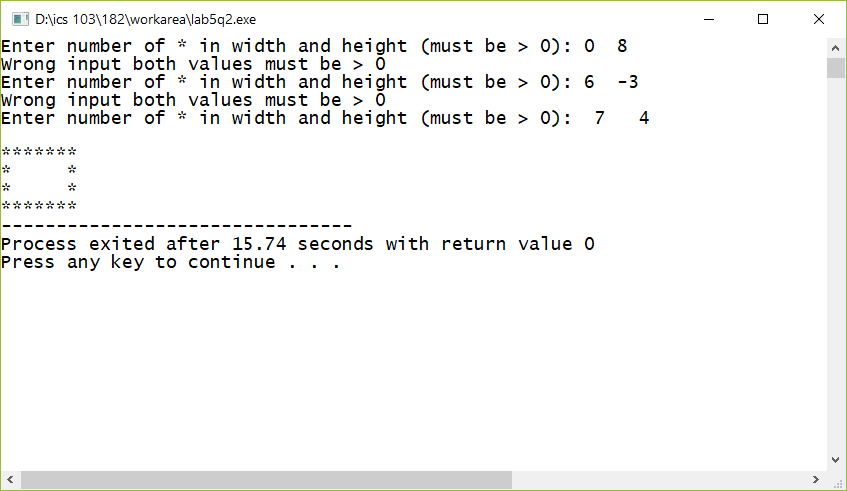
}

for(i=1;i<=width;i++)

printf("\*");

return 0;

}



**Task 05 :**

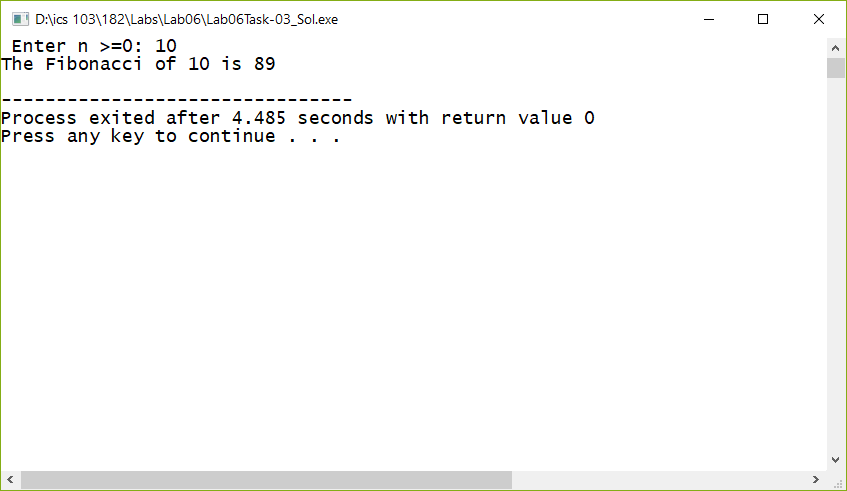
The Fibonacci sequence is, by definition, the integer sequence in which every number after the first two is the sum of the two preceding numbers. The first 12 numbers of the sequence are:

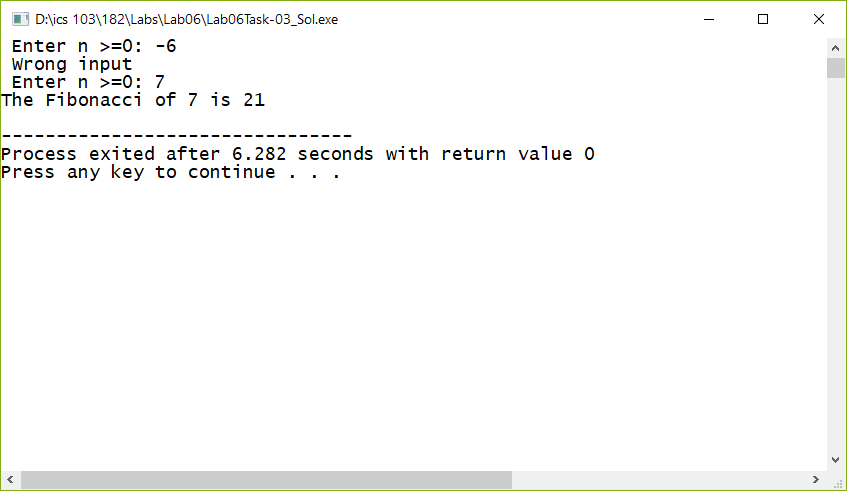
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| fib(n) | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 |

The mathematical formulation is given by:

Note that to find the fibonacci of n>=2, you need to use a loop.

Write a program that reads the value of n (n>=0) from the user. Then, it finds and displays the Fibonacci of n. Your program will contain the main and fib function.





**Task 06:** The probability **p** that in a group of **n** people, there is at least two people who have the same birthday is given by the formula:

**p =**

where **n** is the number of people.

Write an interactive C program that prompts for and reads the probability **p**, it then calls a function **getNumPeople** that receives **p** as an argument and it returns the value of **n**.

**Hint**: Use the larger root of the following quadratic equation:

**n2 – n – 2log10(1 – p)/ log10(364.0/365.0) = 0**

Note: The larger root **r** of the quadratic equation **ax2 + bx + c = 0** is given by:

***Please do the following steps:***

1. Write and call a **void** function that displays the following messages to the user:

“*Given the probability that two people in a group have the same birthday, this program computes the number of people in the group."*

*"Enter non-negative value less than 1.0 for the probability"*

1. Prompt for and read the value of p then pass this value to the function **getNumPeople**
2. In the main, print the value of **n** returned by the function **getNumPeople**. Display the result up to 4 decimal digits. ***(Note: printing must be in the main function and not in the getNumPeople function)***

Sample program runs:

|  |
| --- |
|  |
|  |