**ICS 103 – Computer Programming in C**

# Lab# 5 counting while and for loops

Objectives:

• To learn how to use the C **for-** and **while-statements** for writing counting loops.

• To learn about loop control variables and the three steps needed to control loop repetition.

* **while-statement**

The while loop has the format shown below. If the loop body has more than one statement, then they must be put between curly brackets { }.

|  |  |
| --- | --- |
| Simple statement while-body | Compound-statement while-body |
| while(condition)  statement; | while(condition)  compound-statement |
|  | |

Examples:

|  |  |
| --- | --- |
| while-loop | output |
| int n = 1; // initialization  while(n <= 10){ // test  printf("%d ", n);  n = n+2; // update  } | 1 3 5 7 9 |
| int k = 12; // initialization  while(k > 6) { // test  printf("%d ", k);  k=k-1; // update  } | 12 11 10 9 8 7 |

**Example1**: write a program that reads **n** floating pointnumbers and find their sum, product, and average.

**#include <stdio.h>**

**int main(void){**

**int n ,i;**

**double value, sum,product,average;**

**printf("Enter number of values to process: ");**

**scanf("%d", &n); // assume n > 0**

**sum = 0;**

**product = 1;**

**i = 1; // initialization**

**while(i <= n){ // test**

**printf("Enter value %d: ",i);**

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

**sum = sum + value;**

**product = product \* value;**

**i++; // update**

**}**

**average = sum / n;**

**printf("Sum = %.2f\n",sum);**

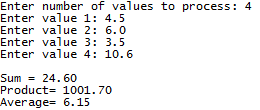
**printf("Product= %.2f\n",product);**

**printf("Average= %.2f\n",average);**

**return 0;**

**}**

Sample run of the above program:



In the above program ***i*** is the controlling variable. It is initialized to **1**. Since we need the loop to repeat **n** times to read **n** values the condition is **i<=n**. The variable ***i*** is incremented by **1** after each iteration.

Note that sum and product variables need to be initialized before the loop; 0 for sum and 1 for product. (Why)?

**Example2**: write a program that reads a positive integer **n** and finds the product of numbers from 1 to **n**. This product is called factorial of **n**.

**#include <stdio.h>**

**int main(void){**

**int n ,I,product;**

**printf("Enter a positive number ");**

**scanf("%d", &n); // assume n > 0**

**product = 1;**

**i = 1; // initialization**

**while(i <= n){ // test**

**product = product \* i;**

**i++; // update**

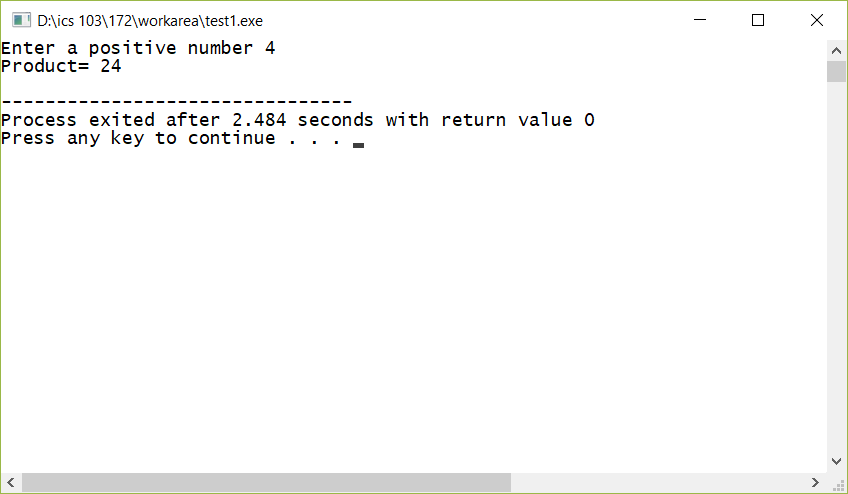
**}**

**rintf("Product= %d\n",product);**

**return 0;**

**}**

Sample run of the above program



24=4x3x2x1

* **for-statement**

In a for-loop the 3 steps of initialization, test (condition) and update are grouped in one place as shown below. Similar to while loop, if the for loop body has more than one statement, then they must be put between curly brackets { }.

**for(initialization; condition; update)**

**for-loop-body**

|  |  |
| --- | --- |
| Simple statement for-body | Compound-statement for-body |
| for(initialization; condition; update)  statement; | for(initialization; condition; update)  {  compound\_statement  } |
|  | |

Examples:

|  |  |
| --- | --- |
| for-loop | output |
| int k;  for(k = 7; k <= 12; k++)  printf("%d ", k); | 7 8 9 10 11 12 |
| int x ;  for(x = 8; x > 2; x=x-1)  if( x % 2 == 0)  printf("%d is even\n", x);  else  printf("%d is odd\n", x); | 8 is even  7 is odd  6 is even  5 is odd  4 is even  3 is odd |

* Equivalent loops

A loop can always be converted to an equivalent loop of a different type. Example:

|  |  |
| --- | --- |
| for loop | equivalent while loop |
| for(initialization;condition; update)  {  statement1;  statement2;  . . .  statementN;  } | initialization;  while(condition)  {  statement1;  statement2;  . . .  statementN;  update;  } |

**Example3:** The while-loop of the **example1** shown above concerning sum, product, and average can be converted to a for-loop.

Here we show only the part that needs modification:

**for(i = 1;i <= n; i = i+1){** // initialization, test, and update in one place

**printf("Enter value %d: ",i);**

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

**sum = sum + value;**

**product = product \* value;**

**}**

**Laboratory Tasks**

**Task 1:** Write a C program that prompts for and reads the number **n** of circles to be processed. If **n ≤ 0** your program must display an error message and terminate; otherwise it does the following for **n** times:

* Prompts for and reads the **volume** of a sphere, it then displays the **surface area** of the sphere with that volume. Assume that each volume is in cubic centimeters.

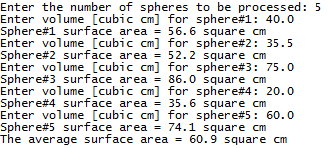
[, ]

The program finally displays the average of the surface areas.

Note:

* Assume that the value entered **n** and for each **volume** is valid.

Sample program run:



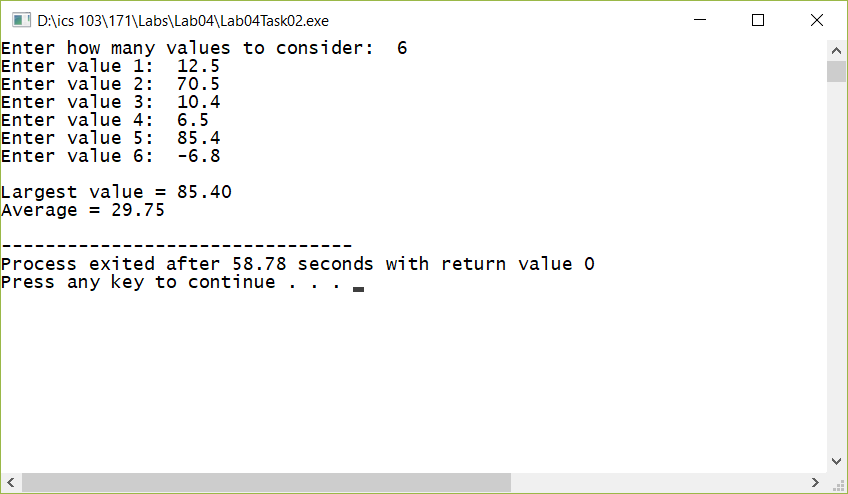
**Task 2:** Write a program that given a collection of Nnumbers will find the largest value, and the average of the **N** numbers.

1. Get the value of N from the user.
2. If N <= 0 display an appropriate error message and terminate the program; otherwise
3. Read the values as entered from the user. (If N =5, then there are 5 values the user is going to enter).
4. Find the Largest and the average of these N values.

**Hint:** To find the largest value, read the first value **before the loop** then assume that this first value entered by the user is the largest value. **Within the loop** compare the assumed largest value with other values **read inside the loop** and change the largest value accordingly.

**Note:** Your solution must not use concepts not covered in ICS 103 so far, like arrays and data files.

Sample program run:



**Task 3:**

Write a program to compute the sum of the following series with n terms.

Sum=2 + 22 + 222 + 2222 +…

The program reads the values n from the user. If n <= 0 or n> 15, it displays an appropriate error message and terminates. Otherwise, it computes and displays the sum.

Note: make your sum and your terms of type double but print the sum without digits after decimal point.

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

**Task 4:** The sum of the following series containing n terms is given by:

where **n** is a positive integer ranging from 1 to 50 and x is in the interval (-2, 2).

The program reads the values n from the user. If n <= 0, it displays an appropriate error message and terminates. Otherwise, it prompts the user to enter the value of x. If the value of x is not in the interval (-2, 2), it displays an appropriate error message and terminates. If the input is valid, it calculates the sum and display it.

Note: Do not use the pow function

Sample program runs:

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

**Additional Exercises:**

**Ex1:**

A positive integer **x** greater than one is a prime number if it has exactly two divisors, namely one and the number itself.[Alternatively, a positive integer x > 1 is a prime number if it has one divisor between 1 and inclusive]. Write a C program that prompts for and reads a positive integer **n** greater than one, it then determines whether the integer is a prime number or not. Your program must validate the input and keep on looping and displaying an appropriate error message as long as the input is not valid.

Prime numbers between 2 and 60 are: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59

Sample program runs:

|  |
| --- |
|  |
|  |

**Ex2:**

The value for for any x in the interval (0, 2]can be determined by the Taylor series:

Write a program that approximate the value of using **n** terms of the above series, where **n** is a positive integer obtained from the user. If n <= 0, display an appropriate error message and terminate the program. Otherwise prompt the user to enter the value of x. If the value of x is not in the interval (0, 2], display an appropriate error message and terminate the program. If the input is valid, it calculates the approximation to and displays it. Your program must also display the value of ln(x) by calling the standard mathematical function log(x).

Sample program runs:

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