Department of
Computer Science and Engineering

Title: Problem solving using for Loop

Computational Thinking and Problem Solving Lab

CSE 100

Green University of Bangladesh

1 Objectives

- To attain knowledge on for loop.
- To identify and solve problems using for loop.
- To implement programs using for loop concept.

2 Overview

In our previous lab class, we have solved several problems using while and do-while loops. In this lab class, another looping concept will be introduced, called **for** loop. Like other loops, it integrates 3 specific statements, namely **initialization, test condition, and increment/decrement operations**. A variable, on which we generally perform these three operations are called **control variable**. Let's demonstrate these three statements along with other general statements using a syntax:

```
1 for (initialization; test condition; increment/decrement)
2 {
3     statement(s);
4 }
```

Here,

initialization: Executed only once just before loop starts. Normally control variable is initialized here.

test condition: Any valid C condition is checked here. As long as this is true statement is repeatedly executed.

increment/decrement operation: Performed after all the statements are executed once. Typically contains incrementing counter or decrementing counter as the case maybe. Again, this operation is generally performed on control variable.

statement(s) (Body of the loop): This is repeatedly executed as long as condition is true. It may be a compound statement also.

Note: All these above statements are optional. Several statements of a loop can be omitted, but they may have consequences, include infinite execution of a loop. Hence, be careful whenever you are omitting a loop related statement. Even though, a statement is omitted, semicolon (;) after that statement must be given.

2.1 When to use for loop

For loop is typically used to repeat statements for a fixed number of times. In other words, when as a programmer we already know how many times the loop is going to iterate or repeat.

2.2 Basic examples

2.2.1 Example 1

The following is an example to display numbers from 1 to 100 using for loop.

```
1 for(int n = 1; n < 101; n++)
2     printf("%d\n",n);
```

This is in effect same as while loop used previously. But, as it combines initialization, test condition and increment/decrement operation together, many programs proffered this loop over others.

2.2.2 Example 2

The following is another example of for loop where the initialization and the test condition statements are omitted.

```
1 /*this is to be terminated when 0 or negative number is given*/
2 for( ; n>0; )
3     scanf("%d",&n);
```

Note: In C language, for and while loops can be used in place one another. Both of them execute statements as long as the condition is true and terminate the loop once condition is false. However, in some cases when number of iterations are not fixed, while is preferable.

2.2.3 Example 3

Another example of for loop where all three specific statements are omitted:

```
1 for(; ;)
2 {
3     statement(s);
4 }
```

This is an example of infinite loop. This can be terminated using a break statement (discussed later in this manual) or by calling an exit() function.

3 Problem solving using for loop

3.1 Problem 1: find number of even and odd numbers in a list

In our first problem, we'll analyze and then write a C program to find number of even and odd numbers in a list. As we know that an even number is a number that is divisible by 2 or a factor of 2, i.e., number %2 == 0. On the other hand, an odd number is a number that is not divisible by 2 or a factor of 2, i.e., number %2 == 0.

3.1.1 Algorithm

Algorithm 1: Algorithm for finding number of even and odd numbers in a list

```
1  Start
2  Declare necessary variables, including n, num, ecount, ocount
   /* Here, n ← how many numbers to check                    */
   /* num ← a number taken from the user                     */
   /* ecount ← even number counts                            */
   /* ocount ← odd number counts                             */
   /*                                                        */
3  Initialize variables like ecount = 0, ocount = 0
4  Read n from a user
5  for iterate i until i < n do
6  |  Read num from a user
7  |  if num % 2 is 0 then
8  |  |  print: num is an even number
9  |  |  Increase ecount by 1
10 |  end
11 |  else
12 |  |  print: num is an odd number
13 |  |  Increase ocount by 1
14 |  end
15 end
16 print: how many even numbers are encountered
17 print: how many odd numbers are encountered
18 End
```

3.1.2 Implementation

```c
#include<stdio.h>

int main()
{
    int n, num, ecount = 0, ocount = 0;
    printf("Enter number of values you would like to check: ");
    scanf("%d", &n);

    for(int i = 0; i < n; i++){
        printf("Enter a value: ");
        scanf("%d", &num);
        if(num % 2 == 0) {
            printf ("%d is an even number\n", num);
            ecount++;
        }
        else {
            printf ("%d is an odd number\n", num);
            ocount++;
        }
    }

    printf("Even count in the list of %d numbers is %d\n", n, ecount);
    printf("Odd count in the list of %d numbers is %d\n", n, ocount);

    return 0;
}
```

3.2 Problem 2: Factorial of a number

In mathematics, the factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n:

$$\text{n!} = \text{n} \times (\text{n-1}) \times (\text{n-2}) \times (\text{n-3}) \cdots \times 3 \times 2 \times 1$$

For example,

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Note: The value of 0! is 1, according to the convention for an empty product.

The factorial operation is encountered in many areas of mathematics, notably in combinatorics, algebra, and mathematical analysis. Its most basic use counts the possible distinct sequences – the permutations – of n distinct objects: there are n!.

3.2.1 Algorithm

Algorithm 2: Algorithm for finding factorial of a number

```
Start
Declare necessary variables like n, fact
Read the number n from a user
Initialize necessary variables, i = 1, fact = 1
for iterate until i == n do
    fact = fact * i
end
print: fact
End
```

3.2.2 Implementation

```c
#include<stdio.h>

int main() {
    int n;
    long int fact = 1;
    printf("Enter n value: ");
    scanf("%d", &n);

    for(int i = 1; i <= n; i++)
        fact = fact * i;

    printf("Factorial of %d is %ld", n, fact);
    return 0;
}
```

3.3 Problem 3: Find out the sum of series $1^2 + 2^2 + \cdots + n^2$

In mathematics, a series is, roughly speaking, a description of the operation of adding infinitely many quantities, one after the other, to a given starting quantity. The study of series is a major part of calculus and its generalization, mathematical analysis. Series are used in most areas of mathematics, even for studying finite structures (such as in combinatorics) through generating functions. In addition to their ubiquity in mathematics, infinite series are also widely used in other quantitative disciplines such as physics, computer science, statistics and finance. [Source: Wikipedia]

3.3.1 Notes

As we can observe from the series is that all the constant values are raised to power 2. It can be implemented in a C program in two ways, namely *i*) constants can be multiplied twice, i.e., 2×2 or 3×3 or so on and *ii*) use the function **pow()**, which is defined in **math.h** header. In our implementation, we are going to use the second one. If you want, you can use the first one as well. In **pow()**, we have to pass two parameters, namely value and power. For example, **pow (5, 2)** means 5 is raised to the power 2 by the function.

3.3.2 Algorithm

Algorithm 3: Algorithm for finding factorial of a number

1 Start
2 Declare necessary variables like n
3 Read the number n from a user
4 **for** *iterate until* $i <= n$ **do**
5 | sum = sum + pow (i, 2)
6 **end**
  /* Printing the series and show the output */
7 **for** *iterate until* $i <= n$ **do**
8 | print the series and the output
9 **end**
10 End

3.3.3 Implementation

```c
#include<stdio.h>
#include<math.h>

int main()
{
    int n, sum = 0;
```

```
    printf("Enter n value: ");
    scanf("%d", &n);

    for(int i = 1; i <= n; i++) {
        sum = sum + pow(i,2);
    }

    for(int i = 1; i <= n; i++) {
        if(i!= n) printf("%d^2 + ", i);
        else printf("%d^2 = %d", i, sum);
    }

    return 0;
}
```

4 Nested loops

In many cases, it has been observed that we have to place one or multiple loops inside another or other loop statement(s). It is called **nested loop**. Programming C allows nested loops as many times as it is necessary. The inner loop or the nested loop is executed for each iteration of the outer loop. Syntax of inner loop is given below:

```
for( ... ) { <--------//Outer loop
    for ( ... ) { <---//Inner loop
        ... <---------//We can have many layers of inner loops
                      //based on the demand of the program
    }
}
```

4.1 Example 1

Let us assume that we would like to display out numbers in the following format:

1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

In this case, we can use nested loop to perform the task easily.

*Note: we can use **for**/**while**/**do-while** as the inner loop and the outer loop. We can also mix various loops together. For instance, we can write a **while** loop as inner loop of outer **for** loop and so on.*

```
#include<stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        printf("\n");
        for (int j = 1 ; j <= 5; j ++)
            printf("%5d", j);
    }
    return 0;
}
```

4.2 Example 2

Again, let us assume that we would like to display out numbers in the following format:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

In this case, we can use nested loop to perform the task easily.

```c
#include<stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        printf("\n");
        for (int j = 1 ; j <= i; j ++)
            printf("%5d", j);
    }
    return 0;
}
```

4.3 Example 3

Furthermore, we would like to display the following format.

```
*
* *
* * *
* * * *
* * * * *
```

```c
#include<stdio.h>

int main(){
    int n = 5;

    for(int i = 0; i < n; i++) {
        for(int j = 0; j <= i; j++)
            printf(" *");
        printf("\n");
    }

    return 0;
}
```

5 Problem solving using nested loops

5.1 Problem: Display the sum of following series $1! + 2! + 3! + \cdots + n!$

Please see Section 3.3 for the definition of series and relevant discussions.

5.1.1 Algorithm

Algorithm 4: Algorithm for finding the summation of a given series

```
1  Start
2  Declare necessary variables like n, sum, and fact
3  Assign values to variables for iterate until i <= n do
4  |  Initialize fact to 1
5  |  for iterate until j <= i do
6  |  |  fact = fact * j
7  |  end
8  |  sum = sum + fact
9  end
10 print: summation of the given series
11 End
```

5.1.2 Implementation

```c
#include<stdio.h>

int main(){

    int n = 5;
    long int sum = 0, fact;

    for(int i = 1; i <= n; i++) {
        fact = 1;
        for(int j = 1; j <= i; j++)
            fact = fact * j;
        sum = sum + fact;
    }

    printf ("Summation of the given series is: %ld\n", sum);

    return 0;
}
```

6 Other statements

There are other statements like **break** and **continue** also used in loops to deal special conditions. In this section, we are going to discuss on these statements.

6.1 Break statement

This is used to terminate a loop when a certain condition is encountered. A loop can be terminated either when condition is false or when you execute break statement. When you have to terminate loop based on some other condition other than condition of the loop then you can use break statement.

6.1.1 Example

Let us assume that we are writing a C program to display sum of 10 numbers or till 0 is given whichever comes first. An implementation of this example is given below:

```c
#include<stdio.h>

int main() {
    int sum=0, n;
    for(int i = 1; i <= 10; i++){
```

```
        printf("Enter a number (0 to stop): ");
        scanf("%d", &n);
        if(n == 0)
            break;
        sum += n;
    }
    printf("Sum = %d",sum);
    return 0;
}
```

In this implementation, we can observe that even though the loop can iterate 10 times, however, when 0 is read from the user — loop termination occurred and summation is printed out.

6.2 Continue statement

This is used to transfer control to the beginning of the loop from within the loop. It is used to skip the statements after continue statement and enter into next iteration of the loop.

6.2.1 Example

Following C program demonstrates continue statement. In this program, a square of positive numbers only calculated. When negative numbers are provided, it skips the calculation.

```
#include <stdio.h>

int main() {
    int n, a, sq;
    printf("\nHow many numbers you want to enter: ");
    scanf("%d", &n);

    for (int i = 1; i <= n; i++) {
        printf("\nEnter a number: ");
        scanf("%d", &a);

        if(a<0)
            continue;

        sq = a * a;
        printf("\nSquare = %d\n", sq);
    }

    return 0;
}
```

7 Lab Task (Please implement yourself and show the output to the instructor)

1. Write a C program to find reverse of a given number following the algorithm mentioned below:
   Step 1: Begin
   Step 2: Display "Enter a number: "
   Step 3: Read n
   Step 4: Initialize rev to 0
   Step 5: Loop "n != 0", do
   Step 5.1: rem = n%10
   Step 5.2: rev = (rev*10) + rem
   Step 5.3: n = n/10
   Step 6: End Loop
   Step 7: Print "Reverse Number: ", rev

Step 8: End

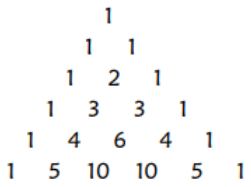
2. Display the numbers in the following format:
   5
   4 4
   3 3 3
   2 2 2 2
   1 1 1 1 1

8 Lab Exercise (Submit as a report)

1. Write a C program to find whether a given number is a prime number or not
   Step 1: Begin
   Step 2: Display "Enter a number: "
   Step 3: Read n
   Step 4: Initialize c to 0
   Step 5: For i = 1 to n, do
   Step 5.1: If "n%i==0"
   Step 5.1.1: Increment c by 1
   Step 5.2: EndIf;
   Step 5.3: Increment i by 1
   Step 6: EndFor;
   Step 7: If "c<=2"
   Step 7.1: Display n " is prime number"
   Step 8: Else
   Step 8.1: Display n " is not prime number"
   Step 9: EndIf;
   Step 10: End.

2. Print Fibonacci series until a given number. For instance, if a user wants to print Fibonacci series until 1000, print all the Fibonacci number below 1000.

3. Display Pascal's Triangle until a given row. For instance, if a user selects row = 6, the pascal triangle for the choice would be something like below:



9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.