ESc101: Multiple statement execution using loops

Instructor: Krithika Venkataramani
Semester 2, 2011-2012

The content of most of these slides are taken from the lecture slides of Prof. Arnab Bhattacharya

Loops are used when instructions are repeated

- Print all numbers between 1 and 100 that are divisible by 7
  - Algorithm
  1. Initialize x = 1
  2. Test if x is divisible by 7
  3. If yes, display x
  4. Increment x
  5. If x <= 100, go back to step 2
- Requires loops - instructions that are repeated a number of times
- Each time (called an iteration), some variable may change
- For a loop to stop, either of these must be specified
  - Number of times the loop runs
  - Stopping condition

while statement can be used for a loop

```
while ( condition )
{
    statements
}
```

- *condition* evaluates to a boolean
- The statements in the loop are executed as long as condition is true
- Any expression acts as condition
- Value of condition, if initially true, must change at some appropriate later point to false
- Otherwise, infinite loop is created

Example of a while-statement

- Print all numbers between 1 and 100 that are divisible by 7

```
x = 1;
while (x <= 100)
{
    if ((x % 7) == 0)
          printf ("%d ", x);
    x++;
}
```


Algorithm: Find sum of first N natural numbers

1. Input N
2. Sum = 0
3. If N < 1, go to step 8
4. I = 1
5. Sum = Sum + I
6. I = I + 1
7. If I <= N, go back to step 5
8. Print Sum

Sum of first N natural numbers using a while loop

```
printf("\nEnter number N');
scanf("%d", &N); /
x= 1; //initializing
sum = 0; //initializing
if (N<1)  //error check: to ensure N is a natural number
    printf("\nN is not a natural number: enter only a natural number");
else //if N is a Natural number
{
    while (x<=N)
    {
          sum = sum + x; //if number is less than or equal to N, add it to sum
          x++; // go to the next number
    }
    printf("\nSum = %d', sum);
}
```


Lab 1: Q2 algorithm using loop

- Take as input 4 numbers. Print arithmetic mean & harmonic mean. Print the maximum of the two means.
- Algorithm:

1. Initialize j = 1, arithmetic_mean = 0, inverse_harmonic_mean = 0
2. Input a number , n
3. Add number, n, to arithmetic_mean
4. If n is positive and harmonic mean is valid, add 1/n to inverse_harmonic_mean
5. Otherwise, harmonic mean is not valid
6. Increment j
7. If j <= 4, go back to Step 2
8. arithmetic_mean = arithmetic_mean/4
9. if valid, divide inverse_harmonic_mean by 4
10. if valid, harmonic mean = 1/inverse_harmonic_mean

Lab 1: Q2 sample solution using while loop

```
#include<stdio.h>
int main()
{
   float n, arithmetic_mean=0, inverse_harmonic_mean=0,
    harmonic_mean = 0;
   int flag = 1, j= 1;
   while (j<=4)
   {
         printf("Enter the %d number",j);
         scanf("%f",&n);
         arithmetic_mean = arithmetic_mean + n;
```


Lab 1: Q2 sample solution using while loop (cont.)

```
    if ((n>0)&&(flag==1))
       inverse_harmonic_mean = inverse_harmonic_mean+ 1/n;
    else
       flag = 0;//invalid harmonic mean
    j++;
}
arithmetic_mean = arithmetic_mean/4;
if (flag==1) //valid harmonic mean
   {
   inverse_harmonic_mean = inverse_harmonic_mean/4;
   harmonic_mean = 1/inverse_harmonic_mean;
   }
```


Lab 1: Q3 sample solution using a loop

- Take a 5 digit integer as input from the user. Count the total number of zeroes in it and print the result.
- Algorithm:

1. Input the integer
2. Initialize zero_count to 0
3. Find the remainder of integer by dividing using 10
4. If remainder is zero, then increment zero_count by 1
5. Divide the integer by 10
6. Use the quotient as the new integer
7. If integer !=0, go to Step 3
8. Display zero_count

Lab 1: Q3 sample solution using a loop

```
int n, count=0;
printf("Enter the FIVE DIGIT integer\n");
scanf("%d",&n);
while (n!=0)
{
 //checking if the last digit is zero
 if(n%10 == 0)
       count++;
 n=n/10; //integer with one less digit
}
//printing the results
printf("Number of zeros: %d\n",count);
```

Algorithm to display digits in reverse order

- Input an integer. Display the digits of the integer in reverse order.
- Algorithm

1. Input an integer
2. If negative integer, take the absolute value and display '-'
3. Find the remainder of division by 10
4. Display the remainder
5. Divide the integer by 10 and use the quotient as new integer
6. If integer is not equal to 0, go back to Step 3

while loop to display digits in reverse order

```
if (n<0)
{
    n = -n; // get absolute value
    printf("-"); //display – for negative integer
}
while (n!=0)
{
    remainder = n%10;
    printf("%d", remainder); //display the last digit
    n = n/10; //new integer has one less digit, without the last digit
}
```


Algorithm to find geometric mean

- Input n positive numbers and find their geometric mean
- geometric mean = $(x_1\, x_2\, x_3 \ldots x_n)^{1/n}$
- Algorithm:

1. Input n
2. Initialize j to 1 and geometric mean to 1.
3. Input number
4. Update geometric mean by multiplying it with the $(\text{number})^{1/n}$
5. Increment j
6. If j < = n, go back to Step 3
7. Display geometric mean

while loop to find Geometric Mean

```
j = 1;
geometric_mean = 1;
while (j<=n)
{
    printf("Enter number %d:",j);
    scanf("%f", &number);
    if (number > 0)
    {//updates and increments only for positive numbers
        geometric _mean = geometric_mean * pow(number, 1.0/n);
        j++;
    }
}
printf("Geometric mean = %f", geometric_mean);
```

Computation of 'power'

- Standard functions to compute power are present
- pow(x,y) function computes x^y
- Requires #include <math.h>
- compilation using gcc –lm
  - e.g. gcc –lm geomean.c

for statement

```
for ( initialization ; condition ; update )
{
    statements
}
```

- *condition* evaluates to a boolean
- The statements in the loop are executed as long as *condition* is true
- *initialization* initializes variables
- *update* updates the condition
- Value of *condition*, if initially true, must change at some appropriate point of time later to false
  - Otherwise, infinite loop is created

Example using for-statement

- Print all numbers between 1 and 100 that are divisible by 7

```
for (x = 1; x <= 100; x++)
{
    if ((x % 7) == 0)
        printf (“%d ”, x);
}
```


Sample program

- Find the sum of first N natural numbers using a for loop.

```
printf(“Enter the natural number N: ”);
scanf(“%d”, &N);
sum = 0;
for (i = 1, i<=N, i++)
{
    sum = sum + i;
}
printf(“\nSum of %d Natural numbers =  %d”,N, sum);
```

Equivalence of while and for statements

- while and for statements are equivalent

```
for ( initialization ; condition ; update )
{
statements ;
}
```

- translates to

```
initialization ;
while ( condition )
{
statements ;
update ;
}
```

- It is a matter of convenience and ease on the choice of loop

for loop to find geometric mean

```
geometric_mean = 1;
for (j = 1; j <=n; j++)
{
    printf("Enter number %d:",j);
    scanf("%f", &number);
    if (number > 0) //updates mean only for positive numbers
        geometric _mean = geometric_mean * pow(number, 1/n);
    else
        j--; //increments only for positive numbers
}
printf("Geometric mean = %f", geometric_mean);
```


for loop to display digits in reverse order

```
if (n<0)
{
    n = -n; // get absolute value
    printf("-"); //display – for negative integer
}
for ( ; n!=0; n=n/10) // there is no initialization
{
    remainder = n%10;
    printf("%d", remainder); //display the last digit
}
```


break brings execution out of loop

- break statement can be used within a loop
- break brings execution out of the loop
- break is generally used when loop is not to be executed under certain circumstances (condition)

```
while (condition1)
{
    statements1;
    if (condition2)
        break;
    statement2;
}
```

```
for (initialization; condition1; update)
{
    statements1;
    if (condition2)
        break;
    statement2;
}
```

break brings execution out of loop

- The loop is exited straightaway using a break statement
- Find the first number between 103 and 145 divisible by 23

```
for (x = 103; x <= 145; x++)
{
    if ((x % 23) == 0)
    {
        printf ("%d", x);
        break ;
    }
}
```

```
x = 103;
while ( x <= 145)
{
    if ((x % 23) == 0)
    {
        printf ("%d", x);
        break ;
    }
    x++;
}
```

- After the number is found, it does not make sense to continue
- break immediately exits the loop

Sample program using break

- Compute the sum of the geometric series for n terms until the sum does not change beyond a specified value
  - $a + a^r + ar^2 + ar^3 + \ldots$
  - r is between 0 and 1

```
sum = 0;
for (i = 1; i <= n; i++)
{
    term = a * pow(r, i - 1);
    if (term <= Spec_value)
        break;
    sum = sum + term;
}
```


continue statement

- continue can be used in loops
- continue allows the next iteration to be executed in the loop
- The statements after continue are not executed in the current iteration of the loop
- continue is generally used when certain statements (e.g. statements2 below) are not to be executed under certain circumstances (e.g. condition2 below)

```
while (condition1)
{
    statements1;
    if (condition2)
        continue;
    statements2;
}
```

```
for (initialization; condition1; update)
{
    statements1;
    if (condition2)
        continue;
    statements2;
}
```


Using continue for updating geometric mean

```
j = 1;
geometric_mean = 1;
while (j<=n) // loop to compute geometric mean of n numbers
{
    printf("Enter number %d:",j);
    scanf("%f", &number);
    if (number <= 0)
        continue; //For non-positive numbers, don't update mean
    geometric _mean = geometric_mean *pow(number,1/n);
    j++; //ensures increment only if number is positive
}
printf("Geometric mean = %f", geometric_mean);
```

Using continue for updating geometric mean

```
geometric_mean = 1;
for (j = 1; j <=n; j++) // loop to compute geometric mean
{
    printf("Enter number %d:",j);
    scanf("%f", &number);
    if (number <= 0)
    {
        j--; //increment count only for positive numbers
        continue; //For non-positive numbers, don't update mean
    }
    geometric _mean = geometric_mean * pow(number, 1/n);
}
printf("Geometric mean = %f", geometric_mean);
```


More on for loop

- for (; ;) – creates an infinite loop
- However, you can exit the loop using break
- Display numbers from 34 to 55 using for (; ;)

```
a = 34;
for ( ; ;)
{
    printf("%d ", a);
    a++;
    if  (a>55)
        break;
}
```


More on for loop

- for (a =0; a<=20;)
  - also creates an infinite loop if value of a does not change to >20 within the loop
  - update statements can be within the loop

```
for (a = 0; a <=20; )
{
    a++;
    printf("%d ", a);
    a = a+1; //multiple update statements
    printf("%d ", a);
}
```


Comma operator

- Lowest precedence of all operators
- comma operator links a list of related expressions together
- left to right evaluation
- Value of the combined expression = value of the right-most expression
  - sum = (x=10, y= 5, x+y);
  - sum = 15
- Can be used for exchanging values
  - t =x, x=y, y = t;
- Used in loops for multiple expressions
  - for (n = 1, m = 1; n<=m; n++, m++)
  - while (n!=0, m!=0) – equivalent to while (m!=0)

do-while loop

```
do
    statement;
while (expression);
```

- statement is executed first, and then expression in evaluated
- if expression is TRUE, the statement is executed again
- if expression is FALSE, the loop terminates
- In general, do-while loops are less frequently used

Krithika Venkataramani (krithika@cse.iitk.ac.in)

do-while loop example

- Print all numbers between 1 and 100 that are divisible by 7

```
x =1;
do
{
    if ((x % 7) == 0)
        printf ("%d ", x);
    x++;
}
while (x<=100)
```

Krithika Venkataramani (krithika@cse.iitk.ac.in)

Nested loops

- Nested loops: Using loops within loops
- Example: print multiplication table
- 1 2 3 … 10
- 2 4 6 … 20
- 3 6 9 … 30
- ..
- 12 24 … 120
- Formatting: printf("%4d", x) – displays an integer in 4 spaces
  - If x has more than 4 digits, all digits are displayed
  - If x has less than 4 digits, spaces are placed before the integer

Krithika Venkataramani (krithika@cse.iitk.ac.in)


```
int row=1, column, y; //program to print multiplication table
do
{
    column = 1;
    do
    {
        y = row *column;
        printf("%4d",y);
        column++;
    }
    while (column <=10)
    printf("\n"); //prints new line after one row of multiplication table
    row++;
}
while (row <=12)
```

Krithika Venkataramani (krithika@cse.iitk.ac.in)

Sample program

- Use nested loops to find the sum of the series below, until the sum does not change more than a specified value for |x|<1

$$\frac{x}{1!} - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \cdots \frac{x^n}{n!}$$

Series sum using nested loops

```
sum = 0;
for (i = 1; ; i++)
{
    factorial = 1;
    power = 1;
    for (j = 1; j <=I; j++)
    {
        factorial = factorial * (-j);
        power = power*x;
    }
    term = -1* power/factorial;
    if  (((term<0)&&(term >= - specified_value)) || ((term>0)&&(term <=
    specified_value)) )
        break;
    sum = sum + term;
}
```


Nested Loop Example

- Write a program to display the following output

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

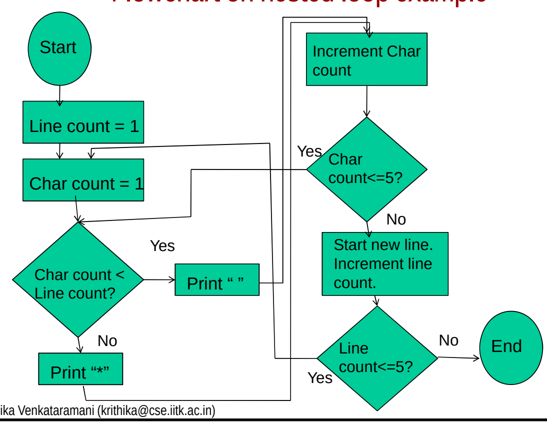
- Algorithm
- Flowchart
- Program

Algorithm for nested loop example

- 5 lines to be displayed. One less star in each line, starting with 5 stars. Star replaced by space in the beginning.

1. Initialize linecount =1.
2. Initialize charactercount=1.
3. If charactercount<linecount, print space.
4. Otherwise print star.
5. Increment charactercount.
6. If charactercount <=5, go back to Step 3.
7. Otherwise increment linecount and display in a new line. Go back to Step 2.

Flowchart on nested loop example



Code for nested loop

```
int j, k;
for(j=1;j<=5;j++) /*for displaying 5 lines*/
{
   for(k=1;k<=5;k++)/*for displaying 5 characters in each line*/
   {
        if(k<j)
                printf(" ");/*display blank space at beginning*/
        else
                printf("*");/*display star at the end*/
   }
   printf("\n"); /*new line after 5 characters*/
}
```