Introduction to Scientific and Engineering Computation

(BIL 102E)

LECTURE 7

LOOPS

One strong side of computers is that they can do *repetitive* jobs *very quickly*, and without getting *bored*.

Doing repetitive blocks of code are called *loops* in computer terminology.

In C, loops are implemented by means of **while, do .. while,** or **for** statements.

The **while** Loop

The purpose of the **while** loop is to repeatedly execute a statement over and over while a given condition is true.

The syntax is as follows:

while (expression)
  statement;

While the expression is nonzero the statement is executed.

For example,

```
i=1;
while (i<5) {
   printf("2 x %d = %2d\n", i, 2*i);
   i++;
}
```

```
2 x 1 =  2
2 x 2 =  4
2 x 3 =  6
2 x 4 =  8
```

Note that in a while loop, first the expression in while statement (the termination condition) is calculated. If the expression is zero than the statement(s) in the loop are not executed.


```
#include <stdio.h>
main()
{
  int c;
  c=' ';
  printf("Enter a character:\n(enter x to exit)\n");
  while (c!='x') {
        c = getchar();
        putchar(c);
  }
  printf("Out of the while loop. Bye!\n");
}
```

```
Enter a character:
(enter x to exit)
A
A
k
k
x
x
Out of the while loop. Bye!
```

The **do .. while** Loop

Sometimes the statement(s) inside the loop is required to be executed at least once before checking for the termination condition. In such cases, the **do .. while** loop can be used.

Syntax:

do
  statement;
while (expression);

The statement is executed first and then executed repetitively while the expression is nonzero.

```
#include <stdio.h>
main() {
  int c;
  printf("Enter a character:\n(enter x to exit)\n");
  do {
        c = getchar();
        putchar(c);
  } while (c!='x');
  printf("Out of the while loop. Bye!\n");
}
```

The same program with **do .. while**. Note that we do not need to initialize *c* here.

Doing things for a number of times:

Many loops require a certain task to be repeated for a number of times. There is a special statement called **for statement** for this purpose. The synatx is as follows:

```
for (expression1; expression2; expression3)
  statement;
```

=

```
expression1;
while(expression2) {
  statement;
  expression3;
}
```

Here, first *expression1* is executed. This generally involves an initialization of (giving first value to) a variable (usually referred as the *counter* or *loop variable*).

Then expression2, which is usually a conditional expression, is evaluated. While this expression evaluates to a nonzero value the for loop is repetitively executed.

At the end of each loop expression3, which is usually an increment or decrement statement is executed.

Examples

```
for(i=1;i<5;i++)
   printf("2 x %d = %2d\n", i, 2*i);
```

This piece of code produces the same result as that of the example given for the **while** loop.

```
#include <stdio.h>
main()
{
  int count;
  for(count=0;count<5;count+=2)
        printf("count=%d \n", count);
  printf("After for loop: count=%d \n", count);
}
```

```
count=0
count=2
count=4
After for loop: count=6
```


Analysis

int count;

count: ?

for (count=0; count<5; count+=2)

count=0 — count: 0

Is (count < 5) → *true* → continue

printf ... — count: 2
count+=2

Loop 1

Is (count < 5) → *true* → continue

printf ... — count: 4
count+=2

Loop 2

Is (count < 5) → true → continue

printf ... — count: 6
count+=2

Loop 3

Is (count < 5) → *false* → exit the **for** loop

}

Remarks

Though not recommended, C allows direct assignments to for variable (counter) inside the loop. For example,

```
for (i = 1;i<17;i+=2) {
  printf ("%d\n", i);
  if (i>5)
    i = i + 2;  /* This is possible */
}
```

```
1
3
5
7
11
15
```

Note that there is no semicolon after the for statement. If you accidentally put a semicolon it won't produce a syntax error since semicolon is a null (do nothing) statement in C. However, your program will not work as you wish it to work.

```
for (i = 1;i<17;i++);
```

→ This means count from 1 to 17 do nothing else.

More Examples

for statement	→ **counter** values
for (i=1;i<=3;i++)	→ *1,2,3*
for (j=1;j<12;j+=3)	→ *1,4,7,10*
for (k=78;k>40;k-=10)	→ *78,68,58,48*
for (p=-15;p<=15;p+=5)	→ *-15,-10,-5,0,5,10,15*
for (r=12;r>10;r-=3)	→ *12*
for (i=1;i>5,i++)	→ (for loop is skipped)
for (i=7;i<3;i--)	→ (for loop is skipped)

Write a program that asks the number of students in a class, asks the mark of each student and calculates the average mark.

ANALYSIS:

1. First the number of students has to be read from the keyboard. (*num_of_studs*)
2. Then, in a for loop with initial value *1* and final value *num_of_studs* the mark of each student (*mark*) should be read and the sum of marks, which is kept in a separate variable (*mark_sum*), should be updated. In order for this idea to work, *mark_sum* has to be initialised to 0.0 before the **for** loop.
3. Finally, the average is found as *mark_sum / num_studs* and printed out.


```c
#include <stdio.h>

main()
{
  int i, num_of_studs;
  float mark, mark_sum=0.0;

  /* Get the number of students */
  printf("How many students in the class?\n");
  scanf("%d",&num_of_studs);

  /* Read the mark of each student and update mark_sum */
  for(i = 1;i<=num_of_studs;i++) {
   printf("What is the mark of Student %d?\n",i);
   scanf("%f",&mark);
   mark_sum = mark_sum + mark;
  }

  /* Print out the result */
  printf("Average of marks : %4.1f\n",
                      mark_sum/num_of_studs);
}
```

Bonus Question 4

Write a program that asks the marks of students in a class until a negative number is entered. Then the average of positive numbers entered should be calculated and printed on screen.


```c
#include <stdio.h>
main()
{
  int i=0;
  float mark=0, mark_sum=0.0;

  /* Read the mark of each student and update mark_sum */
  while (mark>=0){
     mark_sum = mark_sum + mark;
     i++;
     printf("What is the mark of student %d?\n",i);
     scanf("%f",&mark);
  }

  /* Print out the result */
  printf("Average of marks : %4.1f\n",
                        mark_sum/(i-1));
}
```


Examples

Note that you can use if statements in loops, loops in if statements, and nest these in any order you want if necessary.

The following program finds the sum of positive odd numbers between 1 and n where n is a number entered by the user.

```c
#include <stdio.h>
main()
{
  int i,n,sum=0;
  printf("n=?\n");
  scanf("%d",&n);
  if (n>0)
    for(i=1;i<=n;i++)
       if((i%2) == 1)
           sum+=i;
  printf("sum = %d\n",sum);
}
```

Note that we could have used i+=2 here and not use the **if** statement inside the loop.

```
n=?
10
sum = 25
```


Write down a program that finds the factorial of a given number.

```c
#include <stdio.h>
main()
{
 long int i, n, fact=1;
 printf("Enter the number n=");
 scanf("%d",&n);
 for(i=1; i<=n; i++)
 {
   fact *= i;
 }
 printf("n! =  %ld \n",fact);
}
```

We have used **long int** to ensure that large integer numbers can be represented.

ld format specifier is used to format the number as a long decimal number.

```
Enter the number n=10
n! = 3628800
```

Write down a program that asks for an angle and finds the sinus of that angle approximately. Use the following equation

$$Sin(x) \cong \sum_{k=1}^{n} \frac{(-1)^{k+1} x^{2k-1}}{(2k-1)!}$$

Analysis:

- First the angle should be read into a variable (*angle*).
- Then the angle should be converted to radians (x).
- The sinus of the angle should be kept in a variable (sin_x).
- The sinus can be updated in a for loop. In each loop the *kth* term of the sum should be calculated and added to sin_x.
- In order to calculate the *kth* term 3 temporary variables can be used (sign, x_to_2k1, fact). At the beginning of each loop these should give $(-1)^{k+1}$, x^{2k-1}, and (2k-1)! respectively. At the end of each loop the temporary variables should be updated for the next loop.
- The variables sin_x, sign, x_to_2k1 and sin_x should have proper values before starting the for loop.


```
#include <stdio.h>
main()
{
 const int n=7;
 const double pi=3.14159;
 long int k, fact=1;
 double angle, x, x_to_2k1, sign=1, sin_x=0.0;
 printf("Enter an angle in degrees:");
 scanf("%lf",&angle);
 x_to_2k1=x=angle*pi/180.0; /* First convert to radians */
 for(k=1; k<=n; k++)
 {
        sin_x = sin_x + sign * x_to_2k1 / fact;
        /* Update temporary variables */
        sign= -sign;
        x_to_2k1 *= x*x;
        fact *= 2*k*2*(k+1);
 }
  printf("Sinus %lf is %4.2lf. \n",angle,sin_x);
}
```

const is used to define constants (These cannot be change in execution time).

double is used for high precision (instead of float).

$(-1)^{k+1}$ x^{2k-1} $(2k-1)!$

lf format specifier is used to format the number as a **double** precision number.

Nested Loops

It is possible to use loops inside other loops. For example,

```
for (i = 1; i<= 3; i++)
  for (j = i; j<=i+4; j+=2)
     printf ("(%d,%d)\n",i, j);
```

would print (1,1), (1,3), (1,5), (2,2), (2,4), (2,6), (3,3), (3,5), (3,7) on the screen on separate lines.

Write a program that calculates the following double sum:

$$\sum_{\substack{k=1 \\ (\text{odd } k)}}^{n} \sum_{i=1}^{k} xi$$

Here x is a real number and n is a positive integer to be entered by the user.


```
#include <stdio.h>
main(){
  float x, dsum=0.0;
  int i, k, n;

  printf("x=");
  scanf("%f",&x);
  printf("n=");
  scanf("%d",&n);
  for(k=1; k<=n; k+=2)
    for(i=1; i<=k; i++)
      dsum += x * i;

  printf("The result is %5.2f\n", dsum);
}
```

```
x=1.5
n=5
The result is 33
```

Multiple expressions in a **for** statement

It is possible to give more than one statements in expression1 and expression3 parts of a for statement. In such a case, the expressions should be separated by commas. For example,

```
for (i=0, j=10; i!=j; i++, j--)
   printf("(%d,%d)\n", i, j);
```

prints out (0,10), (1,9), (2,8), (3,7), (4,6) on the screen in separate lines.

Breaking Loops

Sometimes it is required to exit a loop. This usually happens when an error is detected, or a certain condition related to the nature of the loop is encountered.

break statement can be used for such purposes.

In execution time, when an **break** statement is encountered the execution continues from the line that follows the loop.

Jumping to the End of Loops

Sometimes it is required to jump to the end of a loop without exiting the loop.

It is possible to use the **continue** statement for such purposes.

In execution time, when a **continue** statement is encountered the execution continues from the brace that closes the loop.

This means *expresion3* in a for loop will be executed and the loop will continue to do its work.

Infinite Loops

Sometimes it is required to exit a loop when a certain condition is met and unless this condition is met the loop is required to continue infinitely. Such loops are called *infinite loops*. In C language, it is possible to use loops with the following syntax in order to implement infinite loops:

```
for (;;) {
[Block of C Statements]
}
```

```
while {
[Block of C Statements]
}
```

Obviously so as to avoid the loop go infinitely there has to be a **break** statement in the loop (most probably inside an **if** construct). It is programmers responsibility to make sure that such a **break** statement is reached in execution time.

Example

Write a program that asks the marks of students in a class and calculates the average mark. The process of entering the marks should be terminated when –1.0 is entered as a mark of a student. Furthermore, the program should make sure that all the marks entered are valid marks (i.e. they are between 0 and 100).


```
#include <stdio.h>
main() {
  int num_of_studs=0;
  float mark_sum=0.0, mark;
  /* Read the mark of each student and update mark_sum */
  for(;;) {
    printf("What is the mark of Student %d?\n",num_of_studs+1);
    scanf("%f",&mark);
    if (mark == -1.0)
      break;
    else if((mark <0.0) || (mark >100.0))
    {
      printf("Invalid mark!\n");
      continue;
    }
    num_of_studs++;
    mark_sum += mark;
  }
  /* Print out the result */
  printf("Number of students :%d\n", num_of_studs);
  printf("Average of marks : %4.1f\n", mark_sum / num_of_studs);
}
```

Infinite loop!

This means continue from **here!**

This means continue from **here!**

A sample output of the program

```
What is the mark of Student 1?
10
What is the mark of Student 2?
30
What is the mark of Student 3?
-34
Invalid mark!
What is the mark of Student 3?
40
What is the mark of Student 4?
-1
Number of students :3
Average of marks : 30.0
```


The **goto** statement

In order for program to continue execution from a certain point in the program the **goto** statement can be used.

The syntax is:

labelname:
  statement1;
  statement2;
  ...
goto labelname;

This kind of programming is very poor and is not recommended. So avoid using **goto** statements in your programs.