**CHAPTER – 4**

**REPETITION**

**Introduction:**

When the exact number of repetitions are known, then there are more convenient methods of looping in C. These looping capabilities enable us to develop concise programs containing repetitive processes without the use of goto statements.

In looping, a sequence of statements are executed until some conditions fro termination of the loop are Satisfied. A program loop therefore consists of two segments, one known as the body of the loop and the other known as the control statement. The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.

Depending on the position of the control statement in the loop, a control structure may be classified either as the entry-controlled loop or as the exit-controlled loop. The flowchart is as shown below:

Entry Entry

Test cond ?

Body of the Loop

Body of the Loop

Test cond?

False

True False

True

(a) Entry controlled loop (b) Exit controlled loop

In the entry-controlled loop or pretest control loop, the control conditions are tested before the start of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed. In the case of an exit-controlled loop or post-test control loop, the test i performed at the end of the body of the loop and therefore the body is executed unconditionally for the first time. The entry-controlled and exit-controlled loops are also known as pre-test and post-test loops respectively.

A looping process, in general, would include the following four steps:

1. Setting and initialization of a condition variable.

2. Execution of the statements in the loop.

3. Test for a specified value of the condition variable for execution of the loop.

4. Incrementing or updating the condition variable.

The test may be either to determine whether the loop has been repeated the specified number of times or to determine whether a particular condition has been met.

The C language provides for three constructs for performing loop operations. They are :

1. The while statement.

2. The do statement.

3. The for statement.

**The while statement:**

The simplest of all the looping structures in C is the while statement. The basic format of the while statement is while (test condition)

{

body of the loop

}

The while is an entry-controlled loop statement. The test-condition is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test-condition is once again evaluated and if it is true, the body is executed once again. This process of repeated execution of the body continues until the test-condition finally becomes false and the control id transferred out of the loop. On exit, the program continues with the statement immediately after the body of the loop.

The body of the loop any have one or more statements. The braces are needed only if the body contains two or more statements.

Example: # include <stdio.h>

# include <conio.h>

main ()

{

int n,r,d=1;

long int s=0;

clrscr ();

printf ("Enter any decimal number");

scanf ("%d",&n);

while(n>0)

{

r=n%2;

s=(s+r)\*d;

n=n/2;

d=d\*10;

}

printf ("The binary no is %ld",s);

getch ();

}

Output: Enter any decimal number 16

The binary no is 10000

**The do-while statement:**

Do statement is used to execute the body of the loop before the test is performed.

Syn: do

{

Body of the loop

}

While (test-condition);

On reaching the do statement, the program proceeds to evaluate the body of the loop first. At the end of the loop, the test-conditions in the while statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again. This process continues as long as the condition is true. When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement.

Since the test-condition is evaluated at the bottom of the loop, the do...while construct provides an exit-controlled loop and therefore the body of the loop is always executed at least once.

Example: # include <stdio.h>

# include <conio.h>

void main ()

{

int a=0,b=1,c,n;

clrscr ();

printf ("Enter the range of the series");

scanf ("%d",&n);

printf ("The fibanocci series are \n");

printf ("%d %d",a,b);

c=a+b;

do

{

printf (" %d",c);

a=b;b=c;c=a+b;

}while (c<=n);

getch ();

}

Output: Enter the range of the series 11

The Fibonacci series are 0 1 1 2 3 5 8

**The For Statement:**

The for loop is another entry-controlled loop that provides a more concise loop control structure. The general form of the for loop is

for ( initialization ; test-condition ; increment )

{

body of the loop

}

The execution of the for statement is as follows:

1. Initialization of the control variables is done first, using assignment statements such as i=1 and

count=0. The variables i and count are known as loop-control variables.

2. The value of the control variable is tested using the test-condition. The test-condition is a

relational expression, such as i<10 that determines when the loop will exit. If the condition is

true, the body of the loop is executed; otherwise the loop is terminated and the execution

continues with the statement that immediately follows the loop.

3. When the body of the loop is executed, the control is transferred back to the for statement after

evaluating the last statement in the loop. Now, the control variable is incremented using an

assignment statement such as i=i+1 and the new value of the control variable is again tested to

see whether if satisfies the loop condition. If the condition is satisfied, the body of the loop is

again executed. This process continues till the value of the control variable fails to satisfy the test-

condition.

In for loop, all the three actions, namely initialization, testing, and incrementing, are placed in the for statement itself, thus making them visible to the programmers and users, in one place.

The for statement and its equivalent of while and do statements are shown below:

**For while do**

For (n=1;n<=10;++n) n=1; n=1;

{ while (n<=10) do

..................... { {

..................... .................. .................

} .................. ..................

n=n+1; n=n+1;

} }

While (n<=10);

**Additional features of for loop:**

The for loop in C has several capabilities that are not found in other loop constructs.

1. More than one variable can be initialized at a time in the for statement. The statements

P=1;

For (n=0;n<17;++n)

Can be rewritten as for (p=1,n=0;n<17;++n)

Here the initialization section has two parts p=1 and n=1 separated by a comma.

2. Like the initialization section, increment section may also have more than one part.

Ex: for (n=1,m=50;n<=m;n=n+1,m=m+1)

{

p=m/n;

Printf (“%d %d %d \n”, n, m, p);

}

3. The test condition may have any compound relation and the testing need not be limited

only to the loop control variable.

Ex: sum=0;

For (i=1;i<20 && sum<100;++i)

{

Sum=sum+i;

Printf (“%d %d”, i, sum);

}

4. We can use expressions in the assignment statements of initialization and increment

sections.

Ex: for (x=(m+n)/2;x>0;x=x/2)

5. Another unique aspect of for loop is that one or more sections can be omitted, if

necessary.

Ex: m = 5;

for (; m!=100 ;)

{

printf (“%d”, m);

m=m+5;

}

Both the initialization and increment sections are omitted in the for statement. The initialization has been done before the for statement and the control variable is incremented inside the loop. In such cases, the sections are left ‘blank’ and also the semicolons separating the sections must remain. If the test-condition is not present, the for statement sets up an ‘infinite’ loop. Such loops can be broken using break or goto statements in the loop.

**Nesting of for Loops:**

Nesting of loops, that is , one for statement within another for statement, is allowed in C. For example, two loops can be nested as follows:

Ex: ...............

..................

for (i=1;i<10;++i)

{

..........................

..........................

for (j=1;j!=5;++j) Outer loop

{

.................... Inner loop

....................

}

..........................

..........................

}

................................

................................

**Jumps in Loops:**

Sometimes, when executing a loop it becomes desirable to skip a part of the loop or leave the loop as soon as a certain condition occurs. For ex, consider the case of searching for a particular name in a list containing 100 names. A program loop written for reading and testing the names 100 times must be terminated as soon as the desired name is found. C permits a jump from one statement to another within a loop as well as a jump out of a loop.

**Jumping Out of a Loop:**

An early exit from a loop can be accomplished by using the break statement or the goto statement. When a break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the break would only exit from the loop containing it. That is, the break will exit only a single loop.

Ex: while (.......) for (.......................)

{ {

................ ......................

................ for (..................)

if (condition) {

break; if (condition)

Exit ...................... exit break;

from ...................... from ......................

loop } inner }

........................... loop ......................

}

**Skipping a Part of a Loop:**

During the loop operations, it may be necessary to skip a part of the body of the loop under certain conditions. Like the break statement, C supports another similar statement called continue statement. The continue , as the name implies, causes the loop to be continued with the next iteration after skipping any statements in between. The continue statement tells the compiler, “Skip The Following Statements And Continue With The Next Iteration”.

Syn: continue;

The use of continue statement in the loops is as shown below:

Ex: while (.......) do

{ {

................ ......................

if (condition) if (..................)

continue; continue;

...................... .....................

...................... ......................

} } while (test-condition);

In while and do while loops, continue causes the control to go directly to the test-condition and then to continue the iteration process. In the case of for loop, the increment section of the loop is executed before the test-condition is evaluated.

**Jumping Out of the Program:**

We can jump out of the loop by using the break statement or goto statement. In a similarly way, we can jump out of a program by using the library function exit().

We can use the exit function as shown below:

..........................

..........................

if (test-condition) exit(0);

..........................

..........................

The exit() function takes an integer value as an argument. Normally zero is used to indicate normal termination and a nonzero value to indicate termination due to some error or abnormal condition. The use of exit function requires the inclusion of the header file <stdlib.h>

**Concise Test Expressions:**

For making the branching decisions, we used test expressions in the if, for, while and do statements that are evaluated and compared with zero. For instance, the expression is true whenever x is not zero, and false when x is zero. Applying ! operator , we can write concise test expressions without using any relational operators.

if (expression==0)

is equivalent to if (!expression)

similarly, if (expression!=0)

is equivalent to if (expression)

Ex: if (m%5==0 && n%5==0) is same as if(!(m%5)&&!(n%5))