**Looping Statements**

**6.1 Define Looping:** In Looping, a sequence of statements are executed until some conditions for termination of the loop are satisfied. A *program loop* consists of two segments. They are:

**1. Body of the loop.**

**2. 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 into two types. They are:

**1. Entry-controlled loop.**

**2. Exit-controlled loop.**

In the **entry-controlled 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**, the test is performed at the end of the body of the loop and 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, 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.

**6.2 List at least three loop statements:**

The C language provides three loop operations. They are:

1. The **while** statement.

2. The **do** statement.

3. The **for** statement.

**6.3 Explain while statement with syntax and sample program:** 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 is transferred out of the loop.
* The body of the loop may have one or more statements.
* The braces are needed only if the body contains two or more statements.

**Example program: Reverse of a given number**

#include<stdio.h>

main()

{

int rev=0,rem,n;

clrscr();

printf("Enter any number\n");

scanf("%d",&n);

while(n!=0)

{

rem=n%10;

rev=rev\*10+rem;

n=n/10;

}

printf("Reverse number=%d",rev);

getch();

}

**6.4 Explain do.....while statement with syntax and sample program:**

The basic format for do......while is

**do**

**{**

**Body of the loop**

**}while(test-condition);**

* **do** statement is an exit-controlled loop statement.
* On reaching the **do** statement, the program proceeds to evaluate the body of the loop first.
* At the end of the loop, the *test-condition* in the **while** statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again.
* When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the **while** statement
* The body of the loop is always executed at least once.

**Example Program:** write a c program to print ‘n’ natural numbers.

#include<stdio.h>

main()

{

int i=1,n

clrscr();

printf("Enter any number\n");

scanf(“%d”,&n);

do

{

printf(“%d”,i);

i++;

}while(i<=n);

**6.5 Explain ‘for’ loop statement with syntax and sample program:** The **for** loop is an *entry-controlled* loop. The general form of **for** loop is:

**for (initialization; test-condition; increment/decrement)**

**{**

**Body of the loop;**

**}**

The execution of the **for** statement is as follows:

1. *Initialization* of the *control variables* is done first, using assignment statement.

2. The value of control variable is tested using the test-condition. The test-condition is a relational expression. 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,* the new value of the control variable is again tested. If the condition is satisfied, the body of the loop is again executed; otherwise it terminated from the loop.

**Additional features of for loop:**

•More than one variable can be initialized at a time in the **for** statement.

Ex: p=1;

for (n=0; n<17; n++) can be rewritten as

for (p=1,n=0; n<17; n++)

•The increment section may also have more than one part.

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

•The test-condition may have any compound relation.

Ex: sum=0;

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

{

Sum=sum+i;

}

* In **for** loop, one or more sections can be omitted.

Ex: m=5;

for(; m!=100;)

{

printf (“%d”,m);

m=m+5;

}

• We can use expressions in the assignment statement of initialization and increment sections.

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

• We can set time delay by using null statement.

Ex: for(j=1000; j>0; j--)

;

**Write a c program for factorial of a given number**.

 main()

 {

int fact=1,i,n;

printf(“enter a number”);

scanf(“%d”,&n);

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

 {

 fact=fact\*i;

 }

Printf(“factorial of a given number=%d”,fact);

 }

**6.6 Define nesting of loops statements:**

A loop statement defined with in another loop statement is called nested loop.

* C language supports loop with in a loop concept.
* The inner loop is executed prior to the outer loop.
* The different nested looping statements are
1. Nested while loop.
2. Nested do-while loop.
3. Nested for loop.

**1. Nested while loop:** In this looping statement, a while statement is defined inside a while statement.

**2. Nested do-while loop:** In this looping statement, a do-while statement is defined inside the do-while statement.

**3. Nested for loop:** In this looping statement, a for statement is defined inside another for statement.

|  |  |  |
| --- | --- | --- |
| **Syntax:** While(condition){ While(condition) { Statements; } Statements;} | **Syntax:**do{ Statements; do { Statements; } While(condition);} While(condition); | **Syntax:**for(initialization;condition;increment){ for(initialization;condition;increment) { Statements; } Statements;} |

**Example programs:**

|  |  |
| --- | --- |
| /\* to print output in particular format \*/main(){ int n,i,t; clrscr(); printf(“enter a number”); scanf(“%d”,&n); for(i=1;i<n;i++) { printf(“\n”); for(t=1;t<=i;t++) {  printf(“%d ”,t); } } getch(); }**Output:** Enter a number: 5 11. 2

1 2 31 2 3 41 2 3 4 5 | /\* to print mathematical tables \*/#include<stdio.h>#define colmax 10#define rowmax 10main(){ int row,column,y; clrscr(); row=1; printf(“multiplication table \n”); printf(“--------------- \n”); do { column=1; do { y=row\*column; printf(“%4d”,y); column=column+1; }while(column<=colmax); printf(“\n”); row=row+1; }while(row<=rowmax); printf(“------------“); } |

**6.7 Compare different loop statements:**

|  |  |  |
| --- | --- | --- |
| **While** | **do..while** | **For** |
| 1.while is an **entry-controlled** loop statement | 1. do..while is an **exit-controlled** loop statement | 1. for is an **entry-controlled** loop statement |
| 2.While loop is a **pre-test** loop. | 2. do while loop is a **post-test** loop. | 2. for loop is a **pre-test** loop. |
| 3**.**syntax**:** while (condition) { //body of the loop } | 3.syntax:do{ //body of the loop}while(condition); | 3.syntax:for (initialization; test-condition;  increment)  { //body of the loop } |
| 4. Condition is tested at the beginning of the loop. | 4. Condition is tested at the endingof the loop. | 4. Condition is tested at the beginning of the loop. |
| 5.It executes only when the condition is true | 5.It executes at least once even the condition is false | 5. It executes only when the condition is true. |
| 6.minimum iterations is 0 | 6.minimum iterations is 1 | 6.minimum iterations is 0 |
| 7.Control variable will be initialized outside the loop | 7.Control variable will be initialized outside the loop | 7. Control variables can be initialized within the loop. |
| 8. Not so flexible . | 8. Not so flexible  | 8. Very flexible structure. |
| 9. e.g.: i=1;while(i<10){ printf(“%d”,i); i++;} | 9.e.g.: i=1;do{ printf(“%d”,i); i++;} while(i<10); | 9.e.g.: for(i=1;i<10;i++){ printf(“%d”,i);} |

**6.8 Differentiate Break and Continuestatements:**

**Break-statement**: (jumping out of the loop)

 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.



**Continue (skipping part of a loop):**

Continue statement causes the loop to be continued with the next iteration after skipping after the continue statement in the body of the loop; “**SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION**”.

The format of the continue statement **continue;**



**Differentiate Break and Continue Statements:**

 Break and continue statements are also known as jump statements.

|  |  |
| --- | --- |
| **Break statement** | **Continue statement** |
| 1.break is a keyword used to terminate the loop (or) exit from the loop | 1. continue is a keyword used for continuing the next iteration of the loop |
| 2.control jumps to the next statement after the loop. | 2. control jumps to the iteration part of the loop. |
| 3.Break can be used in for, while do..while loops and switch statement. | 3.continue can be used in for , while, do..while loops. |
| 4.the general form is **break;** | 4.the general form is **continue;** |
| 5.when break is used in nested loop then only the innermost loop is terminated. | 5.when continue is used in nested loop causes the next loop iteration. |
| 6..more useful than continue statement.7. eg: for(i=1;i<=10;i++) { if(i%5==0) break; else printf(“%d ”,i); }output: 1 2 3 4  | 6.less useful than break statement.7. eg: for(i=1;i<=10;i++) { if(i%5==0) continue; else printf(“%d ”,i); }output: 1 2 3 4 6 7 8 9 |

**6.9 Structured Programming:**

Structured programming is **an approach to the design and development of programs**. It is a discipline of making a program‘s logic easy to understand by using only the basic 3-control structures:

**1. Sequence (straight line) structure**

**2. Selection (branching) structure**

3. **Repetition(looping) structure**

While sequence and loop structures are sufficient to meet all the requirement of programming, the selection structure proves to be more convenient in some situations.

The use of structured programming techniques **helps ensure well –designed programs that are easier to write, read, debug and maintain compared to those that are unstructured**.

Structured programming discourages the implementation of unconditional branching using jump statements such as **goto, break, continue**. In its purest form, structured programming is synonymous with “**goto less programming**”.