UNIT- II

Managing I/O: Input-Output statements, Formatted I/O.
Decision making statements: if, if-else, if-else-if, nested-if, switch
Iterative statements: while, do-while, for
Unconditional statements: break, continue, goto.

1. Explain the formatted I/O functions with example?

Explain unformatted (character oriented) I/O functions with example?

Ans: Managing input and output operations:

Reading, processing and writing of data are the three essential functions of a computer program. Most programs take some data as input and display the processed data. We have two methods of providing data to the program variables. One method is to assign values to variables through the assignment statements like x=5, a=0 and so on. Another method is to use the input function scanf, which can read data from a keyboard. We have used both the methods in programs. For outputting results, we have used extensively the function printf, which sends results out to a terminal.

Input – Output functions:-

- The program takes some I/P- data, process it and gives the O/P.
- We have two methods for providing data to the program

  i) Assigning the data to the variables in a program.
  ii) By using I/P-O/P statements.

C language has 2 types of I/O statements; all these operations are carried out through function calls.

1. Unformatted I/O statements
2. Formatted I/O statements

Unformatted I/O statements:-

getchar( ):- It reads single character from standard input device. This function don‘t require any arguments.

Syntax:- char variable_name = getchar( );

Ex:- char x;

x = getchar( );

putchar ( ):- This function is used to display one character at a time on the standard output device.

Syntax:- putchar(char_variable);

Ex:- char x;

putchar(x);

program:

```
main( )
{
char ch;
printf(—enter a char‖);
ch = getchar( );
printf(—entered char
is‖); putchar(ch);
}
```

Output: enter a char a

entered char is a

getc() :- This function is used to accept single character from the file.

Syntax: char variable_name = getc();

Ex:- char c;

c = getc();

putc():- This function is used to display single character.

Syntax:- putc(char variable_name);

Ex:- char c;

putc(c);

These functions are used in file processing.

gets( ):- This function is used to read group of characters(string) from the standard I/P device.

Syntax:- gets(character array variable);

Ex:- gets(s);

puts( ):- This function is used to display string to the standard O/P device.

Syntax:- puts(character array variables);

Ex:- puts(s);

program:

```
main()

{
char s[10];

puts(―enter name‖);

gets(s);

puts(―print name‖);

puts(s);

}
```

Output: enter name ramu

print name ramu

getch():- This function reads a single character directly from the keyboard without displaying on the screen. This function is used at the end of the program for displaying the output (without pressing (Alt-F5).

Syntax: char variable_name = getch();

Ex:- char c

c = getch();

getche():- This function reads a single character from the keyboard and echoes(displays) it to the current text window.

Syntax:- char variable_name = getche();

Ex:- char c

c = getche();

program:

```
main()
{
char ch, c;
printf(—enter
char‖); ch = getch();
printf(—%c‖, ch);
printf(—enter
char‖); c = getche();
printf(—%c‖,c);
}
```

Output :- enter character a

enter character b

b

Character test functions:-

Function	Test
isalnum(c)	Is c an alphanumeric character?
isalpha(c)	Is c an alphabetic character
isdigit(c)	Is c a digit?
islower(c)	Is c alower case letter?
isprint(c)	Is c a character?
ispunct(c)	Is c a punctuation mark?
isspace(c)	Is c a white space character?
isupper(c)	Is c an upper case letter?
tolower(c)	Convert ch to lower case
toupper(c)	Convert ch to upper case

program:

```
 main()

{ char a;

printf(—enter

char‖); a = getchar();

if (isupper(a))

{

x= tolower(a);

putchar(x);

}

else

putchar(toupper(a));

}
```

Output:- enter char A

a

Formatted I/O Functions: Formatted I/O refers to input and output that has been arranged in a particular format.

Formatted I/P functions → scanf( ) , fscanf()

Formatted O/P functions--- → printf() , fprintf()

scanf( ) :-scanf() function is used to read information from the standard I/P device.

Syntax:- scanf(―controlstring‖, &variable_name);

Control string (also known as format string) represents the type of data that the user is going to accept and gives the address of variable. (char-%c , int-%d , float - %f , double-%lf).Control string and variables are separated by commas. Control string and the variables going to I/P should match with each other.

Ex:- int n;

scanf(―%d‖,&n);

Inputting Integer Numbers:

The field specification for reading an integer number is:

%w d

The percentage sign(%) indicates that a conversion specification follows. w is an integer number that specifies the field width of the number to be read and d known as data type character indicates that the number to be read is in integer mode.

Ex:- scanf(―%2d,%5d‖,&a&b);

The following data are entered at the console:

50 31426

Here the value 50 is assigned to **a** and 31426 to **b**

Inputting Real Numbers:

The field width of real numbers is not to be specified unlike integers. Therefore scanf reads real numbers using the simple specification **%f.**

Ex: scanf(―%f %f‖,&x,&y);

Suppose the following data are entered as input:

23.45 34.5

The value 23.45 is assigned to **x** and 34.5 is assigned to **y**.

Inputting character strings:

The field specification for reading character strings is

%ws or %wc

%c may be used to read a single character.

Ex: scanf(—%s‖,name1);

Suppose the following data is entered as input:

Griet

Griet is assigned to name1.

Formatted Output:

printf( ): This function is used to output any combination of data. The outputs are produced in such a way that they are understandable and are in an easy to use form. It is necessary for the programmer to give clarity of the output produced by his program.

Syntax:- printf(—control string‖, var1,var2……);

Control string consists of 3 types of items

1. Characters that will be printed on the screen as they appear.
2. Format specifications that define the O/P format for display of each item.
3. Escape sequence chars such as \n , \t and \b…..

The control string indicates how many arguments follow and what their types are.

The var1, var2 etc.. are variables whose values are formatted and printed according to the specifications of the control string.

The arguments should match in number, order and type with the format specifications.

O/P of integer number: - Format specification :**%wd**

Format	**O/P**
printf(—%d‖, 9876)	9 8 7 6
printf(—%6d‖,9876)	9 8 7 6
printf(—%2d‖,9876)	9 8
printf(—%-6d‖,9876)	9 8 7 6
printf(—%06d‖,9876)	0 0 9 8 7 6

O/P of real number: %w.pf

w---- Indicates minimum number of positions that are to be used for display of the value.

p------ Indicates number of digits to be displayed after the decimal point.

Format **y=98.7654**	**O/P**
printf(—%7.4f‖,y)	9 8 . 7 6 5 4
printf(—%7.2f‖,y)	9 8 . 7 7
printf(—%-7.2f‖,y)	9 8 . 7 7
printf(—%10.2e‖,y)	9 . 8 8 e + 0 1

O/P of single characters and string:

%wc %ws

Format	O/P
%s	R a J U R a j e s h R a N i
%18s	R a J U R a J E s h R a n i
%18s	R A j U R a j E S h R a n i

2. **What are different types of 'if' statements available in c? Explain them. (Or)**

Describe all the variants of if-else statement with clear syntax and examples. Ans: There are 4 if statements available in C:

1. Simple if statement.
2. if…else statement.
3. Nested if…else statement.
4. else if ladder

1. Simple if statement: Simple if statement is used to make a decision based on the available choice. It has the following form:

Syntax: if (condition)

{

stmt block;

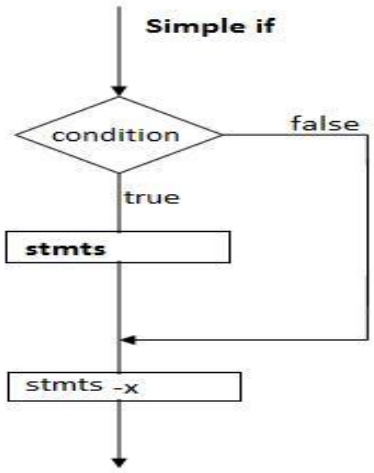
}

stmt-x;

In this syntax,

- if is the keyword. <*condition*>is a relational expression or logical expression or any expression that returns either true or false. It is important to note that the condition should be enclosed within parentheses _(_ and ')'.
- The *stmt block* can be a simple statement or a compound statement or a null statement.
- *stmt-x* is any valid C statement.

The flow of control using simple if statement is determined as follows:



Whenever simple if statement is encountered, first the condition is tested. It returns either true or false. If the condition is false, the control transfers directly to stmt-x without considering the stmt block. If the condition is true, the control enters into the stmt block. Once, the end of stmt block is reached, the control transfers to stmt-x

Program for if statement:

```
#include<stdio.h>

#include<conio.h>

int main()

 {

int age;

printf("enter age\n");

scanf("%d",&age);

if(age>=55)

printf("person is retired\n");

}
```

Output: enter age 57

person is retired

2. if—else statement

if…else statement is used to make a decision based on two choices. It has the following form:

Syntax:

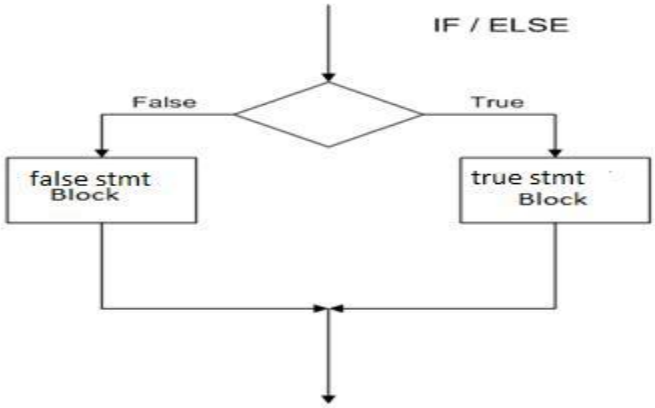
```
if(condition)
        {
        true stmt block;
        }
        else
        {
```

```
false stmt block;
}
stmt-x;
```

In this syntax,

- if and else are the keywords.
- <*condition*>is a relational expression or logical expression or any expression that returns either true or false. It is important to note that the condition should be enclosed within parentheses (and).
- The *true stmt block* and false stmt block are simple statements or compound statements or null statements.
- *stmt-x* is any valid C statement.

The flow of control using if...else statement is determined as follows:



Whenever if...else statement is encountered, first the condition is tested. It returns either true or false. If the condition is true, the control enters into the true stmt block. Once, the end of true stmt block is reached, the control transfers to stmt-x without considering else-body.

If the condition is false, the control enters into the false stmt block by skipping true stmt block. Once, the end of false stmt block is reached, the control transfers to stmt-x.

Program for if else statement:

```
#include<stdio.h>
#include<conio.h>
int main()
```

```
{
int age;
printf("enter age\n");
scanf("%d",&age);
if(age>=55)
printf("person is
retired\n"); else
printf("person is not retired\n");
}
```

Output: enter age 47

person is not retired

3. Nested if—else statement

Nested if…else statement is one of the conditional control-flow statements. If the body of if statement contains at least one if statement, then that if statement is called as ―Nested if…else statement‖. The nested if…else statement can be used in such a situation where at least two conditions should be satisfied in order to execute particular set of instructions. It can also be used to make a decision among multiple choices. The nested if…else statement has the following form:

Syntax:

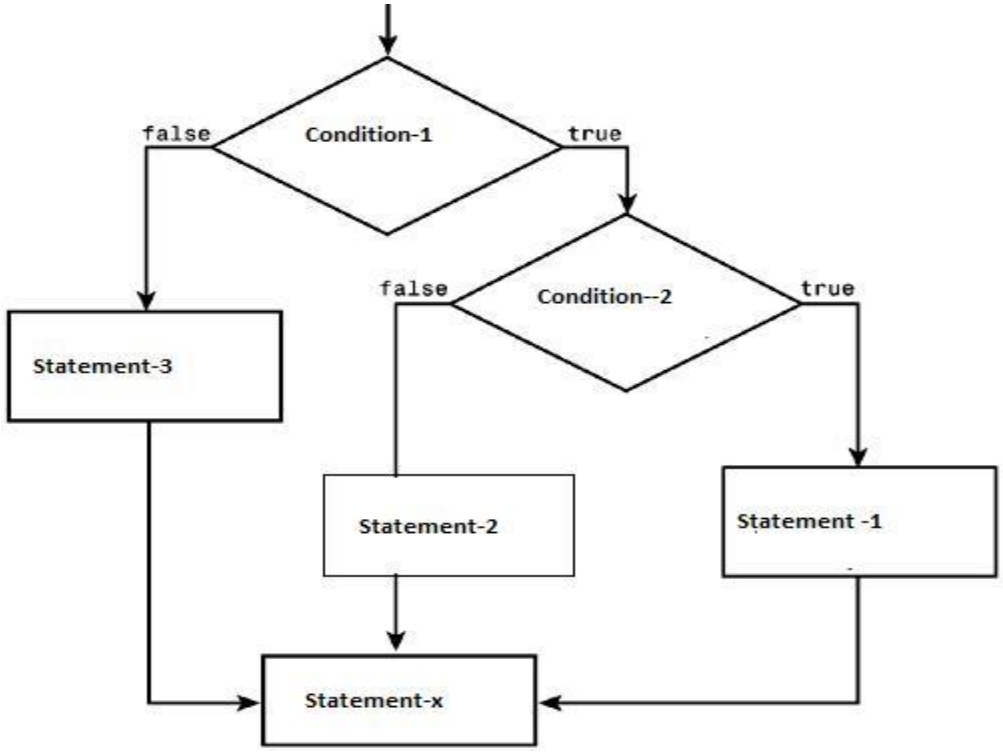
```
if(condition1)
{
if(condition 2)
{
stmt1;
}
else
{
```

```
stmt2;
}
}
else
{
stmt 3;
}
stmt-x;
```

In this syntax,

- if and else are keywords.
- *<condition1>,<condition2> ... <condition>*are relational expressions or logical expressions or any other expressions that return true or false. It is important to note that the condition should be enclosed within parentheses (and).
- *if-body* and *else-body* are simple statements or compound statements or empty statements.
- *stmt-x* is a valid C statement.

The flow of control using Nested if...else statement is determined as follows:



Whenever nested if…else statement is encountered, first <condition1> is tested. It returns either true or false.

If condition1 (or outer condition) is false, then the control transfers to else-body (if exists) by skipping if-body.

If condition1 (or outer condition) is true, then condition2 (or inner condition) is tested. If the condition2 is true, if-body gets executed. Otherwise, the else-body that is inside of if statement gets executed.

Program for nested if… else statement:

```
#include<stdio.h>

#include<conio.h>

void main()

{

int a ,b,c;

printf("enter three values\n");

scanf("%d%d%d",&a,&b,&c);

if(a>b)

{

if(a>c)

printf("%d\n",a);

else

printf("%d\n",b);

}

else

{

if(c>b)
```

```
printf("%d\n",c);
else
printf("%d\n",b);
}
}
```

Output: enter three values 3

2

4

5

4. else—if Ladder

else-if ladder is one of the conditional control-flow statements. It is used to make a decision among multiple choices. It has the following form:

Syntax:

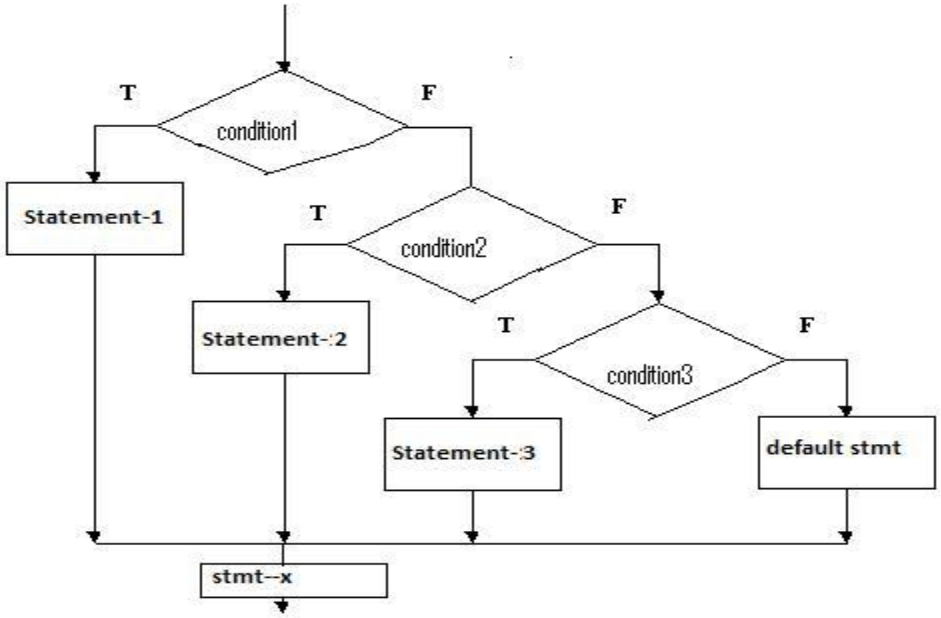
```
if(condition 1)
{
statement 1;
}
else if(condition 2)
{
statement 2;
}
else if(condition 3)
{
statement 3;
}
else if(condition n)
```

```
{
statement n;
}
else
{
default statement;
}
stmt- x;
```

In this syntax,

- if and else are keywords. There should be a space between else and if, if they come together.
- *<condition1>,<condition2>….<condtionN>*are relational expressions or logical expressions or any other expressions that return either true or false. It is important to note that the condition should be enclosed within parentheses (and).
- *statement 1, statement 2,statement 3……,statement n* and *default statement* are either simple statements or compound statements or null statements.
- *stmt-x* is a valid C statement.

The flow of control using else--- if ladder statement is determined as follows:



Whenever else if ladder is encountered, condition1 is tested first. If it is true, the statement 1 gets executed. After then the control transfers to *stmt-x*.

If *condition1* is false, then *condition2* is tested. If *condition2* is false, the other conditions are tested. If all are false, the default stmt at the end gets executed. After then the control transfers to *stmt-x*.

If any one of all conditions is true, then the body associated with it gets executed. After then the control transfers to *stmt-x*.

Example Program:

Program for the else if ladder statement:

```
#include<stdio.h>

#include<conio.h>

void main()

{

int m1,m2,m3,avg,tot; printf("enter

three subject marks");

scanf("%d%d%d",

&m1,&m2,&m3); tot=m1+m2+m3;

avg=tot/3;

if(avg>=75)

{

printf("distinction");

  }

else if(avg>=60 &&avg<75)

{

printf("first class");  }

else if(avg>=50 &&avg<60)
```

```
{
printf("second class");
  }
else if (avg<50)
{
printf("fail");
 }
}
```

Output: enter three subject marks85 80 81

Distinction

3. **Explain the switch statement with Example program.**

Ans: switch statement is one of decision-making control-flow statements. Just like else if ladder, it is also used to make a decision among multiple choices. switch statement has the following form:

Syntax:

```
switch(<exp>)
{   case<exp-val-1>: statements block-1
                          break;
     case<exp-val-2>: statements block-2
                          break;
          case<exp-val-3>: statements block-3
                          break;
                :
          case<exp-val-N>: statements block-N
                          break;
```

---

default: default statements block

}

Next-statement;

In this syntax,

- switch, case, default and break are keywords.

- *<exp>*is any expression that should give an integer value or character value. In other words, it should never return any floating-point value. It should always be enclosed with in parentheses (and). It should also be placed after the keyword switch.

- *<exp-val-1>, <exp-val-2>, <exp-val-3>…. <exp-val-N>* should always be integer constants or character constants or constant expressions. In other words, variables can never be used as *<exp-val>*. There should be a space between the keyword case and *<exp-val>*. The keyword case along with its *<exp-val>* is called as a case label. *<exp-val>* should always be unique; no duplications are allowed.

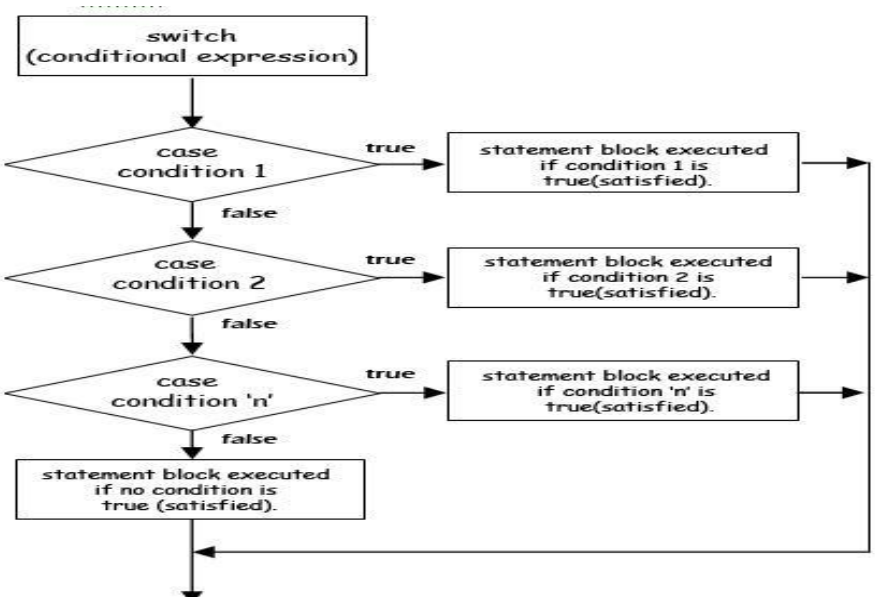
- *statements block-1, statements block-2, statements block-3… statements block-N* and *default statements block* are simple statements, compound statements or null statements. It is important to note that the statements blocks along with their own case labels should be separated with a colon (:)

- The break statement at the end of each statements block is an optional one. It is recommended that break statement always be placed at the end of each statements block. With its absence, all the statements blocks below the matched case label along with statements block of matched case get executed. Usually, the result is unwanted.

- The statement block and break statement can be enclosed with in a pair of curly braces { and }.

- The default along with its statements block is an optional one. The break statement can be placed at the end of default statements block. The default statements block can be placed at anywhere in the switch statement. If they are placed at any other place other than at end, it is compulsory to include a break statement at the end of default statements block.

- *Next-statement* is a valid C statement.



Whenever, switch statement is encountered, first the value of <*exp*> gets matched with case values. If suitable match is found, the statements block related to that matched case gets executed. The break statement at the end transfers the control to the *Next-statement*.

If suitable match is not found, the default statements block gets executed and then the control gets transferred to *Next-statement*.

Example Program:

```
#include<stdio.h>

void main()

{ int a,b,c,ch;

        printf("\nEnter two numbers :");

        scanf("%d%d",&a,&b);

        printf("\nEnter the choice:");

        scanf("%d",&ch);

    switch(ch)

    {

        case 1: c=a+b;
```

```
            break; case 2:
             c=a-b; break;
            case 3: c=a*b;
                        break;

            case 4: c=a/b;
            break; case 5:
                        return;
      }
      printf("\nThe result is: %d",c);
}
```

Output: enter the choice 1

Enter two numbers 4 2

6

4. Write in detail about different types of loop statements in C.

Ans:

While: The simplest of all the looping structures in c is the while statement. The basic format of the while statement is:

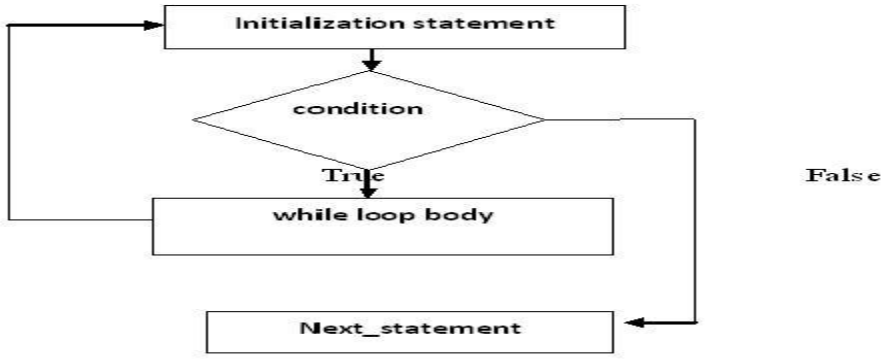
Syntax:

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

The while is an entry –controlled loop statement. The condition is evaluated and if the condition is true then the statements will be executed. After execution of the statements the condition will

be evaluated and if it is true the statements will be executed once again. This process is repeated until the condition becomes false and the control is transferred out of the loop .On exit the program continues with the statement immediately after the body of the loop.

Flow chart:



Program to print n natural numbers using using while

```
#include<stdio.h>

#include<conio.h>

int main()

{

int i,n;

printf("enter the range\n");

scanf("%d",&n);

 i=1;

while(i<=n)

 {

printf("%d",i);

  i=i+1;

  }

  }
```

Output: enter the range 10

12345678910

do-while statement: It is one of the looping control statements. It is also called as *exit-controlled looping control statement*. i.e., it tests the condition after executing the do-while loop body.

The main difference between ―while‖ and ―do-while‖ is that in ―do-while‖ statement, the loop body gets executed at least once, though the condition returns the value false for the first time, which is not possible with while statement. In ―while‖ statement, the control enters into the loop body only when the condition returns true.

Syntax:

Initialization statement;

do

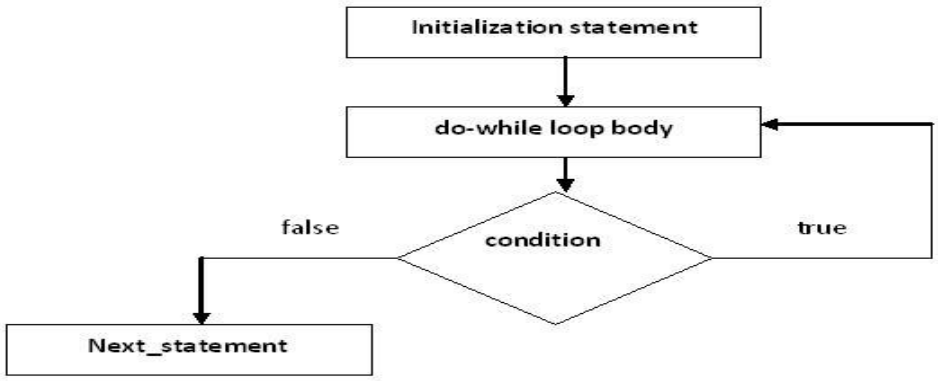
{ statement(s);

} while(<condition>);

next statement;

In this syntax:

while and **do** are the keywords. <condition> is a relational expression or a compound relational expression or any expression that returns either true or false. initialization statement, statement(s) and next_statement are valid _c‘ statements. The statements within the curly braces are called as do-while loop body. The updating statement should be included with in the do-while loop body. There should be a semi-colon (;) at the end of while(<condition>).



Whenever ―do-while‖ statement is encountered, the initialization statement gets executed first. After then, the control enters into do-while loop body and all the statements in that body will be executed. When the end of the body is reached, the condition is tested again with the updated loop counter value. If the condition returns the value false, the control transfers to next statement without executing do-while loop body. Hence, it states that, the do-while loop body gets executed for the first time, though the condition returns the value false.

Program to print n natural numbers using using do while

```
#include<stdio.h>

#include<conio.h>

int main()

{

int i,n;

printf("enter the range\n");

scanf("%d",&n);

 i=1;

do

 {

  printf("%d\n",i);

  i=i+1;

  }

while(i<=n);

  }
```

Output: enter the range 8

12345678

for statement:

It is one of the looping control statements. It is also called as *entry-controlled looping control statement*. i.e., it tests the condition before entering into the loop body. The syntax for ―for‖ statement is as follows:

Syntax:

for(exp1;exp2;exp3)

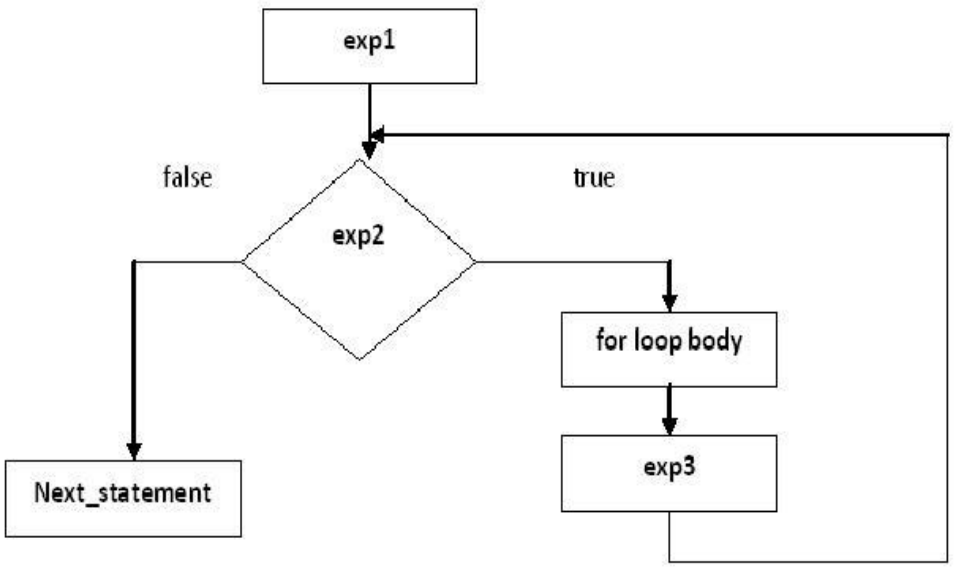
{ for-body;

}

next_statement;

In this syntax,

for is a keyword. exp1 is the initialization statement. If there is more than one statement, then the statements must be separated with commas. exp2 is the condition. It is a relational expression or a compound relational expression or any expression that returns either true or false. The exp3 is the updating statement. If there is more than one statement then, they must be separated with commas. exp1, exp2 and exp3 should be separated with two semi-colons. exp1, exp2, exp3, for-body and next_statement are valid _c' statements. for-body is a simple statement or compound statement or a null statement.



Whenever ―for‖ statement is encountered, first exp1 gets executed. After then, exp2 is tested.

If exp2 is true then the body of the loop will be executed otherwise loop will be terminated.

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 exp3 will be evaluated and the new value is again tested .if it satisfies body of the loop is executed .This process continues till condition is false.

Program to print n natural numbers using for loop

```
#include<stdio.h>

#include<conio.h>

void main()

{

int i,n;

printf("enter the value");

scanf("%d",&n);

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

{

printf("%d\n",i);

}

}
```

Output: enter the value 5

12345

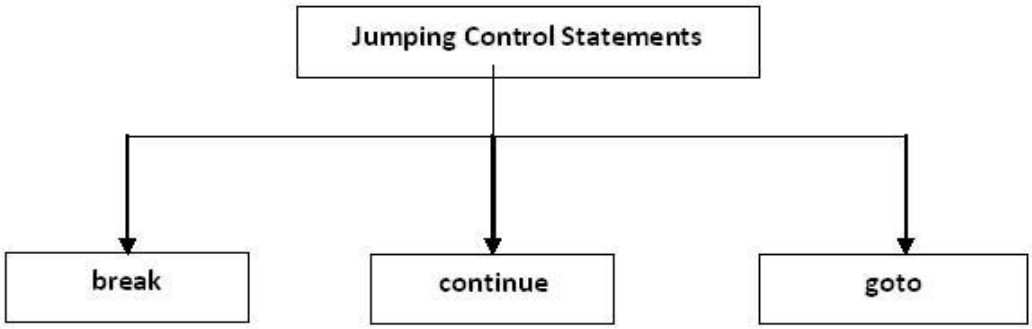
5. Explain Jumping control-flow statements. (Or)
Differentiate between the following with example?
a) Break statement
b) Continue statement
c) Goto statement (Or)

What are the differences between break statement and Continue statement?

Ans:

Jumping control-flow statements are the control-flow statements that transfer the control to the specified location or out of the loop or to the beginning of the loop. There are 3 jumping control

statements:



1. break statement

The ―break‖ statement is used within the looping control statements, switch statement and nested loops. When it is used with the for, while or do-while statements, the control comes out of the corresponding loop and continues with the next statement.

When it is used in the nested loop or switch statement, the control comes out of that loop / switch statement within which it is used. But, it does not come out of the complete nesting.

Syntax for the ―break‖ statement is:

```
break;
```

In this syntax, break is the keyword.

The following representation shows the transfer of control when break statement is used:

Any loop

```
{   statement_1;
    statement_2;
        :
    break;
        :
```

```
}
next_statement;
```

Program for break statement:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i;
for(i=1; i<=10; i++)
{
if(i==6)
break;
printf("%d",i);
}
}
```

Output: 12345

2. continue statement

A continue statement is used within loops to end the execution of the current iteration and proceed to the next iteration. It provides a way of skipping the remaining statements in that iteration after the continue statement. It is important to note that a continue statement should be used only in loop constructs and not in selective control statements.

Syntax for continue statement is:

```
continue;
```

where continue is the keyword.

The following representation shows the transfer of control when continue statement is used:

Any loop

{

statement_1;

statement_2;

:

continue;

:

}

next_statement;

Program for continue statement:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i, sum=0, n;
for(i=1; i<=10; i++)
{
printf("enter any no:");
scanf("%d",&n);
if(n<0)
continue;
else
```

```
sum=sum+n;

printf("%d\n",sum);

}

}
```

Output: enter any no:8

18

3. goto statement

The goto statement transfers the control to the specified location unconditionally. There are certain situations where goto statement makes the program simpler. For example, if a deeply nested loop is to be exited earlier, goto may be used for breaking more than one loop at a time. In this case, a break statement will not serve the purpose because it only exits a single loop.

Syntax for goto statement is:

```
label:
        {
        statement_1;
        statement_2;
                :
        }
        :
goto label;
```

In this syntax, goto is the keyword and label is any valid identifier and should be ended with a colon (:).

The identifier following goto is a statement label and need not be declared. The name of the statement or label can also be used as a variable name in the same program if it is declared appropriately. The compiler identifies the name as a label if it appears in a goto statement and as a variable if it appears in an expression.

If the block of statements that has label appears before the goto statement, then the control has to move to backward and that goto is called as backward goto. If the block of

statements that has label appears after the goto statement, then the control has to move to forward and that goto is called as forward goto.

Program for goto statement:

```
#include<stdio.h>

#include<conio.h>

void main()

{

printf("www.");

goto x;

y:

printf("expert");

goto z;

x:

printf("c

programming"); goto y;

z:

printf(".com");

}
```

Output: www.c programming expert.com

6. **What are the differences between while and do-while statements ?**

Ans:

The main difference between the while and do-while loop is in the place where the condition is to be tested.

In the while loops the condition is tested following the while statement then the body gets executed. Where as in do-while, the condition is checked at the end of the loop. The do-while loop will

execute at least one time even if the condition is false initially. The do-while loop executes until the condition becomes false.

While	do-while
1) It is an entry-controlled control-flow statement. That is, it tests condition before the body gets executed.	1) It is an exit-controlled control-flow statement. That is, it tests condition after the body gets executed.
2) The body gets executed only when the condition is true.	2) The body gets executed at least once though the condition is false for the first time.