2.6 Assignment Statement

Assignment statement can be defined as the statement through which the value obtained from an expression can be stored in a variable.

The general form of assignment statement is

```
< variable name> = < arithmetic expression> ;
```

```
Example:  sum  =  a + b + c;
                tot  =  s1 + s2 + s3;
                area  =  ½ * b* h;
```

2.7 I/O Control Structure (if, If-else, for, while, do-while)

Conditional Statements

The conditional expressions are mainly used for decision making. The following statements are used to perform the task of the conditional operations.

a. if statement.

b. If-else statement. Or 2 way if statement

c. Nested else-if statement.

d. Nested if–else statement.

e. Switch statement.

a. if statement

The **if statement** is used to express conditional expressions. If the given condition is true then it will execute the statements otherwise skip the statements.

The simple structure of '**if**' statement is

```
i.      If (< condtional expressione>)
                statement-1;
                (or)
ii.     If (< condtional expressione>)
                {
```

```
statement-1;
statement-2;
statement-3;
……………
……………
STATEMENT-N
}
```

The expression is evaluated and if the expression is true the statements will be executed. If the expression is false the statements are skipped and execution continues with the next statements.

Example: a=20; b=10;

```
        if ( a > b )
        printf (“big number is %d” a);
```

b. if-else statements

The **if-else** statements is used to execute the either of the two statements depending upon the value of the exp. The general form is

```
if(<exp>)
{
Statement-1;
Statement -2;
…………..          “ SET-I”
……………
Statement- n;
}
else
{
Statement1;
Statement 2;
```

```
        …………..        " SET-II
        ……………
        Statement n;
        }
```

SET - I Statements will be executed if the exp is true.

SET – II Statements will be executed if the exp is false.

Example:

```
        if ( a> b )
        printf ("a is greater than b");
        else
        printf ("a is not greater than b");
```

c. Nested else-if statements

If some situations if may be desired to nest multiple **if-else** statements. In this situation one of several different course of action will be selected.

Syntax

```
if ( <exp1> )
        Statement-1;
else if ( <exp2> )
        Statement-2;
else if ( <exp3> )
        Statement-3;
else
        Statement-4;
```

When a logical expression is encountered whose value is true the corresponding statements will be executed and the remainder of the nested else if statement will be bypassed. Thus control will be transferred out of the entire nest once a true condition is encountered.

The final **else** clause will be apply if none of the exp is true.

d. nestedif-else statement

It is possible to nest if-else statements, one within another. There are several different form that nested if-else statements can take.

The most general form of two-layer nesting is

```
if(exp1)
            if(exp3)
      Statement-3;
            else
      Statement-4;
else
            if(exp2)
      Statement-1;
            else
      Statement-2;
```

One complete **if-else** statement will be executed if **expression1** is true and another complete **if-else** statement will be executed if **expression1** is false.

e. Switch statement

A switch statement is used to choose a statement (for a group of statement) among several alternatives. The switch statements is useful when a variable is to be compared with different constants and in case it is equal to a constant a set of statements are to be executed.

Syntax:

```
Switch (exp)
{
case
      constant-1:
      statements1;
case
      constant-2:
        statements2;
        _______
        _______
    default:
        statement n;
    }
```

Where constant1, constanat2 — — — are either integer constants or character constants. When the switch statement is executed the exp is evaluated and control is transferred directly to the group of statement whose case label value matches the value of the exp. If none of the case label values matches to the value of the exp then the default part statements will be executed.

If none of the case labels matches to the value of the exp and the default group is not present then no action will be taken by the switch statement and control will be transferred out of the switch statement.

A simple switch statement is illustrated below.

Example 1:

```
main()
{
char choice;
printf("Enter Your Color (Red - R/r, White – W/w)");
choice=getchar();
switch(choice= getchar())
{
case'r':
case'R':
printf("Red");
break;
case'w':
case'W':
```

```
printf("white");
break;
default :
printf("no colour");
}
```

Example 2:

```
switch(day)
{
case 1:
printf("Monday");
break;
_______
_______
}
```

2.8 Structure for Looping Statements

Loop statements are used to execute the statements repeatedly as long as an expression is true. When the expression becomes false then the control transferred out of the loop. There are three kinds of loops in **C**.

a) while b) do-while c) for

a. while statement

while loop will be executed as long as the exp is true.

Syntax:
```
while (exp)
{
statements;
}
```

The statements will be executed repeatedly as long as the exp is true. If the exp is false then the control is transferred out of the while loop.

Example:

int digit = 1;

While (digit <=5) FALSE

{

printf ("%d", digit); TRUE

Cond Exp

Statements; ++digit;

}

The while loop is top tested i.e., it evaluates the condition before executing statements in the body. Then it is called entry control loop.

b. do-while statement

The **do-while** loop evaluates the condition after the execution of the statements in the body.

Syntax: do

Statement;

While<exp>;

Here also the statements will be executed as long as the exp value is true. If the expression is false the control come out of the loop.

Example:

-int d=1;

do

{

printf ("%d", d); FALSE

++d;

} **while** (d<=5); TRUE

Cond Exp

statements

exit

The statement with in the do-while loop will be executed at least once. So the **do-while** loop is called a bottom tested loop.

c. for statement

The **for** loop is used to executing the structure number of times. The **for** loop includes three expressions. First expression specifies an initial value for an index (initial value), second expression that determines whether or not the loop is continued (conditional statement) and a third expression used to modify the index (increment or decrement) of each pass.

Note: Generally for loop used when the number of passes is known in advance.

Syntax: **for** (exp1;exp2;exp3)

```
{
        Statement –1;
        Statement – 2;
        ——————; FALSE
        ——————;
        Statement - n; TRUE
}
exp2
        Statements;
exp3
Exit loop
exp1
start
```

Where **expression-1** is used to initialize the control variable. This expression is executed this expression is executed is only once at the time of beginning of loop.

Where **expression-2** is a logical expression. If **expression-2** is true, the statements will be executed, other wise the loop will be terminated. This expression is evaluated before every execution of the statement.

Where **expression-3** is an increment or decrement expression after executing the statements, the control is transferred back to the **expression-3** and updated. There are different formats available in **for loop**. Some of the expression of loop can be omit.

Formate - I

```
for( ; exp2; exp3 )
Statements;
```

In this format the initialization expression (i.e., **exp1**) is omitted. The initial value of the variable can be assigned outside of the **for loop.**

Example 1

```
int i = 1;
for( ; i<=10; i++ )
printf("%d \n", i);
```

Formate - II

```
for( ; exp2 ; )
Statements;
```

In this format the initialization and increment or decrement expression (i.e **expression-1** and **expression-3**) are omitted. The exp-3 can be given at the statement part.

Example 2

```
int i = 1;
for( ; i<=10; )
{
printf("%d \n",i);
i++;
}
```

Formate - III

```
for( ; ; )
Statements;
```

In this format the **three expressions** are omitted. The loop itself assumes the **expression-2**is true. So **Statements** will be executed infinitely.

Example 3

```
int i = 1;
for ( ; i<=10; )
{
printf("%d \n",i);
i++;
}
```

2.9 Nested Looping Statements

Many applications require nesting of the loop statements, allowing on loop statement to be embedded with in another loop statement.

Definition

Nesting can be defined as the method of embedding one control structure with in another control structure.

While making control structure s to be reside one with in another ,the inner and outer control structures may be of the same type or may not be of same type. But ,it is essential for us to ensure that one control structure is completely embedded within another.

```
 /*program to implement nesting*/
 #include <stdio.h>
 main()
{
int a,b,c,
for (a=1,a< 2, a++)
{
printf("%d",a)
for (b=1,b<=2,b++)
{
```

```
print f(%d",b)
for (c=1,c<=2,c++)
{
print f( " My Name is Sunny \n");
                        }
                }
        }
}
```

2.10 Multi Branching Statement (switch), Break, and Continue

For effective handling of the loop structures, C allows the following types of control break statements.

a. Break Statement   b. Continue Statement

a. Break Statement

The break statement is used to terminate the control form the loops or to exit from a switch. It can be used within a for, **while, do-while, for**.

The general format is :

```
break;
```

If **break** statement is included in a **while, do-while** or **for** then control will immediately be transferred out of the loop when the break statement is encountered.

Example

for (; ;) normal loop

```
{
break
Condition
within loop
scanf("%d",&n);
if ( n < -1)
break;
sum = sum + n;
}
```

b. The Continue Statement

The continue statement is used to bypass the remainder of the current pass through a loop. The loop does not terminate when a continue statement is encountered. Rather, the remaining loop statements are skipped and the proceeds directly to the next pass through the loop. The "**continue**" that can be included with in a **while a do-while and a for loop** statement.

General form :

continue;

The **continue** statement is used for the inverse operation of the **break** statement .

Condition

with in loop

Remaining part of loop

continue

Example

```
while (x<=100)
{
if (x <= 0)
{
printf ("zero or negative value found \n");
continue;
}
}
```

The above program segment will process only the positive whenever a zero or negative value is encountered, the message will be displayed and it continue the same loop as long as the given condition is satisfied.

2.11 Differences between Break and Continue

Break	Continue
1. Break is a key word used to terminate the loop or exit from the block. The control jumps to next statement after the loop or block	1. Continue is a keyword used for containing the next iteration of the loop
2. Break statements can be used with for, while, do-while, and switch statement. When break is used in nested loops, then only the innermost loop is terminated.	2. This statement when occurs in a loop does not terminate it rather skips the statements after this continue statement and the control goes for next iteration. 'Continue' can be used with for, while and do- while loop.
3. Syntax:{ statement1; statement2; statement3; break;}	3. Syntax: { statement1; continue; statement2; statement3; break; }
4. Example :Switch (choice){ Case 'y': printf("yes"); break; Case 'n': printf("NO"); break;}	4. Example:- I = 1, j=0;While(i<= 7){ I=I+1; If((I==6) Continue; j = j + 1;}
5. When the case matches with the choice entered, the corresponding case block gets executed. When 'break' statement is executed, the control jumps out of the switch statement.	5. In the above loop, when value of ' i becomes 6' continue statement is executed. So, j= j+1 is skipped and control is transferred to beginning of while loop.

2.12 Unconditional Branching (Go To Statement)

goto statement

The **go to statement** is used to alter the program execution sequence by transferring the control to some other part of the program.

Syntax

Where label is an identifier used to label the target statement to which the control would be transferred the target statement will appear as:

Syntax

```
goto<label>;

label :

statements;
```

Example 1

```
#include <stdio.h>
main();
{
inta,b;
printf ("Enter the two numbers");
scanf ("%d %d",&a,&b);
if (a>b)
gotobig;
else
gotosmall;
big :printf ("big value is %d",a);
gotostop;
small :printf ("small value is %d",b);
gotostop;
stop;
}
```

Simple Programs Covering Above Topics

Practice Programs

1. Write a C program to find out smallest value among A, B,C.

Ans:

```
include <stdio.h>
int a,b,c;
clrscr();
scanf(%d %d %d, &a, &b, &c);
if (a<b)
        {
```

```
        if(a<c)
        printf("a is small/n")
        else
}
```

02. Write a 'C' programe for 5th multiplication table with the help of goto statement.

Ans.

```
#include<stdio.h>
main( )
{
        int t, n = 1, P;
        Printf("Enter table number:");
        Scanf("%d,&t);
        A:
        if (n<=10)
        {
                P=t * n;
                Printf("%d * %d = %d \n", t,n,p);
                n++;
                goto A;
        }
        else
                printf("Out of range");
        }
```

03. Write a 'C' program to find greatest among three numbers.

Ans. #include<stdio.h>

```
void main( )
```

```
{
        int a,b,c;
        printf("enter the values of a,b,c,");
        scanf("%d%d%d", &a,&b,&c);
        if((a>b)&&(c>b))
        {
                if(a>c)
                printf("a is the max no");
                else
                printf("C is the max no");
        }
                else if ((b>c)&&(a>c))
        {
                if(b>a)
                printf("b is the max no");
                else
                printf("a is the max no");
        }
                else if ((b>a)&&(c>a))
        {
                if(b>c)
                printf("b is the max no");
                else
                printf("C is the max no");
        }
}
```