Unit 5. Control Statement

5.1 Conditional Statements,

5.2 Decision Making and Branching (if, if else, nested if else, else if ladder, and switch statements)

5.3 Decision Making and Looping (for, while, and do while loops)

5.4 Exit function

5.5 Break and Continue.

Control statements

- **A Control statements** is a statement which enables to specify the flow of control of a program i.e. , the order in which the instructions in a program must be executed.
- It make it possible to make decisions, to perform tasks repeatedly or to jump from one section of code to another.
- There are four types of control statements in C:
  1. Decision making statements
  2. Selection statements
  3. Iteration statements
  4. Jump statements

Decision Making and Branching

- Decision making and branching statements are implemented in C using the if, if-else, nested if else and if-else if ladders.
- The If Statement: If a single statement or block of statements has to be executed only when the given condition is true, if- statement is used. The syntax of if statement is :
- if (expression)

  Statement;

  Alternatively,

  if(expression)
  {
  statement1;
  statement2;
  statement3;
  … … … …
  }
- If there is only single statement inside if, use of {} is optional but if there are multiple statements, then parentheses must be used.
- The statement/statements inside if will be executed only when the expression evaluates the value : true

The if-else Statement

- The **if-else** statement is used to select one of the two alternatives. The syntax of **if-else** is:
- if(expression)

{

- Statement/statements;

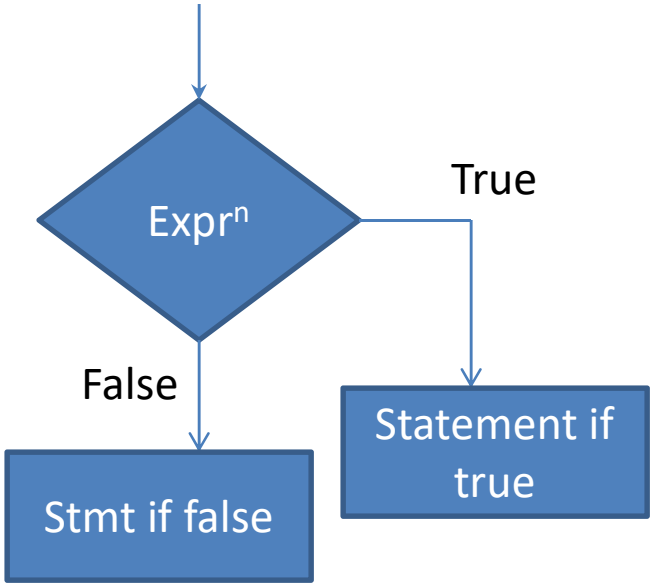
}

else

{

-statement/statements;

}



- If conditional expression in if statement is true, then the statement/statements inside if block will be executed and control gets transferred immediately after the statement in else block.
- If condition is false then the statement/statements inside else block will be executed and control is transferred after the statement after else block

Example: Find a number even or odd

```
#include<stdio.h>
#include<conio.h>
main()
{
      int num,rem;
      printf("Enter an positive integer:");
      scanf("%d",&num);
      rem=num%2;
      if(rem==0)
      {
            printf("%d  is Even number",num);
      }
      else
      {
            printf("%d is Odd Number",num);
      }
      getch();
      return 0;
}
```

Nested if-else statement

- The nested if-else statement includes if-else statement inside if-else statement itself. The syntax of nested if-else is:
- **If(expr1)** **/* if expr1 start */**

```
{
        if(expr2)    /* inner  if exp2  start  */
        {
                     statement/statements;
        }
        else                    /*  inner else block start */
        {
                     statement/statements;
        }                       /* inner else block  end */
}                                                    /* end if expr1 */
else                 /* start of outer else 1 for if expr1 */
{
        if(expr3)
        {
                     statement/statements;
        }
        else
        {
                     statement/statements;
        }
} /*end of else 1*/
```

Nested if-else example.

```
#include<stdio.h>
main()
{
        int num,rem;
        printf("Enter an integer Number:");
        scanf("%d",&num);
        if(num>=0)
        {       rem=num%2;
                if(rem==0)  {    printf("%d is positive even number",num);   }
                else       {    printf("%d is positive odd number",num);    }
        }
        else  {    rem=num%2;
                if(rem==0)  { printf("%d is negative even number",num);   }
                else            {  printf("%d is negative odd number",num);  }
                        }
        return 0;
}
```

The if-else-if ladder

Syntax:

```
If(expr1)
{           statement1/statements ;
}
else if( expr2)
{
            statement2/statements;
}
else if(expr3)
{
            statement3/statements;
}
.
.
else if(expr n)
{
            statement_n/statements;
}
else
{
            statements  when all  expression results false
}
```

Example: if-else if ladder

```
#include<stdio.h>
#define PASS_MARK 40
main()
{
        float score;
        printf("Enter your Exam Score:");
        scanf("%f",&score);
        if(score<PASS_MARK)
                printf("You have FAILED !");
        else if(score<55)
                printf("You have PASSED !");
        else if(score<70)
                printf("You have passed with SECOND DIVISION");
        else if(score<80)
                printf("You have passed with FIRST DIVISION");
        else if(score<=100)
                printf("You have passed with DISTINCTION");
        else
                printf("Your Score is invalid");
        return 0;
}
```

The selection : Switch-case

- The *switch-case* statement is a multi way decision that tests whether an expression matches one of the number of constant expression and selects the statement accordingly.
- *switch case* statements are a substitute for long if statements. The basic format for using switch case is outlined below.

```
switch (expression or variable)
{
        case  constant-expr1:       statement sequence;
                                    break;
        case  constant-expr2:       statement sequence;
                                    break;
        case constant-expr3:   statement sequence;
                                    break;
                ...
        default:
                 statement sequence;
}
```

The selection : Switch-case

- In switch-case, the expression or variable has a value.
- The **case** says that if the **expression** or variable has value which is in after that case, then execute the statements that follows the colon until the **break** is reached.
- Only one **case** is executed since only one value holds for **expression** or variable inside switch.
- For the checking of any one of multiple **case** to be true, the **break** statement between those cases is not used and only statements after listing of **case** are executed.
- The ***break*** is used to break out of the case statements. **break** is a keyword that breaks out of the code block .
- The **break** prevents the program from testing the next **case** statement also.

Switch-case example

```
#include <stdio.h>
#include <stdlib.h>      /* for exit()*/
main()
{
        int input;
        printf("1. Play game\n");
        printf("2. Load game\n");
        printf("3. Play multiplayer\n");
        printf("4. Exit\n");
        printf("Enter your Choice:");
        scanf("%d",&input);
        switch (input)
        {          /*switch block starts */
                case 1: printf("Play game is choosen");
                /*note use of  : and ; */          break;
                case 2: printf("Load game is choosen");
                                break;
                case 3:  printf("Play multiplayer is choosen");
                                break;
                case 4: exit(0);
                default:printf("Error, bad input");
        }         /* end of switch */
}        /* end of main() */
```

The exit() function

- The C library function exit() terminates the calling process immediately , so terminates the running program.
- File buffers are flushed, streams are closed, and temporary files are deleted.
- In C library exit() function has been defined in the header file <stdlib.h> with following structures.

  void exit(int status)
- **So to call exit() function, we should pass the integer parameter to the function.**
- The status Indicates whether the program terminated normally.
- 0 – normal termination (Successful exit) e.g. exit(0);
- EXIT_SUCCESS - successful Termination
- EXIT_FAILURE - Unsuccessful Termination

Another example:

/* this program takes input a alphabetic character and determines that either the character is vowel or consonant and if other character is found than letter it gives appropriate message */

```
#include<stdio.h>
main()
{
        char ch;
        printf("\nEnter  a character:");
        scanf("%c",&ch);
        if((ch>='A'&&ch<='Z')||(ch>='a'&&ch<='z'))
        {
                printf("This is alphabetic character.");
                switch(ch)
                {
                        case 'A': case 'a':
                        case 'E': case 'e':
                        case 'I': case 'i':
                       case 'O': case 'o':
                        case 'U': case 'u':
                        printf("\nThe letter is vowel ");
                        break;
                        default: printf("\nThe letter is consonant.");
                }    /* end switch */
         } /*end if */
        else printf("The character is not alphabetic.");
        return 0;
}   /*end main() */
```

Loops: iterative methods

- The term “iterative method” refers the methods where one or more steps are repeatedly executed more than one times to obtain the require solution.
- This is the most powerful technique to solve the linear problem which has the same operation to be performed more than one times.
- In iterative technique we use looping construct to repeat the same operations.
- For example the algorithm to add the natural number up to n without iterative method can be

    1. Input n
    2. Sum=1+2+3+4+5+6+ ……+n;
    3. Display sum

Loops: iterative methods

- If the number of terms is large this straight forward method is not suitable for the solution. Which is not appropriate algorithm for programming.
- We can write the iterative algorithm to add n natural number as:

1. **Input n ;**
2. **count = 1 ;**
3. **sum=0;**
4. **while count <=n do repeat step 5 to 6**
5. **sum = sum + count;**
6. **count = count +1;**
7. **print sum ;**

- In this case step 5 and 6 are repeatedly executed until the condition is not true. If the condition is not true then rest of the step are executed.
- Hence we can see that writing the algorithm using iterative method is much more easier and practical to solve the problem.

Control Loops in C

- In C iterative algorithms are written using the Loops. Loops are used to repeat a block of code. C gives us a choice of three types of loop, ***while loop, do while loop*** **and** ***for loop***
- The while loop keeps repeating an action until an associated test expr returns false. This is useful where the programmer does not know in advance how many times the loop will be traversed.
- The do while loops is similar, but the test occurs after the loop body is executed. This ensures that the loop body is run at least once.
- The for loop is frequently used, usually where the loop will be traversed a fixed number of times. It is very flexible.
- However we can write an iterative algorithm to solve a problem using any one of these three loops.

The while statement :

- The while statement, also called the while loop, executes a block of statements as long as a specified condition is true. The while statement has the following form:

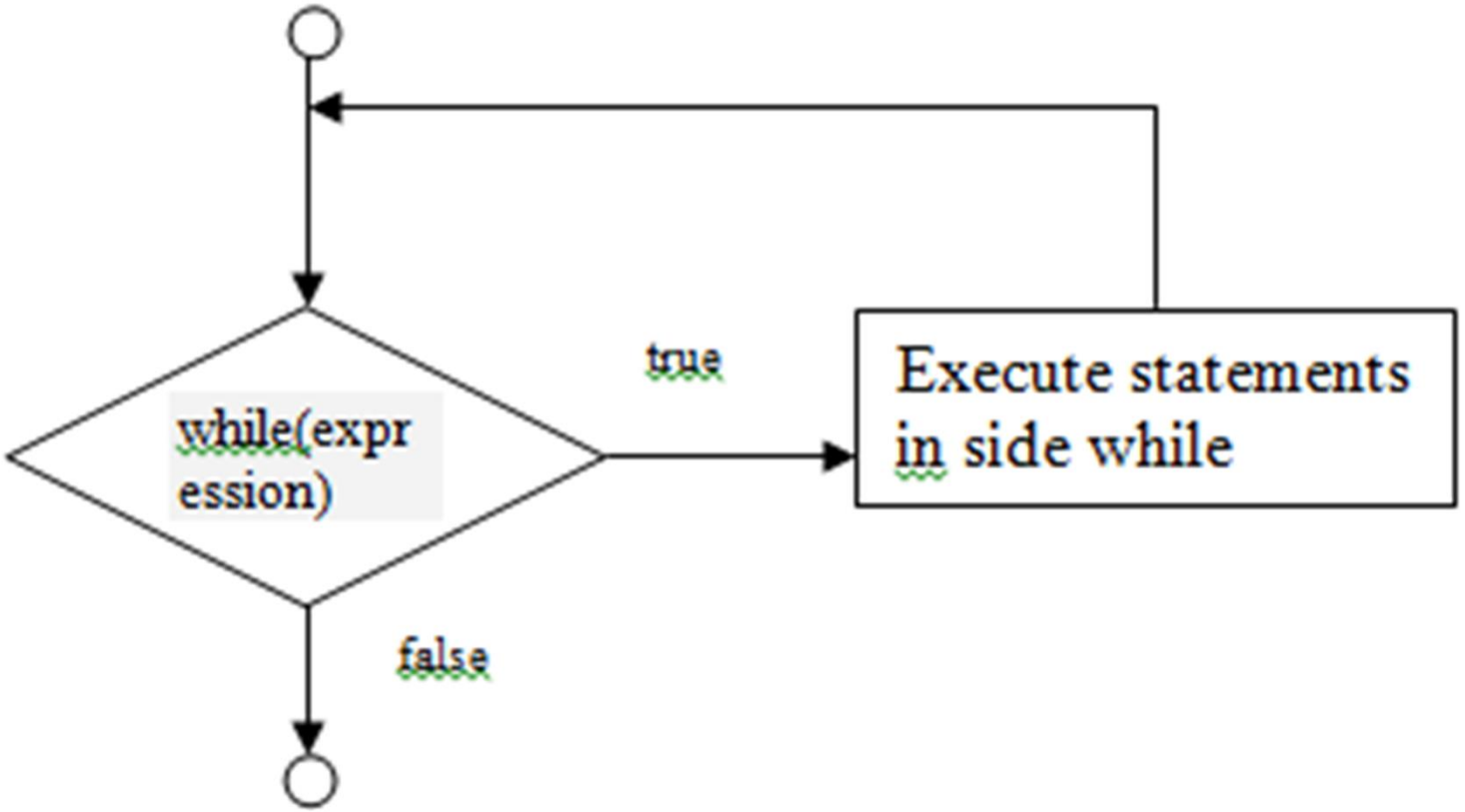
```
while (condition)
    statement;
```

- Here, **while** is keyword and condition is any C expression usually a relational or logical expression and statement is a single or compound C statement. When program execution reaches a **while** statement, the following events occur:

1. The expression condition is evaluated.
2. If condition evaluates to false (0), the **while** statement terminates, and execution passes to the first statement following statement.
3. If condition evaluates to true (nonzero), the C statement(s) in statement inside **while** are executed.
4. Again execution returns to step 1 to test condition.

Flow of Control in while

The flow chart representation of While loop is :



Example A simple while statement.

```
1: /* Demonstrates a simple while statement */
2:
3:   #include <stdio.h>
4:
5:   int count;
6:
7:   void main()
8:   {
9:       /* Print the numbers 1 through 20 */
10:
11:      count = 1;
12:
13:      while (count <= 20)
14:      {
15:          printf("%3d", count);
16:          count++;
17:      }
18:  }
```

Note: The statements in line no: 15 to 16 are executed until count >20

- **/* Sum of first n natural numbers */**

```
#include<stdio.h>
main()
{
        int n, num, sum;
        printf("Enter the value of n:");
        scanf("%d",&n);
        sum=0;  num=1;
        while(num<=n)
        {
                sum+=num;
                num++;
        }
        printf("The sum of %d natural number= %d",n,sum);
        return 0;
}
```

- **/* Find the sum of even and odd numbers from 1 to n */**

```
#include<stdio.h>
main()
{
        int n, num=1,sum_even=0,sum_odd=0;
        printf("Enter  value of n:");
        scanf("%d",&n);
        while(num<=n)
        {
                if(num%2==0)  /*even */
                        sum_even+=num;
                else
                        sum_odd+=num;
                num++;
        }
        printf("The sum of Even numbers: %d", sum_even);
        printf("\nThe sum of odd numbers: %d", sum_odd);
        return 0;
}
```

Nesting while Statements

- Just like if else while statements can also be nested i.e. the while statement itself contains other while statements as our requirements :
- The general syntax of nested while is:

```
while(expression1)
{
    while(expression2)
    {
            statements;
    }
    other statements;
}
```

Use of break statement in while loop:

The break statement is used to break the loop even if the expression of while loop evaluates true. If break statement is encountered inside the while loop, the rest of the part of the loop is skipped i. e. loop is broken out and the control gets transferred outside the loop immediately.

An example:

```
sum = 0;
 i = 1;
 while(i <= 10)
{
        sum += i;
         i++;
}
```

The above loops adds the number 1 to 10.

We can break from while loop using break statement writing lines above using break as :

```
sum = 0;
i = 1;
while(1)
{
        sum += i;
        i++;
        if(i>10)
        break;
}
```

Use of continue statement in loop

- Using continue statements inside a loop, we can skipped some statements and continue the condition test or update (increment or decrement) without executing the other statements in loop. In case of while loop, continue redirects the control of program into the while condition. Example for continue which is used to add the even number up to 10 is shown below.

```
i = 2;
sum = 0;
while(i<=10)
{
        sum += i;
        i +=2;
}
```

```
i = 1;
sum = 0;
while(i <= 10)
{
        if(i%2!=0)
        {
                i++;
                continue;
        }
        sum+= i++;
}
```

These are equivalent blocks

For Loop :

- The for statement is a C programming construct that executes a block of one or more statements a certain number of times.
- A for statement has the following structure:

```
for(initialization statement;    condition statement;       increment/decrement statement )
{
   statement sequences;
}
```

e.g.

```
for(i = 0;i<= 10; i++)
        sum += i;
```

Note: {} must be used in for statement if more than one statements are inside the loop. In above example , the for loop has to execute only one statement: **sum+=i;** so , we have not used left and right curly brackets.

Rules regarding for loop:

- A for loop generally contains three types of statement: initialization, condition and increment/decrement.
- All the three kind of statements in the for loop are optional. It is permitted to write:

  for(; ;){ ….. …. } /* null statement initialization, condition */
- But , it is absolutely necessary to put two semicolons in for loop even if there are no initialization and condition statements
- There can be more than one initialization statements and increment /decrement statements. In such cases each statements should be separated by comma ‘,’ . for example we can write :

  for(sum = 0,i = 1; i <= 10;sum += i, i++)

  ; /* null statement inside for */

- There may not be single statement following for loop, even that case, a semicolon ‘;’ is must be put after the for loop indicating that the for loop executes a null statement.

How for loop works ?

Step 1: Initialization statements are executed .
these are executed only once.

Step 2: Condition is tested

Step 3: If condition is true, the statements associated with for loop are executed. If condition is false, exit from the for loop.

Step 4: Increments/decrement statement are executed

Step 5: Go to step 2

A simple example of for loop:

/ This program adds numbers from 1 to 20 and display the sum in screen */*

```
 1: #include<stdio.h>
2:
3: void main()
4: {
5:  int sum,i;
6:  sum = 0;
7:  for(i=1;i<=20;i++)
8:    sum += i;
9:  printf("The sum is  : %d" , sum);
10:}  /*end of main */
```

The Alternatives for the for statement in line no 6 to 8 of above program can be written as:

```
.    for(sum = 0, i= 1; i<=20; sum+=i; i++)
                ;
```

```
sum = 0;
 i=1;
for(;i <= 20;sum += i,i++)
;
```

```
sum = 0;
i=1;
for( ; i <= 20; i++)
 sum += i;
```

```
sum = 0;
i=1;
for( ; i <= 20;)
{
   sum += i;
   i++;
}
```

```
sum = 0;
i = 1;
for(; ;)
{
   sum += i;
   i++;
   if(i>20)
        break;
}
```

- *In fifth case(rightmost box above) the for(;;) loop is infinite loop, as condition part is missing. Which is controlled by using break statement.*

Use of continue and break

```
/* A simple program that adds the odd numbers from 1 to 99 */
 #include<stdio.h>
 main()
{
        int sum = 0, i;
         for(i=1;i++)
        {
                 if(i%2 == 0)
                         continue;
                sum += i;
                if(i >= 99)
                         break;
         }        /*end of for loop */
        printf("The sum of 1,3,… ,99 is : %d",sum);
        return 0;
}      /*end of main() */
```

***break**: breaks the loop and control goes out from the loop .*

***continue**: goes to execute the increment or decrement part of the for loop without executing rest of statements in loop.*

The do while statement:

- ***do while*** is another type of looping construct in C in which statements are executed at least once even if the condition is not true.
- It executes the statements in the loop block at once and checks the condition . Until the condition is not true it executes the loop repeatedly and when condition is false, the loop exits.
- The general construct of 'do while' statement is :

```
do{
        Statements sequences;
}while(condition );     /* note the semicolon */
```

- condition is any C expression, and statement is a single or compound C statement. When program execution reaches a do...while statement, the following events occur:

1. The statements in statements sequences are executed.

2. condition is evaluated. If it's true, execution returns to step 1. If it's false, the loop terminates.

The for statement in the above program

```
for(sum = 0;i =1; i <= 20;i++)
    Sum += i;
```

Can be written using 'do while' loop as:

```
sum = 0;
i = 1;
do{
  sum += i;
  i++;
 }while(i <= 20 );
```

In while statement, condition is tested at the beginning and if the condition is false the statements in loop are not executed of the loop but in 'do while' the condition is tested at the end of the loop after executing the statements at once.

Use of continue in while, do while, and for loop

- In case of 'while' and 'do while' loop when the continue encountered, it skips the remaining of the statements of loop and goes to check condition.
- In case of 'for' loop, when the continue is encountered, the remaining statement of the loop are skipped and increment statements of for loop executed and then the condition is tested.
- The 'continue' statement is applied to loop only not in 'switch'
- The 'break' statement exits the loop skipping the remaining statements in loop in all types of loop.