

Unit 3 Introduction to Programming

Structure:

- 3.1 Introduction
 - Objectives
- 3.2 Structure of C Program
- 3.3 Compilation and Execution of a C Program
- 3.4 Decision Making and Branching Statements
- 3.5 Break Statement
- 3.6 Continue Statement
- 3.7 Switch Statement
- 3.8 Goto Statement
- 3.9 Iterative Statements
- 3.10 Examples
- 3.11 Summary
- 3.12 Terminal Questions
- 3.14 Answer to Self-Assessment and Terminal Questions

3.1 Introduction

The C language programming creates a new programming generation. It is very important to learn and implement. It is a very sensitive and interesting language. Like natural language, when we want to learn a computer programming language, we must follow some structure or procedure.

There are some tools through which we can write programs, making decisions, looping and branching. To run the program we required compiler. There is a description of Turbo C 3.0 compiler, through which we can write and execute program. The detailed steps are given.

Objectives:

At the end of this unit, you will be able to:

- explain the structure of a C program.
- expose the Turbo C 3.0 compiler environment
- write programs using conditional statements.

3.2 Structure of C Program

Please follow the sequence of instructions that should be maintained to be a C Programmer.

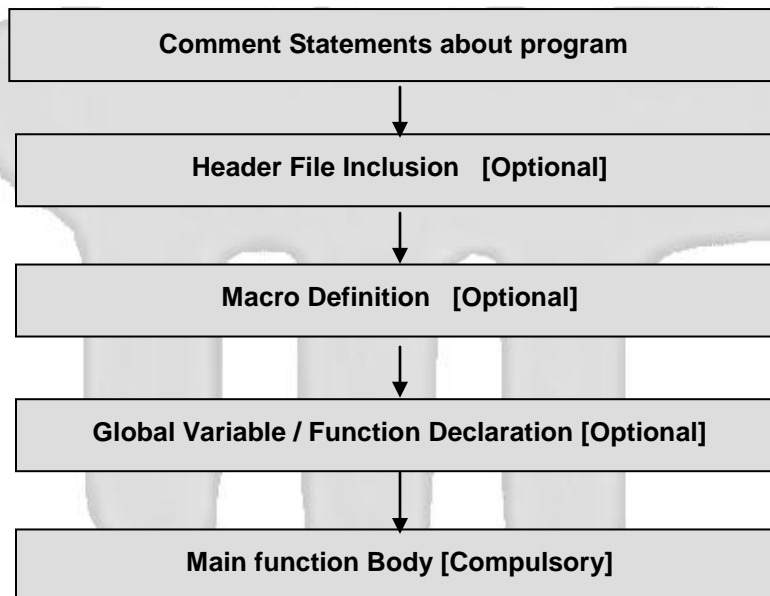


Fig. 3.1

Let us start with the compulsory portion, that is, the **main function body**. In the main function, there are a number of parts out of which the basic parts are as follows:

i) Starting of main function

The compulsion function is the main function that would be done as

```
// program for addition of two integer numbers  
void main()  
{
```

ii) Variable declaration

To declare variable we must have the awareness of variable name convention, data type and syntax for declaration etc.

```
int a,b,c;  
Or can initialize value at the time of declaration as:  
int a=20,b=40,c;
```

iii) Message for accepting value

To display any values or messages into the screen or to file we have several functions called output functions. The functions are given below:

printf(),sprintf(),cprintf(),puts(),putw(),fwrite(), fprintf() etc.

The syntax of printf() is given below:

printf("message/ scan format" [, list of variables]);

Note: The scan format varies upon the type of variable which is already defined in section 1.5. The following statement will show user a message that prompt the user to enter two values through keyboard.

```
printf(" Please enter two values : ");
```

iv) Inputs from the Keyboard

To input the data into the program, we have a number of functions called input functions which are given below:

scanf() , gets() , getch() , getche() , getchar() , getc() , getw() etc.

The syntax of scanf() is given below and others will be discussed later.

```
scanf("scan format",&variables);
```

Note: The scan format varies upon the type of variable which is already defined in Section 1.6. The following statement will accept two integer values from keyboard and store to variables a and b [enter two values in the same line separated by space].

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

v) Process according to problem given

Now we need to add the two numbers. So we need the knowledge of the operators and expressions (refer Section 1.7). Here we need the arithmetic operator '+' for addition.

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

vi) Show Output on monitor

To display the output, we again need the output function which has been discussed earlier. The statement is

```
printf("The sum of %d and %d is = %d",a,b,c);
```

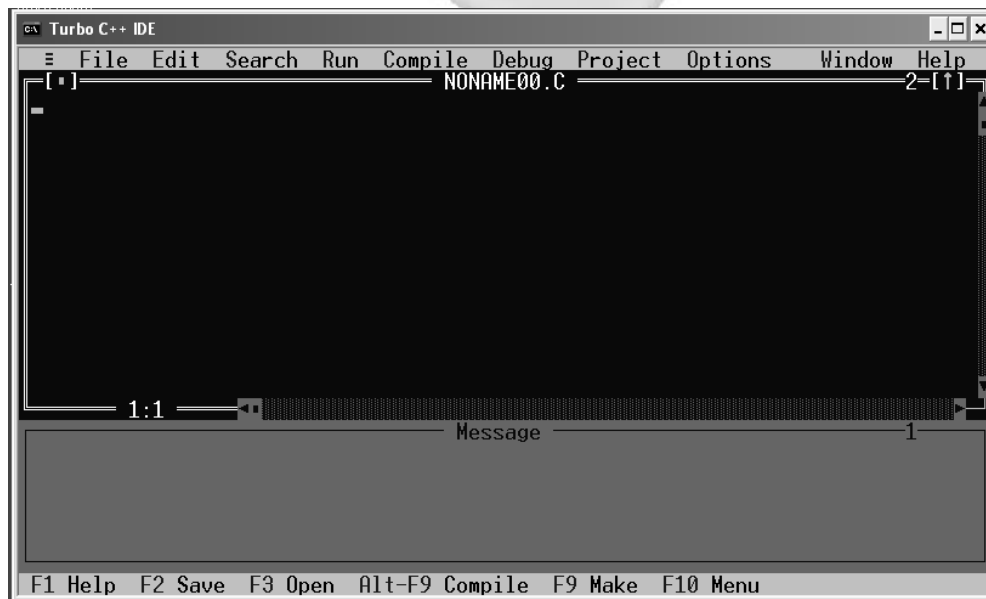
vii) Closing of the main function

Now the time is to close the main program and run the program for desired output. Close the main function with a '}' as given below:

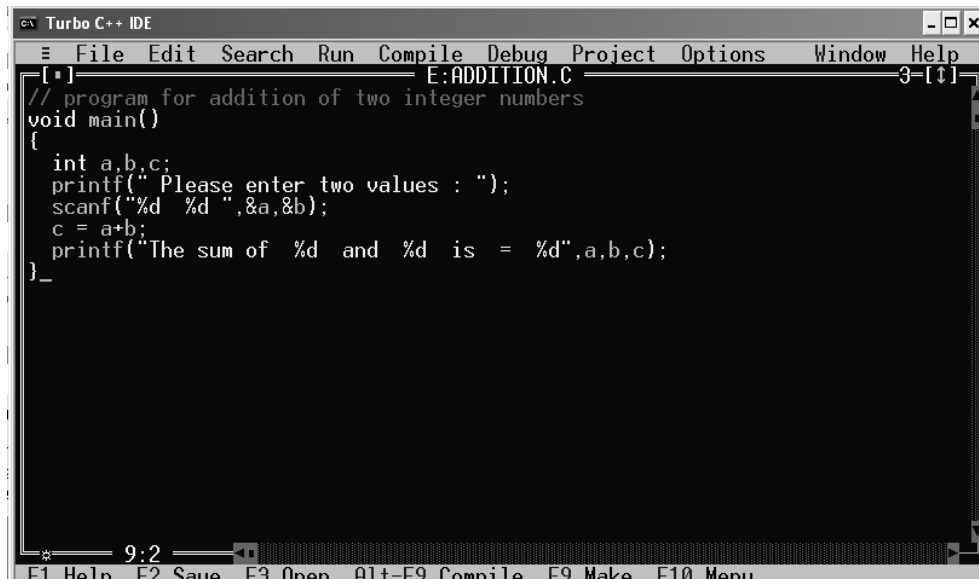
```
}
```

3.3 Compilation and Execution of a C Program

1) Open Turbo C compiler :



2) Write or enter program into this editor. In the shaded area the enter program has been written. After that save from file menu by the name **addition.c**

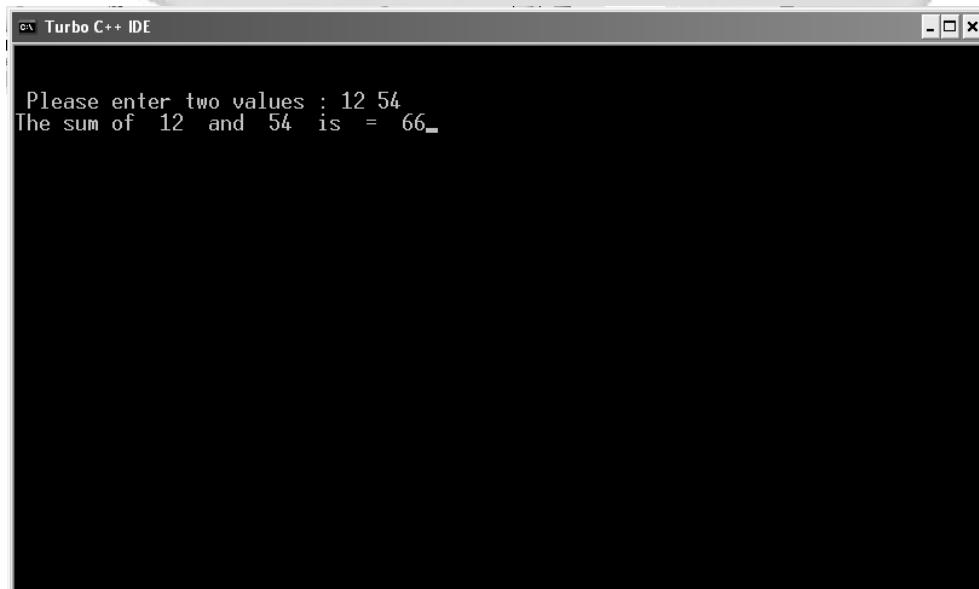


```

Turbo C++ IDE
E:ADDITION.C
// program for addition of two integer numbers
void main()
{
    int a,b,c;
    printf(" Please enter two values : ");
    scanf("%d %d",&a,&b);
    c = a+b;
    printf("The sum of %d and %d is = %d",a,b,c);
}

```

3. Compile and run by pressing **Ctrl+F9** or Click on Run menu



```

Turbo C++ IDE
Please enter two values : 12 54
The sum of 12 and 54 is = 66_

```

3.4 Decision Making and Branching Statements

The decision making and branching is one of the important concepts in programming. These statements let us to select one of the many possible criteria involved in the problem domain. The if and switch statements are belong to this category of statement and they are discussed in the subsequent sections.

i) if statement

There are four types of if statements

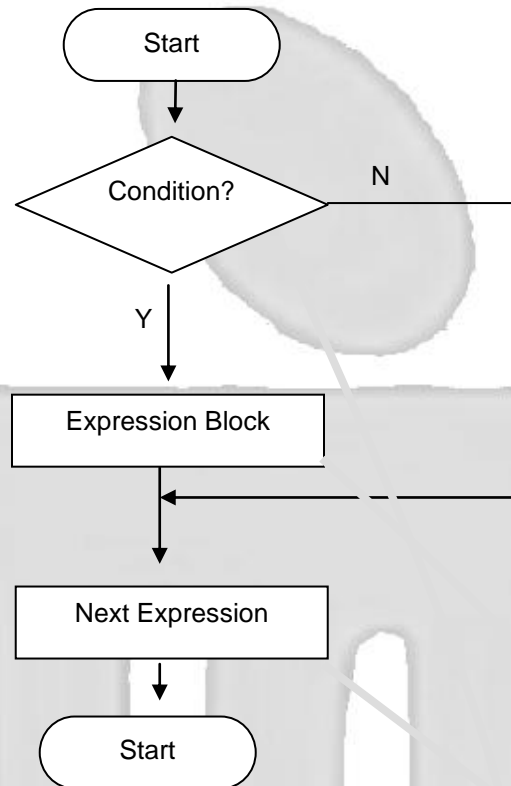
- a) **Simple if statement:** This if statement has only true condition expression block.

Syntax:

```
if(<condition>)  
{  
    <Expression Block>;  
}  
    <Next Expression>;
```

Manipal

The following flowchart illustrate the operation of a simple if statement.



Example:

```
void main()
{
    int age = 18;
    if(age>=13 && age<=19)
    {
        printf("Teen Ager ");
    }
    printf("\nBye Bye");
}
```

Output: Teen Ager
Bye Bye

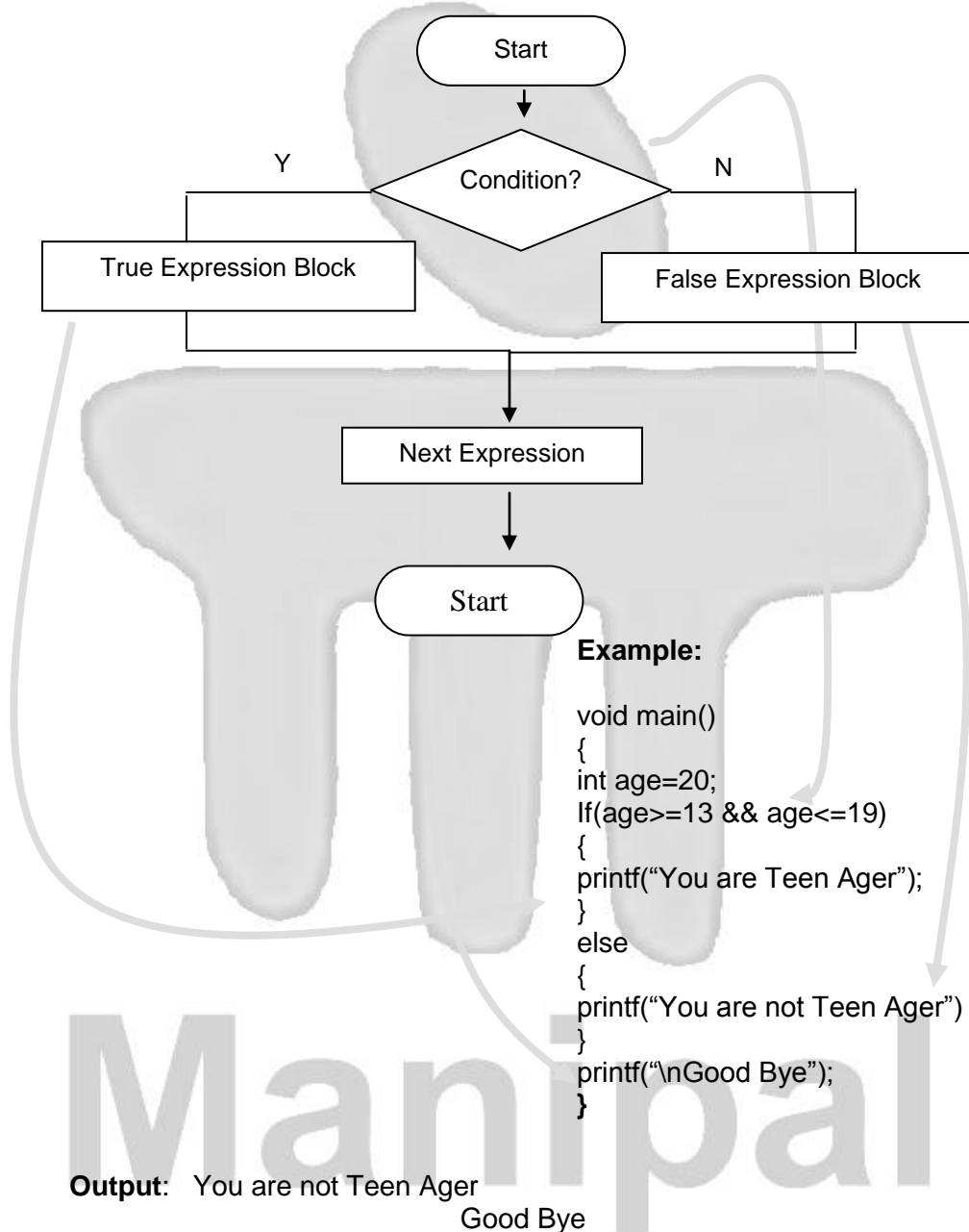
b) Complex if statement: This if statement has true and false condition expression block.

Syntax:

```
if(<condition-1>
{
    <true expression block>;
}
else
{
    <false expression block>;
}
```

Manipal

The following flowchart illustrate the operation of a complex if statement.

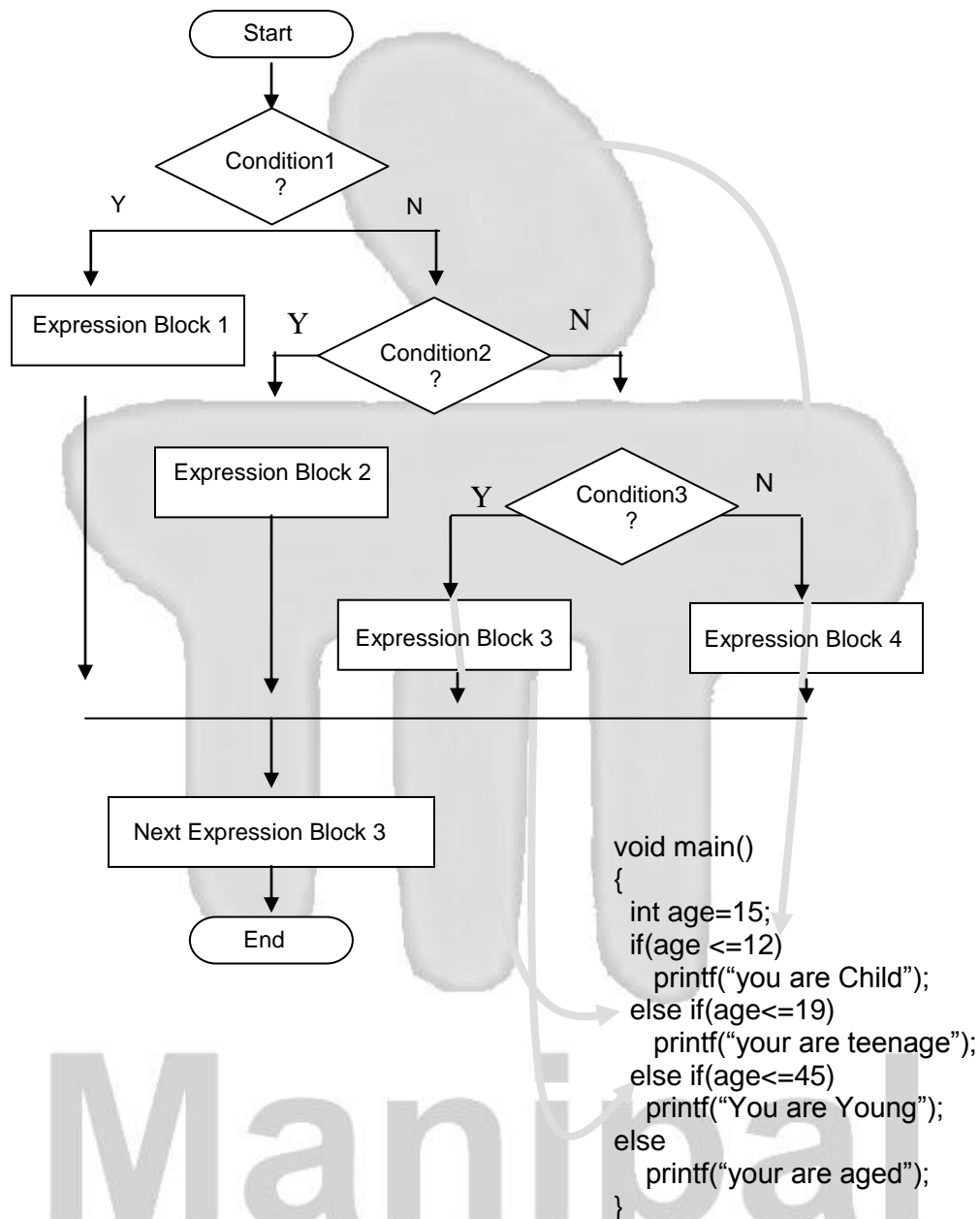


c) Compound if Statement: The compound if statement is the checking of multiple conditions.

```
if(<condition-1>
{
    <expression block -1>; }
    else if (<condition-2>)
    {
        <expression block -2>; }
    else if (<condition-3>)
    {
        <expression block -3>; }
    -----
    -----
else
`{
    <expression block -n>; }
```

Manipal

The following flowchart illustrate the operation of a compound if statement.

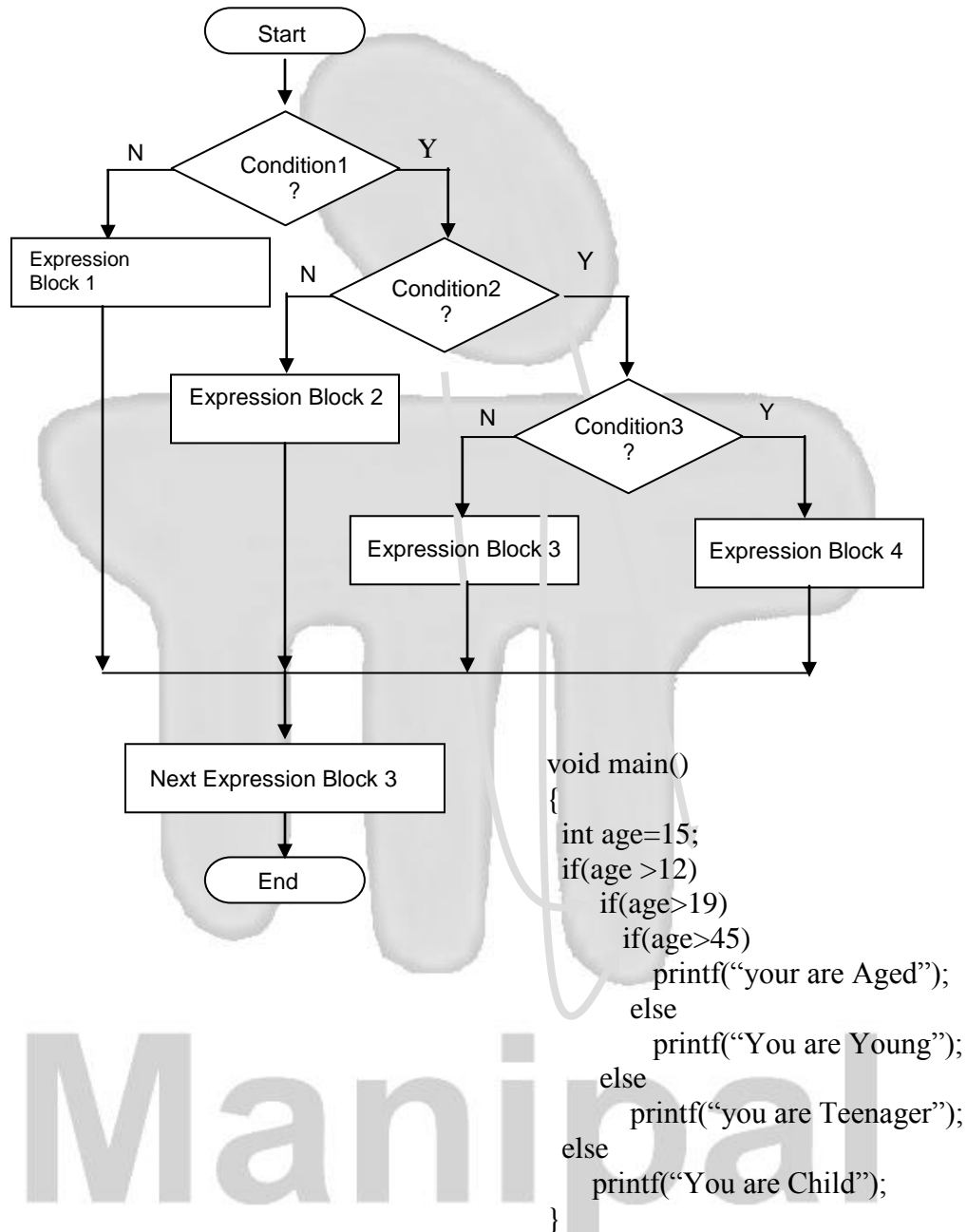


d) **Nested if statement:** The nested if statement is the simultaneous multiple condition checking operation. It is almost opposite of compound statement.

```
if(<condition-1>
    if(<condition-2>
        if(<condition-3>
            {<expression block -1>; }
        else
            {<expression block -2>; }
    else
        {<expression block -3>; }
else
    { <expression block -n>; }
```

Manipal

The following flowchart illustrate the operation of a nested if statement.



3.5 Break Statement

A break statement lets you end an iterative (do, for, or while) statement or a switch statement and exit from it at any point other than the logical end. A break may only appear on one of these statements.

Syntax: break;

In an iterative statement, the break statement ends the loop and moves control to the next statement outside the loop. Within nested statements, the break statement ends only the smallest enclosing do, for, switch, or while statement.

In a switch statement, the break passes control out of the switch body to the next statement outside the switch statement.

3.6 Continue Statement

A continue statement ends the current iteration of a loop. Program control is passed from the continue statement to the end of the loop body.

Syntax: continue;

A continue statement can only appear within the body of an iterative statement.

The continue statement ends the processing of the action part of an iterative (do, for, or while) statement and moves control to the loop continuation portion of the statement. For example, if the iterative statement is a for statement, control moves to the third expression in the condition part of the statement, then to the second expression (the test) in the condition part of the statement.

Within nested statements, the continue statement ends only the current iteration of the do, for, or while statement immediately enclosing it.

/* This example shows a continue statement in a for statement.*/

```
#include <stdio.h>
#define SIZE 5

int main(void)
{
    int i;
    static float rates[SIZE] = { 0.45, 0.95, 1.70, 4.00, 0.50 };

    printf("Rates over 1.00\n");
    for (i = 0; i < SIZE; i++)
    {
        if (rates[i] <= 1.00) /* skip rates <= 1.00 */
            continue;
        printf("rate = %.2f\n", rates[i]);
    }

    return(0);
}
```

The program produces the following output:

```
Rates over 1.00
rate = 1.70
rate = 4.00
```

The following example shows a continue statement in a nested loop. When the inner loop encounters a number in the array strings, that iteration of the loop ends. Processing continues with the third expression of the inner loop. The inner loop ends when the '\0' escape sequence is encountered.

/* This program counts the characters in strings that are part of an array of pointers to characters. The count excludes the digits 0 through 9.*/


```
#include <stdio.h>
#define SIZE 3

int main(void)
{
    static char *strings[SIZE] = { "ab", "c5d", "e5" };
    int i;
    int letter_count = 0;
    char *pointer;
    for (i = 0; i < SIZE; i++)          /* for each string */
        for (pointer = strings[i]; *pointer != '\0'; ++pointer)
            /* for each character */
            {
                /* if a number */
                if (*pointer >= '0' && *pointer <= '9')
                    continue;
                letter_count++;
            }
    printf("letter count = %d\n", letter_count);

    return(0);
}
```

The program produces the following output:

letter count = 5

3.7 Switch Statement

The switch and case statements help control complex conditional and branching operations. The switch statement transfers control to a statement within its body.

Control passes to the statement whose case constant-expression matches the value of switch (expression). The switch statement can include any

number of case instances, but no two case constants within the same switch statement can have the same value. Execution of the statement body begins at the selected statement and proceeds until the end of the body or until a break statement transfers control out of the body.

Use of the switch statement usually looks something like this:

```
switch ( expression )  
{  
    declarations  
    .  
    .  
    .  
    case constant-expression :  
        statements executed if the expression equals the  
        value of this constant-expression  
    .  
    .  
    .  
    break;  
    default :  
        statements executed if expression does not equal  
        any case constant-expression  
}
```

You can use the break statement to end processing of a particular case within the switch statement and to branch to the end of the switch statement. Without break, the program continues to the next case, executing the statements until a break or the end of the statement is reached. In some situations, this continuation may be desirable.

The default statement is executed if no case constant-expression is equal to the value of switch (expression). If the default statement is omitted, and no case match is found, none of the statements in the switch body are

executed. There can be at most one default statement. The default statement need not come at the end; it can appear anywhere in the body of the switch statement. A case or default label can only appear inside a switch statement.

The type of switch expression and case constant-expression must be integral. The value of each case constant-expression must be unique within the statement body.

The case and default labels of the switch statement body are significant only in the initial test that determines where execution starts in the statement body. Switch statements can be nested. Any static variables are initialized before executing into any switch statements.

Useful tips about the usage of switch:

- a) The expression must be either integer or character type.
- b) If the constant-expression is character type, it must be enclosed with ' '.
- c) constant-expressions may be arranged in order.
- d) The constant-expressions may be a combination of integer and character data types.
- e) For multiple statements in a single case, enclose of braces is not necessary.
- f) The break statement is necessary for all case unless the all cases will be executed from matched case.
- g) No condition is allowed in expression and constant-expressions.
- h) The switch statement may be used to create menu application.

Example:

```
void main()  
{  
    float a,b,c;  
    char ch;
```

```
printf("M A I N   M E N U\n");
printf("A.....Addition\n");
printf("D.....Division\n");
printf("M.....Multiplication\n");
printf("Q.....Quotient\n");
printf("-----\n");
printf("Choice Please ... : "); ch=getche();
printf("Enter 1st Number "); scanf("%f",&a);
printf("Enter 2nd Number "); scanf("%f",&b);
```

```
switch(ch)
{
case 'a': /* Addition */
case 'A':
    c=a+b; break;
case 'd': /* difference */
case 'D':
    c=a/b; break;
case 'm': /* multiplication */
case 'M':
    c=a*b; break;
case 'q': /* quotient */
case 'Q':
    c=floor(a/b); break;
default:
    printf("Invalid Input "); getch(); exit();
} /* end of switch */
printf("The Result is = %f",c);
}
```

3.8 Goto Statement

The goto statement transfers control to a label. The given label must reside in the same function and can appear before only one statement in the same function.

Syntax

```
statement:  
    labeled-statement  
    jump-statement  
jump-statement:  
    goto identifier ;  
labeled-statement:  
    identifier : statement
```

A statement label is meaningful only to a goto statement; in any other context, a labeled statement is executed without regard to the label.

A jump-statement must reside in the same function and can appear before only one statement in the same function. The set of identifier names following a goto has its own name space so the names do not interfere with other identifiers. Labels cannot be redeclared. See Name Spaces for more information.

It is good programming style to use the break, continue, and return statement in preference to goto whenever possible. Since the break statement only exits from one level of the loop, a goto may be necessary for exiting a loop from within a deeply nested loop.

The following example demonstrates the goto statement:

Example:

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

```
int i, j;

for ( i = 0; i < 10; i++ )
{
    printf_s( "Outer loop executing. i = %d\n", i );
    for ( j = 0; j < 3; j++ )
    {
        printf_s( " Inner loop executing. j = %d\n", j );
        if ( i == 5 )
            goto stop;
    }
}
/* This message does not print: */
printf_s( "Loop exited. i = %d\n", i );
stop: printf_s( "Jumped to stop. i = %d\n", i );
}
```

In this example, a goto statement transfers control to the point labeled stop when i equals 5.

3.9 Iterative Statements

Iterative statements perform the same set of statements zero or more time depends on the condition. are repetition of job. When we want to do the same type of job in a finite or infinite number of times then we take the help of Iterative statements. There are three different types of Iterative statements are used in C. They are

- i) For statement ii) while statement iii) do-while statement

i) for statement

Syntax:

```
for([<initialization>] ; [<condition>] ; [<increment>])
{
```

```
< Body of the Loop>;  
}
```

The **<initialization>** is optional because we can initialize before the for ().

The **<condition>** is optional because we can write any condition within the for ().

The **<increment>** is optional because we can write the increment statement within the for()

We can initialize multiple variables but cannot define multiple condition, if defined, it will execute the first one within the single for.

Example: Print first 20 natural numbers

```
void main()  
{  
    int i;  
    for(i=1;i<=20;i++)  
    {  
        printf("%d ",i);  
    }  
}
```

Output:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

ii) while statement

It is another looping statement which checks the condition first and then executes the body of the loop. The body of the loop may executed depends on the condition. If the condition is false during first check itself, the body of the loop is executed zero time.

Syntax:

```
<initialization of loop counter>;  
While (condition)  
{  
    <body of the loop>;  
    <Increment>;  
}
```

Example: Print first 20 natural numbers

```
void main()  
{  
    int i=1;  
    while(i<=20)  
    {  
        printf("%d ",i);  
        i=i+1;  
    }  
}
```

Output:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

iii) do-while statement

It is the third type of iterative statement where the body of the loop will be executed first and then checks the condition for further execution. So, the body of the loop will be executed atleast once.

Syntax:

```
do  
{  
    <body of the loop>;  
    <increment>;  
} while(<condition>;)
```


Example: Print first 20 natural numbers

```
void main()
{
    int i=1;
    do
    {
        printf("%d ",i);
        i=i+1;
    }while(i<=20);
}
```

Output:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

3.11 Examples

Q1: Write a program to print all even numbers between 100 to 1 [File Name: u3q1.c]

Answer:

```
void main()
{
    int i;
    clrscr();
    printf("List of Even Numbers between 1 to 100\n");
    printf("=====\n");
    for(i=100;i>=0;i-=2)
    {
        printf("%d ",i);
    }
}
```

Q2: Write a program to print all prime numbers between 1 to 100 [File Name : u3q2.c]

Answer:

```
/* Program for prime numbers */
```

```
void main()
```

```
{
```

```
    int i,j,ount;
```

```
    clrscr();
```

```
    for(i=2;i<=100;i++)
```

```
    {
```

```
        count=0;
```

```
        for(j=1;j<=i;j++)
```

```
        {
```

```
            if(i%j==0)
```

```
                count=count+1;
```

```
        }
```

```
        if(count==2)
```

```
            printf("%d ",i);
```

```
    }
```

```
    getch();
```

```
}
```

Q3: Write a program to check the leap year

[File Name : u3q3.c]

Answer:

```
void main()
```

```
{
```

```
int n;
```

```
clrscr();
```

```
printf("Enter year : "); scanf("%d",&n);
```

```
if((y%4==0 && y%100 !=0 || y%400==0)==1)
    printf("Yes! Leap Year ");
else
    printf("No! Not Leap Year ");
getch();
}
```

3.12 Summary

Language C is very sensitive and interesting language. Like natural language, when we want to learn a computer programming language we must follow some structure or procedure. There are some tools through which we can write programs, making decisions, looping and branching. To run the program we required compiler. There is a description of Turbo C 3.0 compiler, through which we can write and execute program. The detail steps are given. This unit covers input output functions, if structures, switch - case statements and the detail explanation about how to write program and run in C compilers.

Self Assessment Questions

1. scanf() is an output function. (True / False)
2. Turbo C is a Compiler. (True / False)
3. Decision making is done by if statements. (True / False)
4. Break statement terminates the program execution.
(True / False)
5. Break statement is compulsory in switch-case statement.
(True / False)

3.13 Terminal Questions

1. Justify the requirement for different types of iterative statements.
2. Why should we want to avoid goto statement?
3. What is the difference between switch statement and nested if statement?
4. Write C programs for the following:
 - a) Write a program to add all numbers between 100 and 200 which contain the digit 5.
 - b) Write a program that reverses of given number.
 - c) Write a program to find the sum of the digits in a given number.
 - d) Write a program which prints all Fibonacci numbers between m and n where m and n are given input.
 - e) Write a program that displays all perfect numbers.

3.14 Answer to Self Assessment and Terminal Questions

Self Assessment Questions

1. False
2. True
3. True
4. False
5. False

Terminal Questions

1. Section 3.9
2. Section 3.8
3. Sections 3. And 3.7