DECISION MAKING

INTRODUCTION

Control instructions in c

Control instructions specify the various instructions in a program are to be executed by the computer. Simply, control instructions determine the flow of control in a program.

There are four types of control instructions in C. They are:

(a) Sequence Control Instruction.

(b) Selection or Decision Control Instruction.

(c) Repetition or Loop Control Instruction.

(d) Case Control Instruction.

- The Sequence control instruction ensures that the instructions are executed in the same order in which they appear in the program.
- Decision and Case control instructions allow the computer to take a decision as to which instruction is to be executed next.
- The Loop control instruction helps computer to execute a group of statements repeatedly.

Decision Control Instruction

If you want to have a situation like set of instructions to be executed in one situation, and an entirely different set of instructions to be executed in another situation. This kind of situation is dealt in C programs using a decision control instruction. As mentioned earlier, a decision control instruction can be implemented in C using:

- The **if** statement.
- The **if-else** statement.
- The conditional operators.

Topic 1 if statement

Syntax

```
if (condition)
{
        Statements;
}
```

If key word is used to implement the decision control statements, the general form of the if statement is,

```
if(condition)
    execute statement ;
```

If statement is formed by using assignment and the relational operators. The result of using these operators is always simply the observation of true or false.

Expression	Is true if
X==Y	X is equal to Y
X!=Y	X is not equal Y
X < Y	X less than Y
X >Y	X greater than Y
X<=Y	X is less than or equal to Y
X>=Y	X is greater than or equal to Y

We express a condition using C's 'relational' operators. The relational operators allow us to compare two values to see whether they are equal to each other, unequal, or whether one is greater than the other.

The relational operators should be familiar to you except for the equality operator == and the inequality operator! =. Note that = is used for assignment, whereas, == is used for comparison of two quantities.

An example which demonstrates the **if statement,**

Example

```
main()
{
   int i;
   printf ("enter the value of variable I");
    scanf("%d",&i);
     if(i<10)
            printf ( "Welcome") ;
}
```

On execution of this program, if you type a number less than 10, you get a message on the screen through **printf( )**. If you type some other number the program doesn't do anything.

Flow chart

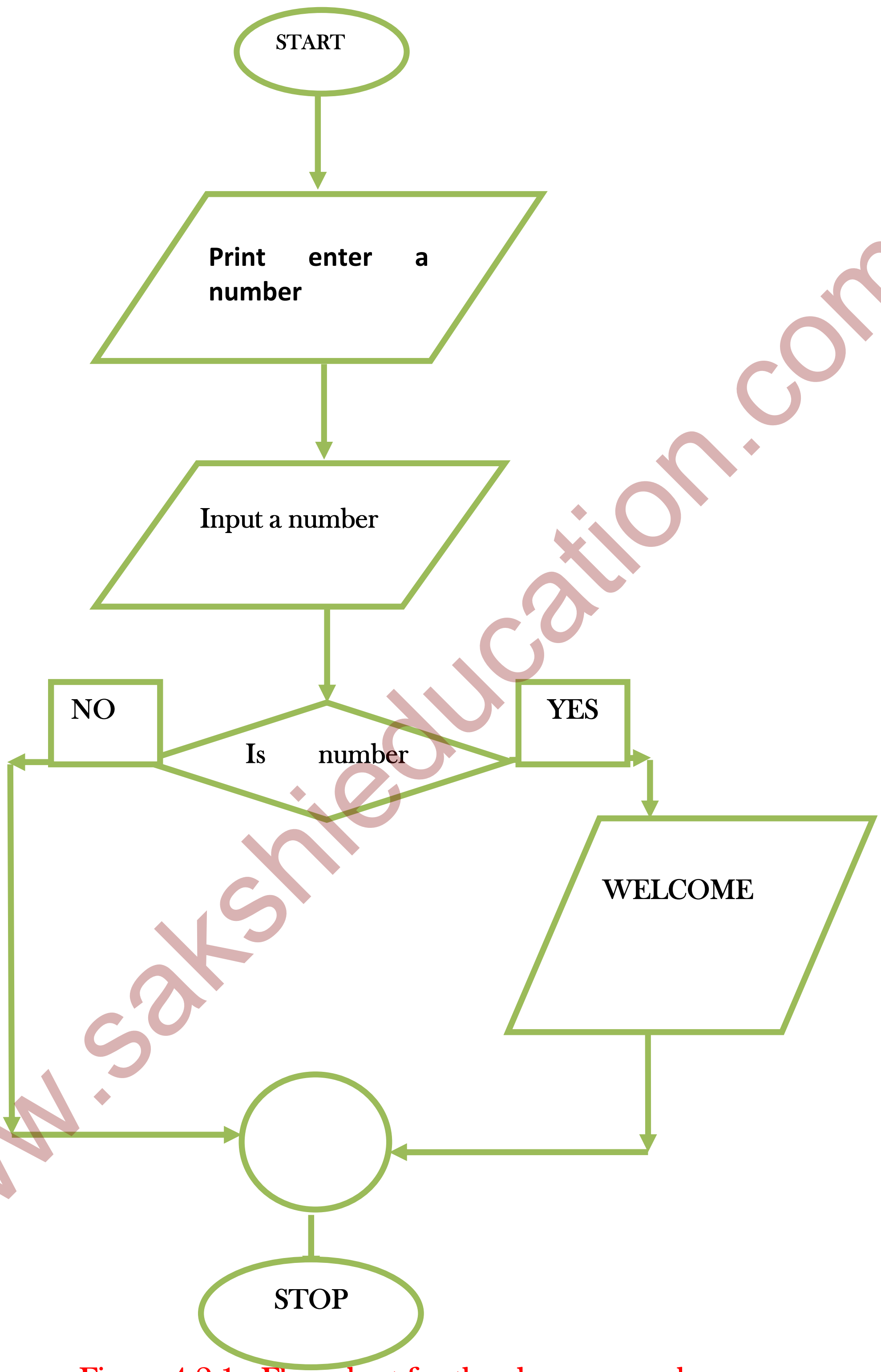

Figure 4.2.1:- Flow chart for the above example

Here the expression can be any valid expression including a relational expression. We can even use arithmetic expressions in the **if** statement. For example all the following **if** statements are valid

```
if ( 3 + 2 % 5 )
        printf ( "This works" ) ;
if ( a = 10 )
        printf ( "Even this works" ) ;
if ( -5 )
        printf ( "Surprisingly even this works" ) ;
```

Note that in C a non-zero value is considered to be true, whereas a 0 is considered to be false. In the first **if**, the expression evaluates to *5* and since *5* is non-zero it is considered to be true. Hence the **printf( )** gets executed.

In the second **if**, 10 gets assigned to **a** so the **if** is now reduced to **if (a)** or **if (10)**. Since 10 is non-zero, it is true hence again **printf( )** goes to work.

In the third **if**, -5 is a non-zero number, hence true. So again **printf( )** goes to work. In place of -5 even if a float like 3.14 were used it would be considered to be true. So the issue is not whether the number is integer or float, or whether it is positive or negative. Issue is whether it is zero or non-zero.

Multiple statements within if

Do you think that there can be multiple statements within if statement?

Yes, there is a chance of having multiple statements within if statement. In this case the statements are placed within a pair of braces.

Example: - Incrementing i variable value.

```
main()
{       //start of main program
        int i;  //declaration of i variable
        printf("enter the value of i");
        scanf("%d",&i);     //input the i value
        if(     i>0    )     //if condition
        {       //start of if condition
                i = i +1;           //incrementing the i value
                printf("the i value after incrementing is %d",i);       //printing    the    i
value as output
        }       //end of if condition
}       // end of the program
```

Observe that here the two statements to be executed on satisfaction of the condition have been enclosed within a pair of braces. If a pair of braces is not used then the C compiler assumes that the programmer wants only the immediately next statement after the **if** to be executed on satisfaction of the condition. In other words we can say that the default scope of the **if** statement is the immediately next statement after it.

Topic 2 if – else Statement

Syntax

```
if(condition)
{
        True statements;
}
else
{
        False statements;
}
```

'If' statement by itself will execute a single statement, or a group of statements, when the expression following **if** evaluates to true. It does nothing when the expression evaluates to false. Can we execute one group of statements if the expression evaluates to true and another group of statements if the expression evaluates to false? Of course! This is what the purpose of the else is statement that is demonstrated in the following example:

Example

```
#include <stdio.h>

int main ()
{
  /* local variable definition */
```

```
   int a = 100;

   /* check the boolean condition */
   if( a < 20 )
   {
      /* if condition is true then print the following */
      printf("a is less than 20\n" );
   }
   else
   {
      /* if condition is false then print the following */
      printf("a is not less than 20\n" );
   }
   printf ("value of a is : %d\n", a);

   return 0;
}
```

Output :-

```
a is not less than 20;
value of a is : 100
```

Topic 3 Nested if-else

If we write an entire **if-else** construct within either the body of the **if** statement or the body of an **else** statement. This is called 'nesting' of **ifs**.

Syntax

```
if(condition)
{
          if(condition)
          {
                    Statements;
          }
          else
          {
                    statements;
          }
}
else
{
          statements;
}
```

In above syntax, the condition is checked first. If it is true, then the program control flow goes inside the braces and again checks the next condition. If it is true then it executes the block of statements associated with it else executes else part.

Example

```
#include <stdio.h>

#include <conio.h>

void main()

{
        int   no;
        clrscr();
        printf ("\n Enter Number :");
        scanf ("%d",&no);
        if(no>0)
        {
                printf ("\n\n Number is greater than 0 !");
        }
        else
        {
                if(no==0)
                {
                        printf("\n\n It is 0 !");
                }
                else
                {
                    printf("Number is less than 0 !");
                }
            }
            getch();
}
```

Forms of if statements

a. if(condition)

```
   Statement;
```

b. if (condition)

```
   {
       Statement1;
       Statement2;
                -
                -
       Statement n;
   }
```

c. If (condition)

```
   {
       Do this;
   }
   else
   {
       Do this
```

```
}
```

d.
```
if ( condition ) {

    do this ;


    and this ;
 }
 else
 {
    do this ;
    and this ;
 }
```

e.
```
 if ( condition ) do this ;
else
{
    if ( condition )
            do this ;
    else
    {
            do this ;
            and this ;
    }
 }
```

f.
```
if ( condition )
{
    if ( condition )
```

```
                do this ;
            else
            {
                do this ;
                and this ;
            }
        }
        else
        do this ;
```

Note: - Keyword break is not syntactical part of if-else statement. So we cannot use break keyword in if-else statement. This keyword can be use in case of loop or switch case statement.

Note: - It is bad programming practice to write constant as a condition in if clause. Hence compiler will show a warning message: Condition is always true. Since condition is always true, so else clause will never execute. Program control cannot reach at else part. So compiler will show another warning message: **Unreachable code.**

Note: - In case of **if – if else – if else …** Statement if first if clause is true the compiler will never check rest of the if else clause and so on.

Topic 4 Conditional Operators

Introduction

Ternary operators come under the conditional operators. **: and?** Together forms the **ternary operator** are ternary operators. It is called ternary operators because it takes three arguments. These conditional operators evaluate an expression. So if it is true it returns a result 1 and else if it is false then returns result 2.

Syntax:-

Condition? result1: result2;

Example

10==5? 11: 12 // returns 12, since 10 not equal to 5.

CASE CONTROL STRUCTURE

Decisions using switch

Introduction

What is a switch in c language actually?

The control statement that allows us to make a decision from the number of choices is called a **switch.**

- There are three keyword “switch, case and default” makes up the control statement.

Syntax:-

```
switch ( integer expression )
{
        case constant 1 :
        do this ;
        case constant 2 :
        do this ;
        case constant 3 :
        do this ;
        default :
        do this ;
}
```

- The integer expression following the keyword switch is any C expression that will yield an integer value. It could be an integer constant like 1, 2 or 3, or an expression that evaluates to an integer.
- The keyword case is followed by an integer or a character constant. Each constant in each case must be different from all the others.
- Do this statements are simple valid statements.

What happens when switch programs runs?

1. The expression followed by the switch keyword is evaluated.
2. The value that gives from the result of the evaluated expression is then matched ,one by one, against the constant values that follow the case key word.
3. When a match is found, the program executes the statements following that case, and all subsequent case and default statements as well.
4. If no match is found with any of the case statements, only the statements following the default are executed.

Example

```
main( )
{
        int i = 2 ;
        switch ( i )
        {
                case 1 :
                        printf ( "I am in case 1 \n" ) ;
                case 2 :
```

```
                printf ( "I am in case 2 \n" ) ;
        case 3 :
                printf ( "I am in case 3 \n" ) ;
        default :
                printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

```
I am in case 2
I am in case 3
I am in default
```

The program prints case 2 and 3 and the default case. Well, yes. We said the switch executes the case where a match is found and all the subsequent cases and the default as well.

If you want that only case 2 should get executed, it is up to you to get out of the switch then and there by using a break statement.

Topic 2 Switch – Break

Switch executes the case where a match is found and all the subsequent **cases** and the **default** as well. If you want that only case 2 should get executed, it is up to you to get out of the **switch** then and there by using a **break** statement. The following example shows how this is done.

Note

There is no need for a **break** statement after the **default**, since the control comes out of the **switch** anyway.

```
main( )
{
        int  i = 2 ;
        switch ( i ) //switch selection statement
        {
                case 1 :
                        printf ( "I am in case 1 \n" ) ;
                        break ;
                case 2 :
                        printf ( "I am in case 2 \n" ) ;
                        break ;
                case 3 :
                        printf ( "I am in case 3 \n" ) ;
                        break ;
                default:
                        printf ( "I am in default \n" ) ;
        }
}
```

The output of this program would be:

```
I am in case 2
```

Note 1

The earlier program that used **switch** may give you the wrong impression that you can use only cases arranged in ascending order, 1, 2, 3 and default. You can in fact put the cases in any order you please. Here is an example of scrambled case order:

```
main( ) {
    int i = 22 ; //declaration and initialization of i variable
    switch ( i )
    {
        case 121 :
           printf ( "I am in case 121 \n" ) ;
           break ;
        case 7 :
           printf ( "I am in case 7 \n" ) ;
           break ;
        case 22 :
           printf ( "I am in case 22 \n" ) ;
           break ;
        default :
           printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

```
I am in case 22
```

Note 2

You are also allowed to use **char** values in **case** and **switch** as shown in the following program:

```
main( )
{
    char c = 'x' ;
    switch ( c )
    {
            case 'v' :
                    printf ( "I am in case v \n" ) ;
                    break ;
            case 'a' :
                    printf ( "I am in case a \n" ) ;
                    break ;
            case 'x' :
                    printf ( "I am in case x \n" ) ;
                    break ;
            default :
                    printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

```
I am in case x
```

In fact here when we use ‘v’, ‘a’, ‘x’ they are actually replaced by the ASCII values (118, 97, and 120) of these character constants.

Note 3

At times we may want to execute a common set of statements for multiple **cases**. How this can be done is shown in the following example.

```
main( )
{
    char ch ;
    printf ( "Enter any of the alphabet a, b, or c " ) ;
    scanf ( "%c", &ch ) ;
    switch ( ch )
    {
            case 'a' :
            case 'A' :
                    printf ( "a as in ashar" ) ;
                    break ;
            case 'b' :
            case 'B' :
                    printf ( "b as in brain" ) ;
                    break ;
            case 'c' :
            case 'C' :
                    printf ( "c as in cookie" ) ;
                    break ;
                    default :
                            printf ( "wish you knew what are alphabets" ) ;
    }
}
```

Here, we are making use of the fact that once a **case** is satisfied the control simply falls through the **case** till it doesn't encounter a **break** statement. That is why if an alphabet **a** is entered the **case 'a'** is satisfied and since there are no statements to be executed in this **case** the control automatically reaches the next **case** i.e. **case 'A'** and executes all the statements in this **case.**

Note 4

Even if there are multiple statements to be executed in each **case** there is no need to enclose them within a pair of braces (unlike **if**, and **else**).

Note 5

Every statement in a **switch** must belong to some **case** or the other. If a statement doesn't belong to any **case** the compiler won't report an error. However, the statement would never get executed. For example, in the following program the **printf( )** never goes to work.

```
main ( )
{
   int i, j ;
   printf ( "Enter value of i" ) ;
   scanf ( "%d", &i ) ;
   switch ( i )
   {
          printf ( "Hello" ) ;
          case 1 :
                 j = 10 ;
                 break ;
          case 2 :
                 j = 20 ;
                 break ;
   }
}
```

Note 6

If we have no **default** case, then the program simply falls through the entire **switch** and continues with the next instruction (if any,) that follows the closing brace of **switch.**

Disadvantage of SWITCH over IF statement

The disadvantage of **switch** is that one cannot have a case in a **switch** which looks like:

Case i <= 20:

All that we can have after the case is an **int** constant or a **char** constant or an expression that evaluates to one of these constants. **Even a float is not allowed.**

Advantage of SWITCH over IF statement

The advantage of **switch** over **if** is that it leads to a more structured program and the level of indentation is manageable, more so if there are multiple statements within each **case** of a **switch.**

Note 7:-

We can check the value of any expression in a **switch.** Thus the following **switch** statements are legal.

```
switch ( i + j * k )
switch ( 23 + 45 % 4 * k )
switch ( a < 4 && b > 7 )
```

Expressions can also be used in cases provided they are constant expressions. Thus **case** 3 + 7 is correct, however, **case a** + **b** is incorrect.

Note 8:-

The **break** statement when used in a **switch** takes the control outside the **switch.** However, use of **continue** will not take the control to the beginning of **switch** as one is likely to believe.

Note 9:-

The **switch** statement is very useful while writing menu driven programs.

There are something's that you cannot do with the switch (disadvantages of switch)

- A float expression cannot be tested using a switch.
- Cases can never have variable expressions (for example it is wrong to say case a +3 :) .
- Multiple cases cannot use same expressions.

A switch with 10 cases would work faster than an equivalent if-else ladder.

A switch with 2 cases would work slower than if-else ladder.