Course: Problem Solving Techniques using C

Course Code: 24BTPHY104

Module 2

2. 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

2.1 Unformatted I/O statements

getchar( ):- It reads single character from standard input device. This function donot 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 charǁ);
ch = getchar( );
printf(—entered char
isǁ); 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 a lower-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
charǁ); a = getchar();
if (isupper(a))
{
x= tolower(a);
putchar(x);
}
else
putchar(toupper(a));
}
```

Output: - enter char A
a

2.2 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 %fll,&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(—%sll,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 stringll, 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.

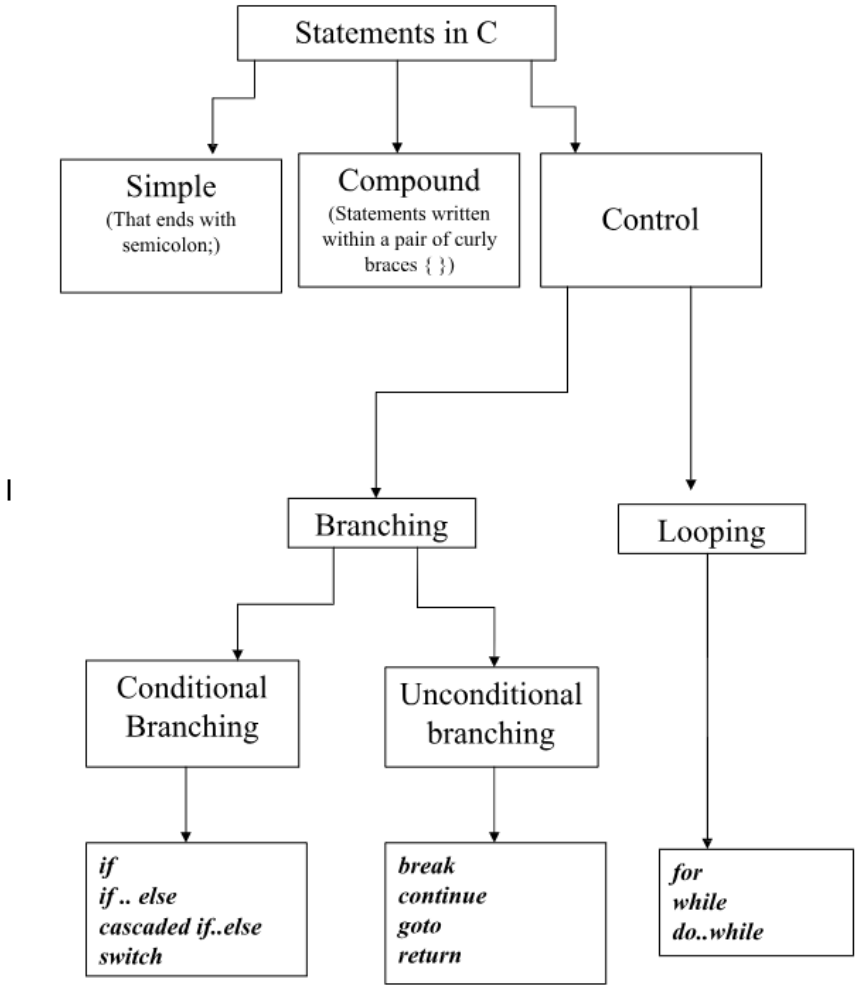
Format **y=98.7654**	**O/P**
printf(—%7.4f‖,y)	9 8 . 7 6 5 4
printf(—%7.2f‖,y)	9 8 . 7 7
printf(—%-7.2f‖,y)	9 8 . 7 7
printf(—%10.2e‖,y)	9 . 8 8 e + 0 1

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

Decision making, Branching and Looping



2.3 Decision Making Statements

If and Switch Statements

We have a number of situations where we may have to change the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met.

2.3.1The if Statement

This is basically a “one-way” decision statement. This is used when we have only one

alternative.

The syntax is shown below:

```
if(expression)
{
        statement1;
}
```

Firstly, the expression is evaluated to *true* or *false*.

If the expression is evaluated to *true*, then statement1 is executed. If the expression is evaluated to *false*, then statement1 is skipped. The flow diagram is shown below:

Example 1: Program to illustrate the use of if statement

```
#include<stdio.h>
main()
{
        int a,b;
        printf("Enter two numbers");
        scanf("%d%d",&a,&b):
        if a>b
                printf(" a is greater");
        if b>a
                printf("b is greater");
}
```

Example 2: Program to illustrate the use of if statement.

```
#include<stdio.h>void
main()
{
        int n;
        printf("Enter any non-zero integer: \n");
        scanf("%d", &n)
        if(n>0)
                printf("Number is positive number");if(n<0)
                printf("Number is negative number");
}
```

Output:

```
Enter any non-zero integer:7
Number is positive number
```

2.3.2 The if else Statement

This is basically a “two-way” decision statement.

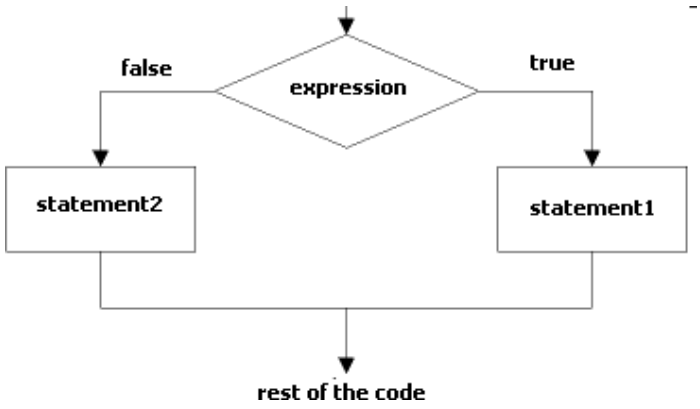
This is used when we must choose between two alternatives. The syntax is shown below:

```
if(expression)
{
        statement1;
}
else
{
        statement2;
}
```

Firstly, the expression is evaluated to true or false.

If the expression is evaluated to *true*, then statement1 is executed. If the expression is evaluated to *false*, then statement2 is executed.

The flow diagram is shown below:



Example: Program to illustrate the use of if else statement.

```
#include<stdio.h>
void main()
{
        int n;
        printf(“Enter any non-zero integer: \n”);
        scanf(“%d”, &n)
        if(n>0)
                printf(“Number is positive number”);
        else
```

```
}

printf("Num
ber is
negative
number")
```

Output:
Enter any non-zero integer:7
Number is positive number

Example 2: Program to illustrate if else statement to find greatest of two numbers.

```
#include<stdio.h>
main()
{
        int a,b;
        printf(“Enter two numbers”);
        scanf(“%d%d”,&a,&b):
        if a>b
                printf(“a is greater”)
        else
                printf(“b is greater”);
}
```

Example 3: program to check whether an integer is odd or even

```
#include <stdio.h>
 int main() {
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);

    // True if the remainder is 0
    if (number%2 == 0) {
        printf("%d is an even integer.",number);
    }
    else {
        printf("%d is an odd integer.",number);
    }

    return 0;
}
```

Output:

Enter an integer: 7
7 is an odd integer.

Example 4: Program to check vowel or consonant

```
#include <stdio.h>
int main() {
    char c;
    int lowercase_vowel, uppercase_vowel;
    printf("Enter an alphabet: ");
    scanf("%c", &c);

    // evaluates to 1 if variable c is a lowercase vowel
    lowercase_vowel = (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');

    // evaluates to 1 if variable c is a uppercase vowel
    uppercase_vowel = (c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U');

    // evaluates to 1 (true) if c is a vowel
    if (lowercase_vowel || uppercase_vowel)
        printf("%c is a vowel.", c);
    else
        printf("%c is a consonant.", c);
    return 0;
}
```

OUTPUT:

```
Enter an alphabet: G
G is a consonant.
```

Example 5: Program to Find the Largest Number Among Three Numbers

```
#include <stdio.h>

int main() {

  double n1, n2, n3;

  printf("Enter three different numbers: ");
  scanf("%lf %lf %lf", &n1, &n2, &n3);

  // if n1 is greater than both n2 and n3, n1 is the largest
  if (n1 >= n2 && n1 >= n3)
    printf("%.2f is the largest number.", n1);

  // if n2 is greater than both n1 and n3, n2 is the largest
  if (n2 >= n1 && n2 >= n3)
    printf("%.2f is the largest number.", n2);

  // if n3 is greater than both n1 and n2, n3 is the largest
    if (n3 >= n1 && n3 >= n2)
    printf("%.2f is the largest number.", n3);
```

```
  return 0;
}
```

OUTPUT:
Enter three different numbers: 4 2 7
7.00 is the largest number.

Example 6: Program to check whether the number is positive or negative

```
#include <stdio.h>

int main() {

    double num;
    printf("Enter a number: ");
    scanf("%lf", &num);
    if (num <= 0.0) {
        if (num == 0.0)
            printf("You entered 0.");
        else
            printf("You entered a negative number.");
    }
    else
        printf("You entered a positive number.");

    return 0;
}
```

Output:
Enter a number: 4
You entered a positive number.

Example 7: Program to find the factorial of n

```
#include <stdio.h>
int main() {
    int n, i;
    unsigned long long fact = 1;
    printf("Enter an integer: ");
 scanf("%d", &n);

    // shows error if the user enters a negative integer
    if (n < 0)
        printf("Error! Factorial of a negative number doesn't exist.");
    else {
        for (i = 1; i <= n; ++i) {
            fact *= i;
        }
```

```
        printf("Factorial of %d = %llu", n, fact);
    }

    return 0;
}
```

OUTPUT:
Enter an integer: 10
Factorial of 10 = 3628800

2.3.3 The nested if Statement

An if-else statement within another if-else statement is called nested if statement.

This is used when an action has to be performed based on many decisions. Hence, it is called as multi-way decision statement.

The syntax is shown below:

```
if(expr1)
{
if(expr2)
statement1
                    else
                            statement2
}
else
{

if(expr3)
statement3
else

}                   State
                    ment
                    4
```

• Here, firstly expr1 is evaluated to true or false.

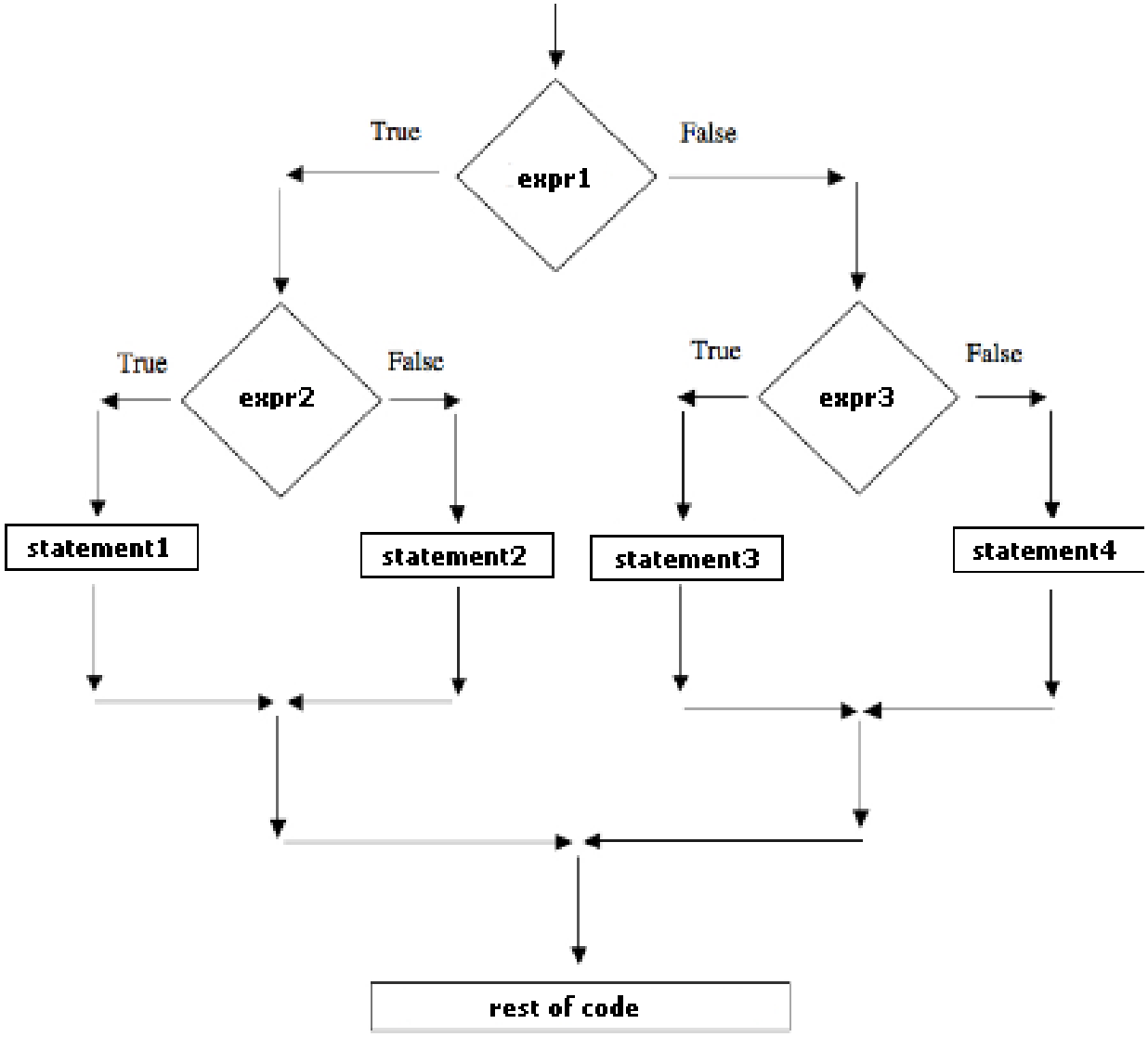
If the expr1 is evaluated to *true*, then expr2 is evaluated to true or false. If the expr2 is evaluated to *true*, then statement1 is executed.

If the expr2 is evaluated to *false*, then statement2 is executed.

If the expr1 is evaluated to *false*, then expr3 is evaluated to true or false. If the expr3 is evaluated to *true*, then statement3 is executed.

If the expr3 is evaluated to *false*, then statement4 is executed.

The flow diagram is shown below:



Example1: Program to select and print the largest of the 3 numbers using nested "if-else"statements.

```
#include<stdio.h>void main()
{
int a,b,c;
printf("Enter Three Values: \n");
scanf("%d %d %d ", &a, &b, &c);
printf("Largest Value is: ") ;
 if(a>b)
{
if(a>c)
printf(" %d ", a);
                                        else
                                                printf(" %d ", c);
}
else
{

if(b>c)
printf(" %d", b);
else
                        printf(
}                       " %d",
}                       c)
```

Example 2:

Problem Statement: Determine the grade of a student based on their score.

- **If the score is above 90, it's an 'A'.**
- **If the score is between 80 and 90, check if it's above 85 for an 'A-' or else 'B+'.**

  Below 80, if it's above 70, it's a 'C', otherwise, it's a 'D'.

```
#include  <stdio.h>
int main() {
    int score = 88;
    // Check if score is greater than 90
    if (score  >  90) {
        printf("Grade: A\n");
     } else {
     // Score is 90 or below, check if score is greater than 80
        if (score  >  80) {
// Score is between 81 and 90, check if score is above 85
            if (score  >  85) {
                printf("Grade: A-\n");
            } else {
                printf("Grade: B+\n");
            }
        } else {
            // Score is 80 or below, check if score is above 70
            if (score  >  70) {
                printf("Grade: C\n");
            } else {
                printf("Grade: D\n");
            }
        }
    }
    return 0;
}
```

2.3.4The Cascaded if-else statement (The else if ladder statement)

This is basically a “multi-way” decision statement.

This is used when we must choose among many

alternatives.The syntax is shown below:

```
if(expression1)
{
        statement1;
```

```
}
else if(expression2)
{
        statement2;
}
else if(expression3)
{
        statement3
}

else

   {

 Default statement
```

The expressions are evaluated in order (i.e. top to bottom).If an expression is evaluated to true, then

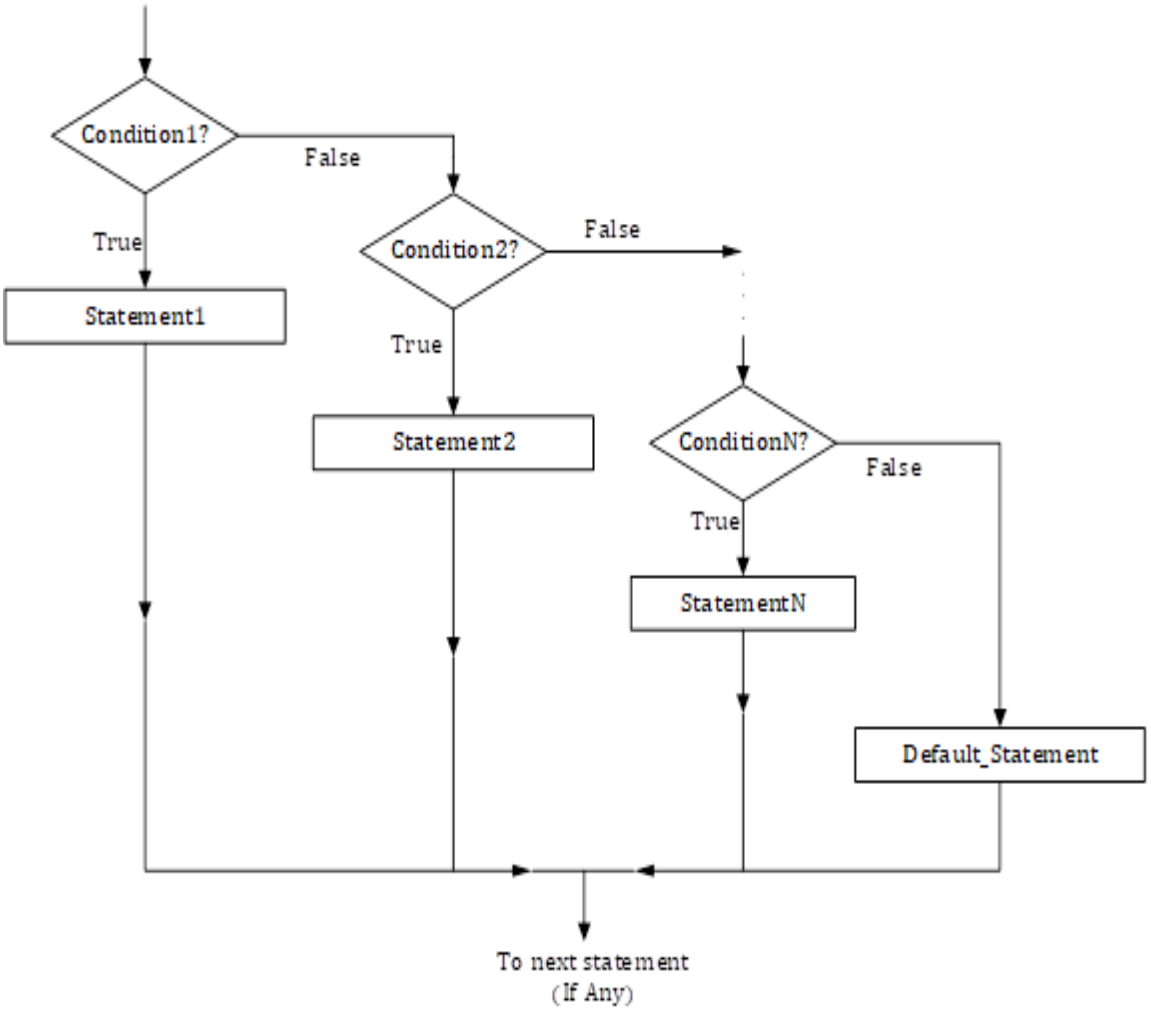
→ Statement associated with the expression is executed &

→ Control comes out of the entire else if ladder

For ex, if exprression1 is evaluated to *true*, then statement1 is executed.

If all the expressions are evaluated to false, the last statement4 (default case) is executed.

Flowchart of if-else-if Ladder statement



Example: Program to find largest from three numbers given by user to explain working of if-else-if statement or ladder.

```
#include<stdio.h>
int main()
{
        int a,b,c;
        printf("Enter three numbers: \n");
        scanf("%d%d%d", &a, &b, &c);
        if(a>b && a>c)
        {
         printf("Largest = %d", a);
        }
        else if(b>a && b>c)
        {
          printf("Largest = %d", b);
        }
```

```
        else
        {
         printf("Largest = %d", c);
        }
        return(0);
}
```

Output

```
Enter three numbers:
12
33
17
Largest = 3
```

2.3.5 The Switch Statement

This is a multi-branch statement similar to the if - else ladder (with limitations) but clearer and easier to code.

Syntax:

```
switch (expression)
{
        case constant1:     statement1;
                            break;

        case constant2:     statement2;
                            break;

        . . .

        default:            statement;
}
```

The value of expression is tested for equality against the values of each of the constants specifiedin the **case** statements in the order written until a match is found. The statements associated withthat case statement are then executed until a break statement or the end of the switch statement is encountered.

When a break statement is encountered execution jumps to the statement immediately followingthe switch statement.

The default section is optional -- if it is not included the default is that nothing happens and execution simply falls through the end of the switch statement.

The switch statement however is limited by the following

- Can only test for equality with **integer constants** in case statements.
- No two case statement constants may be the same.
- Character constants are automatically converted to integer.

Rules for switch statement in C language

- The *switch expression* must be of an integer or character type.
- The *case value* must be an integer or character constant.
- The *case value* can be used only inside the switch statement.
- The *break statement* in switch case is not must. It is optional.
- If there is no break statement found in the case, all the cases will be executed present after the matched case.
- It is known as *fall through* the state of C switch statement.

- We are assuming that there are following variables.

```
int x,y,z;
char a,b;
float f;
```

Valid Switch	Invalid Switch	Valid Case	Invalid Case
switch(x)	switch(f)	case 3;	case 2.5;
switch(x>y)	switch(x+2.5)	case 'a';	case x;
switch(a+b-2)		case 1+2;	case x+2;
switch(func(x,y))		case 'x'>'y';	case 1,2,3;

Example: - Program to simulate a basic calculator.

```
#include <stdio.h>void
main()
 {
    double num1, num2, result ;char
    op ;

    printf ( " Enter number operator number\n" ) ;scanf ("%f %c
    %f", &num1, &op, &num2 ) ; switch ( op )
    {
        case '+' :   result = num1 + num2 ;
                                    break ;
        case '-' :            result = num1 - num2 ;
                                    break ;
        case '*' :   result = num1 * num2 ;
                                    break ;
        case '/' :   if ( num2 != 0.0 ) {
                                        result = num1 / num2 ;break ;
                                        }
        // else we allow to fall through for error message

        default :   printf ("ERROR -- Invalid operation or division
                                        by 0.0" ) ;
    }
       printf( "%f %c %f = %f\n", num1, op, num2, result) ;
}
```

Note : The break statement need not be included at the end of the case statement body if it is logically correct for execution to fall through to the next case statement (as in the case of divisionby 0.0) or to the end of the switch statement (as in the case of default :).

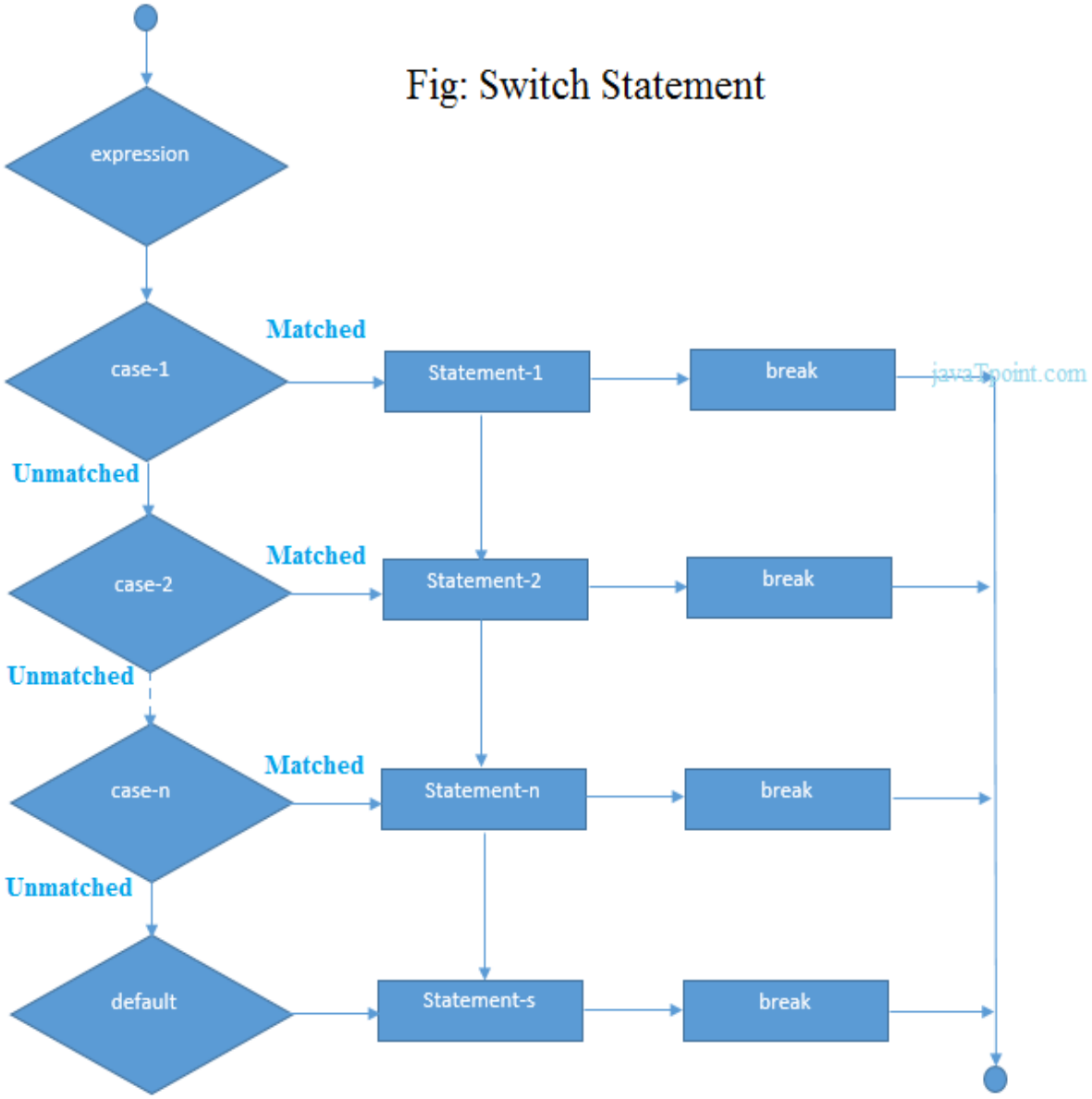


Fig: Switch Statement

/* program to implement switch */

```
#include<stdio.h>
main()
{
int marks,index;
char grade[10];
printf("Enter your marks");
scanf("%d",&marks);
```

```
index=marks/10;
switch(index)
{
case 10 :
case 9:
case 8:
case 7:
case 6: grade=”first”;
break;
case 5 : grade=”second”;
break;
case 4 : grade=”third”;
break;
default : grade =”fail”;
break;
}
printf(“%s”,grade);
}
```

2.4 Ternary operator

This is a special shorthand operator in C and replaces the following segment.

```
if (condition)
expr_1 ;
else
expr_2 ;
```

with the more elegant:

Syntax: *condition? expr_1: expr_2;*

The? operator is a ternary operator in that it requires three arguments. One of the advantages of the? operator is that it reduces simple conditions to one simple line of code which can be thrown unobtrusively into a larger section of code.

For Example: - to get the maximum of two integers, x and y, storing the larger in max.

```
max =   x >= y   ? x   :   y   ;
```

The alternative to this could be as follows

```
if ( x > = y   )
        max = x ;
else
        max = y ;
```

giving the same result but the former is a little bit more sufficient.

2.5 Looping structure in C

2.5.1 For statement

The for statement is most often used in situations where the programmer knows in advance how many times a particular set of statements are to be repeated. The for statement is sometimes termed a counted loop.

Syntax: **for ([initialisation] ; [condition] ; [increment])**
{
[statement body];
}

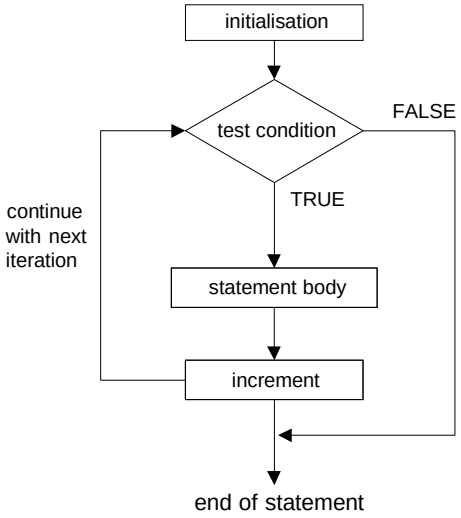
Initialisation :- this is usually an assignment to set a loop counter variable for example.

Condition :- determines when loop will terminate.

Increment :- defines how the loop control variable will change each time the loop is executed.

Statement body :- can be a single statement, no statement or a block of statements.

The *for* statement executes as follows :-



Note: The square braces above are to denote optional sections in the syntax but are not part of thesyntax. The semi-colons must be present in the syntax.

For Example: Program to print out all numbers from 1 to 100.

```
#include <stdio.h>void

main()
{
   int x ;

   for ( x = 1; x <= 100; x++ )printf( "%d\n", x ) ;
}
```

Curly braces are used in C to denote code blocks whether in a function as in main() or as the bodyof a loop.

For Example: - To print out all numbers from 1 to 100 and calculate their sum.

```
#include <stdio.h>void
main()
{
   int x, sum = 0 ;

   for ( x = 1; x <= 100; x++ )
   {
        printf( "%d\n", x ) ;sum +=
        x ;
   }
   printf( “\n\nSum is %d\n”, sum ) ;
   }
```

Multiple Initialization

C has a special operator called the **comma operator** which allows separate expressions to be tiedtogether into one statement.

For example, it may be tidier to initialize two variables in a for loop as follows: -

```
for (x = 0, sum = 0; x <= 100; x++)
        {
            printf("%d\n", x) ;sum += x
            ;
        }
```

Any of the four sections associated with a for loop may be omitted but the semi-colons must bepresent always.

For Example: -

```
for ( x = 0; x < 10;                )
        printf( "%d\n", x++ ) ;
...
x = 0 ;
for ( ; x < 10; x++ ) printf( "%d\n",
        x ) ;
```

An infinite loop may be created as follows

```
    for ( ;   ;   )
        statement body ;
```

or indeed by having a faulty terminating condition.

Sometimes for statement may not even have a body to execute as in the following examplewhere we just want to create a time delay.

```
for ( t = 0; t < big_num ; t++ )   ;
```

or we could rewrite the example given above as follows

```
for ( x = 1; x <= 100; printf( "%d\n", x++ ) ) ;
```

The initialization, condition and increment sections of the for statement can contain any valid Cexpressions.

```
for ( x = 12 * 4 ; x < 34 / 2 * 47 ; x += 10 )printf(
“%d “, x ) ;
```

It is possible to build a nested structure of for loops, for example the following creates a largetime delay using just integer variables.

```
unsigned int x, y;

for (x = 0; x < 65535; x++ )for ( y = 0; y < 65535;
    y++);
```

Example :

Program to produce the following table of values

```
#include  <stdio.h>void main()
{
    int j, k ;

    for ( j = 1; j <= 5; j++ )
    {
       for ( k = j ; k < j + 5; k++ )
         {
            printf( "%d    ", k ) ;
       }
       printf( "\n" ) ;
    }
}
```

```
1  2  3  4  5
2  3  4  5  6
3  4  5  6  7
4  5  6  7  8
5  6  7  8  9
```

C Program to Display Fibonacci Sequence

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

  int i, n;

  // initialize first and second terms
  int t1 = 0, t2 = 1;

  // initialize the next term (3rd term)
  int nextTerm = t1 + t2;
    // get no. of terms from user
    printf("Enter the number of terms: ");
  scanf("%d", &n);
```

```
  // print the first two terms t1 and t2
  printf("Fibonacci Series: %d, %d, ", t1, t2);

  // print 3rd to nth terms
  for (i = 3; i <= n; ++i) {
    printf("%d, ", nextTerm);
    t1 = t2;
    t2 = nextTerm;
    nextTerm = t1 + t2;
  }

  return 0;
}
```

Output

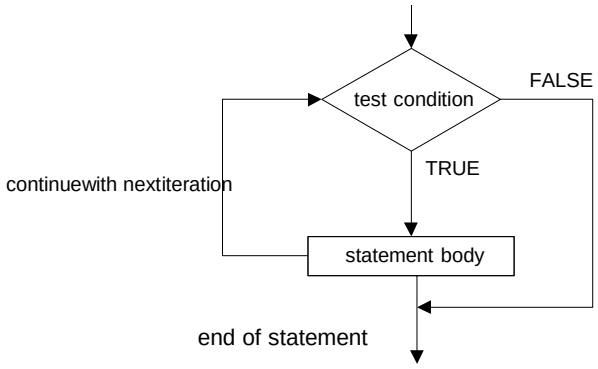
```
Enter the number of terms: 10
Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
```

2.5.2 While statement

The while statement is typically used in situations where it is not known in advance how many iterations are required.

Syntax**:**

```
while (condition)
{
        statement body;
}
```



Example 1: C program to print numbers from 1 to 5 using while loop

```
#include <stdio.h>
int main()
{
  int i = 1;
```

```
  while (i <= 5)
  {
    printf("%d\n", i);
    ++i;
  }
  return 0;
}
```

Output:

1

2

3

4

5

Example 2: Program to sum all integers from 100 down to 1.

```
#include  <stdio.h>void main()
{
    int sum = 0, i = 100 ;

    while ( i )
        sum += i-- ;// note the use of postfix decrement operator! printf (“Sum is %d \n”, sum);
}
```

where it should be recalled that any non-zero value is deemed TRUE in the condition section ofthe statement.

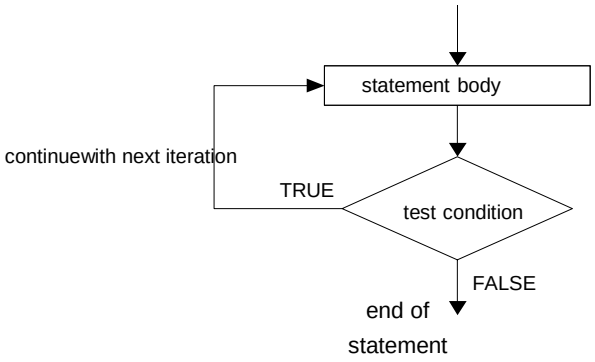
2.5.3 Do while

The terminating condition in the “for” and “while” loops are always tested before the body of the loopis executed -- so of course the body of the loop may not be executed at all.

In the ***do while*** statement on the other hand the statement body is always executed at leastonce as the condition is tested at the end of the body of the loop.

Syntax:

```
do
{
        statement body;
}  while (condition);
```



Example: To read in a number from the keyboard until a value in the range 1 to 10 is entered.

```
int i ;do
{
    scanf( "%d\n", &i ) ;
}       while ( i < 1 && i > 10 ) ;
```

In this case we know at least one number is required to be read so the do-while might be the natural choice over a normal while loop.

2.6 Break statement

When a break statement is encountered inside a while, for, do/while or switch statement the statement is immediately terminated, and execution resumes at the next statement following the statement.

For Example: -

```
                ...
for ( x = 1 ; x <= 10   ; x++   )
{
        if ( x > 4   )
                break;

        printf( “%d   “ , x ) ;
}
printf(   "Next   executed\n"   );//Output   :   “1                    2      3      4      NextExecuted”
```

2.7 Continue statement

The continue statement terminates the current iteration of a while, for or do/while statement and resumes execution back at the beginning of the loop body with the next iteration.

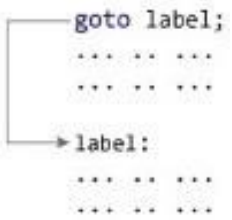
For Example: -

```
for ( x = 1; x <= 5; x++ )
{
        if ( x == 3 )
                continue ; printf( "%d ", x ) ;
}
printf( "Finished Loop\n" ) ; // Output : "1 2 4 5 Finished Loop"
```

2.8 Goto statement

goto statement can be used to branch unconditionally from one point to another in the program.The goto requires a label in order to identify the place where the branch is to be made. A label isany valid variable name and must be followed by a colon(:). The label is placed immediately before the statement where the control is to be transferred.

The syntax is shown below:



Example: Program to detect the entered number as to whether it is even or odd. Use goto statement.

```
#include<stdio.h>void main()
{
        int x;
        printf("Enter a Number: \n"); scanf("%d", &x);
                                        if(x % 2 = = 0)
                                            goto even;
        else
                goto odd;
  even:printf("%d is Even Number");return;
  odd:printf(" %d is Odd Number");
}
```

Output:

Enter a Number:5

5 is Odd Number.

2.9 C Programs

2.9.1 C Program to Print Pyramids and Patterns

Example 1: Half Pyramid of *

```
*
* *
* * *
* * * *
* * * * *
```

```
#include <stdio.h>
int main() {
   int i, j, rows;
   printf("Enter the number of rows: ");
   scanf("%d", &rows);
   for (i = 1; i <= rows; ++i) {
      for (j = 1; j <= i; ++j) {
         printf("* ");
      }
      printf("\n");
   }
   return 0;
}
```

Example 2: Half Pyramid of Numbers

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
#include <stdio.h>
int main() {
   int i, j, rows;
   printf("Enter the number of rows: ");
   scanf("%d", &rows);
   for (i = 1; i <= rows; ++i) {
      for (j = 1; j <= i; ++j) {
         printf("%d ", j);
      }
      printf("\n");
   }
   return 0;
}
```

Example 3: Inverted half pyramid of *

```
* * * * *
* * * *
* * *
* *
*
```

```
#include <stdio.h>
int main() {
   int i, j, rows;
   printf("Enter the number of rows: ");
   scanf("%d", &rows);
   for (i = rows; i >= 1; --i) {
      for (j = 1; j <= i; ++j) {
         printf("* ");
      }
      printf("\n");
   }
   return 0;
}
```

2.9.2 C Program to Check Whether a Number is Palindrome or Not

```
#include <stdio.h>
int main() {
  int n, reversed = 0, remainder, original;
    printf("Enter an integer: ");
    scanf("%d", &n);
    original = n;

    // reversed integer is stored in reversed variable
    while (n != 0) {
        remainder = n % 10;
        reversed = reversed * 10 + remainder;
        n /= 10;
    }

    // palindrome if orignal and reversed are equal
    if (original == reversed)
        printf("%d is a palindrome.", original);
    else
        printf("%d is not a palindrome.", original);

       return 0;
}
```

Output:

```
Enter an integer: 1001
1001 is a palindrome.
```

2.9.3 C Program to Calculate the Power of a Number

```
#include <stdio.h>
int main() {
    int base, exp;
    long double result = 1.0;
    printf("Enter a base number: ");
    scanf("%d", &base);
    printf("Enter an exponent: ");
    scanf("%d", &exp);

    while (exp != 0) {
        result *= base;
        --exp;
    }
    printf("Answer = %.0Lf", result);
    return 0;
}
```

OUTPUT:

```
Enter a base number: 3
Enter an exponent: 4
Answer = 81
```