UNIT- II
Program Control Constructs

Decision Making and Branching Statements

Decision making in C

Decision making is about deciding the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met. C language handles decision-making by supporting the following statements,

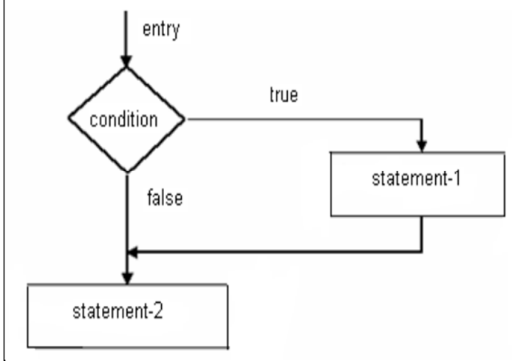
a. if statement
b. switch statement
c. conditional operator statement
d. goto statement

The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are,

- Simple if statement
- If....else statement
- Nested if....else statement
- else if statement

Simple if Statement

Simple if statement is used to execute some statements for a particular condition. The general form of if statement and flow chart is shown below:

Syntax	Flow chart
if (condition) { statement-1; } *statement -2;*	

Where condition is a relational or a logical expression. The condition must be placed in parentheses. Statement-1 can be either simple or compound statement (group of statements). The value of condition is evaluated first. The value may be either true or false. If the condition is true, the statement – 1 is executed and then the control is transferred to statement – 2. If the condition is false, the control is transferred directly to the statement-2 without executing the statement-1.

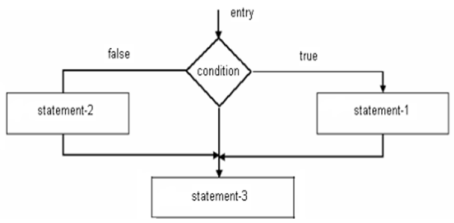
The conditional statement should not the terminated with Semi-colons (ie ;)

Example 1: Write a program to check equivalence of two numbers. Use "if" statement	*Example 2: Write a Program to check whether a given number is less than 20.*
# include<stdio.h> # include<conio.h> void main( ) { int m,n; clrscr( ); printf("\n Enter two numbers:"); scanf("%d %d", &m, &n); if(m= =n) printf(""\n two numbers are equal""); getch(); }	#include <stdio.h> # include<conio.h> void main () { int a; scanf("%d", &a); if(a < 20) { printf("a is less than 20\n"); } printf("value of a is : %d\n", a); }

if …. else Statement

The simple if statement will execute a single statement, or a group of statements, if the condition lis true. If the condition is false, it does nothing.

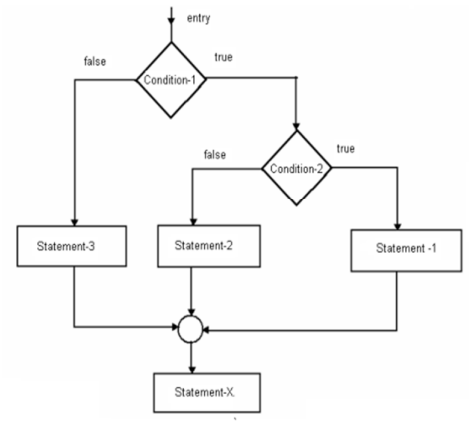
If …else statement is used to execute one group of statements if the condition is true. The other groups of statements are executed when the condition is false. General form of if…else statement and flow chart is shown below:

Syntax	Flow chart	Explanation
if (condition) { statement-1; } else { statement-2; } *statement -3;*		statement-1 and statement-2 can be simple or compound statements. If the condition is true, the statement-1 is executed; otherwise the statement-2 is executed. In either case, either statement-1 or statement-2 will be executed, not both. In both the cases, the control is transferred subsequently to statement-3.

Example 1: Write a program to print the given number is even or odd.	*Example2: Develop a program to accept two numbers and find largest number and print.*
# include<stdio.h> void main( ) { int n; printf(""Enter a number:""); scanf(""%d"", &n); if((n%2)==0) printf(""\n The given number is EVEN" "); else printf(""\n The given number is ODD" "); }	# include<stdio.h> void main( ) { int a,b; printf("Enter Two numbers:"); scanf("%d%d", &a,&b); if(a>b) printf(""\n %d is largest number"",a); else printf(""\n %d is largest number"",b); }

Nested ifs

A nested if statement is an if statement which is within another if – block or else – block. If an if…else is contained completely within another construct, then it is called nesting of if's.

Syntax	Flow chart
.if (condition-1) { if (condition-2) { statement-1; } else { statement-2; } else { statement -3; } } Statement – 4;	

If the condition-1 is false, the statement-3 will be executed. Otherwise it continues to test condition- 2. If the condition-2 is true, the statement-1 will be executed; otherwise the statement-2 will be executed and then the control is transferred to statement-4. Second if…else construction may be nested in the if block or else block of the first if…else construction. The statements between the keywords if and else are called if block and statements after the else form the else block.

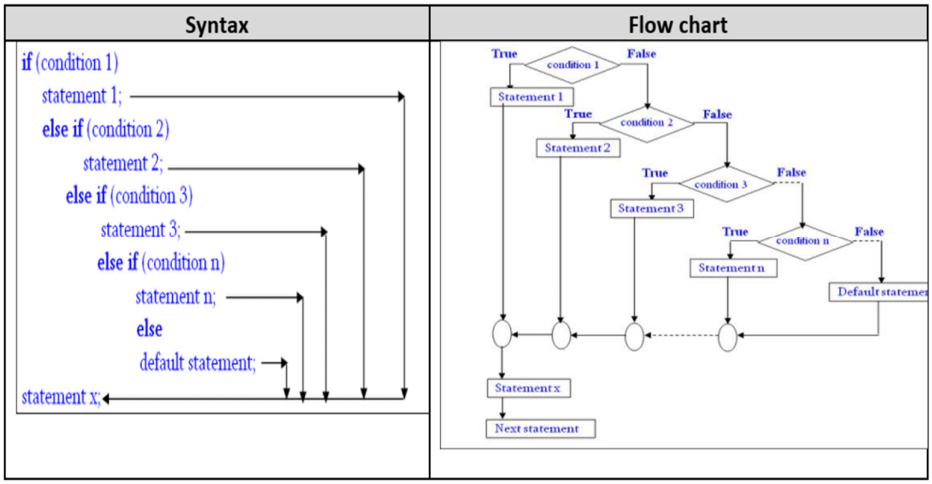
Example: Program to select and print the largest of the three float numbers using nested “ifelse” statements.

```
# include<stdio.h>
# include<conio.h>
void main( )
 {
float a,b,c;
printf(“Enter Three Values:”);
scanf(“%f%f%f ”, &a, &b, &c);
printf(“\n Largest Value is:”) ;
if(a>b)
{
if(a>c)
printf(“ %f ”, a);
else
printf(“ %f ”, c);
}
else
{
if (b>c)
printf(“ %f ”, b);
else
printf(“ %f ”, c);
 }
getch( ); }
```

In the above example second if….else construct is used within the else block and if block of the first if…else construct.

Else if ladder

The else-if ladder is mutiway branching statement. It is used to test many conditions.



The conditions are evaluated from top to bottom. When a true condition is evaluated, the statement associated with it is executed and the rest of the ladder is omitted. If none of the conditions is true then default statement is executed. If the default statement is not present, no action taken place when all other conditions are false.

Example

The marks obtained by a student in 5 different subjects are input through the keyboard. The student gets a division as per the following rules:

(i) Percentage above or equal to 60 - First division

(ii) Percentage between 50 and 59 - Second division (iii) Percentage between 40 and 49 - Third division

(iv) Percentage less than 40 - Fail . Write a program to calculate the division obtained by the student.

```
main( )
{
Int m1, m2, m3, m4, m5, per ;
per = ( m1+ m2 + m3 + m4+ m5 ) / 5 ;
if ( per >= 60 )
printf ( "First division" ) ;
else if ( per >= 50 )
printf ( "Second division" ) ;
else if ( per >= 40 )
printf ( "Third division" ) ;
else
printf ( "fail" ) ;
}
```

If the first condition (per >= 60) is true, then “First division” is printed. and the control will come out of the loop. If the first condition is false, then the next statement (per >= 50) will be evaluated. Depending upon the result, the next statement is executed or control will come out of the loop. The above procedure is continued until the last else if.

Switch Statement

1. The switch statement causes a particular group of statements to be chosen from several available groups.
2. The selection is based upon the current value of an expression which is included within the switch statement.
3. The switch statement is a multi-way branch statement.
4. In a program if there is a possibility to make a choice from a number of options, this structure is useful.
5. The switch statement requires only one argument of int or char data type, which is checked with number of case options.
6. The switch statement evaluates expression and then looks for its value among the case constants.
7. If the value matches with case constant, then that particular case statement is executed.
8. If no one case constant not matched then default is executed.
9. Here switch, case, break and default are reserved words or keywords.
10. Every case statement terminates with colon ":".
11. In switch each case block should end with break statement

```
switch (expression)
{
case constant1 ;
        statement – 1 ;
break;
case constant2 ;
        statement – 2 ;
        break;
default:
        default statement ;
}
 statement –x;
```

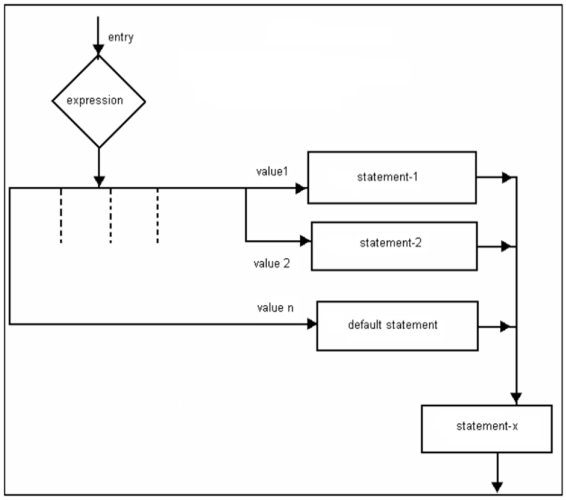
Where statement-1, statement-2 are statement lists and may contain zero or more statements. The expression following switch must be enclosed in parentheses and the body of the switch must be enclosed within curly braces. Expression may be a variable or integer expression.

Case labels can be only integer or character constants.

case labels do not contain any variable names. case labels must all be different . case labels end with a semicolon.

The switch structure starts with the switch keyword. It contains one block which contains the different cases. Each case contains different statements to be executed corresponding to different conditions and ends with the break statement. Break statement transfers the control out of the switch structure

to statement – x. If the value of variable is "constant1", the "case constant1: "is executed. If value is "constant2", the "case constant2" is executed and so on. If the value of the variable does not correspond to any case, the default case is executed.



Example

```
void main ( )
 {
char ch; int a,b,c=0;
printf (“Enter the two values”);
scanf(“%d%d”,&a, &b);
printf (“enter the operator ( + - * / )”) ;
ch = getchar();
switch(ch)
{
case ‘’+’’ : c = a + b; break;
case ‘’-‘’ : c = a – b; break;
case ‘’*’’ : c = a*b; break;
case ‘’/’’ : c = a/b; break;
default :
 printf (“The operator is invalid”) ;
}
printf (“The result is %d”, c);
}
```

RULES FOR FORMING SWITCH STATEMENT

1. The order of the case may be in any order. It is not necessary that order may be in as ascending or descending.
2. Mixing of integer and character constants in different cases is allowed.
3. If multiple statements are to be executed in each case, there is no need to enclose within a pair of braces.
4. If default statement is not present, then the program continues with the next instruction.
5. switch statement may occur within another switch statement.
6. If the break is not used in a certain case, the statements in the following cases are also executed irrespective of whether that case value is true or not. This execution will continue till a break is encountered. For example, consider the following program,

```
switch( ch )
{
case ‘’a’’:
case ‘’b’’:
case ‘’i’’ : printf (“The vowel i \n”);
```

```
case ''o'': printf ("The vowel o");
break ;
case ''u'':
default : printf ( Not vowel );
}
```

7. In the above example, if the value of ch = 'i', then the display will be
The vowel i
The vowel o

8. When one of the case statements is evaluated as true, all statements are executed until a break statement is executed.

goto Statement

The goto statement is used to transfer the control from one point to another. The general form of the goto statement is

```
goto label;
```

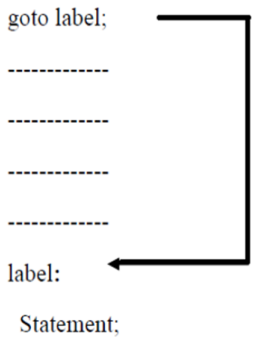
Where label is an identifier. Label is the name given to the target place to which the control will be transferred. Control may be transferred to any other statement within the program. The label is placed immediately before the statement where the control is to be transferred. The label can be anywhere in the program either before or after the goto label statement. The general form of the target place is

```
Label: statement (s)
```

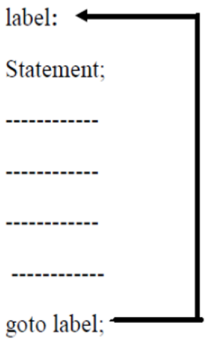
The goto statement transfers the control without checking any condition. That is why this statement is sometimes called as unconditional goto statement. Goto statement may be useful for exiting from any levels of nesting in one jump.

It is possible to have a forward jump or a backward jump.

- If the "label:" is before the statement "goto label;" a loop will be formed and some statements will be executed repeatedly. Such a jump is known as a „backward jump".
- If the "label:" is placed after the "goto label;" some statements will be skipped and the jump is known as a "forward jump".



Forward jump



Backward jump

Example

```
main ( )
{
int x = 10; loop:
printf ("x is %d \n", x );
```

```
x++;
if ( x < 100 ) goto loop;
printf ("End of the goto statement");
}
```

Looping Statements

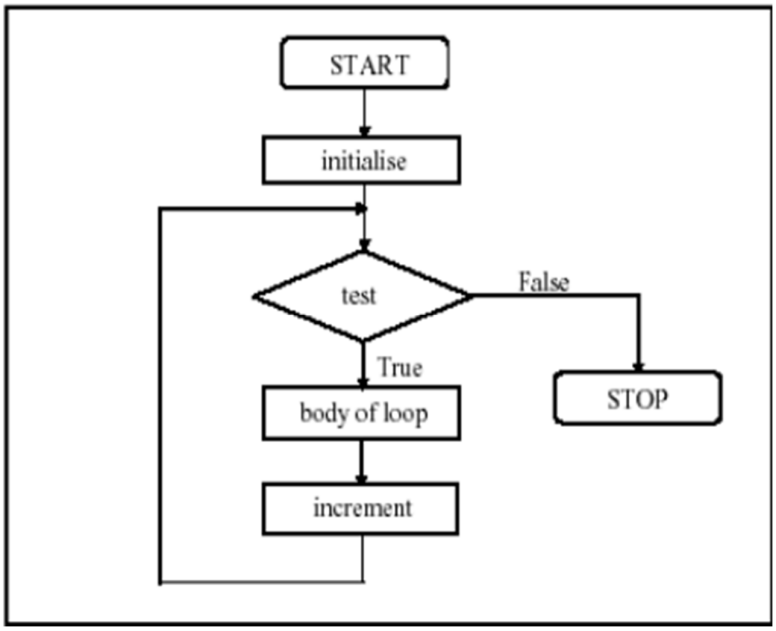
Loops are used to execute a same set of instructions for many times. The number of times a loop executed is given by some condition. The loop structures available in C are

1) for loop 2) while loop and 3) do…while loop

for loop

The for loop is used to repeat a statement or block of statements for a known number of times. The general format of the for loop

```
for ( initialization; condition test; increment)
{
statements(s)
 }
```



Initialization is an assignment statement, that is used to set the loop control variable. The condition test is a relational or logical expression which determines when to end the loop. The increment is a unary or an assignment expression. This section is used to alter the value of the variable initially assigned by initialization. These three sections must be separated by semicolon. The statement which forms the body of the loop can be either a single statement or a compound statement (group of statements).

When the "for statement" is executed, the value of

Conditional test is evaluated and tested before each pass through the loop. Incrementation is carried out at the end of each pass.

The for loop continues to execute as long as the value of the conditional test is true. When the value of condition becomes false, the program comes out of the for loop

Execution of the for loop

1. Initialization of control variable is done first.

2. The value of control variable is tested using condition test. If the value is true, the body of the loop is executed; otherwise the loop is terminated.

3. After the body of the loop is executed, the control is transferred back to for statement. Control variable is altered and now the new value is tested. This process continues till the value of the control variable fails to satisfy the test condition.

Example: To print odd numbers from 1 to 13

```
main()
{
int i;
for ( i=1; i<=13; i=i+2)
printf("%d", i); }
```

Output : 1 3 5 7 9 11 13

The above for loop initialized the integer variable i to 1 and increments it by 2 every time the loop is executed.

In for loop, the conditional test is always performed at the starting of the loop. The body of the loop is not executed if the condition is false in the beginning itself. Thus the minimum number of loop execution is zero.

All the three sections need not be present in the for statement. But semicolons are necessary and must be shown. If the first section is omitted, the initialization is to be done before the for loop. If the third section is omitted, the variable is incremented in the body of the loop. If the second section is omitted, it will assumed that the permanent value is 1; then the loop will continue indefinitely.

While loop

The while loop construct contains only the condition. The general format of the while loop is

```
Initialization Expression;
while( Test Condition)
{
Body of the loop
Updation Expression
}
```

Where body of the loop is either an empty statement, a single statement or a block of statements. The condition may be any expression. The condition value may be zero or non-zero.

1. The while is an entry – controlled loop statement.
2. The test condition is evaluated and if the condition is true, then the body of the loop is executed.
3. The execution process is repeated until the test condition becomes false and the control is transferred out of the loop.
4. On exit, the program continues with the statement immediately after the body of the loop.
5. The body of the loop may have one or more statements.
6. The braces are needed only if the body contains two or more statements.
7. It‟is a good practice to use braces even if the body has only one statement.

do…..while loop

The do while loop is sometimes called as the do loop in C. Unlike for and while loops, this loop checks the condition at the end of the loop. So the body of the loop is executed at least once, even if the condition is false initially. The general form of do…while loop is.

```
Initialization Expression;
do{ Body of the loop
Updation Expression; }
while ( Test Condition);
```

The do…while loop repeats until the condition becomes false. In the do…while, the body of the loop is executed first, then the condition is checked. When the condition becomes false, control is transferred to the statement after the loop.

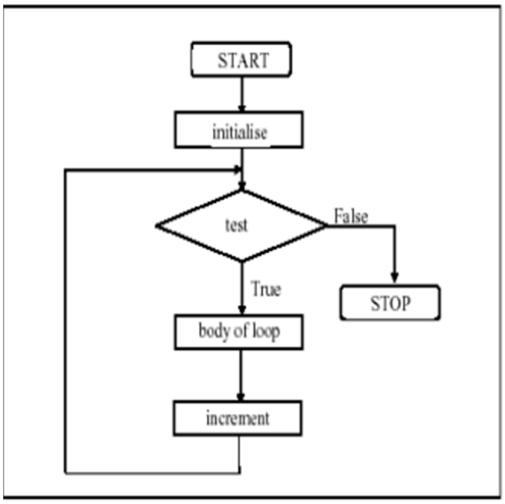
Difference between while loop and do-while loop

The difference between the while and do…while is illustrated by the following program segments

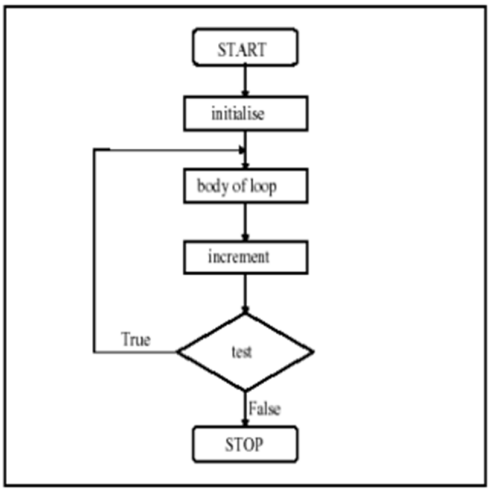
while loop

```
main ( )
{
while ( 100 < 10 )
printf (“False”);
}
```

In the above program, the condition is false for the first line. So the printf ( ) will not be executed at all.



do…while loop




```
main ( )
{
do
{
printf (“False”);
```

```
}
while (100 < 10 );
}
```

In the above program, the printf statement will be executed only once.

Nested loops

A loop can also be used within loops.

1. i.e. one for statement within another for statement is allowed in C. (or „C" allows multiple for loops in the nested forms).
2. In nested for loops one or more for statements are included in the body of the loop.

Two loops can be nested as follows.

Syntax:

```
for( initialize ; test condition ; increment) /* outer loop */
{
for(initialize ; test condition ; increment) /* inner loop */
{Body of loop; }
}
```

For example, to find the factorial value of number between 1 to 10, the program i

```
main ( )
{
int i, j, factorial;
for ( i=1; i<=10; i++ )
{
factorial = 1;
for ( j = 1; j<= i; j ++ )
{factorial = factorial * j;}
printf ("Factorial value of %d is %d", i, factorial ) ;
}
}
```

One loop can be completely contained within the other. But there can be no overlap. Each loop must have different control variable.

Comparison between the Three loops, for, while, do ….. While

S.No.	Topics	For loop	While loop	Do...while loop
1	Initialization of condition variable	In the parenthesis of the loop.	Before the loop.	Before the loop or in the body of the loop.
2	Test condition	Before the body of the loop.	Before the body of the loop.	After the body of the loop.
3	Updating the condition variable	After the first execution.	After the first execution.	After the first execution.
4	Type	Entry controlled loop.	Entry controlled loop.	Exit controlled loop.
5	Loop variable	Counter.	Counter.	Sentinel & counter

Break Statement

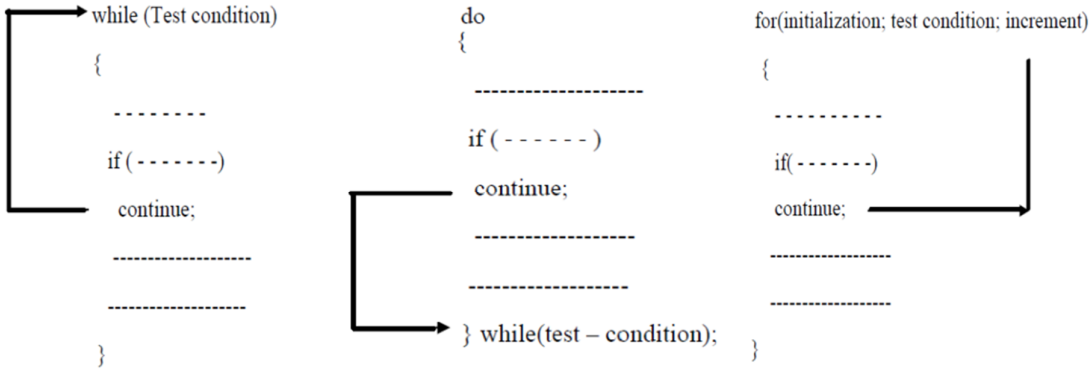
1. A break statement terminates the execution of the loop and the control is transferred to the statement immediately following the loop.
2. The break statement is used to terminate loops or to exit from a switch.
3. It can be used within a for, while, do-while, or switch statement.
4. The break statement is written simply as break;

Example

```
main()
{
int ch = 65;
for(;;ch++)
{
printf("%c", ch);
if (ch == 97) break;
}
}
```

Continue Statement

1. The continue statement is used to bypass the remainder of the current pass through a loop.
2. The loop does not terminate when a continue statement is encountered.
3. Instead, the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop.
4. The continue statement can be included within a while, a do-while, a for statement.
5. It is simply written as "continue".
6. The continue statement tells the compiler "Skip the following Statements and continue with the next Iteration".
7. In "while" and "do" loops continue causes the control to go directly to the test – condition and then to continue the iteration process.
8. In the case of "or" loop, the updation section of the loop is executed before test-condition, is evaluated.



Example

```
main()
{ int number, sum=0, count=0;
while (count <= 10)
{
printf("\nEnter the number: ");
scanf("%d",&number);
if(number <=0 ) continue;
sum + = number;
count++;
} }
```

The above program is used to find the sum of positive numbers.

I/O Functions

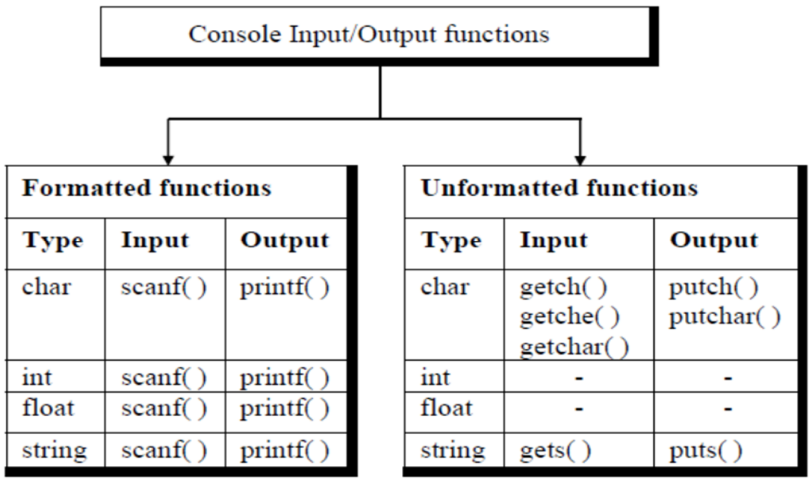
C has no input and output statement to perform the input and output operations. C language has a collection of library functions for input and output (I/O) operations. The input and output functions are used to transfer the information between the computer and the standard input / output devices.

Console I/O functions - functions to receive input from keyboard and write output to VDU.

Formatted and Unformatted Functions

The basic difference between formatted and unformatted I/O functions is that the formatted functions allow the input read from the keyboard or the output displayed on the VDU to be formatted as per our requirements. For example, if values of average marks and percentage marks are to be displayed on the screen, then the details like where this output would appear on the screen, how many spaces would be present between the two values, the number of places after the decimal points etc., can be controlled using formatted functions.

Console functions



Formatted functions		
Type	**Input**	**Output**
char	scanf( )	printf( )
int	scanf( )	printf( )
float	scanf( )	printf( )
string	scanf( )	printf( )

Unformatted functions		
Type	**Input**	**Output**
char	getch( ) getche( ) getchar( )	putch( ) putchar( )
int	-	-
float	-	-
string	gets( )	puts( )

Printf () Function

printf ( ) function is used to display data on the monitor. The printf ( ) function moves data from the computer's memory to the standard output device. The general format of printf ( ) function is

```
printf ( "control string", list of
```

Comma is used to separate the control string from variable list.

The control string can contain : -the characters that are to be displayed on the screen.
-format specifiers that begin with a % sign.
-escape sequences that begin with a backward slash (\) sign.

Examples

```
printf (""%f, %f, %d, %d"", i, j, k + i, 5);
printf ("Sum of %d and %d is %d"", a, b, a + b );
printf (""Two numbers are %d and % d"",a,b );
```

printf ( ) never supplies a newline automatically. Therefore multiple printf ( ) statements may be used to display one line of output. A new line can be introduced by the new line character \n.

Conversion Characters or Format Specifiers

The character specified after % is called a conversion character. Conversion character is used to convert one data type to another type.

Conversion Character	Meaning
%c	Data item is displayed as a single character
%d	Data item is displayed as a signed decimal integer
% e	Data item is displayed as a floating-point value with an exponent
% f	Data item is displayed as a floating-point value without an exponent
%i	Data item is displayed as a signed decimal integer
%o	Data item is displayed as an octal integer, without a leading zero
%s	Data item is displayed as a string
%u	Data item is displayed as an unsigned decimal integer
%x	Data item is displayed as a hexadecimal integer, without the leading Ox

Formatted Printf Functions

Formatted printf function is used for the following purposes:

1. To print values at the particular position of the screen.
2. To insert the spaces between the two values.
3. To give the number of places after the decimal point.

The above things can be achieved by adding modifier to the format specifiers. Modifiers are placed between the percent sign (%) and the format specifier. A maximum field width, the number of decimal places, left justification can be specified by using the modifiers.

The number placed between percent sign (%) and the format specifier is called a field width specifier.

For outputting Integer Numbers

The general format of the field width specifier is

```
%Wd
```

Where W is the minimum field width. If the value to be printed is greater than the specified field width, the whole value is displayed. If the output is shorter than the length specified, remaining spaces will be filled by blank spaces and the value is right justified. The following example illustrates the output of the number 1234 under different formats.

Format	Output					
printf(“”%d””,1234)	1	2	3	4		
printf(“”%6d””,1234)			1	2	3	4
printf(“”%2d””,1234)	1	2	3	4		
printf(“”%-6d””,1234)	1	2	3	4		
printf(“”%4d””,-1234)	-	1	2	3	4	
printf(“”%06d””,1234)	0	0	1	2	3	4

Scanf() Function

scanf( ) function is used to read the value from the input device. The general format of scanf ( ) function is

```
scanf (“Control string”, list of address of arguments );
```

The list of address of arguments represents the data items to be read. Each variable name must be preceded by an ampersand (&). The arguments are actually pointer which indicate where the data items are stored in the memory. Example of a scanf function is

scanf (“%d %f %c””, &a, &b, &ch);

The control string consists of format specifier, white space character and non-white space character.The format specifier in scanf are very similar to those used with printf ( ).

Important points while using scanf are

1. Every basic data type variable must be preceded by (&) sign. In the case of string and array data type, the data name is not preceded by the character &.
2. Text to be printed is not allowed in scanf statement. For example scanf (“Enter the number %d”, &a); is not valid.
3. The data items must correspond to the arguments.
4. If two or more data items are entered, they must separated by white space characters. (blank spaces, tabs or new line characters). Data items may continue into two or more lines. For example, for the statement

scanf (“”%s %d %f””, name, &regno, &avg);

the data items can be entered in the following methods.

(i) Arul
12345
85.56

(ii) Arul 12345
85.56

(iii) Arul
12345 85.56

(iv) Arul 12345 85.56

Unformatted Functions

getchar () function

getchar() function is used to read one character at a time from the standard input device. When the getchar() function is called, it waits until a key is pressed and assigns this character as a value to getchar function. The value is also echoed on the screen.

The getchar() function does not require any argument. But a pair of empty parentheses must follow the word getchar. In general, the getchar function is written as

```
variable name = getchar();
```

Where variable name is a valid C identifier that has been declared as char type.
For example,

```
char letter ;
letter = getchar ();
```

will assign the character ‘A’ to the variable letter when pressing A on the keyboard.
getchar ( )
accepts all the characters upto the pressing of enter key, but reads the first character only.

putchar( ) function

Single character can be displayed using the function putchar ( ). The function putchar ( ) stands for “put character” and uses a argument. The general form of the putchar( ) function is

```
putchar (argument);
```

The argument may be a character variable or an integer value or the character itself contained within a single quote.

Examples

```
void main() {
char x = ‘’A’’;
```

```
putchar (x);
putchar (''B'');
}+
```

gets() function

gets() accepts any line of string including spaces from the standard Input device (keyboard). gets() stops reading character from keyboard only when the enter key is pressed.

Syntax for gets()

```
gets(variable_name);
```

puts() function

puts displays a single / paragraph of text to the standard output device.

Syntax for puts in C :

```
puts(variable_name);
```

Sample program :

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a[20];
gets(a);
puts(a);
getch();
}
```

1. gets is used to receive user input to the array. gets stops receiving user input only when the Newline character (Enter Key) is interrupted.
2. puts is used to display them back in the monitor.

Recursion

Recursion is the process in which a function repeatedly calls itself to perform calculations. Forexample consider the following:

```
main()
{
printf("This is an example of recursion.\n");
main(); }
```

When executed this program will produce an output which is something like this,

This is an example of recursion.

This is an example of recursion.

Execution is terminated abruptly; otherwise the execution will continue indefinitely.

Recursion is a special case of function call where a function calls itself. These are very useful in the situations where solution can be expressed in terms of successively applying same operation to the subsets of the problem. For example, a recursive function to calculate factorial of a number n is given below:

The following function calculates factorials recursively:

```
int fact(int n)
{
int factorial;
if(n==1||n==0)
return(1);
else
factorial=n*fact(n-1);
```

```
return (factorial);
}
```

Assume n=4, we call fact(4)
Since n is not equal to 1 0, factorial=n*fact(n-1)
Factorial=4*fact(3) (again call fact function with n=3)
=4*3*fact(2) (again call fact function with n=2)
=4*3*2*fact(1) (again call fact function with n=1)
=4*3*2*1 (terminating condition)
=24
Always have a terminating condition with a recursive function call otherwise function will never return.