Module II

Character Set

As every language contains a set of characters used to construct words, statements, etc. C language also has a set of characters which include **alphabets, digits**, and **special symbols**. C language supports a total of 256 characters. C language character set contains the following set of characters...

1. Alphabets
2. Digits
3. Special Symbols

Alphabets

C language supports all the alphabets from the English language. Lower and upper case letters together support 52 alphabets.

Lower case letters - **a to z**

Upper case letters - **A to Z**

Digits

C language supports 10 digits which are used to construct numerical values in C language.

Digits - **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

Special Symbols

Special Symbols - **~ @ # $ % ^ & * ( ) _ - + = { } [ ] ; : ' " / ? . > ,< \ | tab newline space NULL bell backspace vertical tab etc.,**

Token

Every C program is a collection of instructions and every instruction is a collection of some individual units. Every smallest individual unit of a c program is called token. Every instruction in a c program is a collection of tokens. Tokens are used to construct c programs and they are said to the basic building blocks of a c program.

In a c program tokens may contain the following

1. Keywords
2. Identifiers
3. Operators
4. Special Symbols
5. Constants
6. Strings
7. Data values

Keywords

Keywords are specific reserved words in C each of which has a specific feature associated with it. Keywords are always in lowercase.

There are total of 32 keywords in C:

auto	**break**	**case**	**char**	**const**	**continue**
default	**do**	**double**	**else**	**enum**	**extern**
float	**for**	**goto**	**if**	**int**	**long**
register	**return**	**short**	**signed**	**sizeof**	**Static**
struct	**switch**	**typedef**	**union**	**unsigned**	**void**
volatile	**while**				

Identifier

An identifier is a collection of characters which acts as the name of variable, function, array, pointer, structure, etc.

Rules for Creating Identifiers

1. An identifier can contain letters (Upper case and lower case), digit and underscore symbol only.
2. An identifier should not start with a digit value. It can start with a letter or an underscore.
3. Should not use any special symbols in between the identifier even whitespace.
4. Keywords should not be used as identifiers.

5. There is no limit for the length of an identifier. However, the compiler considers the first 31 characters only.
6. An identifier must be unique in its scope.

Data Types

Data types in the c programming language are used to specify what kind of value can be stored in a variable. The memory size and type of the value of a variable are determined by the variable data type.

In the c programming language, data types are classified as follows

1. Primary data types (Basic data types OR Predefined data types)
2. Derived data types (Secondary data types OR User-defined data types)
3. Enumeration data types
4. Void data type

I. Primitive Data Types

- **char:** The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
- **int:** As the name suggests, an int variable is used to store an integer.
- **float:** It is used to store decimal numbers (numbers with floating point value) with single precision.
- **double:** It is used to store decimal numbers (numbers with floating point value) with double precision.

DATA TYPE	MEMORY (BYTES)	FORMAT SPECIFIER
Int	2	%d
Char	1	%c
Float	4	%f
Double	8	%lf

Note: Size of each data types varies based on machines.

Data Type Qualifier

The basic data types can be augmented by the use of the data type ***qualifiers*** short, long, signed and unsigned. For example, integer quantities can be defined as short int, long int or unsigned int.

The interpretation of a qualified integer data type will vary from one C compiler to another, though there are some commonsense relationships. Thus, a short int may require less memory than an ordinary int or itmay require the same amount of memory as an ordinary int, but it will never exceed an ordinary int in word length. Similarly, a long int may require the same amount of memory as an ordinary int or it may require more memory, but it will never be less than an ordinary int.

If short int and int both have the same memory requirements (e.g. 2 bytes), then long int will generally have double the requirements (e.g. 4 bytes). Or if int and long int both have the same memory requirements (e.g. 4 bytes) then short int will generally have half the memory requirements (e.g. 2bytes).

An unsigned int has the same memory requirements as an ordinary int. However, in the case of an ordinary int, the leftmost bit **is** reserved for the sign. With an unsigned int, all of the bits are used to represent the numerical value. Thus, an unsigned int can be approximately twice as large as an ordinary int. For example, if an ordinary int can vary from -32,768 to **+32,767**, then an unsigned int will be allowed to vary from 0 to **65,535.**

II. void data type

The void data type means nothing or no value. Generally, the void is used to specify a function which does not return any value. We also use the void data type to specify empty parameters of a function.

Variables

A *variable* is an identifier that is used to represent some specified type of information within a designated portion of the program. Variables in a c programming language are the named

memory locations where the user can store different values of the same datatype during the program execution.

Declaration

A declaration associates a group of variables with a specific data type. All variables must be declared before they can appear in executable statements. A declaration consists of a data type, followed by one or more variable names, ending with a semicolon.

Example

int a,b,c ; float d; char e;

Escape Sequence

Certain nonprinting characters, as well as the backslash (\) and the apostrophe (‘), can be expressed in terms of ***escape sequences.*** An escape sequence always begins with a backward slash and is followed by one or more special characters. For example, a line feed **(LF),** which is referred to as a ***newline*** in C, can be represented as \n. Such escape sequences always represent single characters, even though they are written in terms of two or more characters.

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote

Constants

A constant is a value or variable that can't be changed in the program, for example: 10, 20, 'a', 3.4, "c programming" etc.

There are different types of constants in C programming.

List of Constants in C

Constant	Example
Integer Constant	10, 20, 450 etc.
Real or Floating-point Constant	10.3, 20.2, 450.6 etc.
Octal Constant	021, 033, 046 etc.
Hexadecimal Constant	0x2a, 0x7b, 0xaa etc.
Character Constant	'a', 'b', 'x' etc.
String Constant	"c", "c program".

Symbolic Constants

A symbolic constant is a name that substitutes for a sequence of characters. The characters may represent a numeric constant, a character constant or a string constant. When a program is compiled, each occurrence of a symbolic constant is replaced by its corresponding character sequence. Symbolic constants are usually defined at the beginning of a program.

Syntax: #define name text

Example: #define pi 3.14, #define SQR(x) x*x

Console I/O operation

The printf() and scanf() functions are used for input and output in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).

printf() function

The **printf() function** is used for output. It prints the given statement to the console.

The syntax of printf() function is given below:

printf("format string",*argument_list);*

scanf() function

The **scanf() function** is used for input. It reads the input data from the console.

scanf("format string",*argument_list);*

The **format string** can be %d (integer), %c (character), %s (string), %f (float) etc.

Structure of C program

```
/* program to calculate the area of a circle */          /* title (comment) */
#include <stdio.h>                                       / * library file access */
void main( )                                             / * function heading */
{
        float radius,area;                               /* variable declarations */
        printf (“radius =”);                             / * output statement (prompt) /*
        scanf (“%f”, &radius) ;                          /* input statement * /
        area = 3.14 * radius * radius;                   /* assignment statement */
        printf ("area = %f",area) ;                      / * output statement */
}
```

Operators

Operators are the foundation of any programming language.

C has many built-in operator types and they are classified as follows:

1. **Arithmetic Operators**: These are the operators used to perform arithmetic/mathematical operations on operands. Examples: (+, -, *, /, %,++,–). Arithmetic operator are of two types:
   I. **Unary Operators**: Operators that operates or works with a single operand are unary operators. For example: (++ , –)
   II. **Binary Operators**: Operators that operates or works with two operands are binary operators. For example: (+ , – , * , /)

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

```
        printf("Enter the value of a and b");
        scanf("%d%d",&a,&b);

        // Uniary Operator
        printf("Unary Minus=%d\n",-a);
        printf("Increment =%d\n ",a++);
        printf("Decrement =%d\n ",a--);

        // Binary Operator
        printf("Sum =%d\n ",a+b);
        printf("Difference =%d\n ",a-b);
        printf("Mul =%d\n ",a*b);
        printf("Division =%d\n ",a/b);
        printf("Modulo =%d\n ",a%b);
}
```

Output
Enter the value of a and b20 10
Unary Minus=-20
Increment =20
Decrement =21
Sum =30
Difference =10
Mul =200
Division =2
Modulo =0

Relational Operators:

These are used for comparison of the values of two operands.

1. **Equal to operator:** Represented as **'=='**, the equal to operator checks whether the two given operands are equal or not. If so, it returns 1. Otherwise it returns 0. For example, **5==5** will return 1.
2. **Not equal to operator:** Represented as **'!='**, the not equal to operator checks whether the two given operands are equal or not. If not, it returns 1. Otherwise it returns 0. It is the exact boolean complement of the **'=='** operator. For example, **5!=5** will return 0.
3. **Greater than operator:** Represented as **'>'**, the greater than operator checks whether the first operand is greater than the second operand or not. If so, it returns 1. Otherwise it returns 0. For example, **6>5** will return 1.

4. Less than operator: Represented as **'<'**, the less than operator checks whether the first operand is lesser than the second operand. If so, it returns 1. Otherwise it returns 0. For example, **6<5** will return 0.
5. **Greater than or equal to operator:** Represented as **'>='**, the greater than or equal to operator checks whether the first operand is greater than or equal to the second operand. If so, it returns 1 else it returns 0. For example, **5>=5** will return 1.
6. **Less than or equal to operator: Represented as '<='**, the less than or equal to operator checks whether the first operand is less than or equal to the second operand. If so, it returns 1 else 0. For example, **5<=5** will also return 1.

Logical Operators:

They are used to combine two or more conditions/constraints or to complement the evaluation of the original condition under consideration. They are described below:

1. **Logical AND operator:** The **'&&'** operator returns 1 when both the conditions under consideration are satisfied. Otherwise it returns 0. For example, **a && b** returns 1 when both a and b are 1 (i.e. non-zero).
2. **Logical OR operator:** The **'||'** operator returns 1 even if one (or both) of the conditions under consideration is satisfied. Otherwise it returns 0. For example, **a || b** returns 1 if one of a or b or both are 1 (i.e. non-zero). Of course, it returns 1 when both a and b are 1.
3. **Logical NOT operator:** The **'!'** operator returns 1 the condition in consideration is not satisfied. Otherwise it returns 0. For example, **!a** returns 1 if a is 0, i.e. when a=0.

Short-Circuiting in Logical Operators:

In case of **logical AND**, the second operand is not evaluated if first operand is false. For example, 1>5 && 5<10, since 1>5 is false, second expression will not be evaluated.

In case of **logical OR**, the second operand is not evaluated if first operand is true. For example, 1<5 || 5>10, since 1<5 is true, second expression will not be evaluated.

Assignment Operators:

Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and right side operand of the assignment operator is a value. The value on the right side must be of the same data-type of variable on the left side otherwise the compiler will raise an error.

Different types of assignment operators are shown below:

i. **"="**: This is the simplest assignment operator. This operator is used to assign the value on the right to the variable on the left.

For example: a = 10;

ii. **"+="**: This operator is combination of '+' and '=' operators. This operator first adds the current value of the variable on left to the value on right and then assigns the result to the variable on the left.

Example: (a += b) can be written as (a = a + b)

If initially value stored in a is 5. Then (a += 6) = 11.

iii. **"-="**: This operator is combination of '-' and '=' operators. This operator first subtracts the value on right from the current value of the variable on left and then assigns the result to the variable on the left.

Example: (a -= b) can be written as (a = a - b)

If initially value stored in a is 8. Then (a -= 6) = 2.

iv. **"*="**: This operator is combination of '*' and '=' operators. This operator first multiplies the current value of the variable on left to the value on right and then assigns the result to the variable on the left.

Example: (a *= b) can be written as (a = a * b)

If initially value stored in a is 5. Then (a *= 6) = 30.

v. **"/="**: This operator is combination of '/' and '=' operators. This operator first divides the current value of the variable on left by the value on right and then assigns the result to the variable on the left.

Example: (a /= b) can be written as (a = a / b)

If initially value stored in a is 6. Then (a /= 2) = 3.

Bitwise Operators:

The Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands. The mathematical operations such as addition, subtraction, multiplication etc. can be performed at bit-level for faster processing.

1. **& (bitwise AND)** in C takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
2. **| (bitwise OR)** in C takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.
3. **^ (bitwise XOR)** in C takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
4. **<< (left shift)** in C takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.
5. **>> (right shift)** in C takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.
6. **~ (bitwise NOT)** in C takes one number and inverts all bits of it.

```
#include <stdio.h>
int main()
{
   // a = 5(00000101), b = 9(00001001)
   unsigned char a = 5, b = 9;

   // The result is 00000001
   printf("a = %d, b = %d\n", a, b);
   printf("a&b = %d\n", a & b);

   // The result is 00001101
   printf("a|b = %d\n", a | b);

   // The result is 00001100
   printf("a^b = %d\n", a ^ b);

   // The result is 11111010
   printf("~a = %d\n", a = ~a);

   // The result is 00010010
   printf("b<<1 = %d\n", b << 1);
```

```
    // The result is 00000100
    printf("b>>1 = %d\n", b >> 1);

    return 0;
}
```

Output

```
a = 5, b = 9
a&b = 1
a|b = 13
a^b = 12
~a = 250
b<<1 = 18
b>>1 = 4
```

Swap two number using XOR operation

```
#include<stdio.h>
void main()
{
        int a,b;
        printf("Enter the value of a and b");
        scanf("%d%d",&a,&b);
        printf("Before Swapping\n");
        printf("a=%d\tb=%d\n",a,b);
        a=a^b;
        b=a^b;
        a=a^b;
        printf("After Swapping\n");
        printf("a=%d\tb=%d\n",a,b);
}
```

Output

Enter the value of a and b10 20

Before Swapping

a=10 b=20

After Swapping

a=20 b=10

Notes:

1) The left-shift and right-shift operators are equivalent to multiplication and division by 2 respectively.

2) & operator can be used to quickly check if a number is odd or even. The value of expression (x & 1) would be non-zero only if x is odd, otherwise the value would be zero.

3) **The left shift and right shift operators should not be used for negative numbers**. If any of the operands is a negative number, it results in undefined behaviour. For example results of both -1 << 1 and 1 << -1 is undefined. Also, if the number is shifted more than the size of integer, the behaviour is undefined. For example, 1 << 33 is undefined if integers are stored using 32 bits.

4) **The bitwise operators should not be used in place of logical operators.** The result of logical operators (&&, || and !) is either 0 or 1, but bitwise operators return an integer value. Also, the logical operators consider any non-zero operand as 1.

Conditional Operator

Conditional operator is of the form *Expression1 ? Expression2 : Expression3* . Here, Expression1 is the condition to be evaluated. If the condition(Expression1) is *True* then we will execute and return the result of Expression2 otherwise if the condition(Expression1) is *false* then we will execute and return the result of Expression3. We may replace the use of if..else statements by conditional operators.

Largest of two numbers using conditional operator (ternary operator)

```
#include<stdio.h>
void main()
{
        int a,b,large;
        printf("Enter the value of a and b");
        scanf("%d%d",&a,&b);
        large = a>b ? a : b;
        printf("Largest=%d\n",large);
}
```

Largest of tthree numbers using conditional operator (ternary operator)

```
#include<stdio.h>
void main()
{
        int a,b,c,large;
```

```
        printf("Enter the value of a,b and c");
        scanf("%d%d%d",&a,&b,&c);
        large = a>b && a>c ? a : b>a && b>c ?b:c;
        printf("Largest=%d\n",large);
}
```

sizeof operator:

sizeof is a much used in the C/C++ programming language. It is a compile time unary operator which can be used to compute the size of its operand. The result of sizeof is of unsigned integral type which is usually denoted by size_t. Basically, sizeof operator is used to compute the size of the variable.

```
#include <stdio.h>
void main()
{
   printf("%d\n", sizeof(char));
   printf("%d\n", sizeof(int));
   printf("%d\n", sizeof(float));
   printf("%d", sizeof(double));
}
```

Output

1

4

4

8

Control Flow Statement

C provides two styles of flow control:

- Branching
- Looping

Branching is deciding what actions to take and looping is deciding how many times to take a certain action.

Branching

1) if statement

This is the simplest form of the branching statements. It takes an expression in parenthesis and a statement or block of statements. if the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

Syntax

- **if**

```
if (expression)
 {
   Block of statements;
 }
```

- **if--else**

```
if (expression)
 {
   Block of statements;
 }
else
 {
   Block of statements;
 }
```

- **if—else if**

```
if (expression)
 {
   Block of statements;
 }
else if(expression)
 {
   Block of statements;
 }
else
 {
   Block of statements;
 }
```

Q) Write a program to test whether a number even or odd?

```
#include <stdio.h>
void main()
{
   int n;
        printf("Enter the value of n");
        scanf("%d",&n);
        if( n%2 == 0)
        {
                printf("Even");
        }
        else
        {
                printf("Odd");
        }
}
```

Q) Write a program to test whether a number is negative or positive or zero?

```
#include <stdio.h>
void main()
{
   int n;
        printf("Enter the value of n");
        scanf("%d",&n);
        if( n > 0)
        {
                printf("Positive");
        }
        else if(n < 0)
        {
                printf("Negative");
        }
        else{
                printf("Zero");
        }
}
```

Q) Write a program to find largest of two numbers?

```
#include <stdio.h>
void main()
{
   int a,b;
        printf("Enter the value of a and b");
        scanf("%d%d",&a,&b);
        if(a>b)
        {
                printf("%d is larger than %d",a,b);
        }
        else
        {
                printf("%d is larger than %d",b,a);
        }
}
```

Q) Write a program to find largest of three numbers?

```
#include <stdio.h>
void main()
{
   int a,b,c;
        printf("Enter the value of a and b");
        scanf("%d%d",&a,&b,&c);
        if(a>b && a>c)
        {
```

```
                printf("%d is largerest",a);
        }
        else if(b>a && b>c)
        {
                printf("%d is largerest",b);
        }
        else
        {
                printf("%d is largerest",c);
        }
}
```

Q) Write a program to test whether number is even or odd using AND operator?

```
#include <stdio.h>
void main()
{
   int n;
        printf("Enter the value of n");
        scanf("%d",&n);
        if( n & 1 == 0)
        {
                printf("Even");
        }
        else
        {
                printf("Odd");
        }
}
```

Q) Write a program to test whether number is even or odd using shift operator?

```
#include <stdio.h>
void main()
{
   int n;
        printf("Enter the value of n");
        scanf("%d",&n);
        if( (n >> 1) <<1 == n)
        {
                printf("Even");
        }
        else
        {
                printf("Odd");
        }
}
```

switch statement

The switch statement is much like a nested if .. else statement. Its mostly a matter of preference which you use, switch statement can be slightly more efficient and easier to read.

Syntax

```
switch( expression )
    {
      case constant-expression1:      statements1;break;
      case constant-expression2:      statements2;break;
      case constant-expression3:      statements3;break;
      default : statements4;
    }
```

Q) Write a program to read number between 1 and 7 and display the day corresponding to the number.

```
#include <stdio.h>
void main()
{
   int n;
        printf("Enter the value of n");
        scanf("%d",&n);
        switch(n)
        {
                case 1: printf("Sunday");
                        break;
                case 2: printf("Monday");
                        break;
                case 3: printf("Tuesday");
                        break;
                case 4: printf("Wednesday");
                        break;
                case 5: printf("Thursday");
                        break;
                case 6: printf("Friday");
                        break;
                case 7: printf("Saturday");
                        break;
                default:printf("Wrong Choice");
        }
}
```

goto statement(Unconditional Jump)

The goto statement allows us to transfer control of the program to the specified label.

Syntax

```
goto label;
... .. ...
... .. ...
label:
statement;
```

Q) Write a program to read a number and check whether a number is odd or even. If the number is negative goto end.

```
#include <stdio.h>
void main()
{
   int n;
        printf("Enter the value of n");
        scanf("%d",&n);
        if(n<0)
                goto END;
        if(n % 2 ==0)
                printf("Even\n");
        else
                printf("Odd\n");
        END:printf("End");
}
```

Output

```
Enter the value of n10
Even
End
```

```
Enter the value of n-10
End
```

Looping

Loops provide a way to repeat commands and control how many times they are repeated. C provides a number of looping way.

while loop

The most basic loop in C is the while loop. A while statement is like a repeating if statement. Like an If statement, if the test condition is true: the statements get executed. The difference is

that after the statements have been executed, the test condition is checked again. If it is still true the statements get executed again. This cycle repeats until the test condition evaluates to false.

Syntax

```
while ( expression )
{
  Single statement
  or
  Block of statements;
}
```

Q) Write a program to read a number and find sum of digits of a numbers.

```
#include <stdio.h>
void main()
{
   int n,sum=0,temp;
        printf("Enter the value of n");
        scanf("%d",&n);
        while ( n > 0 )
        {
                temp = n%10;
                sum = sum + temp;
                n = n/10;
        }
        printf("Sum = %d",sum);
}
```

Q) Write a program to read a number and find reverse of a number.

```
#include <stdio.h>
void main()
{
   int n,rev=0,temp;
        printf("Enter the value of n");
        scanf("%d",&n);
        while ( n > 0 )
        {
                temp = n%10;
                rev = rev * 10 + temp;
                n = n/10;
        }
        printf("Reverse = %d",rev);
}
```

Q) Write a program to read a number and check whether it is palindrome or not.

```
#include <stdio.h>
void main()
{
   int n,rev=0,temp,org;
        printf("Enter the value of n");
        scanf("%d",&n);
        org=n;
        while ( n > 0 )
        {
                temp = n%10;
                rev = rev * 10 + temp;
                n = n/10;
        }
        if( org == rev)
        {
                printf("Palindrome");
        }
        else
        {
                printf("Not Palindrome");
        }
}
```

Q) Write a program to read a number and check whether a number is Armstrong or not.

```
#include <stdio.h>
#include <math.h>
void main()
{
   int n,sum=0,temp,org;
        printf("Enter the value of n");
        scanf("%d",&n);
        org=n;
        // To find count number of digits in the number
        while( n > 0)
        {
                count++;
                n=n/10;
        }
        while ( n > 0 )
        {
                temp = n%10;
                rev = sum + pow(temp,count);
                n = n/10;
        }
```

```
        if( org == sum)
        {
                printf("Armstrong");
        }
        else
        {
                printf("Not Armstrong");
        }
}
```

for loop

for loop is similar to while, it's just written differently. for statements are often used to process lists such a range of numbers:

Basic syntax of for loop is as follows:

```
for( expression1; expression2; expression3)
{
  Single statement
  or
  Block of statements;
}
```

In the above syntax:

- expression1 - Initializes variables.
- expression2 - Conditional expression, as long as this condition is true, loop will keep executing.
- expression3 - expression3 is the modifier which may be simple increment of a variable.

Q) Write a program to find sum of first n natural numbers?

```
#include <stdio.h>
void main()
{
        int n,i,sum=0;
        printf("Enter the value of n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                sum = sum + i;
        }
```

```
        printf("Sum = %d",sum);
}
```

Q) Write a program to find sum of first n natural numbers?

```
#include <stdio.h>
void main()
{
        int n,i,fact=1;
        printf("Enter the value of n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                fact = fact * i;
        }
        printf("Factorial = %d",fact);
}
```

Q) Write a program to read n natural numbers and find largest even number and largest odd number using without using division or modulo?

```
#include <stdio.h>
void main()
{
   int n,a,i;
     int le = -1, lo=-1;
     printf("Enter the value of n");
     scanf("%d",&n);
     for(i=0;i<n;i++)
     {
             printf("Enter the value of a");
             scanf("%d",&a);
             if(a & 1 ==0)
             {
                     if(le<a)
                             le = a;
             }
             else
             {
                     if(lo<a)
                             lo = a;
             }
     }
}
```

do – while statement

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated. The syntax of the do...while loop is:

```
do
{
  // statements inside the body of the loop
}
while (expression);
```

Q) Write a program to find sum of first n natural numbers using do..while ?

```
#include <stdio.h>
void main()
{
        int n,i,sum=0;
        printf("Enter the value of n");
        scanf("%d",&n);
        i=1;
        do
        {
                sum = sum + i;
                i++;
        }while(i<=n);
        printf("Sum = %d",sum);
}
```

Difference between while and do while statements

WHILE	DO-WHILE
Condition is checked first then statement(s) is executed.	Statement(s) is executed, thereafter condition is checked.
It might occur statement(s) is executed zero times, If condition is false.	At least once the statement(s) is executed.
No semicolon at the end of while. while(condition)	Semicolon at the end of while. while(condition);
If there is a single statement, brackets are not required.	Brackets are always required.
while loop is entry controlled loop.	do-while loop is exit controlled loop.

WHILE	DO-WHILE

while(condition)
{
statement(s);
}
	do
	{
	statement(s);
	}
	while(condition);

#include <stdio.h>
void main()
{
int i=-1;
while(i>0)
{
printf("Test");
}
printf("End");
}
Output
End
	#include <stdio.h>
	void main()
	{
	int i=-1;
	do
	{
	printf("Test");
	}while(i>0);
	printf("End");
	}
	Output
	Test
	End