2.1 Introduction of C

"C' seems a strange name for a programming language but it is one of the most widely used languages in the world. C was introduced by Dennis Ritchie in 1972 at Bell Laboratories as a successor of the language called B (Basic Combined Programming Language – BCPL). Since C was developed along with the UNIX operating system it is strongly associated with UNIX. UNIX was developed at Bell Laboratories and it was written almost entirely in C.

For many years, C was used mainly in academic environments. However with the release of C compilers for commercial use and increasing popularity of UNIX, it began to gain wide-spread interest among computer professionals. Various languages such as C++, Visual C++, Java and C# have branched away from C by adding object-orientation and GUI features. Today C compilers are available for a number of operating systems including all flavours of UNIX, Linux, MS-DOS, Windows and Apple Mac.

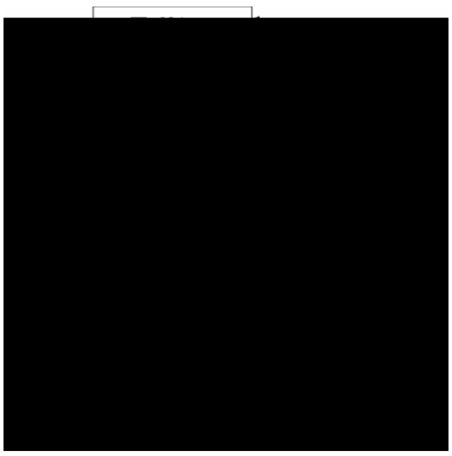
C is a robust language whose rich set of built-in functions and operations can be used to write any complex program. C is well suited to write both commercial applications and system software since it incorporates features of high-level languages and Assembly language. Programs written in C are efficient and fast. Most C programs are fairly *portable;* that is with little or no modification and compiling, C programs can be executed on different operating systems.

The syntax and coding style of C is simple and well structured. Due to this reason most of the modern languages such as C++, Java and C# inherit C coding style . Therefore it is one of the best languages to learn the art of programming. C is also suitable for many complex engineering applications.

2.2 Steps in Developing a Program in C

A programmer uses a text editor to create and modify files containing the C source code. A file containing source code is called a *source file* (C source file s are given the extension **.c).** After a C source file has been created, the programmer must invoke the C compiler before the program can be executed. If the compiler finds no errors in the source code it produces a file containing the machine code (this file referred as the executable file).

The compilation of a C program is infact a three stages process; preprocessing, compiling and linking.



Preprocessing is performed by a program called the *preprocessor.* It modifies the source code (in memory) according to *preprocessor directives* (example: **#define)** embedded in the source code. It also strips comments and unnecessary white spaces from the source code. Preprocessor does not modify the source code stored on disk, every thing is done on the copy loaded into the memory.

Compilation really happens then on the code produced by the preprocessor. The compiler translates the preprocessor-modified source code into *object code* (machine code). While doing so it may encounter syntax errors. If errors are found it will be immediately notifie d to the programmer and compiling will discontinue. If the compiler finds any non-standard codes or conditions which are suspicious but legitimate it will notify to the programmer as warnings and it continues to compile. A well written program should not have any compilation errors or warnings.

Linking is the final step and it combines the program object code with other object codes to produce the executable file. The other object codes come from *run-time libraries*, other libraries, or object files that the programmer has created. Finally it saves the executable code as a file on the disk. If any linker errors are encountered the executable file will not be generated.

2.3 An Example C Program

In order to see the structure of a C program, it is best to start with a simple program. The following code is a C program which displays the message "Hello, World!" on the screen.

```
/*My First Program */
#include <stdio.h>
void main ( )
{
      printf("Hello World!");
}
```

The heart of this program is the function **printf,** which actually does the work. The C language is built from functions like **printf** that execute different tasks. The functions must be used in a framework which starts with the word **main(),** followed by the block containing the function(s) which is marked by braces ({}). The **main()** is a special function that is required in every C program.

The first line starting with characters **/*** and ending with characters ***/** is a *comment.* Comments are used by programmers to add remarks and explanations within the program. It is always a good practice to comment your code whenever possible . Comments are useful in program maintenance. Most of the programs that people write needs to be modified after several months or years. In such cases they may not remember why they have written a program in such a manner. In such cases comments will make a programmer's life easy in understanding the source code and the reasons for writing them. Compiler ignores all the comments and they do not have any effect on the executable program.

Comments are of two types; *single line* comments and *block* comments. Single line comments start with two slashes **{//}** and all the text until the end of the line is considered a comment. Block comments start with characters **/*** and end with characters ***/.** Any text between those characters is considered a block of comments.

Lines that start with a hash **(#)** symbol are called *directives* for the preprocessor. A directive is not a part of the actual program; it is used as a command to the preprocessor to direct the translation of the program. The directive **#include** appears in all programs as it refers to the *standard input output header* file **(stdio.h).** Here, the header file **stdio.h** includes information about the **printf( )** function. When using more than one directive each must appear on a separate line.

A header file includes data types, macros, function prototypes, inline functions and other common declarations. A header file does not include any implementation of the functions declared. The C preprocessor is used to insert the function definitions into the source files and the actual library file which contains the function implementation is linked at link time. There are prewritten libraries of functions such as **printf()** to help us. Some of these functions are very complex and long. Use of prewritten functions makes the programmers life easier and they also allow faster and error free development (since functions are used and tested by many programmers) development.

The function **printf** is embedded into a statement whose end is marked by a semicolon (;). Semicolon indicates the end of a statement to the compiler.

The C language is case sensitive and all C programs are generally written in lowercase letters. Some of the special words may be written in uppercase letters.

2.4 C Tokens

The smallest element identified by the compiler in a source file is called a *token.* It may be a single character or a sequence of characters to form a single item. Tokens can be classified as *keywords, literals, identifiers, operators,* etc. Literals can be further classified as numeric constants, character constants and string constants.

Language specific tokens used by a programming language are called *keywords.* Keywords are also called as *reserved words.* They are defined as a part of the programming language therefore cannot be used for anything else. Any user defined literals or identifiers should not conflict with keywords or compiler directives. Table 2.1 lists keywords supported by the C language.

Table 2.1 – The C language keywords

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

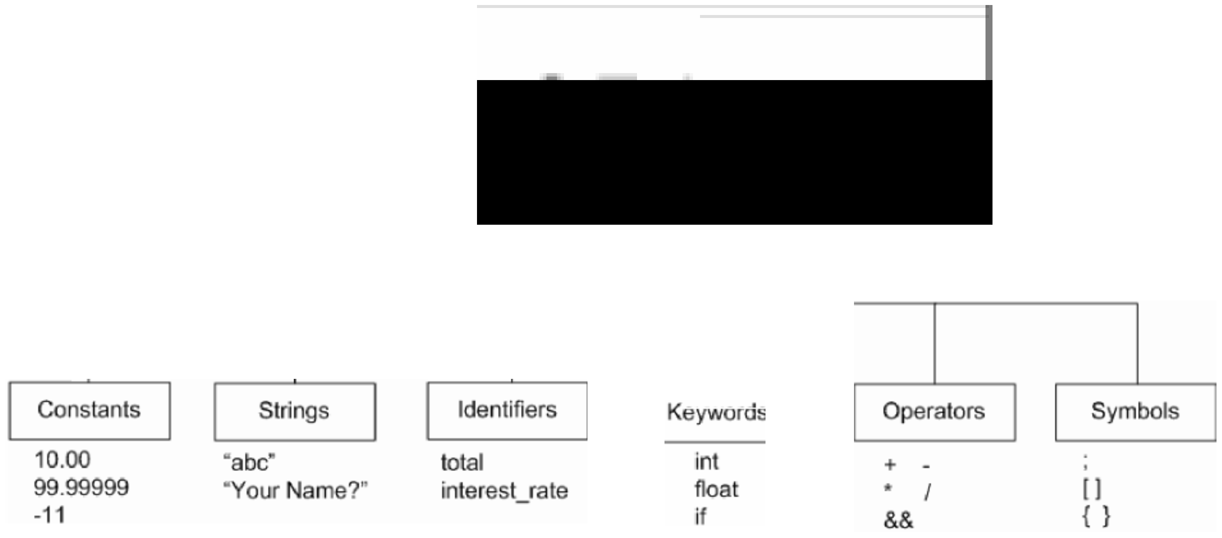
Literals are factual data represented in a language. Numeric constants are an uninterrupted sequence of digits (possibly containing a period). Numerical values such as 123, 10000 and 99.99 are examples. Character constants represents a single character and it is surrounded by single quotation mark ('). Characters such as 'a', 'A', '$' and '4' are examples. A sequence of characters surrounded by double quotation marks (inverted comma "") is called a *string* constant. A statement such as “I like ice cream.” is a string constant.

Identifiers are also referred as names. A valid identifier is composed of a letter followed by a sequence of letters, digits or underscore **(_)** symbols. An identifier must begin with a letter and the rest can be letters, digits or underscores. Identifies are case sensitive; therefore the identifier **a bc** is different from **ABC** or **Abc.**

C identifiers can be very long and so it allows descriptive names like “number_of_students” and “Total_number_of_cars_produced_per_year”. Sometimes a C compiler may consider only the first 32 characters in an identifier. While defining identifiers programmers should follow some of the naming standards for better readability of the program. One such standard is the use of underscores symbol (_) to combine two words (example: sub_total).

Operators are used with operands to build expressions. For example **“4+5”** is an expression containing two operands **(4** and **5)** and one operator **(+** symbol). C supports large number of mathematical and logical operators such as **+, -, *, /, %, ^, &, && , |, ||,** etc. Operators will be discussed in chapter 3.

The C language uses several symbols such as semicolons (;), colons (:), commas (,), apostrophes (‘), quotation marks (“”), braces ([]), brackets ({}), and parentheses (()) to group block of code as a single unit.



2.4.1 Escape Codes

Escape codes are special characters that cannot be expressed otherwise in the source code such as new line, tab and single quotes. All of these characters or symbols are preceded by an inverted (back) slash **(\\).** List of such escape codes are given in Table 2.2. Note that each of these escape codes represents one character, although they cons ists of two characters.

Table 2. 2 – Escape codes

Escape Code	**Meaning**
\a	Audible alert (bell)
\b	Back space
\f	Form feed
\n	New line
\t	Horizontal tab
\v	Vertical tab
\’	Single quote (‘)
\”	Double quote (“)
\?	Question mark (?)
\\\\	Backslash (\\)

Bitwise Operators

They are also called low level operators. C supports six bitwise operators and they are listed in table below. Their precedence (priority) is lower than arithmetic, relational and logical operators

Operator	**Action**
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	One’s complement
>>	Right shift
<<	Left shift

Bitwise operators are used for manipulating data at bit level. These operators are used for testing the bits , or shifting them to the left or to the right . Bitwise operators can be applied only one integer –type operands and not to float or double. There are three types of bitwise operators:

- Bitwise Logical operators

- Bitwise shift operators
- One’s complement operator
- Bitwise Logical operators

Bitwise logical operators performs logical ANDing between two operands . There are three logical bitwise operators:

Bitwise AND (&)

Bitwise OR (|)

Bitwise Exclusive OR(^)

Bitwise AND(&): the result of ANDing opration is 1 if the both the bit have a value of 1 ; otherwise it is 0. let us consider two variable n1=60 and n2 =15. the binary representations of these two variables are:

N1= 0000 0000 0011 1100

N2= 0000 0000 0000 1111

If we execute the statement,

N3=N1&N2

The result will be .

N3=0000 0000 0000 1100

Although the resulting bit pattern represents the decimal number 12, there is no plain connection between the decimal values of these variables

Bitwise OR(|): the result of ORing operation is 1 if eihter of the bit have a value of 1 ; otherwise it is 0. let us consider two variable n1=60 and n2 =15. the binary representations of these two variables are:

N1= 0000 0000 0011 1100

N2= 0000 0000 0000 1111

If we execute the statement,

N3=N1|N2

The result will be .

N3=0000 0000 0011 1111

Again , although the resulting bit pattern represents the decimal number 63, there is no plain connection between the decimal values of these variables

Bitwise Exclusive XOR(^): the result of exclusive ORing operation is 1 only if one of the bit have a value of 1; otherwise it is 0. let us again consider the two variables n1=60 and n2=15. if we execute the statement,

N3=n1^n2;

Then the result will be

```
n1    0000 0000 0011 1100

n2    0000 0000 0000 1111

      ---------------------

       0000 0000  0011 0011
```

Again , although the resulting bit pattern represents the decimal number 53, there is no plain connection between the decimal values of these variables .

- The following example shows the operation of these three logical bitwise operation:

```
#include <stdio.h>

#include <conio.h>

void main()

{

int n1=60,n2=15,AND,OR,XOR;

clrscr();

AND=n1&n2;

OR=n1|n2;

XOR=n1^n2;
printf("AND=%d\n",AND);

printf("OR=%d\n",OR);

printf("XOR=%d\n",XOR);

getch();

}
```

- Bitwise Shift Operators

Bitwise shift operators are used to move bit patterns either to left or to the right. There are two bitwise shift operators:

- Left shift (<<)
- Right shift (>>)

Left shift (<<): the left shift operation causes the operand to be shifted to the left by some bit positions. The general of left-shift operation is -

Operand << n

The bits in the operand are shifted to the left by n positions. The leftmost n bits in the original bit pattern will be lost and the rightmost n bits empty positions will be filled with 0s. For example if n1=60, than if we execute the statement,

```
   n2=n1<<3;
```

The result is calculated as;

```
   n1                  0000  0011  1100

Shift1                 0000 0111  1 000

Shift 2                0000 1111 0000

Shift 3               0001 1110 0000
```

Right shift (>>):   the ritht shift operation causes the operand to be shifted to the right by some bit positions. The general of right-shift operation is -

Operand >> n

The bits in the operand are shifted to the right by n positions. The leftmost n bits will be lost the empty leftmost n bits positions will be filled 0s , if the operand an unsigned integer . If the operation is assigned , then the operand is machine dependent, for example if unsigned int n1=60, then if we execute the statement,

N2=n1>>3;

The result is calculate as;

n1	0000 0000 0011 1100
Shift 1	0000 0000 00011110
Shift 2	0000 0000 0000 1111
Shift 3	0000 0000 0000 0111

The following example shows the operation of these two shift Bitwise perators:

```
#include <stdio.h>

   #include <conio.h>

  void main()

  {

  unsigned int n1=60,left,right;

  clrscr();

  left=n1<<3;

  right=n1>>3;

  printf("left=%d", left);
  printf("right=%d",right);

  getch();

  }
```

- Bitwise one's compliment

Bitwise one's compliment operator is a unary operator which inverts all the bits presented by its operand. This means that all 0s become 1s and all 1s become 0s. For example , if n1=60 than if we execute the statement,

n2=~n1

Than the result is calculated as,

n1= 0011 1100

n2= 1100 0011

2.5 Data Types

A data type in a programming language is a set of data with values having predefined characteristics such as integers and characters. The language usually specifies the range of values for a given data type, how the values are processed by the computer and how they are stored. Storage representations and machine instructions to handle data types differ form machine to machine.

The variety of data types available allows the programmer to select the type appropriate to the needs of the application as well as the machine. C supports a number of data types; if they are not enough programmers can also define their own data types. C supports three classes of data types:

1. Primitive (or basic) data types – these are the fundamental data types supported by the language. These can be classified as integer types, floating point types and character types.
2. User defined data types – based on the fundamental data types users can define their own data types. These include type defined data types (using **typedef** keyword) and enumerated types (using **enum** keyword).
3. Derived data types – programmers can derive data types such as arrays, structures, unions and pointers by combining several data types together.

2.5.1 Primitive Data Types

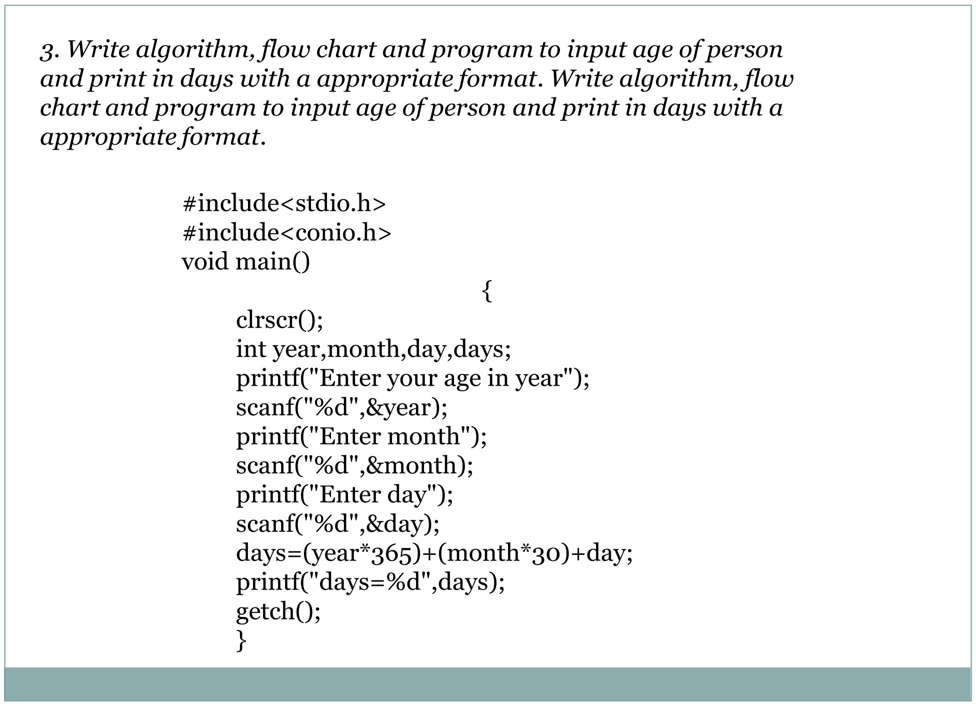
The C language supports five primitive data types; namely integers **(int)** floating point numbers **(float),** double precision floating point numbers **(double),** characters **(char)** and void **(void).** Many of these data types can be further extended as **long int** and **long double.** Each of these data types requires different storage capacities and has different range of values depending on the hardware (see Table 2.3). Character **(char)** type is considered as an integer type and actual characters are represented based on their ASCII value.

Table 2.3 – Basic C data types (on a 32-bit machine)

Data Type	**Size in Bits**	**Range of values**
Char	8	-128 to +127
Int	32	-2147483648 to 2147483647
Float	32	3.4e-38 to 3.4e+38 (accuracy up to 7 digits)
Double	64	1.7e-308 to 1.7e+308 (accuracy up to 15 digits)
Void	0	Without value (null)

Table 2.4 – Basic C data types and modifiers (on a 32-bit machine)

Data Type	**Size in Bits**	**Range of values**
Char	8	-128 to +127
unsigned char	8	0 to 255



3. Write algorithm, flow chart and program to input age of person and print in days with a appropriate format. Write algorithm, flow chart and program to input age of person and print in days with a appropriate format.

```
#include<stdio.h>
#include<conio.h>
void main()
                                        {
       clrscr();
       int year,month,day,days;
       printf("Enter your age in year");
       scanf("%d",&year);
       printf("Enter month");
       scanf("%d",&month);
       printf("Enter day");
       scanf("%d",&day);
       days=(year*365)+(month*30)+day;
       printf("days=%d",days);
       getch();
       }
```

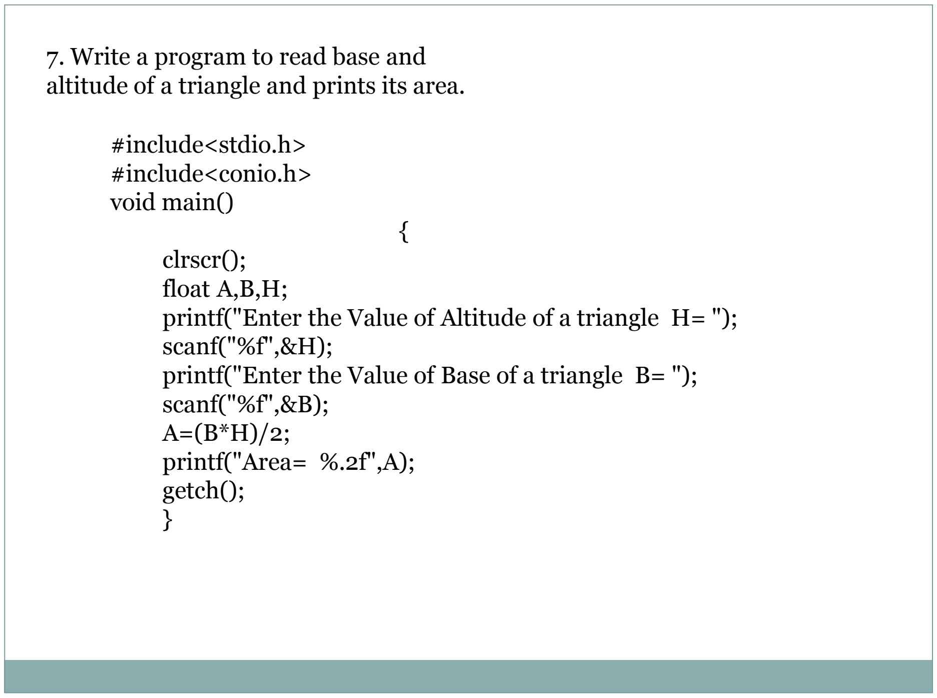
4. Write algorithm, flow chart and program to input length & breadth of a room and calculate and print its area and perimeter.

```
#include<stdio.h>
#include<conio.h>
void main()
                                {
       clrscr();
       int l,b,A,P;
       printf("Enter the L");
       scanf("%d",&l);
       printf("Enter the B");
       scanf("%d",&b);
       A=l*b;
       P=2*(l+b);
```

```
        printf("%d,%d",A,P);
        getch();
        }
```

5. Write a program to read the radius of a sphere and compute its surface area and volume.

```
include<math.h>
#include<stdio.h>
#include<conio.h>
void main()
                                {
        clrscr();
        float A,V,r;
        float PI=3.14;
        printf("Enter the Value of Radious :");
        scanf("%f",&r);
        A=4*PI*pow(r,2);
        V=4/3*PI*pow(r,3);
        printf("Area :%.2f\n",A);
        printf("Volume :%.2f",V);
        getch();
        }
```



7. Write a program to read base and
altitude of a triangle and prints its area.

```
#include<stdio.h>
#include<conio.h>
void main()
                                {
        clrscr();
        float A,B,H;
        printf("Enter the Value of Altitude of a triangle  H= ");
        scanf("%f",&H);
        printf("Enter the Value of Base of a triangle  B= ");
        scanf("%f",&B);
        A=(B*H)/2;
        printf("Area=  %.2f",A);
        getch();
        }
```

8. Write a program to calculate total amount of (p)kept in bank for (n) years at the rate of (r) simple interest per annum.

```
#include<stdio.h>
#include<conio.h>
void main()
                                    {
       clrscr();
       float I,P,T,R;
       printf("Enter the Value of P = ");
       scanf("%f",&P);
       printf("Enter the Value of T= ");
       scanf("%f",&T);
       printf("Enter the Value of R= ");
       scanf("%f",&R);
       I=(P*T*R)/100;
       printf("Interest=  %.2f",I);
       getch();
       }
```

10. Write program to input 5 digit integer and print sum of digit in it.

```
#include<stdio.h>
#include<conio.h>
void main(){
       clrscr();
       int sum,d1,d2,d3,d4,d5,n,n1,n2,n3,n4,n5;
       printf("Enter the Value of Five Digit = ");
       scanf("%d",&n);
       d1=n%10;
       n1=n/10;
       d2=n1%10;
       n2=n1/10;
       d3=n2%10;
       n3=n2/10;
       d4=n3%10;
       n4=n3/10;
       d5=n4%10;
       sum=d1+d2+d3+d4+d5;
       printf("sum=  %d",sum);
       getch();
       }
```

12. Write algorithm, flow chart and program to find out the given number is negative or positive.

```
#include<stdio.h>
#include<conio.h>
void main()
                                        {
       clrscr();
       int n;
       printf("enter the value of n= ");
       scanf("%d",&n);
       if(n>=0)
       printf("positive number  ");
       else
       printf("negative number  ");
       getch();
       }
```

13. Write algorithm, flow chart and program to find out the given number is odd or even.

```
#include<stdio.h>
#include<conio.h>
void main()
                                        {
       clrscr();
       int x;
       printf("Enter the value of x= ");
       scanf("%d",&x);
       if(x%2==0)
       printf("even number  ");
       else
       printf("odd number  ");
       getch();
       }
```

```
#include <stdio.h>
 #include <conio.h>
 void  main()
 {
 icte:
 clrscr();
 int s,a,b,sum,product,division;
 //printf("enter any two numbers a and b:\n
");
 //scanf("%d%d",&a,&b);
 printf("case 1: sum\n");
 printf("case 2:produt\n");
 printf("case 3: division\n");
 scanf("%d",&s);
 switch(s)

 {
 case 1:
 printf("enter any two numbers a and b:\n
");
 scanf("%d%d",&a,&b);
 sum=a+b;
 printf("Sum=%d",sum);
 break;
case 2:
 printf("enter any two numbers a and b:\n ");
 scanf("%d%d",&a,&b);
 product=a*b;
 printf("product=%d",product);
 break;
 case 3:
 printf("enter any two numbers a and b:\n ");
 scanf("%d%d",&a,&b);
 division=a%b;
 printf("division=%d",division);
 }
  getch();
  goto icte;
  }
```

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

 int
s,a=4,b=4,sum,product
,division;
 printf("enter any case
:");
 scanf("%d",&s);
 switch(s)
 {
 case 1:
 sum=a+b;

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

```
case 2:
 product=a*b;
 printf("product=%d",product);
 break;
 case 3:
 division=a%b;
 printf("division=%d",division);
 }
  getch();

  }
```