**C Programming Language**

C programming language was invented by Dennis Ritchie at the Bell Laboratories in 1972. It was invented for implementing UNIX operating system. C is most widely used programming language even today. C Programming is an ANSI/ISO standard and powerful programming language for developing real time applications. All other programming languages were derived directly or indirectly from C programming concepts.

**History of C Language**

* C programming language features were derived from an earlier language called “B” (Basic Combined Programming Language – BCPL)
* In 1978, Dennis Ritchie and Brian Kernighan published the first edition “The C Programming Language” and commonly known as K&R C
* In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or “ANSI C”, was completed late 1988.

**C PROGRAMMING LANGUAGE STANDARDS:**

* C89/C90 standard – First standardized specification for C language was developed by the American National Standards Institute in 1989. C89 and C90 standards refer to the same programming language.
* C99 standard – Next revision was published in 1999 that introduced new features like advanced data types and other changes.

**C11 AND EMBEDDED C LANGUAGE:**

* C11 standard adds new features to C programming language and library like type generic macros, anonymous structures, improved Unicode support, atomic operations, multi-threading and bounds-checked functions. It also makes some portions of the existing C99 library optional and improves compatibility with C++.
* Embedded C includes features not available in C like fixed-point arithmetic, named address spaces, and basic I/O hardware addressing.
* Operating systems, C compiler and all UNIX application programs are written in C language
* It is also called as procedure oriented programming language. The C language is reliable, simple and easy to use. C has been coded in assembly language.

**FEATURES OF C PROGRAMMING LANGUAGE:**

C language is one of the powerful language. Below are some of the features of C language.

* Reliability
* Portability
* Flexibility
* Interactivity
* Modularity
* Efficiency and Effectiveness

**USES OF C PROGRAMMING LANGUAGE:**

The C programming language is used for developing system applications that forms a major portion of operating systems such as Windows, UNIX and Linux. Below are some examples of C being used.

* Database systems
* Graphics packages
* Word processors
* Spreadsheets
* Operating system development
* Compilers and Assemblers
* Network drivers
* Interpreters

**WHICH LEVEL IS C LANGUAGE BELONGING TO?**

There are 3 levels of programming languages. They are,

**Middle Level languages:**Middle level languages don’t provide all the built-in functions found in high level languages, but provides all building blocks that we need to produce the result we want. Examples: C, C++

**High Level languages:**High level languages provide almost everything that the programmer might need to do as already built into the language. Example: Java, Python

**Low Level languages:**Low level languages provides nothing other than access to the machines basic instruction set. Example: Assembler

**Structure oriented language:**

* In this type of language, large programs are divided into small programs called functions
* Prime focus is on functions and procedures that operate on the data
* Data moves freely around the systems from one function to another
* Program structure follows “Top Down Approach”

Examples: C, Pascal, ALGOL and Modula-2

**Non structure oriented language:**

There is no specific structure for programming this language. Examples: BASIC, COBOL, FORTRAN

**BASIC STRUCTURE OF A C PROGRAM:**

Structure of C program is defined by set of rules called protocol, to be followed by programmer while writing C program. All C programs are having sections/parts which are mentioned  below.

* Documentation section
* Link Section
* Definition Section
* Global declaration section
* Function prototype declaration section
* Main function
* User defined function definition section

**DESCRIPTION FOR EACH SECTION OF THE C PROGRAM:**

|  |  |
| --- | --- |
| Sections | Description |
| Documentation section | We can give comments about the program, creation or modified date, author name etc in this section. The characters or words or anything which are given between “/\*” and “\*/”, won’t be considered by C compiler for compilation process.These will be ignored by C compiler during compilation. Example : /\* comment line1 comment line2 comment 3 \*/ |
| Link Section | Header files that are required to execute a C program are included in this section |
| Definition Section | In this section, variables are defined and values are set to these variables. |
| Global declaration section | Global variables are defined in this section. When a variable is to be used throughout the program, can be defined in this section. |
| Function prototype declaration section | Function prototype gives many information about a function like return type, parameter names used inside the function. |
| Main function | Every C program is started from main function and this function contains two major sections called declaration section and executable section. |
| User defined function section | User can define their own functions in this section which perform particular task as per the user requirement. |

**WRITE A C PROGRAM WITH EXPLANATION**

Below are few commands and syntax used in C programming to write a simple C program.

|  |  |
| --- | --- |
| C Basic commands | Explanation |
| #include <stdio.h> | This is a preprocessor command that includes standard input output header file(stdio.h) from the C library before compiling a C program |
| int main() | This is the main function from where execution of any C program begins. |
| { | This indicates the beginning of the main function. |
| /\*\_some\_comments\_\*/ | whatever is given inside the command “/\*   \*/” in any C program, won’t be considered for compilation and execution. |
| printf(“Hello\_World! “); | printf command prints the output onto the screen. |
| getch(); | This command waits for any character input from keyboard. |
| return 0; | This command terminates C program (main function) and returns 0. |
| } | This indicates the end of the main function. |

All syntax and commands in C programming are case sensitive. Also, each statement should be ended with semicolon (;) which is a statement terminator.

**STEPS TO WRITE C PROGRAMS AND GET THE OUTPUT**

Below are the steps to be followed for any C program to create and get the output. This is common to all C program and there is no exception whether its a very small C program or very large C program.

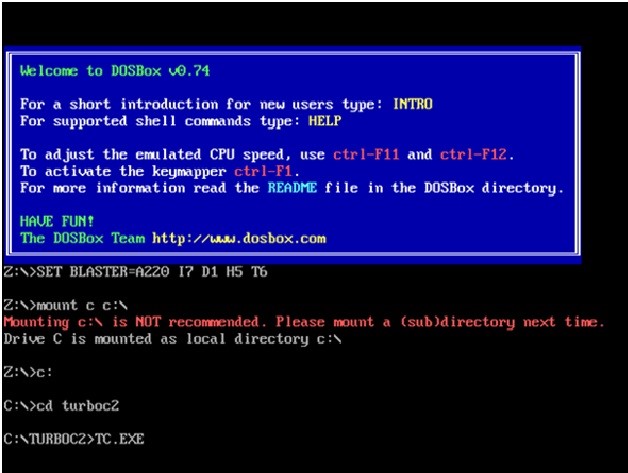
* Create
* Compile
* Execute or Run
* Get the Output

**How to install C Software?**

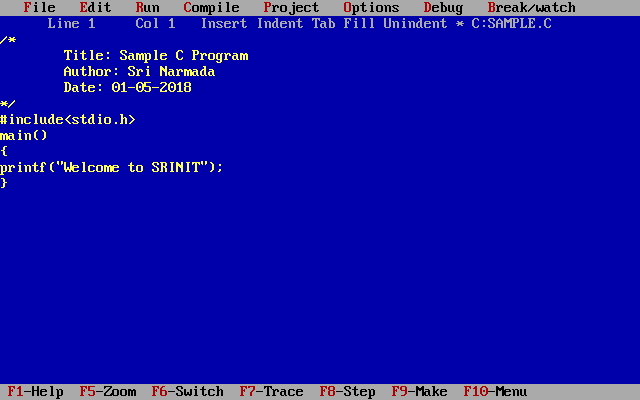
Open the cd double click on dosbox software folder. Double click on DOSBox0.74-win32 – installer. click yes. Click next, click next, click install, click close. It creates a dosbox icon in desktop. Copy the turboc2 folder from cd paste into desktop. Then cut turboc2 from desktop paste into C drive. If it asks administrator permission click yes. Then remove the cd.

**How to open C Software?**

Double click on dosbox icon in desktop it opens.

[](https://2.bp.blogspot.com/-DY7JefccxlU/WQyDo05Ql_I/AAAAAAAAAVE/Nr9s09UzT0QTSIDwWUpY8aSz0UB6bwHGwCEw/s1600/1.jpg)

Z:\>mount c c:\ enter  
Z:\>c: enter  
C:\>cd turboc2 enter  
C:\turboc2>tc.exe enter  
Note:- All underline letters are we have to type.  
It opens.  
For full screen use alt+enter.

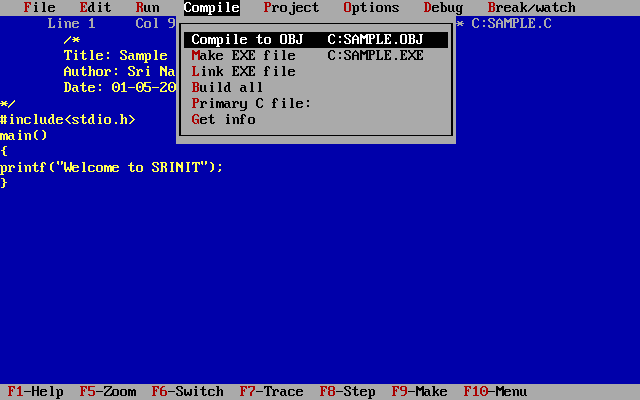
Press esc two times. Then write the program  
  


**How to save?**

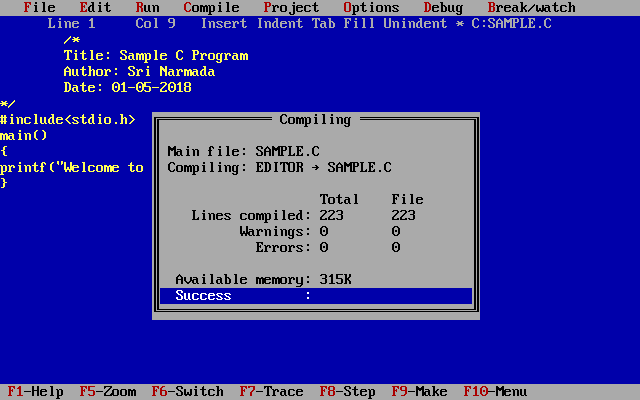
Press F2 functional key  
C:\turboc2\sample.c enter

**How to compile?**

Alt+c enter

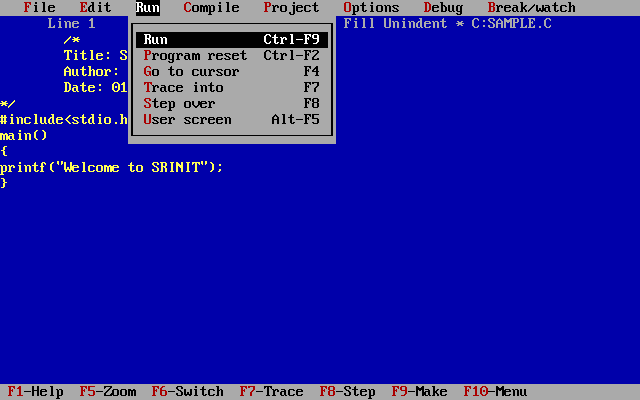


If it having errors or warnings enter  
Then it shows.  
If it is not having any errors or warnings  
It shows success press any key Enter



**How to Run (or) Execute?**

Alt+r enter

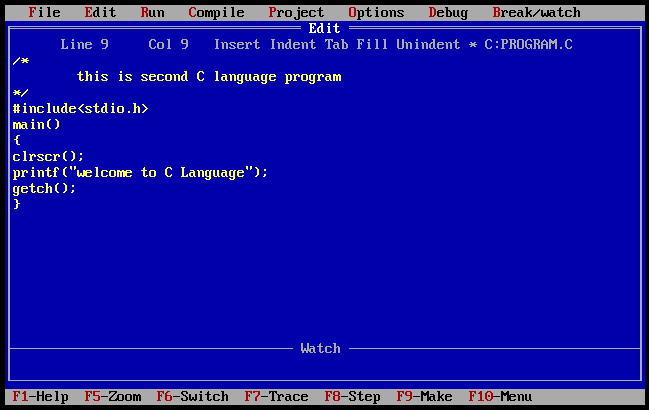


**How to See the output?**

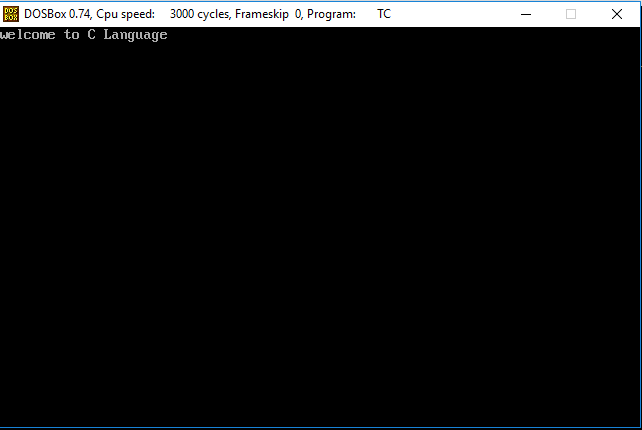
Alt+f5 functional key



**Write a program with clrscr and getch function**

[](https://1.bp.blogspot.com/-EDDXO3wREMo/WQySwGGQ7sI/AAAAAAAAAVo/NPIVfsWW0xclUKDsbCjFhnrfOZ58-eIUgCLcB/s1600/2.png)

**Output**

[](https://2.bp.blogspot.com/-GpW9hksnmRo/WQyS8YMuC6I/AAAAAAAAAVs/J6VYZMZYAuQ0E_jqbbI3EsM_FKtOx5FWgCLcB/s1600/3.png)

**Explanation of the program**

# include <stdio.h> is the link section or source file or header file.

# is a preprocessor. The Preprocessor command tells the compiler to include the stdio.h file in the program. include is a directive.  
Include means just what you would think it does.  
Std means standard.  
I means input.  
O means output.  
H means header file.  
Preprocessor helps to link to header file.  
main():- Program execution stars from the main().  
clrscr():- It clears the screen.  
printf():-  
The output functions are used to display the values on the console. (or) It will read the values from the variables and displayed on the monitor.  
getch():-  
getch() is used to read a single character from the user without echoing.

**C Character Set**

Whenever we write any C program then it consists of different statements.

Character Set Consists Of –

|  |  |
| --- | --- |
| Types | Character Set |
| Lowercase Letters | a-z |
| Uppercase Letters | A to Z |
| Digits | 0-9 |
| Special Characters | !@#$%^&\* |
| White Spaces OR Escape Sequences | Tab Or New line Or Space |

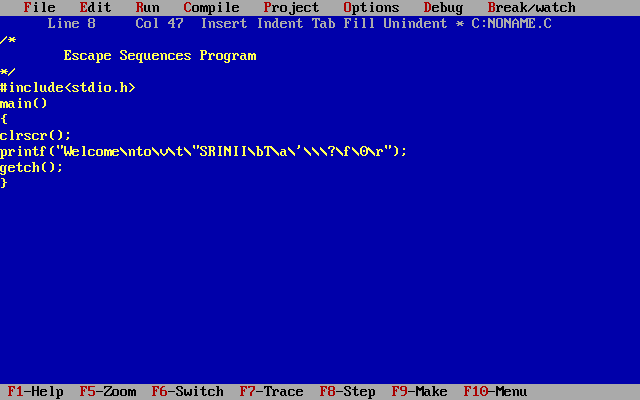
Valid C Characters : Special Characters are

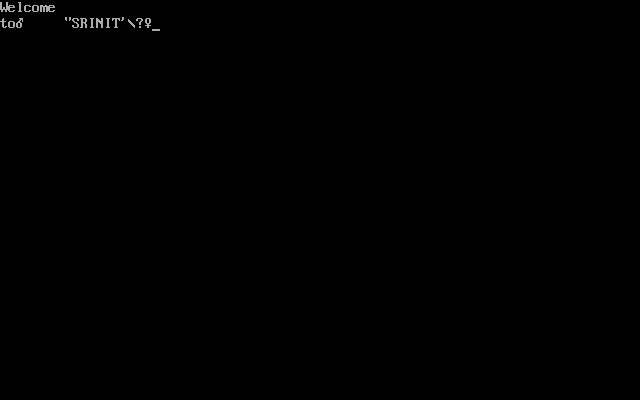
|  |  |
| --- | --- |
| Symbol | Meaning |
| ~ | Tilde |
| ! | Exclamation mark |
| # | Number sign |
| $ | Dollar sign |
| % | Percent sign |
| ^ | Caret |
| & | Ampersand |
| \* | Asterisk |
| ( | Left parenthesis |
| ) | Right parenthesis |
| \_ | Underscore |
| + | Plus sign |
| | | Vertical bar |
| \ | Backslash |
| ` | Apostrophe |
| – | Minus sign |
| = | Equal to sign |
| { | Left brace |
| } | Right brace |
| [ | Left bracket |
| ] | Right bracket |
| : | Colon |
| ” | Quotation mark |
| ; | Semicolon |
| < | Opening angle bracket |
| > | Closing angle bracket |
| ? | Question mark |
| , | Comma |
| . | Period |
| / | Slash |

**Escape Sequences**

An escape sequence in C language is a sequence of characters that doesn't represent itself when used inside string literal or character. C uses some backslash characters in output functions for formatting the output. It is composed of two or more characters starting with backslash \ But it is treated as  a single character.

|  |  |  |
| --- | --- | --- |
| Escape Sequence | Meaning |  |
| \a | Alarm or Beep | (beep sound) |
| \b | Backspace |  |
| \f | Form Feed |  |
| \n | New Line |  |
| \r | Carriage Return |  |
| \t | Tab (Horizontal) | (default 8 spaces) |
| \v | Vertical Tab | (default 1 line) |
| \\ | Backslash |  |
| \' | Single Quote |  |
| \" | Double Quote |  |
| \? | Question Mark |  |
| \nnn | octal number |  |
| \xhh | hexadecimal number |  |
| \0 | Null |  |





**Data Types**

C data types are defined as the data storage format that a variable can store a data to perform a specific operation.

Data types are used to define a variable before to use in a program.

Size of variable, constant and array are determined by data types.

There are four data types in C language. They are,

|  |  |
| --- | --- |
| Types | Data Types |
| Basic data types | int, char, float, double |
| Enumeration data type | Enum |
| Derived data type | pointer, array, structure, union |
| Void data type | Void |

**1. BASIC DATA TYPES IN C LANGUAGE:**

**1.1. INTEGER DATA TYPE:**

Integer data type allows a variable to store numeric values.

“int” keyword is used to refer integer data type.

The storage size of int data type is 2.

It varies depend upon the processor in the CPU that we use.  If we are using 16 bit processor, 2 byte  (16 bit) of memory will be allocated for int data type.

int (2 byte) can store values from -32,768 to +32,767

If you want to use the integer value that crosses the above limit, you can go for “long int” and “long int” for which the limits are very high.

long int (4 byte) can store values from -2,147,483,648 to +2,147,483,647.

Note:

We can’t store decimal values using int data type.

If we use int data type to store decimal values, decimal values will be truncated and we will get only whole number.

In this case, float data type can be used to store decimal values in a variable.

**1.2. CHARACTER DATA TYPE:**

Character data type allows a variable to store only one character.

Storage size of character data type is 1. We can store only one character using character data type.

“char” keyword is used to refer character data type.

For example, ‘A’ can be stored using char datatype. You can’t store more than one character using char data type.

**1.3. FLOATING POINT DATA TYPE:**

Floating point data type consists of 2 types. They are,

float

double

**1. FLOAT:**

Float data type allows a variable to store decimal values.

Storage size of float data type is 4. This also varies depend upon the processor in the CPU as “int” data type.

We can use up-to 6 digits after decimal using float data type.

The range for float datatype is from 1.2E-38 to 3.4E+38.

For example, 10.456789 can be stored in a variable using float data type.

**2. DOUBLE:**

Double data type is also same as float data type which allows up-to 15 digits after decimal using double data type.

The range for double datatype is from 2.3E-308 to 1.7E+308.

**3. Long Double**

Long Double data type is also same as double data type which allows up-to

19 decimal places after decimal using long double data type.

The range for double datatype is from 3.4E-4932 to 1.1E+4932.

**MODIFIERS IN C LANGUAGE:**

There are 5 modifiers available in C language. They are,

short

long

signed

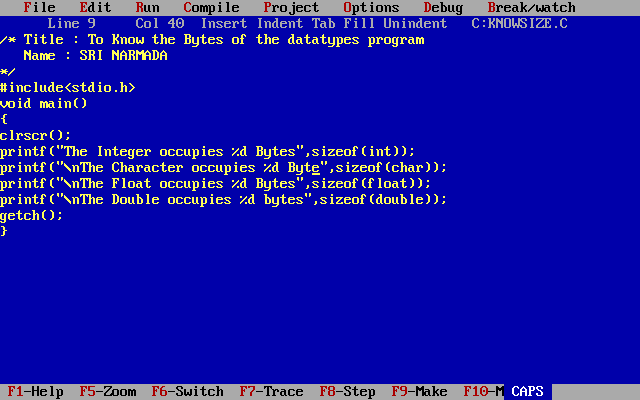
unsigned

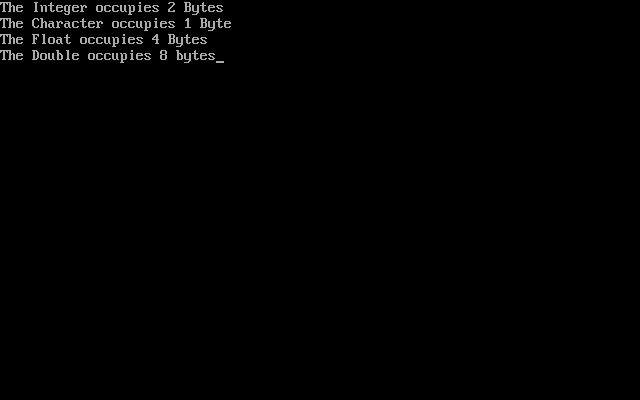
Below table gives the detail about the storage size of each C basic data type in 16 bit processor.

|  |  |
| --- | --- |
| C Data types / storage Size | Range |
| char / 1 | –127 to 127 |
| int / 2 | –32,767 to 32,767 |
| float / 4 | 1.2E-38 to 3.4E+38 with six digits of precision |
| double / 8 | 2.3E-308 to 1.7E+308 with fifteen digits of precision |
| long double / 10 | 3.4E-4932 to 1.1E+4932 with nineteen digits of precision |
| long int / 4 | –2,147,483,647 to 2,147,483,647 |
| short int / 2 | –32,767 to 32,767 |
| unsigned short int / 2 | 0 to 65,535 |
| signed short int / 2 | –32,767 to 32,767 |
| signed long int / 4 | –2,147,483,647 to 2,147,483,647 |
| unsigned long int / 4 | 0 to 4,294,967,295 |

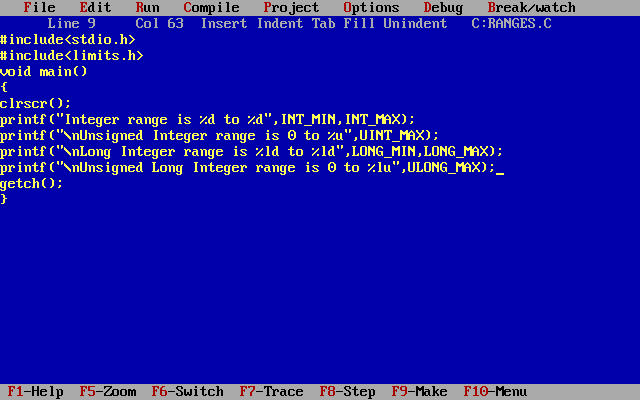
**/\* SIZEOF() FUNCTION IN C LANGUAGE \*/**

sizeof() function is used to find the memory space allocated for each C data types.





**Write a Program to Know the ranges.**



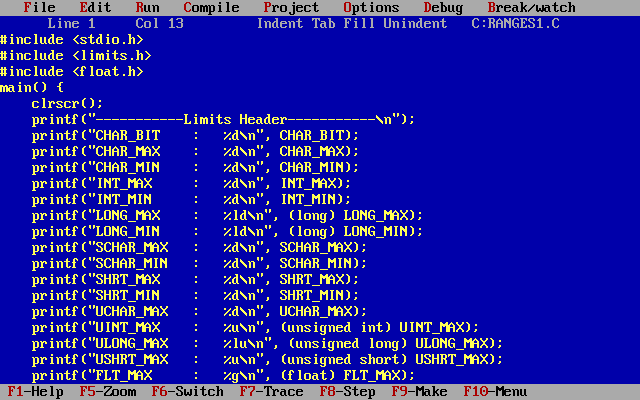


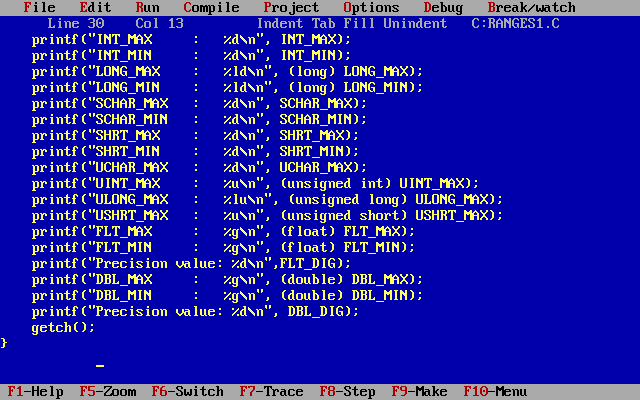
In C programming language, printf() function is used to print the “character, string, float, integer, octal and hexadecimal values” onto the output screen.

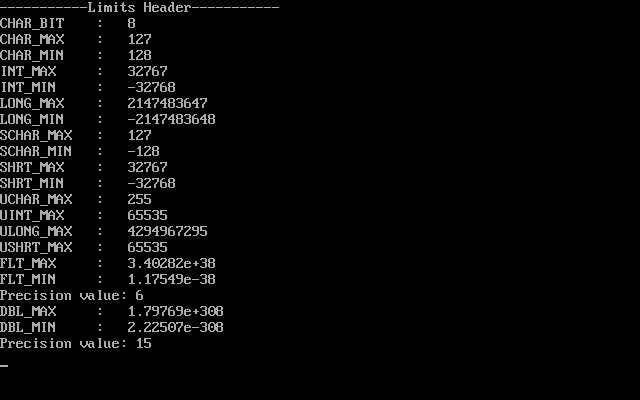
We use printf() function with %d format specifier to display the value of an integer variable.

Similarly %c is used to display character, %f for float variable, %s for string variable, %lf for double and %x for hexadecimal variable.

The header file float.h defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs.







**2. ENUMERATION DATA TYPE:**

Enumeration data type consists of named integer constants as a list.

It start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.

Enum syntax in C:  
enum identifier [optional{ enumerator-list }];

Enum example in C:

enum month { Jan, Feb, Mar }; or  
/\* Jan, Feb and Mar variables will be assigned to 0, 1 and 2 respectively by default \*/  
enum month { Jan = 1, Feb, Mar };  
/\* Feb and Mar variables will be assigned to 2 and 3 respectively by default \*/  
enum month { Jan = 20, Feb, Mar };  
/\* Jan is assigned to 20. Feb and Mar variables will be assigned to 21 and 22 respectively by default \*/

The above enum functionality can also be implemented by “#define” preprocessor directive as given below. Above enum example is same as given below.

#define Jan 20;  
#define Feb 21;  
#define Mar 22;

**C – ENUM EXAMPLE PROGRAM:**

#include <stdio.h>

int main()

{

   enum MONTH { Jan = 0, Feb, Mar };

   enum MONTH month = Mar;

   if(month == 0)

   printf("Value of Jan");

   else if(month == 1)

   printf("Month is Feb");

   if(month == 2)

   printf("Month is Mar");

}

**3. DERIVED DATA TYPE IN C LANGUAGE:**

Array, pointer, structure and union are called derived data type.

**4. VOID DATA TYPE IN C LANGUAGE:**

Void is an empty data type that has no value.

This can be used in functions and pointers. To know how to use void data type in function with simple call by value and call by reference example programs.

**Variable**

C variable is a named location in a memory where a program can manipulate the data. This location is used to hold the value of the variable. The value of the C variable may get change in the program. C variable might be belonging to any of the data type like int, float, char etc.

**RULES FOR NAMING C VARIABLE:**

* Variable name must begin with letter or underscore.
* Variables are case sensitive
* They can be constructed with digits, letters but in first place cannot allow digits.
* No special symbols are allowed other than underscore.
* s1, height, \_value are some examples for variable name

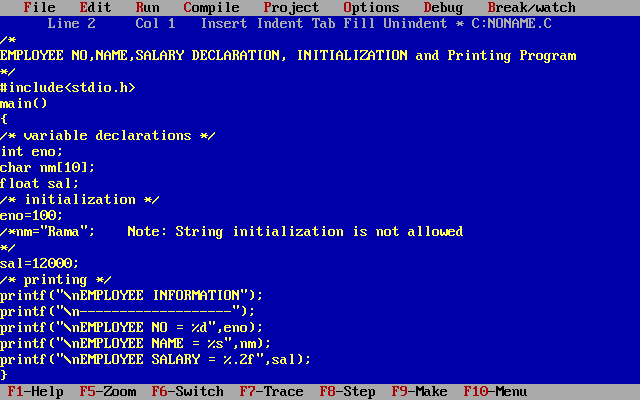
**DECLARING & INITIALIZING C VARIABLE:**

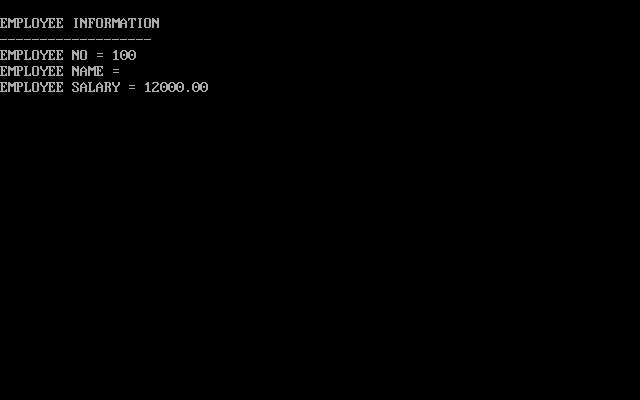
Variables should be declared in the C program before to use.

Memory space is not allocated for a variable while declaration. It happens only on variable definition.

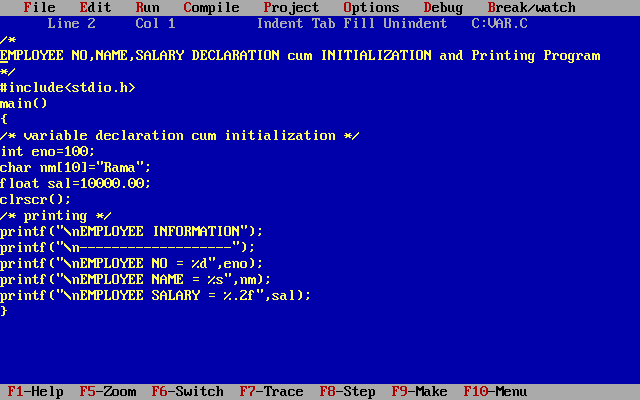
Variable initialization means assigning a value to the variable.

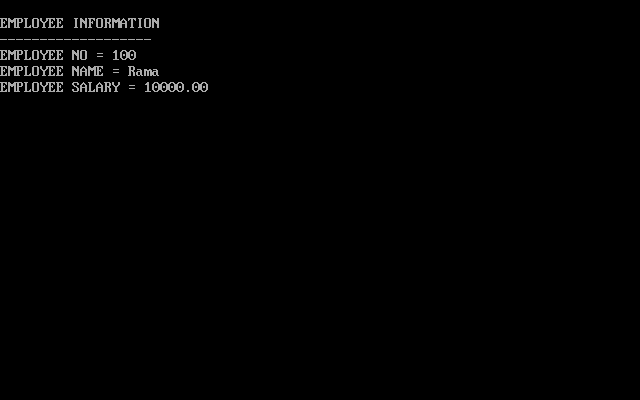
|  |  |
| --- | --- |
| Type | Syntax |
| Variable declaration | data\_type variable\_name; Example: int x; float y, z; char flat, ch; |
| Variable initialization | variable\_name = value;  Example: x = 50, y = 30.0f;z=13.3f; flat = ‘x’; ch=’l’; |





|  |  |
| --- | --- |
| Type | Syntax |
| Variable declaration cum initialization | data\_type variable\_name=value; int eno=100; |





**printf and scanf in C Language**

printf() and scanf() functions are inbuilt library functions in C programming language which are available in C library by default. These functions are declared and related macros are defined in “stdio.h” which is a header file in C language. We have to include “stdio.h” file as shown in below C program to make use of these printf() and scanf() library functions in C language.

1. **PRINTF() FUNCTION:**

In C programming language, printf() function is used to print the “character, string, float, integer, octal and hexadecimal values” onto the output screen.

We use printf() function with %d format specifier to display the value of an integer variable. Similarly %c is used to display character, %f for float variable, %s for string variable, %lf for double and %x for hexadecimal variable. To generate a newline,we use “\n” in C printf() statement.

**Note:**

C language is case sensitive. For example, printf() and scanf() are different from Printf() and Scanf(). All characters in printf() and scanf() functions must be in lower case.

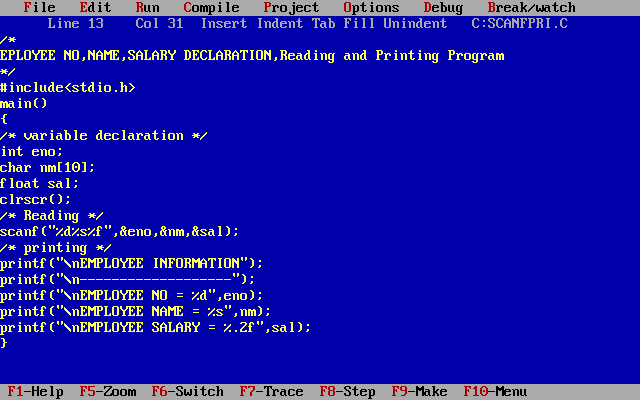
**2. SCANF() FUNCTION:**

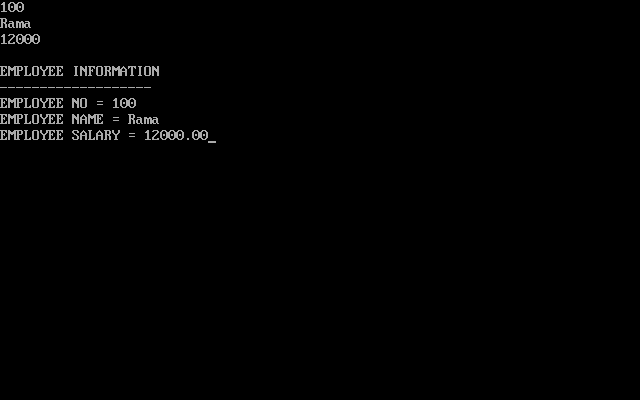
scanf() function is used to read character, string, numeric data from keyboard.

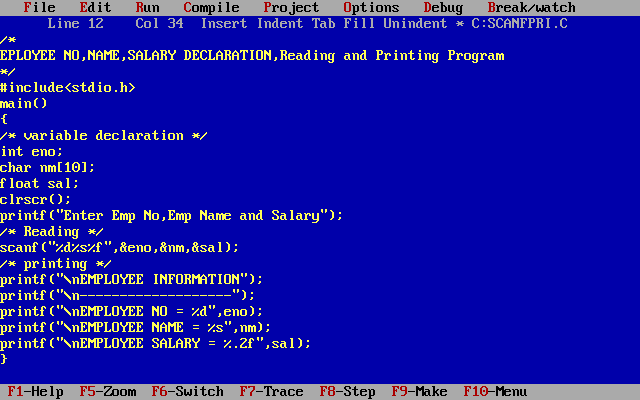
Consider below example program where user enters a character. This value is assigned to the variable “ch” and then displayed.

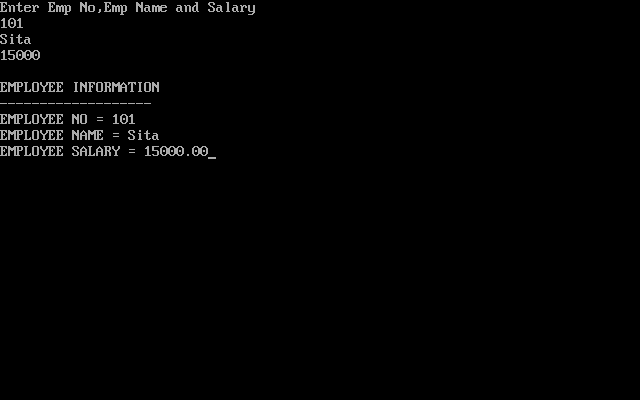
Then, user enters a string and this value is assigned to the variable “str” and then displayed.

**EXAMPLE PROGRAM FOR PRINTF() AND SCANF() FUNCTIONS**









The format specifier %d is used in scanf() statement. So that, the value entered is received as an integer and %s for string.

Ampersand is used before variable name “eno” in scanf() statement as &eno.

It is just like in a pointer which is used to point to the variable.

**C TOKENS:**

C tokens are the basic buildings blocks in C language which are constructed together to write a C program. Each and every smallest individual units in a C program are known as C tokens.

C tokens are of six types. They are,

* Keywords
* Identifiers
* Constants
* Strings
* Special symbols  (eg: (), {} etc),
* Operators              (eg: +, /,-,\* etc)

**1. KEYWORDS:**

Keywords are pre-defined words in a C compiler.

Each keyword is meant to perform a specific function in a C program.

Since keywords are referred names for compiler, they can’t be used as variable name.

C language supports 32 keywords which are given below.

|  |  |
| --- | --- |
| [auto](http://fresh2refresh.com/c/c-storage-class-specifiers/) | [double](http://fresh2refresh.com/c/c-data-types/) |
| [int](http://fresh2refresh.com/c/c-data-types/) | [struct](http://fresh2refresh.com/c/c-structures/) |
| [const](http://fresh2refresh.com/c/c-constants/) | [float](http://fresh2refresh.com/c/c-data-types/) |
| [short](http://fresh2refresh.com/c/c-data-types/) | [unsigned](http://fresh2refresh.com/c/c-data-types/) |
| [break](http://fresh2refresh.com/c/c-case-control-statements/) | [else](http://fresh2refresh.com/c/c-decision-control/) |
| [long](http://fresh2refresh.com/c/c-data-types/) | [switch](http://fresh2refresh.com/c/c-case-control-statements/) |
| [continue](http://fresh2refresh.com/c/c-case-control-statements/) | [for](http://fresh2refresh.com/c/c-loop-control-statements/) |
| [signed](http://fresh2refresh.com/c/c-data-types/) | [void](http://fresh2refresh.com/c/c-data-types/) |

|  |  |
| --- | --- |
| [case](http://fresh2refresh.com/c/c-case-control-statements/) | [enum](http://fresh2refresh.com/c/c-data-types/) |
| [register](http://fresh2refresh.com/c/c-storage-class-specifiers/) | [typedef](http://fresh2refresh.com/c/c-typedef/) |
| [default](http://fresh2refresh.com/c/c-case-control-statements/) | [goto](http://fresh2refresh.com/c/c-case-control-statements/) |
| [sizeof](http://fresh2refresh.com/c/c-data-types/) | [volatile](http://fresh2refresh.com/c/c-type-qualifiers/) |
| [char](http://fresh2refresh.com/c/c-data-types/) | [extern](http://fresh2refresh.com/c/c-storage-class-specifiers/) |
| [return](http://fresh2refresh.com/c/c-function/c-function-arguments-and-return-values/) | [union](http://fresh2refresh.com/c/c-union/) |
| [do](http://fresh2refresh.com/c/c-loop-control-statements/) | [if](http://fresh2refresh.com/c/c-decision-control/) |
| [static](http://fresh2refresh.com/c/c-storage-class-specifiers/) | [while](http://fresh2refresh.com/c/c-loop-control-statements/) |

2. **IDENTIFIERS:**

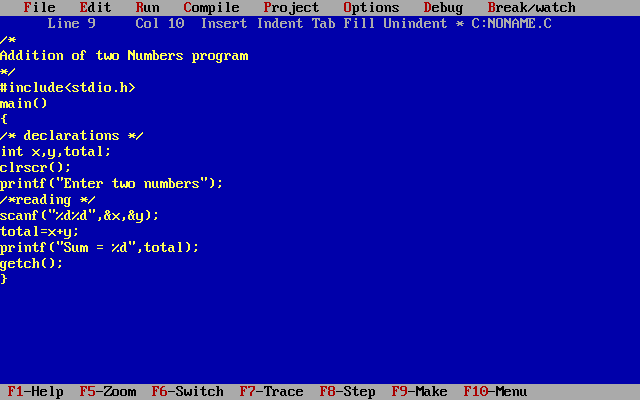
Each program elements in a C program are given a name called identifiers.

Names given to identify Variables, functions and arrays are examples for identifiers. eg. x is a name given to integer variable in above program.

**RULES FOR CONSTRUCTING IDENTIFIER NAME IN C:**

1. First character should be an alphabet or underscore.
2. Succeeding characters might be digits or letter.
3. Punctuation and special characters aren’t allowed except underscore.
4. Identifiers should not be keywords.

**C TOKENS EXAMPLE PROGRAM:**



where,

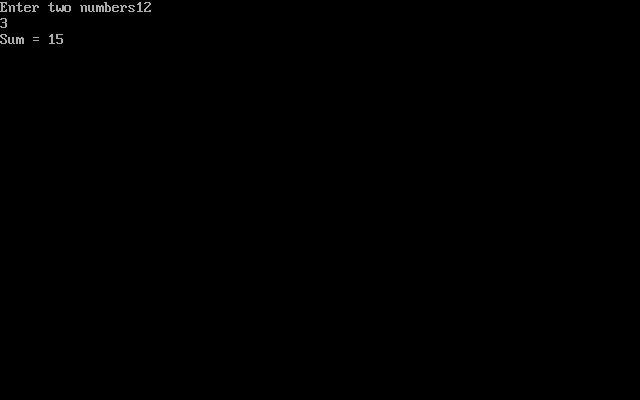
main – identifier

{,}, (,) – delimiter

int – keyword

x, y, total – identifier

main, {, }, (, ), int, x, y, total – tokens



**Constant**

C Constants are also like normal variables. But, only difference is, their values can not be modified by the program once they are defined.

Constants refer to fixed values. They are also called as literals

Constants may be belonging to any of the data type.

Syntax:

const data\_type variable\_name; (or) const data\_type \*variable\_name;

**TYPES OF C CONSTANT:**

Integer constants

Real or Floating point constants

Octal & Hexadecimal constants

Character constants

String constants

**Backslash character constants**

|  |  |
| --- | --- |
| Constant type | data type (Example) |
| Integer constants | int (53, 762, -478 etc ) unsigned int (5000u, 1000U etc) long int, (483,647 2,147,483,680) |
| Real or Floating point constants | float (10.456789) doule (600.123456789) |
| Octal constant | int (Example: 013 /\*starts with 0 \*/) |
| Hexadecimal constant | int (Example: 0x90 /\*starts with 0x\*/) |
| character constants | char (Example: ‘A’, ‘B’, ‘C’) |
| string constants | char (Example: “ABCD”, “Hai”) |

**RULES FOR CONSTRUCTING C CONSTANT:**

1. INTEGER CONSTANTS:

An integer constant must have at least one digit.

It must not have a decimal point.

It can either be positive or negative.

No commas or blanks are allowed within an integer constant.

If no sign precedes an integer constant, it is assumed to be positive.

The allowable range for integer constants is -32768 to 32767.

2. REAL CONSTANTS IN C:

A real constant must have at least one digit

It must have a decimal point

It could be either positive or negative

If no sign precedes an integer constant, it is assumed to be positive.

No commas or blanks are allowed within a real constant.

3. CHARACTER AND STRING CONSTANTS IN C:

A character constant is a single alphabet, a single digit or a single special symbol enclosed within single quotes.

The maximum length of a character constant is 1 character.

String constants are  enclosed within double quotes.

4. BACKSLASH CHARACTER CONSTANTS:

There are some characters which have special meaning in C language.

They should be preceded by backslash symbol to make use of special function of them.

Given below is the list of special characters and their purpose.

|  |  |
| --- | --- |
| Backslash\_character | Meaning |
| \b | Backspace |
| \f | Form feed |
| \n | New line |
| \r | Carriage return |
| \t | Horizontal tab |
| \” | Double quote |
| \’ | Single quote |
| \\ | Backslash |
| \v | Vertical tab |
| \a | Alert or bell |
| \? | Question mark |
| \N | Octal constant (N is an octal constant) |
| \XN | Hexadecimal constant (N – hex.dcml cnst) |

**HOW TO USE CONSTANTS IN A C PROGRAM?**

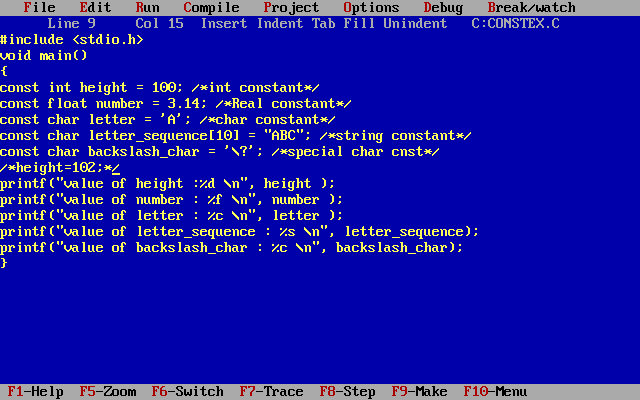
We can define constants in a C program in the following ways.

By “const” keyword

By “#define” preprocessor directive

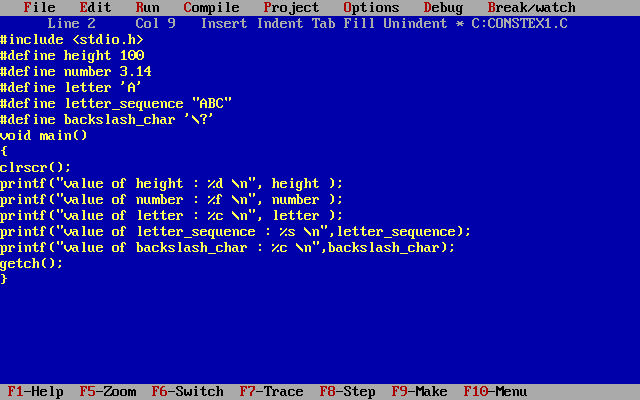
Please note that when you try to change constant values after defining in C program, it will through error.

**1. EXAMPLE PROGRAM USING CONST KEYWORD:**





**2. EXAMPLE PROGRAM USING #DEFINE PREPROCESSOR DIRECTIVE IN C:**





**Operators**

C language supports a rich set of built-in operators. Operators are used in programs to manipulate data and variables. Operators, functions, constants and variables are combined together to form expressions.

Consider the expression A + B \* 5. where, +, \* are operators, A, B  are variables, 5 is constant and A + B \* 5 is an expression.

**TYPES OF OPERATORS:**

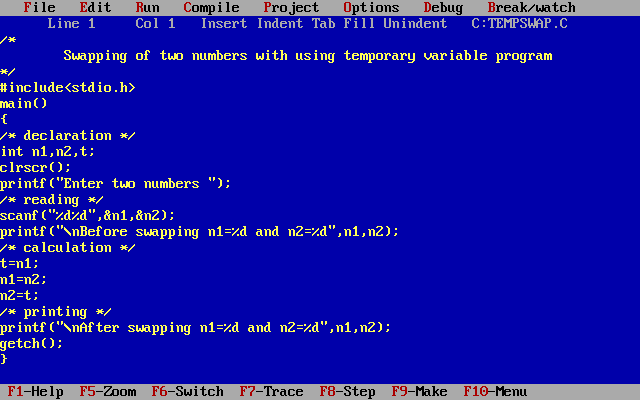
C language offers many types of operators. They are,

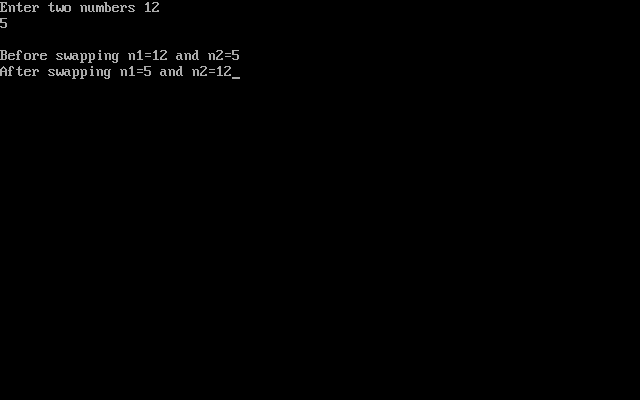
1. Assignment operator
2. Arithmetic operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators
8. Special operators

**Assignment Operator**

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

|  |  |
| --- | --- |
| Operator | Description |
| = | assigns values from right side operands to left side operand |

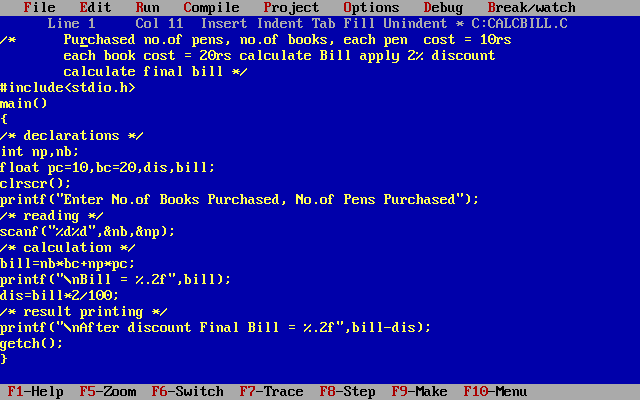




**Arithmetic Operators**

C Arithmetic operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus in C programs.

|  |  |
| --- | --- |
| Arithmetic Operators/Operation | Example |
| + (Addition) | A+B |
| – (Subtraction) | A-B |
| \* (multiplication) | A\*B |
| / (Division) | A/B |
| % (Modulus) | A%B |

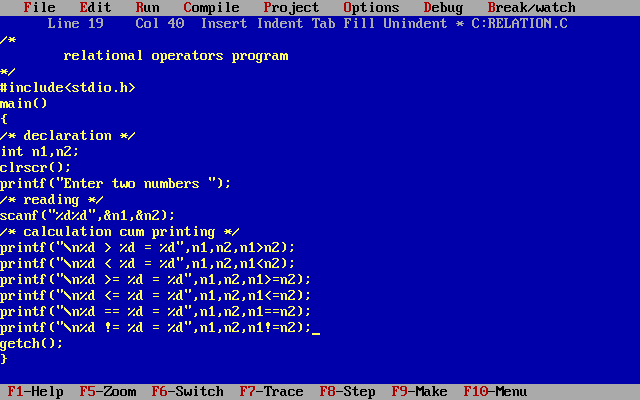


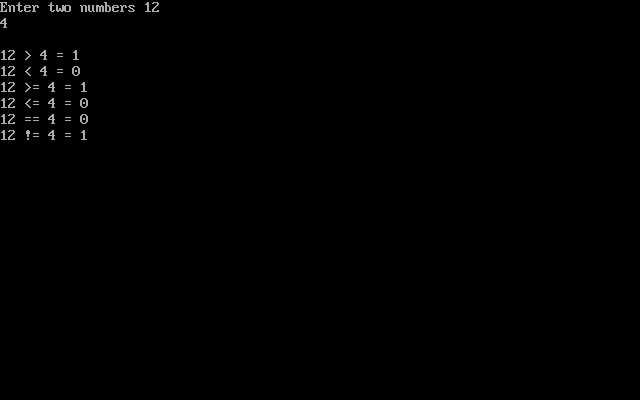


**Relational operators**

We can compare the value stored between two variables and depending on the result we can follow different blocks using Relational Operators. If the condition is true it executes 1 otherwise 0 shows in the result.

|  |  |
| --- | --- |
| Operator | Description |
| == | Check if two operand are equal |
| != | Check if two operand are not equal. |
| > | Check if operand on the left is greater than operand on the right |
| < | Check operand on the left is smaller than right operand |
| >= | check left operand is greater than or equal to right operand |
| <= | Check if operand on left is smaller than or equal to right operand |





**Logical Operators**

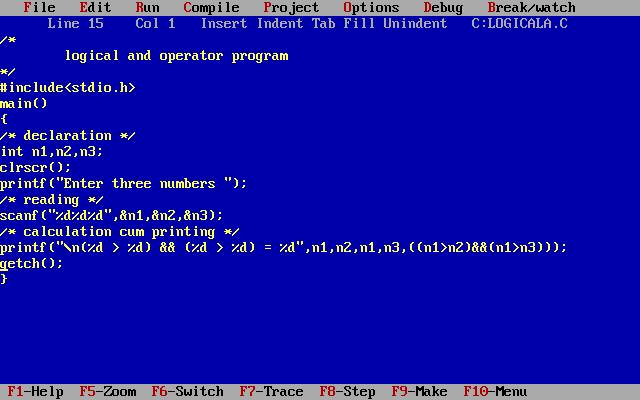
Whenever we use if statement then we can use [relational operator](http://www.c4learn.com/c-programming/c-relational-operator/) which tells us the result of the comparison, so if we want to compare more than one conditions then we need to use logical operators. Suppose we need to execute certain block of code if and only if two conditions are satisfied then we can use Logical Operators.

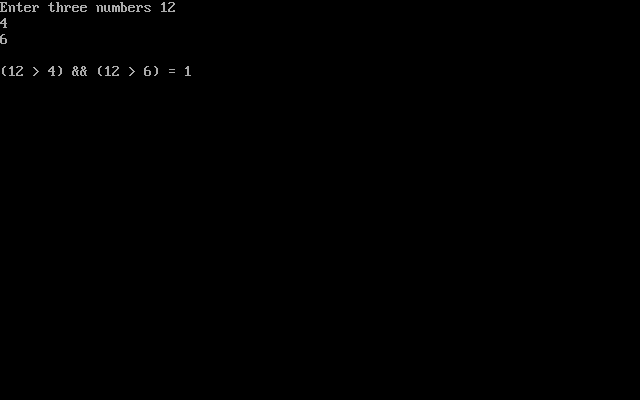
|  |  |
| --- | --- |
| Operator | Name of the Operator |
| && | And Operator |
| || | Or Operator |
| ! | Not Operator |

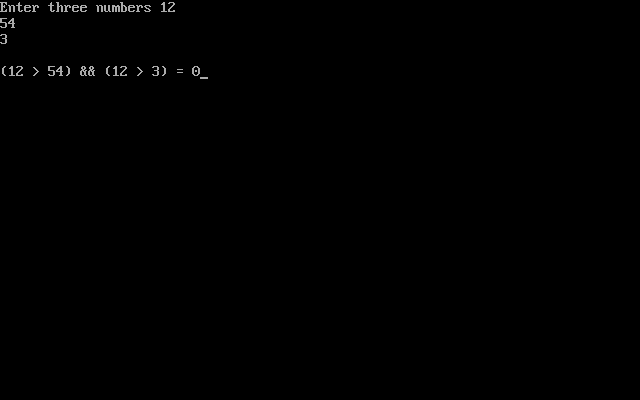
**Logical And:-**  If both the conditions are true then only true otherwise false.

**Truth Table of &&**

|  |  |  |
| --- | --- | --- |
| **Condition1** | **Condition2** | **Result** |
| **1** | **1** | **1** |
| **1** | **0** | **0** |
| **0** | **1** | **0** |
| **0** | **0** | **0** |



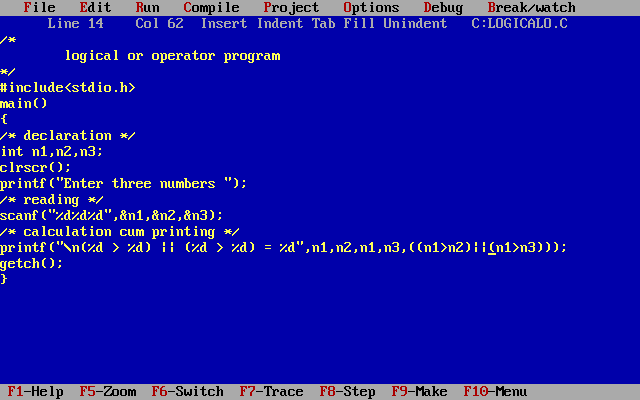


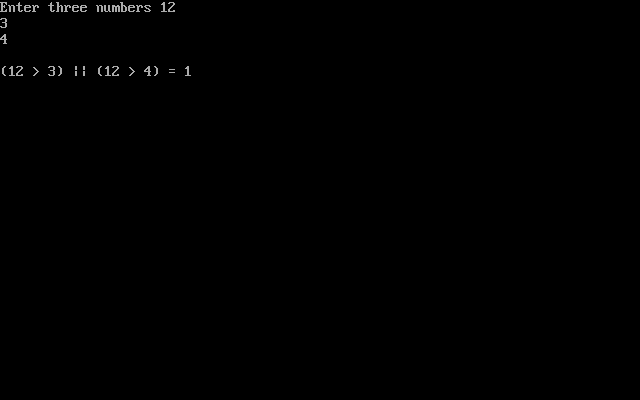


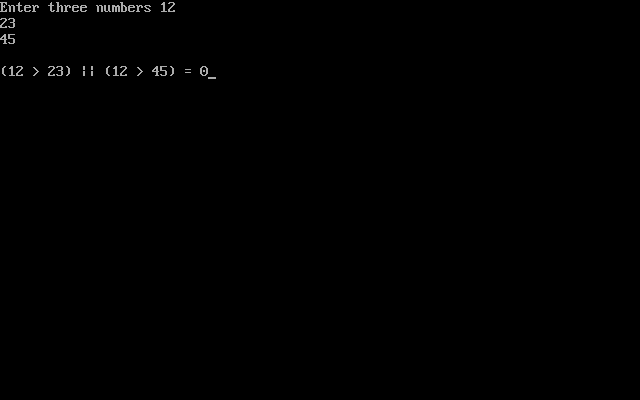
**Logical Or:-**  If both the conditions are false then only false otherwise true.

**Truth Table of ||**

|  |  |  |
| --- | --- | --- |
| **Condition1** | **Condition2** | **Result** |
| **1** | **1** | **1** |
| **1** | **0** | **1** |
| **0** | **1** | **1** |
| **0** | **0** | **0** |



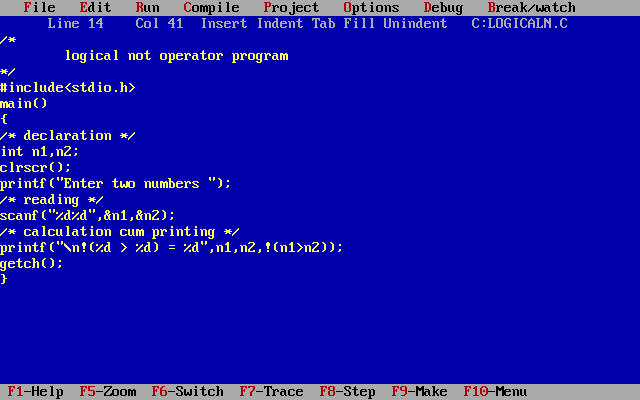


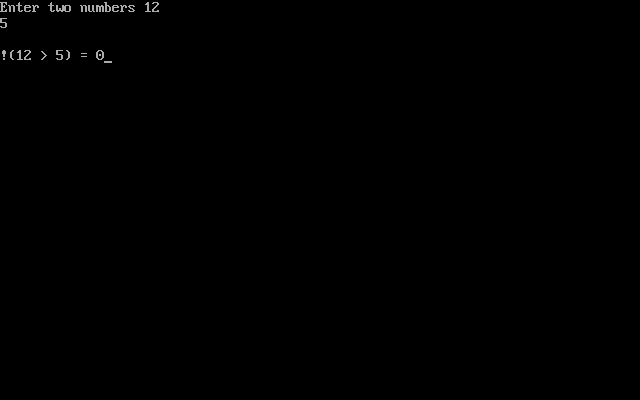


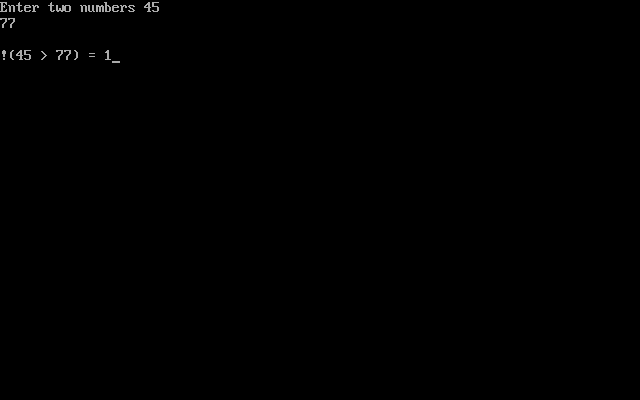
**Logical Not:-**  If the condition is true it shows false and vice versa.

**Truth Table of !**

|  |  |
| --- | --- |
| **Condition** | **Result** |
| **1** | **0** |
| **0** | **1** |







**Bitwise Operators**

During computation, mathematical operations like: addition, subtraction, addition and division are converted to bit-level which makes processing faster and saves power. Bitwise operators are used in C programming to perform bit-level operations.

|  |  |
| --- | --- |
| Operators | Meaning of operators |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |
| ~ | Bitwise complement |
| << | Shift left |
| >> | Shift right |

**Bitwise And (&) Truth Table**

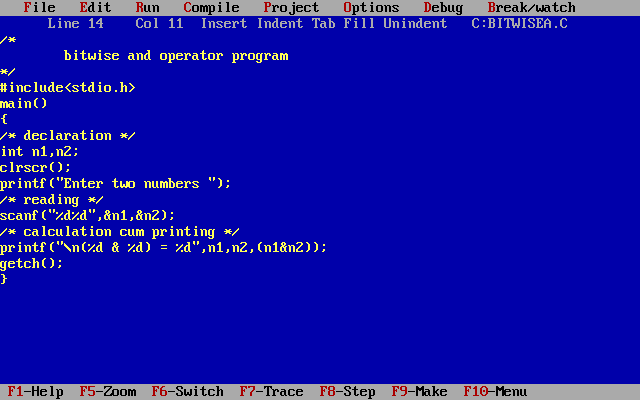
|  |  |  |
| --- | --- | --- |
| **Condition1** | **Condition2** | **Result** |
| **1** | **1** | **1** |
| **1** | **0** | **0** |
| **0** | **1** | **0** |
| **0** | **0** | **0** |

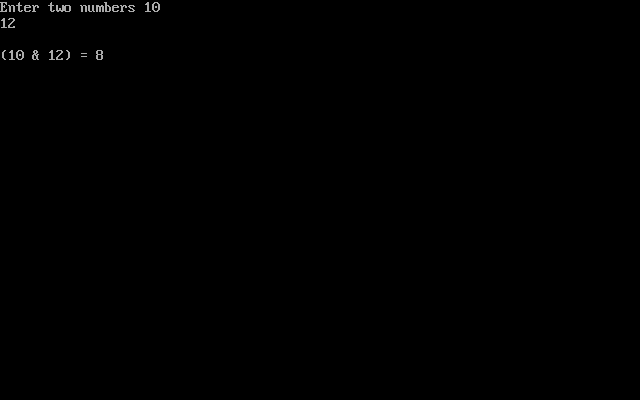
A = 10 = 1 0 1 0

B = 12 = 1 1 0 0

A & B = 10 & 12 = 1 0 0 0

A & B = 10 & 12 = 8





**Bitwise Or (|) Truth Table**

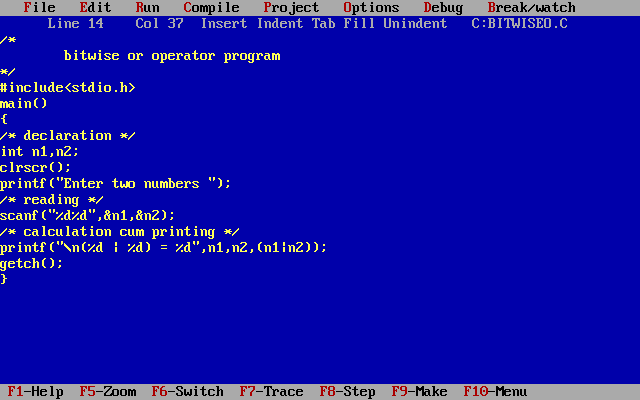
|  |  |  |
| --- | --- | --- |
| **Condition1** | **Condition2** | **Result** |
| **1** | **1** | **1** |
| **1** | **0** | **1** |
| **0** | **1** | **1** |
| **0** | **0** | **0** |

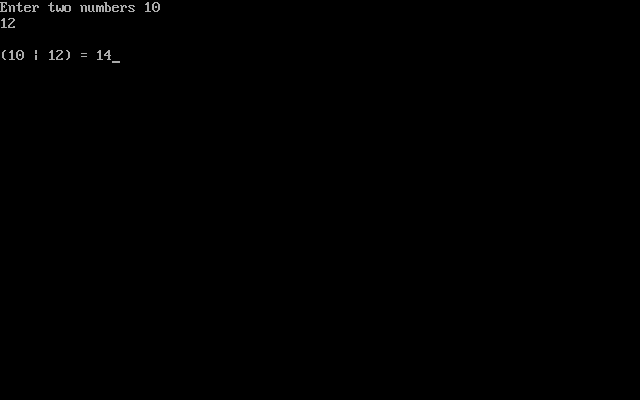
A = 10 = 1 0 1 0

B = 12 = 1 1 0 0

A | B = 10 | 12 = 1 1 1 0

A | B = 10 | 12 = 14





**Bitwise Exclusive Or (^) Truth Table**

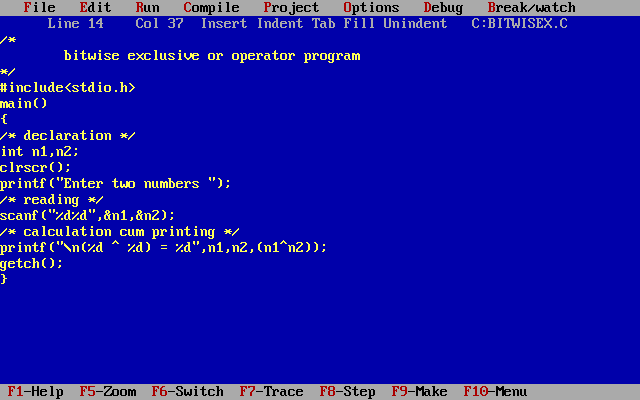
|  |  |  |
| --- | --- | --- |
| **Condition1** | **Condition2** | **Result** |
| **1** | **1** | **0** |
| **1** | **0** | **1** |
| **0** | **1** | **1** |
| **0** | **0** | **0** |

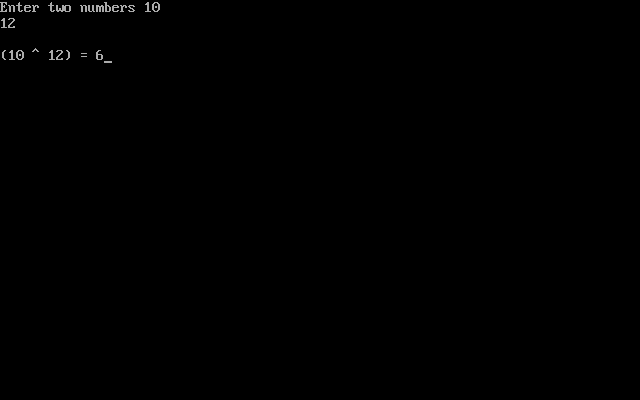
A = 10 = 1 0 1 0

B = 12 = 1 1 0 0

A ^ B = 10 ^ 12 = 0 1 1 0

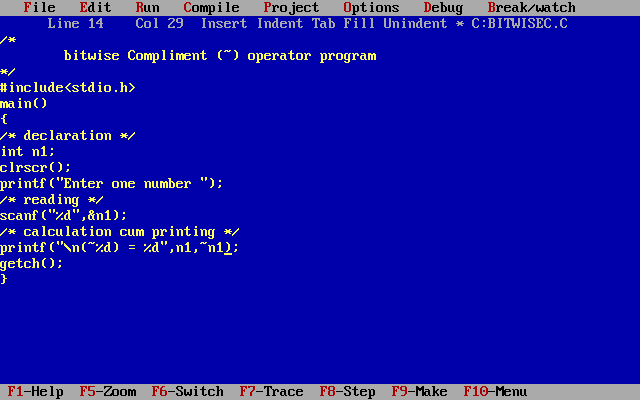
A ^ B = 10 ^ 12 = 6

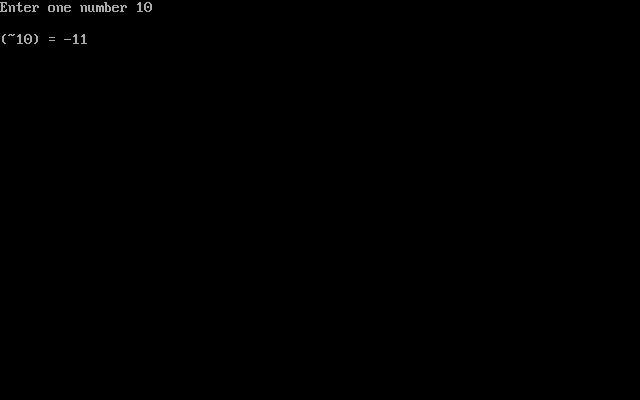


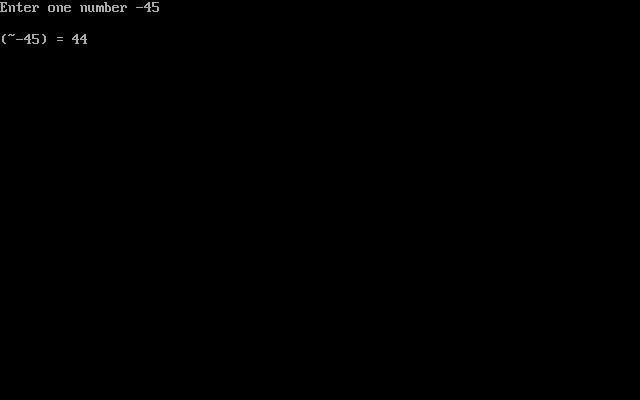


**Bitwise complement**

**Formula:** N=-(N+1)

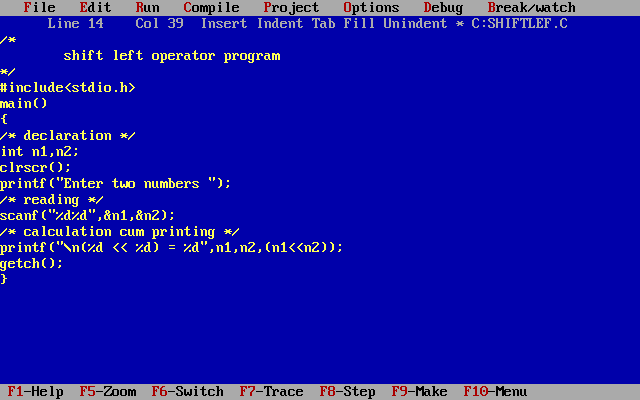


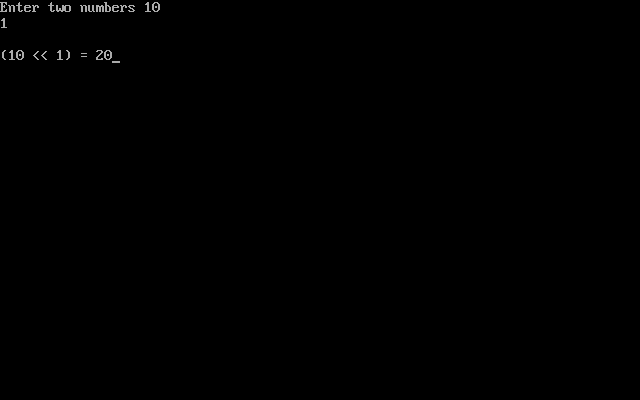


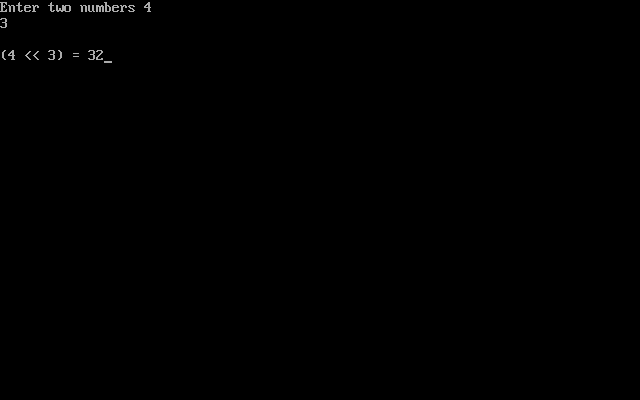


**(Shift Left)<<**:- << contains before one number and after one number (Ex:- n1<<n2). << means multiplied with 2. (n1 is multiplied with 2 n2 times).

Ex:-

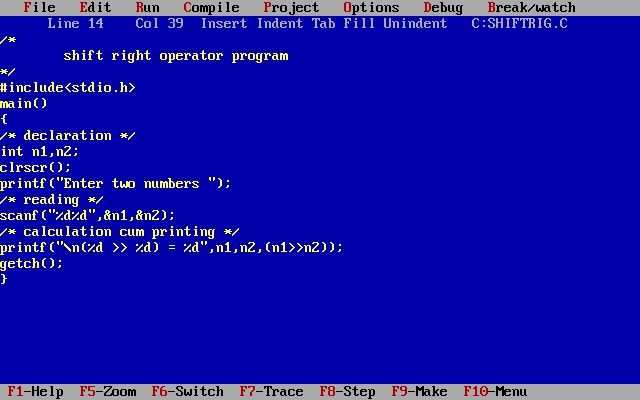


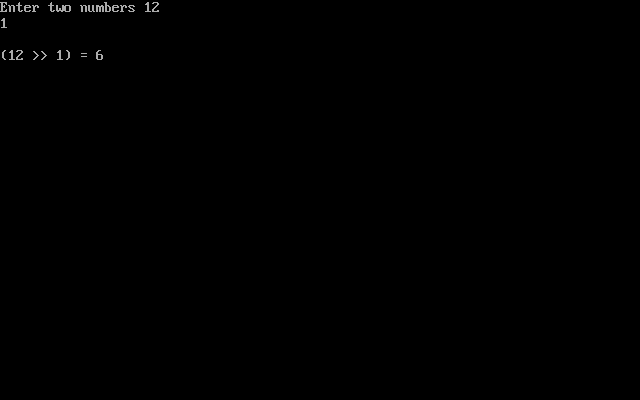




**(Shift Right)<<**:- >> contains before one number and after one number (Ex:- n1>>n2). >> means divided by 2. (n1 is divided with 2 n2 times).

Ex:-





**Ternary Operator (or) Conditional Operator (?:)**

Conditional operators return one value if condition is true and returns another value is condition is false.

**Conditional Operator Syntax**

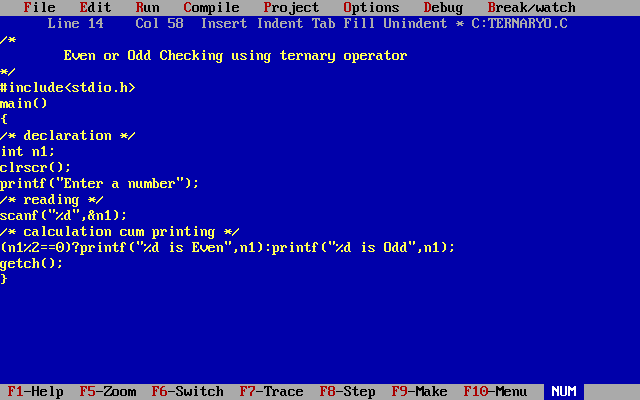
conditionalExpression ? expression1 : expression2

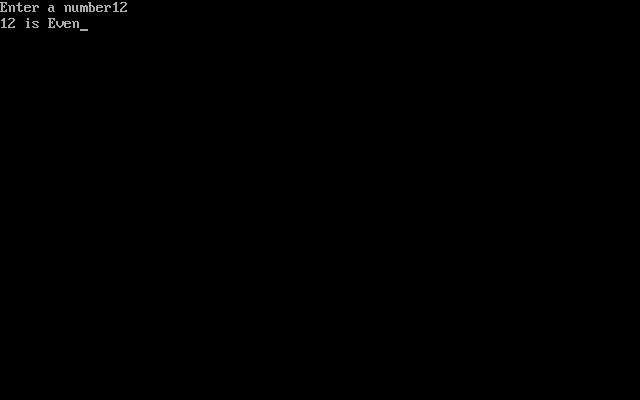
*The conditional operator works as follows:*

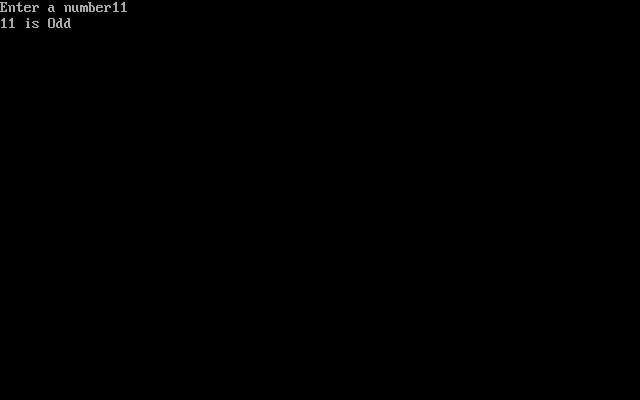
The first expression conditionalExpression is evaluated first. This expression evaluates to 1 if it's true and evaluates to 0 if it's false.

If conditionalExpression is true, expression1 is evaluated.

If conditionalExpression is false, expression2 is evaluated.





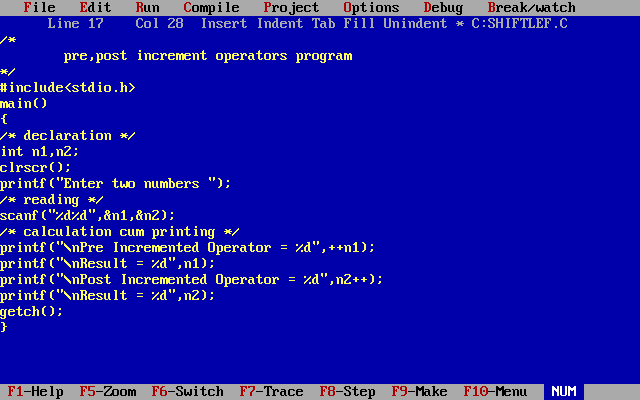


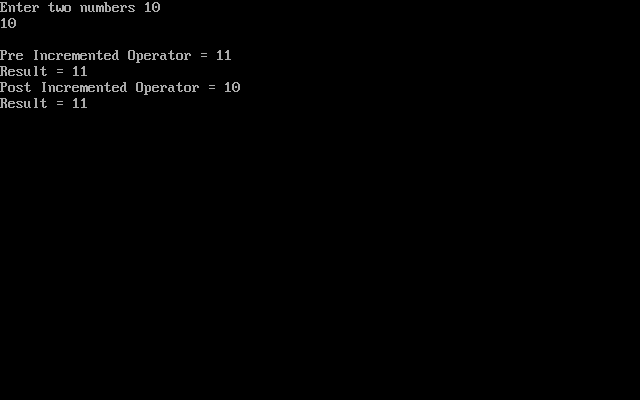
**Increment/decrement Operators**

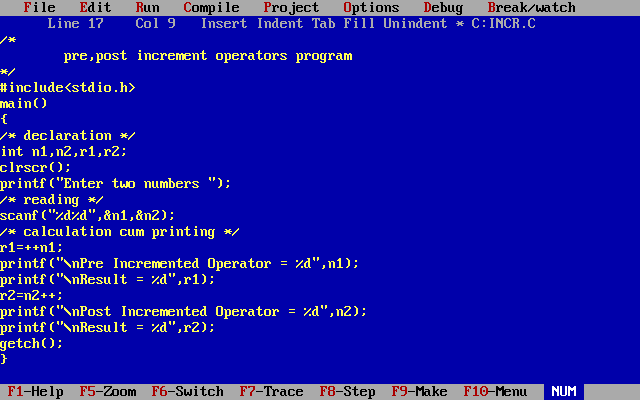
Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.

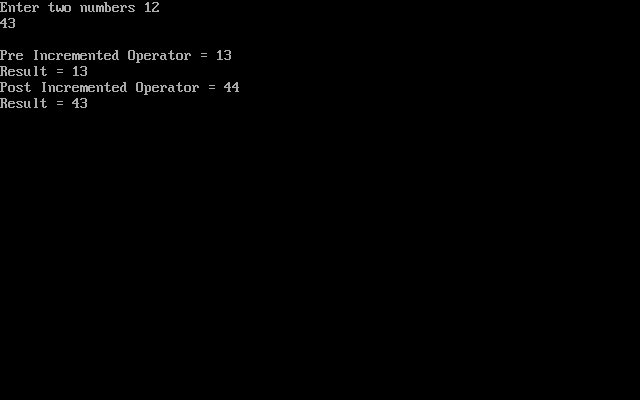
Syntax:  
Increment operator: ++var\_name; (or) var\_name++;  
Decrement operator: – -var\_name; (or) var\_name – -;

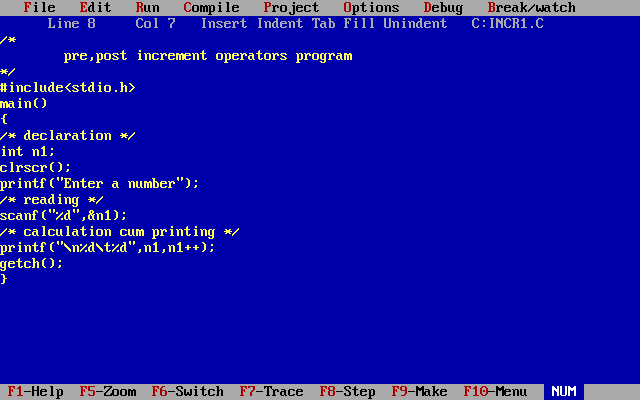
Example:  
Increment operator :  ++ i ;    i ++ ;

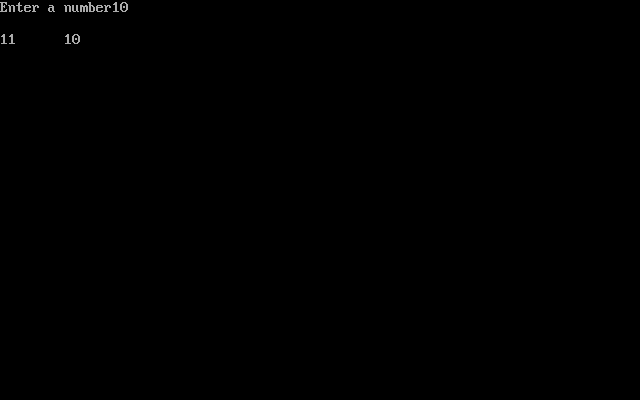


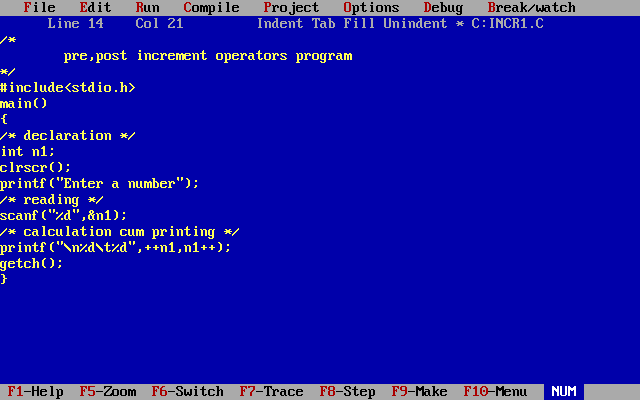


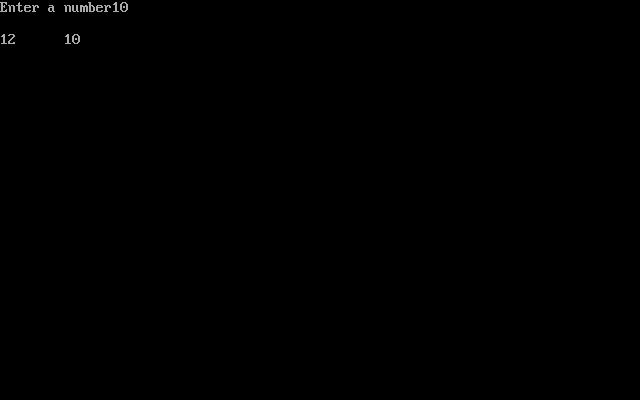












Decrement operator :  – – i ;   i – – ;

**DIFFERENCE BETWEEN PRE/POST INCREMENT & DECREMENT OPERATORS:**

Below table will explain the difference between pre/post increment and decrement operators in C programming language.

|  |  |
| --- | --- |
| Operator | Operator/Description |
| Pre increment operator (++i) | value of i is incremented before assigning it to the variable i |
| Post increment operator (i++) | value of i is incremented after assigning it to the variable i |
| Pre decrement operator (--i) | value of i is decremented before assigning it to the variable i |
| Post decrement operator (i--) | value of i is decremented after assigning it to variable i |

**Special operators:**

Below are some of the special operators that the C programming language offers.

|  |  |
| --- | --- |
| Operators | Description |
| & | This is used to get the address of the variable.  Example : &a will give address of a. |
| \* | This is used as pointer to a variable.  Example : \* a  where, \* is pointer to the variable a. |
| Sizeof () | This gives the size of the variable.  Example : size of (char) will give us 1. |

**Decision Making statement**

In decision making statements, group of statements are executed when condition is true.  If condition is false, then else part statements are executed. There are 3 types of decision making control statements in C language. They are,

* if statements
* if else statements
* else if ladder statements
* nested if statements

**C if statement**

if (testExpression)

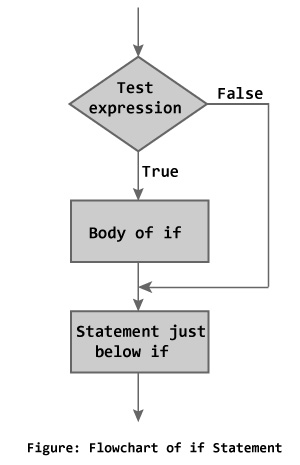
{

// statements

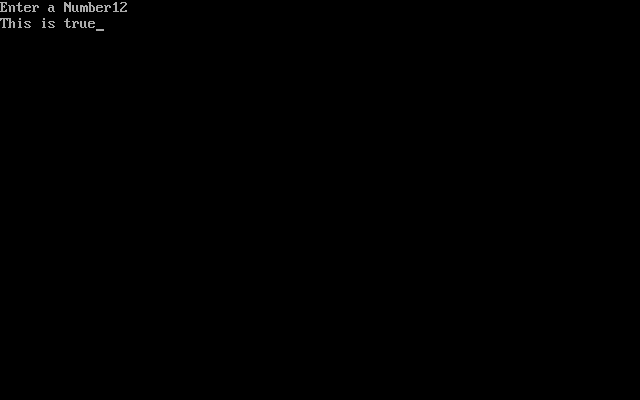
}

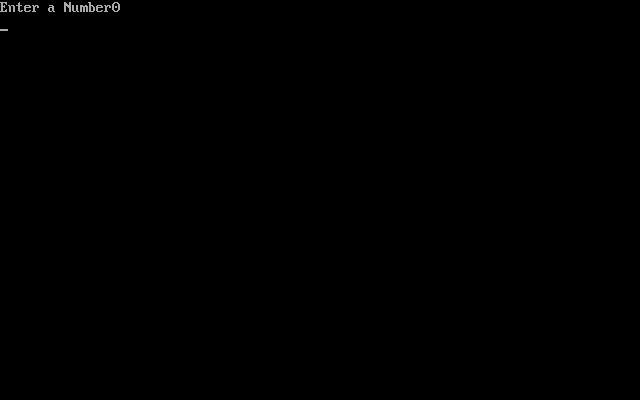
The if statement evaluates the test expression inside the parenthesis. If the test expression is evaluated to true (nonzero), statements inside the body of if is executed. If the test expression is evaluated to false (0), statements inside the body of if is skipped from execution.

**Flowchart of if statement**









**syntax 2:**

if(condition)

{

statement1;

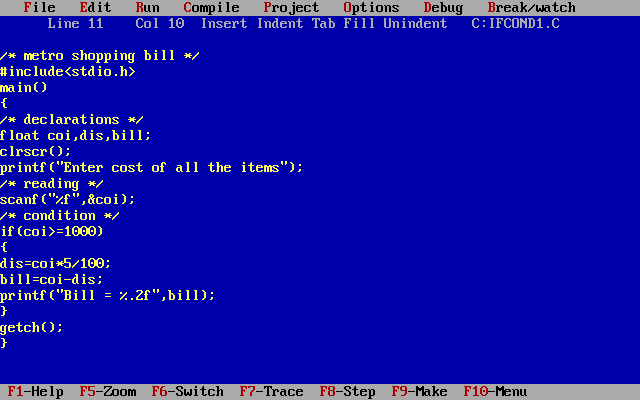
statement2;

}

If the condition is true then statement1 and statement2 will be executed.

If the statement is a single statement {} are not required.

If the statements are more than one {} are mandatory.







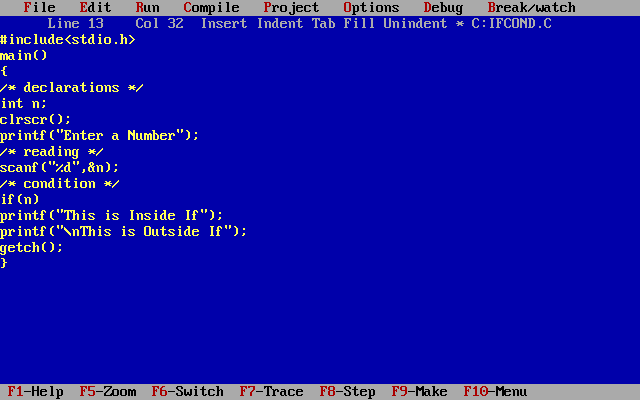
**Syntax for both types:**

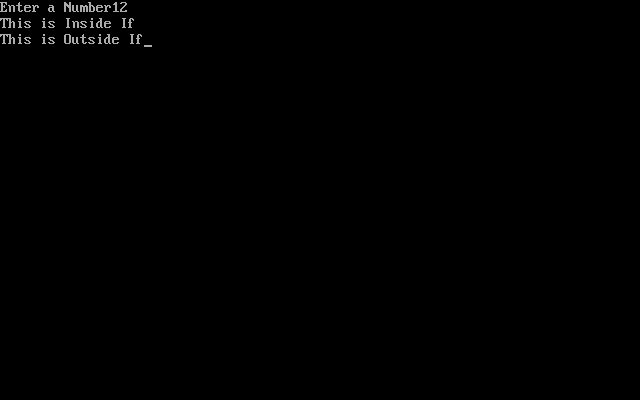
1:- if(condition)

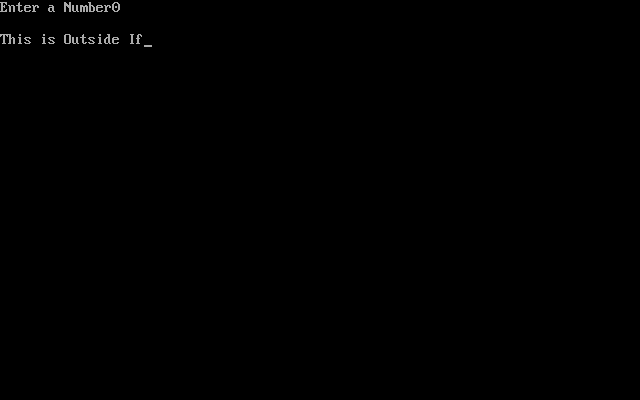
statement1;

statement n;

If the condition is true statement1 and statement n will be executed. If the condition is fail statement n will be executed. Because statement is common if the condition is true or false.







2:- if(condition)

{

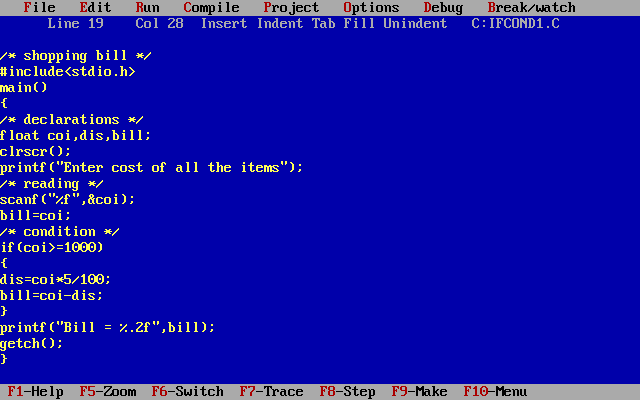
statement1;

statement2;

}

statement n;

If the condition is true it executes statement1, statement2 and statement n will be executed. If the condition is fail statement n will be executed.







**if...else statement**

The if...else statement executes some code if the test expression is true (nonzero) and some other code if the test expression is false (0)

**Syntax of if...else**

if (testExpression)

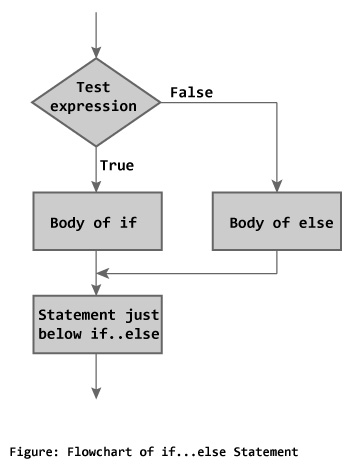
// codes inside the body of if

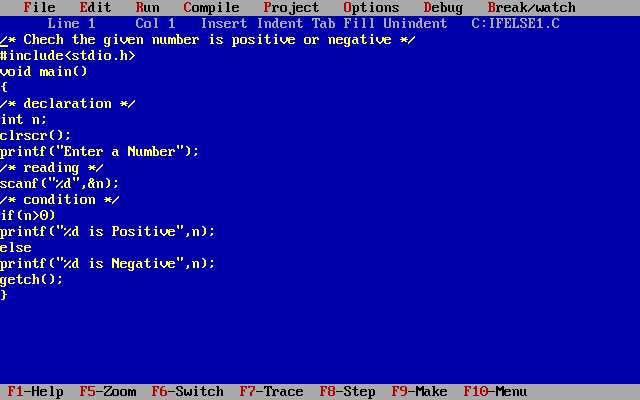
else

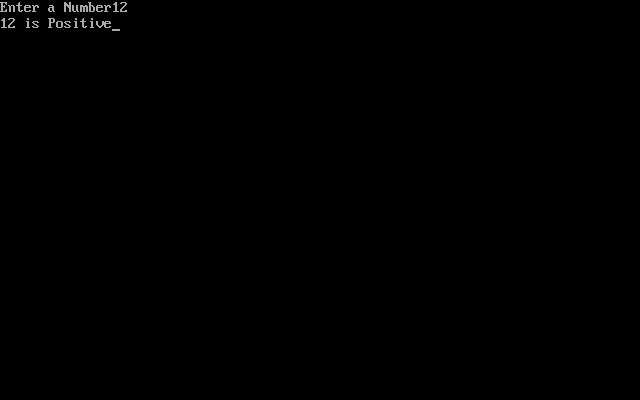
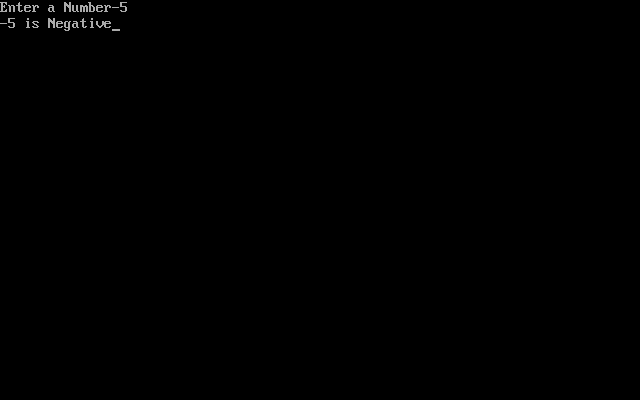
// codes inside the body of else

If test expression is true, codes inside the body of if statement is executed and, codes inside the body of else statement is skipped. If test expression is false, codes inside the body of else statement is executed and, codes inside the body of if statement is skipped.

Flowchart of if...else statement





**syntax 2:**

if( condition)

{

statement 1;

statement 2;

}

else

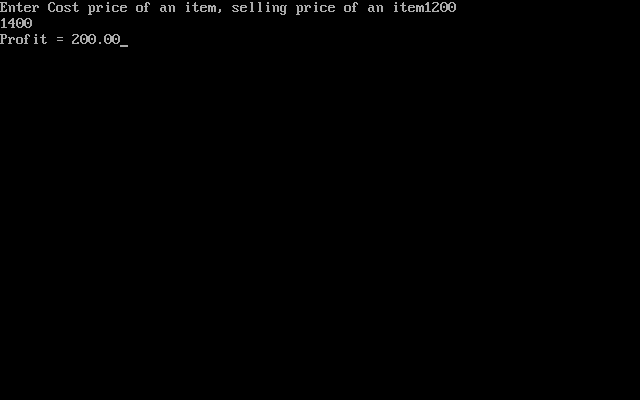
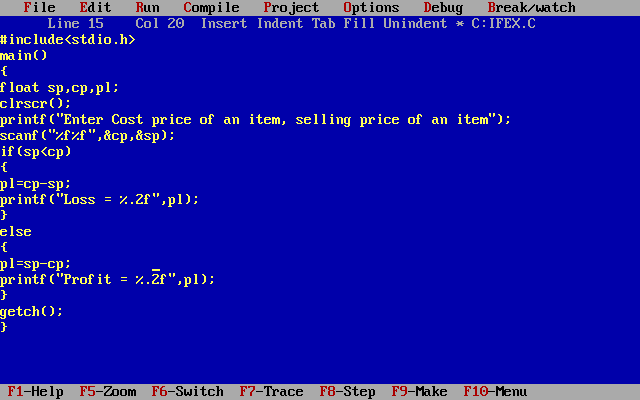
{

statement 3;

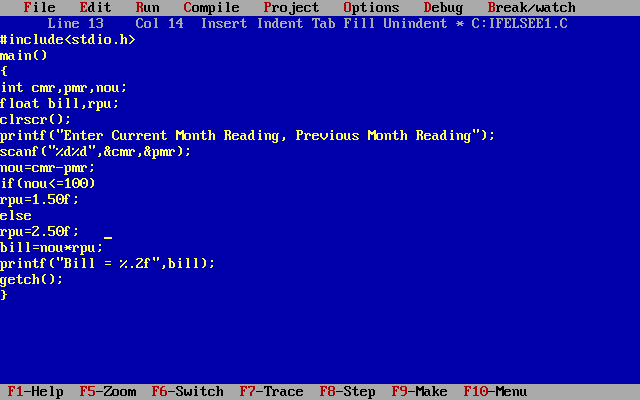
statement 4;

}

If the condition is true then statement1 and statement2 will be executed. If the condition is false then statement3 and statement4 will be executed.











**else if ladder**

if(<condition1>)

Statement1;

else if(<condition2>)

statement2;

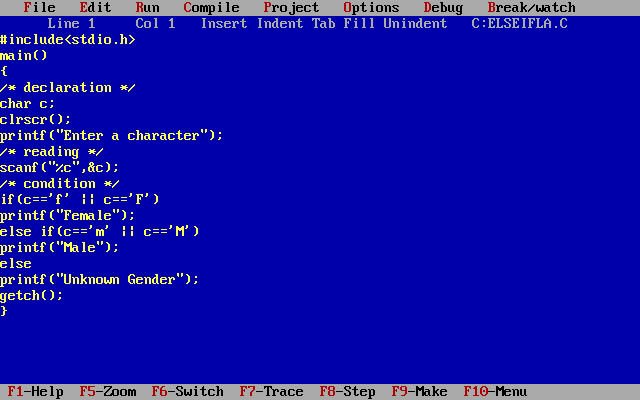
else if(<condition3>)

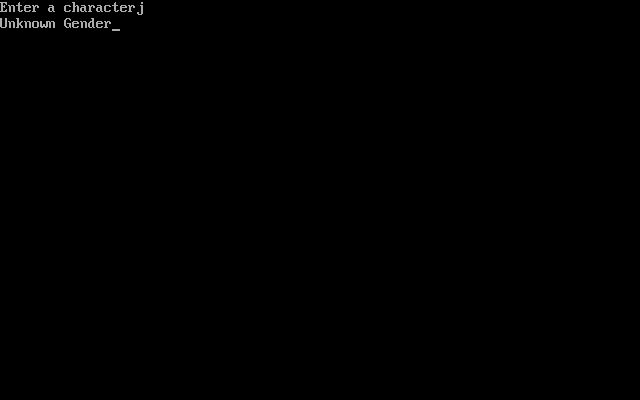
statement3;

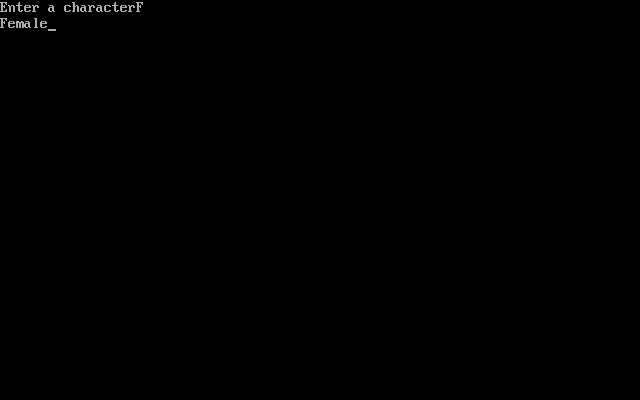
else

statement4;

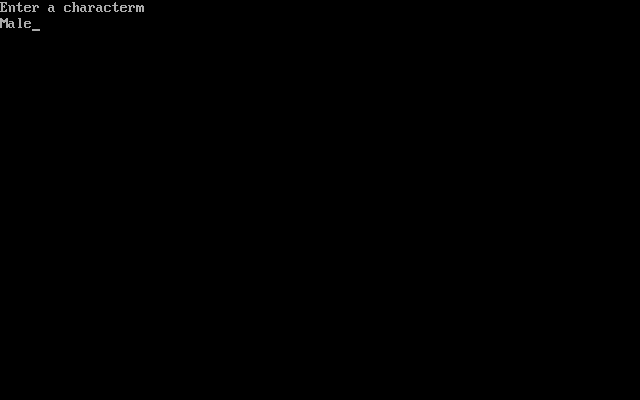
If the condition1 is true then statement1 will be executed and the remaining statements will be skipped. If the condition1 is false then it will check the condition2, if it is true then statement2 will be executed and the remaining statements will be skipped. And so.on if all conditions are false then it will execute statement4. Else part is optional. If all the conditions are fail there is no result.













**Nested-if statement**

Nested if statement contains the no. of if conditions without else part. If the condition is true or false it checks the remaining conditions also.

if(<condition1>)

Statement1;

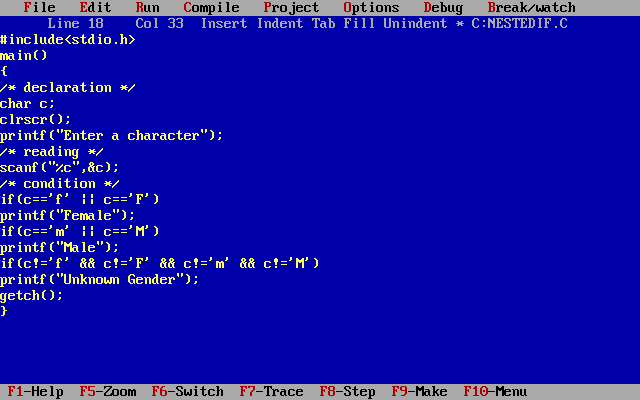
if(<condition2>)

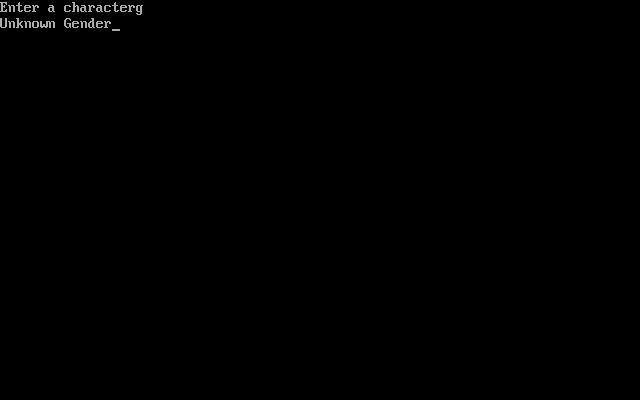
statement2;

if(<condition3>)

statement3;

If the condition1 is true then statement1 will be executed. If the condition1 is true or false then it will check the condition2, if it is true then statement 2 will be executed if condition 1 is false and condition 2 is true. If the condition1 and condition2 are true statement 1, statement 2 also executed and so. On for remaining.





**Nested if....else statement**

The general form of a nested if...else statement is,

if( expression )

{

if( expression1 )

{

statement block1;

}

else

{

statement block2;

}

}

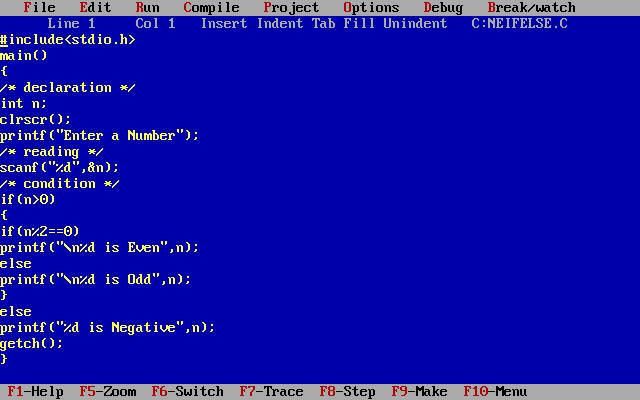
else

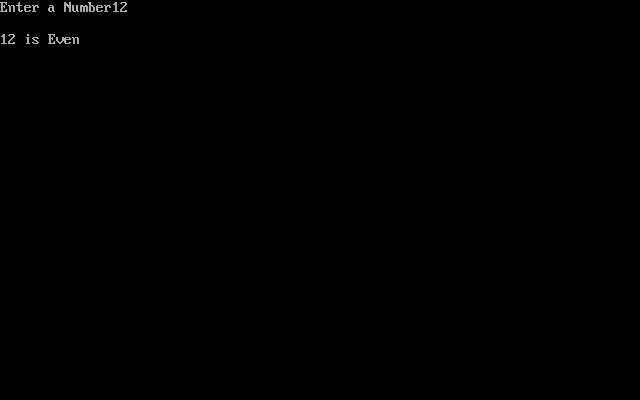
{

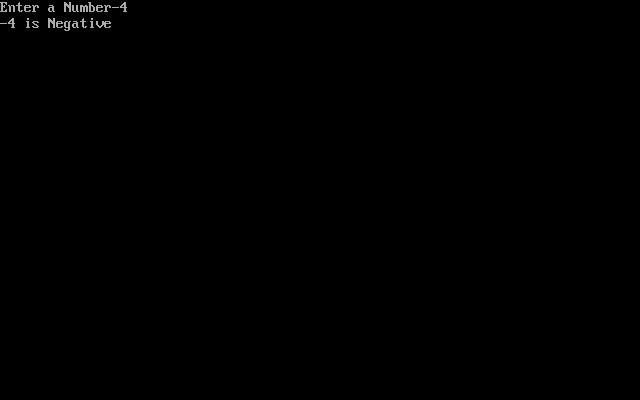
statement block3;

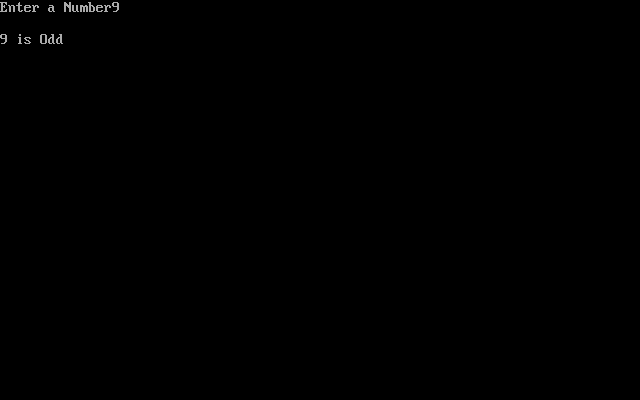
}

if expression is false then statement-block3 will be executed, otherwise the execution continues and enters inside the first if to perform the check for the next if block, where if expression 1 is true the statement-block1 is executed otherwise statement-block2 is executed.









**Case control statements**

The statements which are used to execute only specific block of statements in a series of blocks are called case control statements.

There are 4 types of case control statements in C language. They are,

* switch
* break
* continue
* goto

**Why we should use Switch Case ?**

One of the classic problem encountered in nested if-else / else-if ladder is called problem of Confusion. It occurs when no matching else is available for if. As the number of alternatives increases the Complexity of program increases drastically. To overcome this, C Provide a multi-way decision statement called ‘Switch Statement‘.

if(Condition 1)

Statement 1

else

{

Statement 2

if(condition 2)

{

if(condition 3)

statement 3

else

if(condition 4)

{

statement 4

}

}

else

{

statement 5

}

}

**Switch Case**

switch(expression)

{

case value1 :

body1

break;

case value2 :

body2

break;

case value3 :

body3

break;

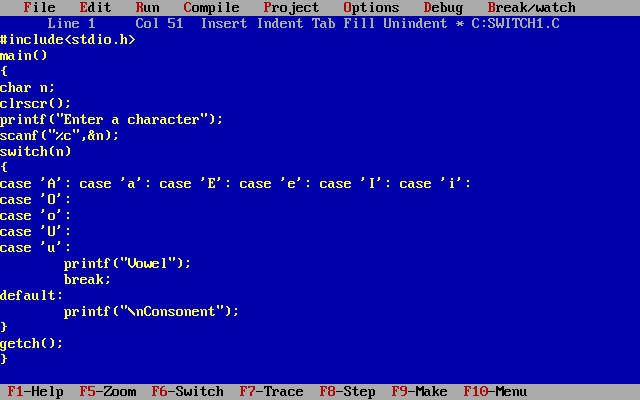
default :

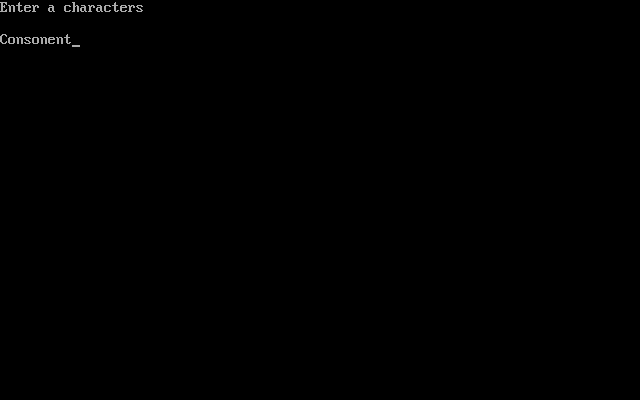
default-body

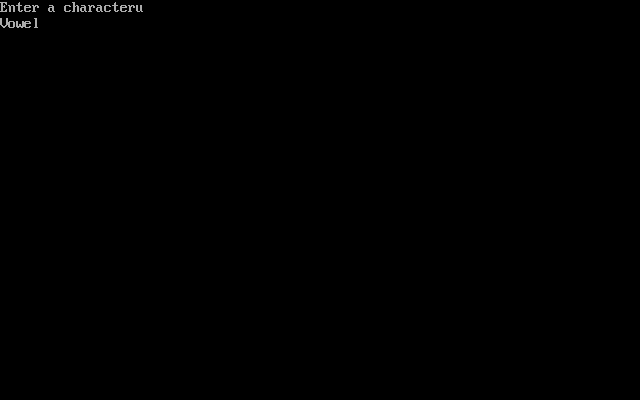
break;

}

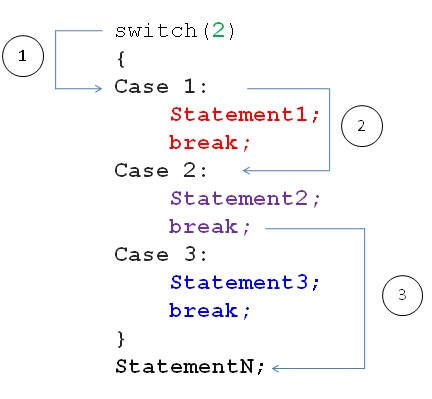
next-statement;







**Flow Diagram:**

[](http://img.c4learn.com/2010/06/Syntax-of-Switch-Case-Stetement-in-C-Programming.jpg)  
\*Steps are Shown in Circles.

**How it works?**

Switch case checks the value of expression/variable against the list of case values and when the match is found, the block of statement associated with that case is executed

Expression should be Integer Expression / Character

Break statement takes control out of the case.

Break Statement is Optional.

**Rules of Using Switch Case**

* Case Label must be unique
* Case Labels must ends with Colon
* Case labels must have constants / constant expression
* Case label must be of integral Type ( Integer,Character)
* Case label should not be ‘floating point number ‘
* Switch case should have at most one default label
* Default label is Optional
* Default can be placed anywhere in the switch
* Break Statement takes control out of the switch
* Two or more cases may share one break statement
* Nesting ( switch within switch ) is allowed.
* Relational Operators are not allowed in Switch Statement.
* Macro Identifier are allowed as Switch Case Label.
* Const Variable is allowed in switch Case Statement.
* Empty Switch case is allowed.

**Syntax of Switch Case:**

switch ( expression )

{

case label1 :

body1

break;

case label2 :

body2

break;

case label3 :

body3

break;

default :

default-body

break;

}

next-statement;

**Rule 1 : Case Label must be unique**

int id = 3 ;

switch(id)

{

case 1:

printf("C Programming Language");

break;

case 2:

printf("C++ Programming Language");

break;

case 2:

printf("Web Technology");

break;

default :

printf("No student found");

break;

}

**Rule 2 : Case Labels must ends with Colon**

case 1 :

printf("C Programming Language");

break;

**Rule 3 : Case labels must have constants / constant expression**

case 1+1:

case 'A':

case 67:

these are allowed examples of switch case labels , however variables are not allowed in switch case labels.

case var :

case num1 :

case n1+n2 :

**Rule 4 : Case label must be of integral Type ( Integer,Character) whereas Case label should not be ‘floating point number ‘**

case 10:

case 20+20:

case 'A':

case 'a':

these are allowed examples and following are illegal examples –

case 10.12:

case 7.5:

**Rule 5 : Switch case should have at most one default label**

switch(roll)

{

case 1:

printf("C Programming Language");

break;

case 2:

printf("C++ Programming Language");

break;

case 2:

printf("Web Technology");

break;

default :

printf("Default Version 1");

break;

default :

printf("Default Version 2");

break;

}

It violets first rule.

**Rule 6 : Default label is Optional**

switch(roll)

{

case 1 :

printf("C Programming Language");

break;

case 2 :

printf("C++ Programming Language");

break;

case 2 :

printf("Web Technology");

break;

}

default statement is optional. It can be neglected.

**Rule 7 : Default can be placed anywhere in the switch**

switch(roll)

{

case 1 :

printf("C Programming Language");

break;

default:

printf("No Student Found");

break;

case 2 :

printf("C++ Programming Language");

break;

case 2 :

printf("Web Technology");

break;

}

**Rule 8 : Break Statement takes control out of the switch**

**Rule 9 : Two or more cases may share one break statement**

switch(alpha)

{

case 'a':

case 'A':

printf("Alphabet A");

break;

case 'b':

case 'B':

printf("Alphabet B");

break;

}

**Rule 10 : Nesting ( switch within switch ) is allowed**

switch(alpha)

{

case 'a':

case 'A':

printf("Alphabet A");

break;

case 'b':

case 'B':

switch(alpha)

{

}

break;

}

nesting of switch case is allowed in C.

**Rule 11 : Relational Operators are not allowed in Switch Statement.**

switch(num)

{

case >15:

printf("Number > 15");

break;

case =15:

printf("Number = 15");

break;

case <15:

printf("Number < 15");

break;

}

relational operators are not allowed as switch label.

**Rule 12 : Macro Identifier are allowed as Switch Case Label.**

#define MAX 2

switch(num)

{

case MAX:

printf("Number = 2");

break;

}

as preprocessor will replace occurrence of MAX by constant value i.e 2 therefor it is allowed.

**Rule 13 : Const Variable is not allowed in switch Case Statement.**

int const var = 2;

switch(num)

{

case var:

printf("Number = 2");

break;

}

**2. BREAK STATEMENT:**

Break statement is used to terminate the while loops, switch case loops and for loops from the subsequent execution.

Syntax: break;

**3. CONTINUE STATEMENT:**

Continue statement is used to continue the next iteration of for loop, while loop and do-while loops.  So, the remaining statements are skipped within the loop for that particular iteration.

Syntax : continue;

**4. GOTO STATEMENT:**

The goto statement is rarely used because it makes program confusing, less readable and complex. Also, when this is used, the control of the program won’t be easy to trace, hence it makes testing and debugging difficult.

**C – goto statement**

When a goto statement is encountered in a C program, the control jumps directly to the label mentioned in the goto statement.

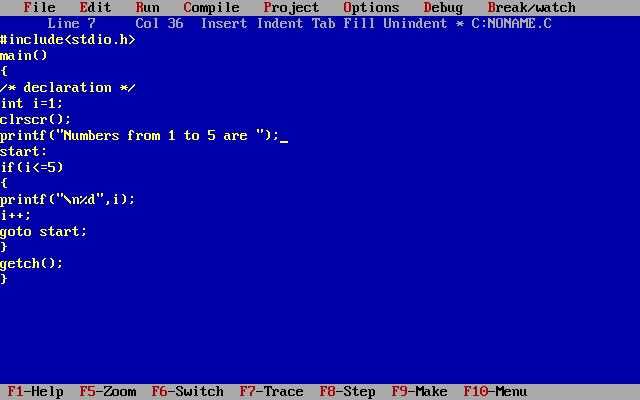
*Syntax of goto statement*

goto label\_name;

..

..

label\_name: C-statements





**Control statements:-**

Loop control statements in C are used to perform looping operations until the given condition is true. Control comes out of the loop statements once condition becomes false.

*TYPES OF LOOP CONTROL STATEMENTS:*

There are 3 types of loop control statements in C language. They are,

while

do-while

for

**while loop**

The syntax of a while loop is:

while (testExpression)

{

//codes

}

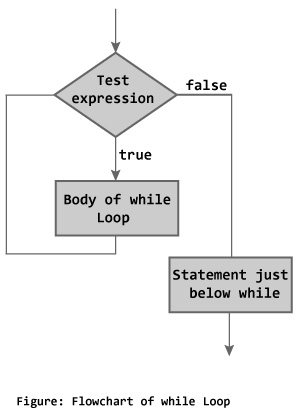
where, testExpression checks the condition is true or false before each loop.

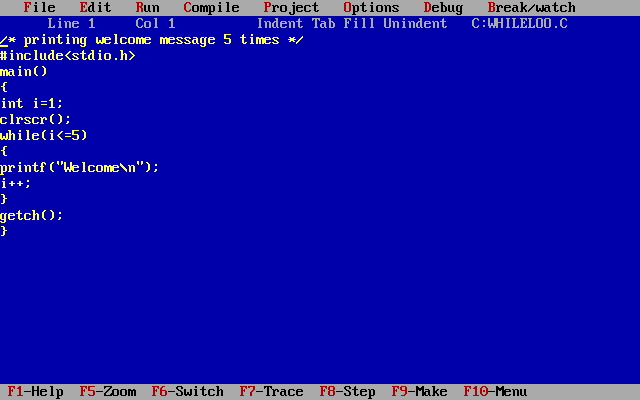
*How while loop works?*

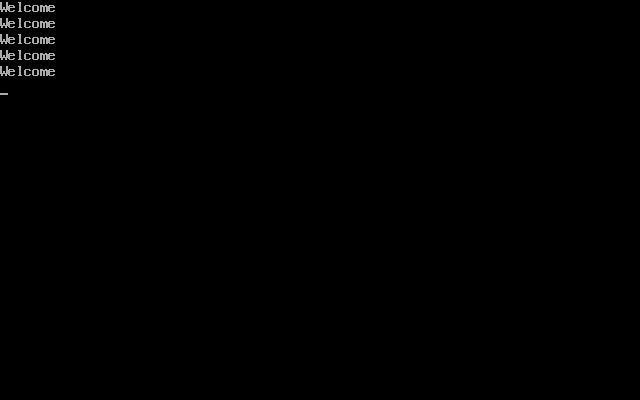
The while loop evaluates the test expression.

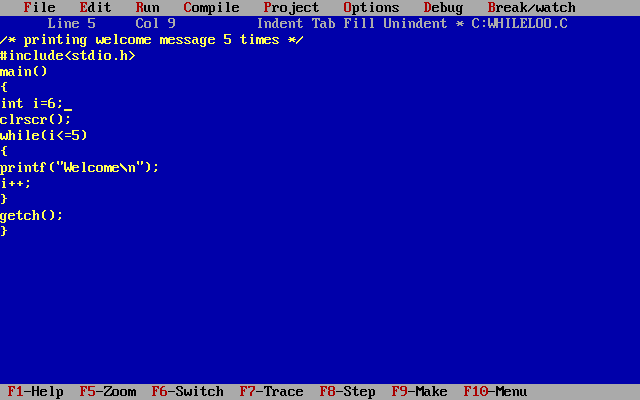
If the test expression is true (nonzero), codes inside the body of while loop are executed. The test expression is evaluated again. The process goes on until the test expression is false. When the test expression is false, the while loop is terminated.

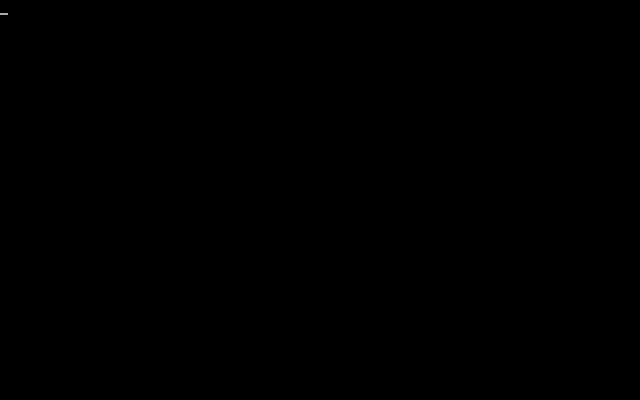
Flowchart of while loop











**do...while loop**

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed once, before checking the test expression. Hence, the do...while loop is executed at least once.

*do...while loop Syntax*

do

{

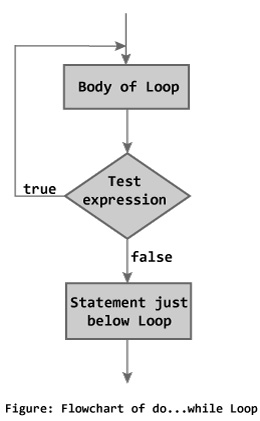
// codes

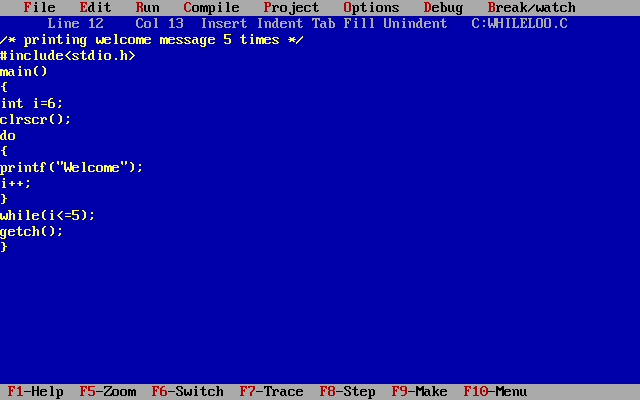
}

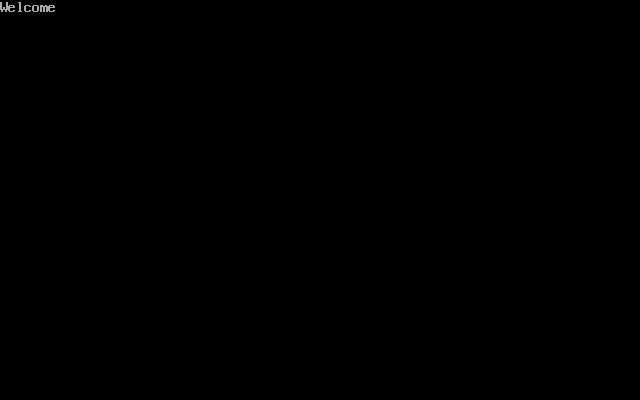
while (testExpression);

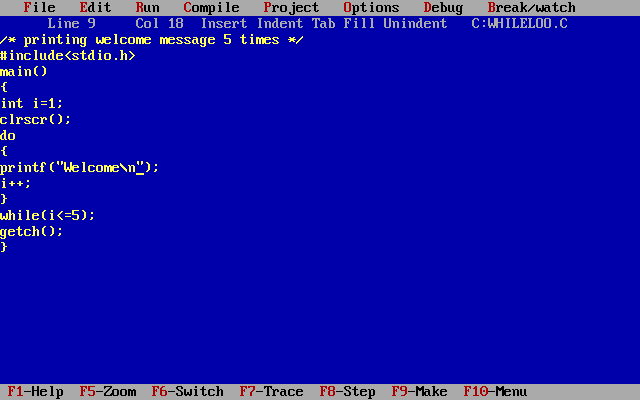
*How do...while loop works?*

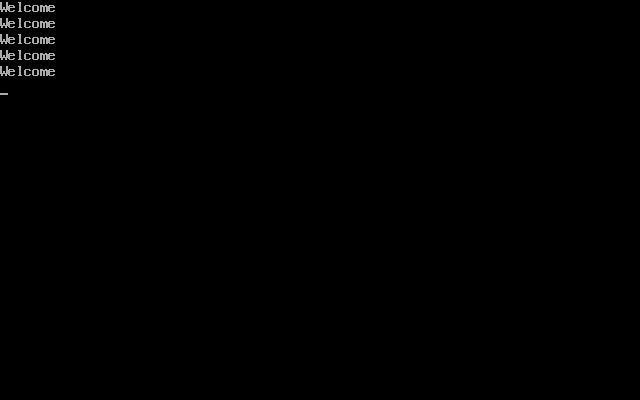
The code block (loop body) inside the braces is executed once. Then, the test expression is evaluated. If the test expression is true, the loop body is executed again. This process goes on until the test expression is evaluated to 0 (false). When the test expression is false (nonzero), the do...while loop is terminated.











**for Loop**

 The syntax of for loop is:

for (initializationStatement; testExpression; updateStatement)

{

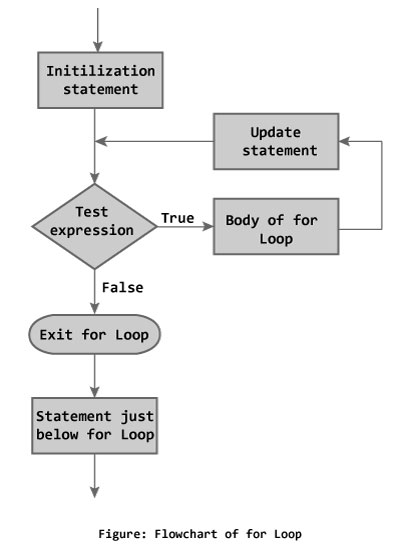
// codes

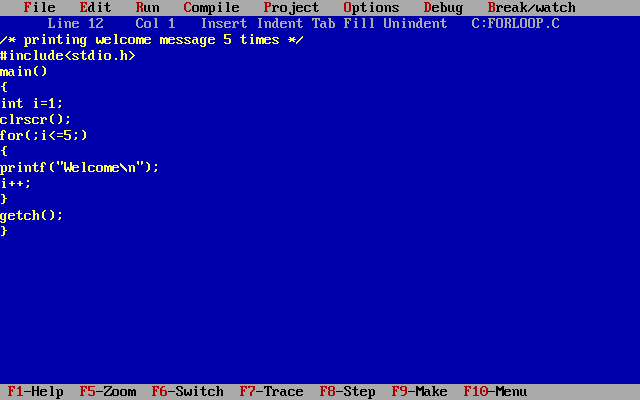
}

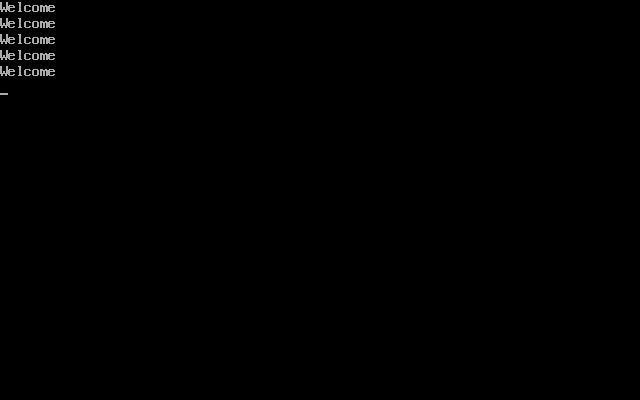
*How for loop works?*

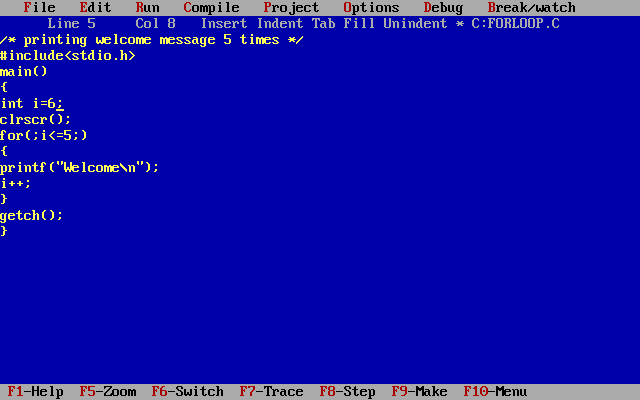
The initialization statement is executed only once. Then, the test expression is evaluated. If the test expression is false (0), for loop is terminated. But if the test expression is true (nonzero), codes inside the body of for loop is executed and the update expression is updated. This process repeats until the test expression is false. The for loop is commonly used when the number of iterations is known. To learn more on test expression (when test expression is evaluated to nonzero (true) and 0 (false)).

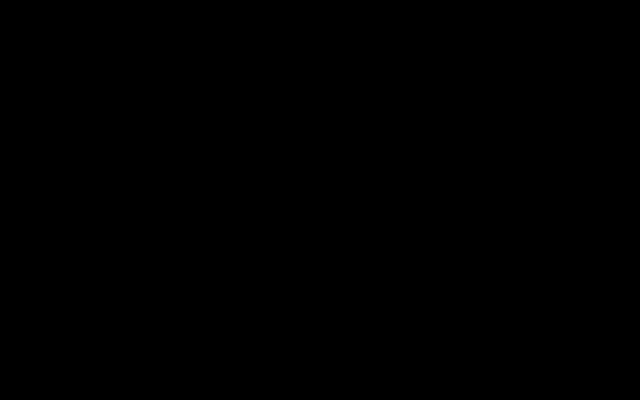
*for loop Flowchart*

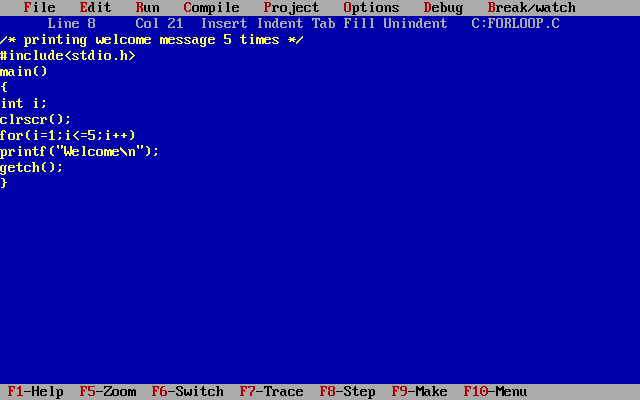


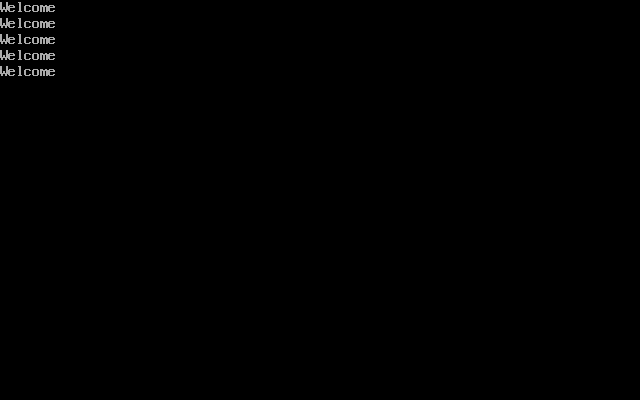




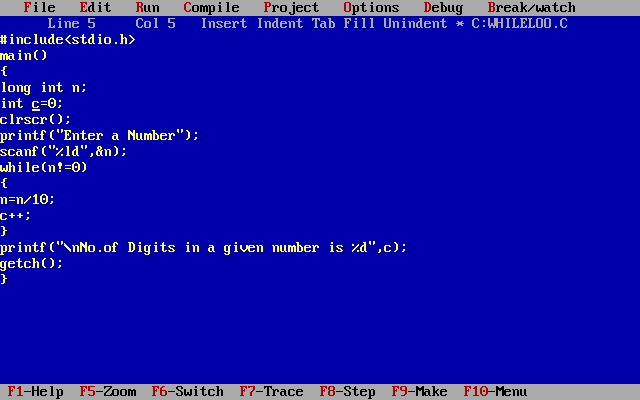


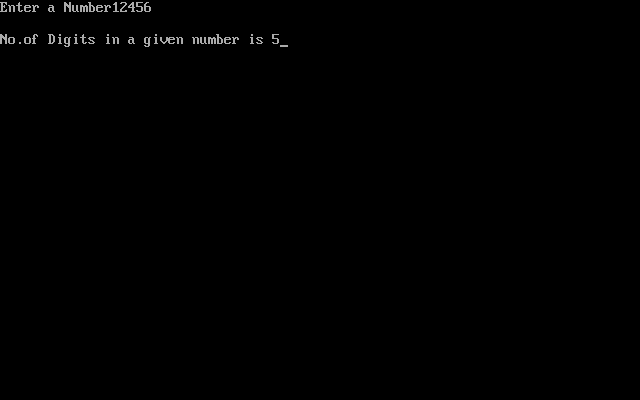


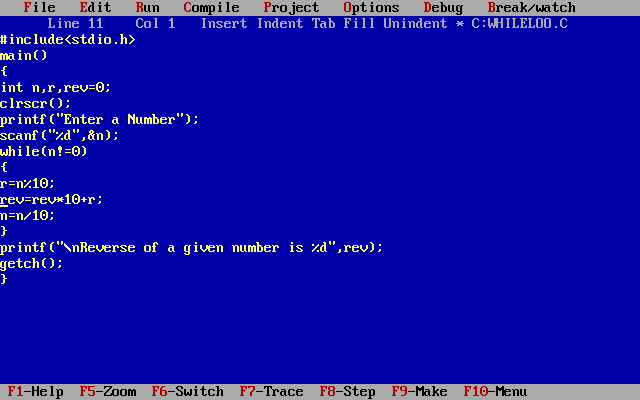




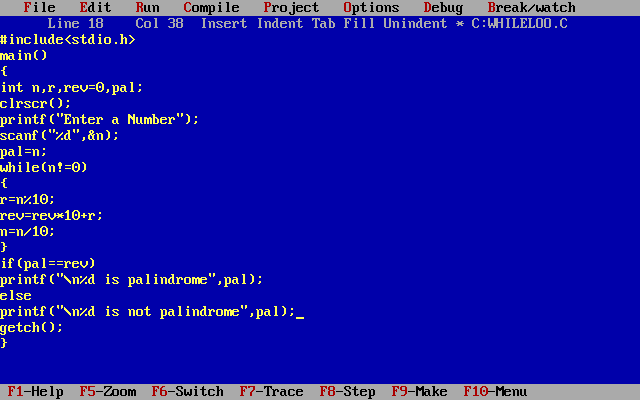
**Write a program read a number check how many digits in a given number.**

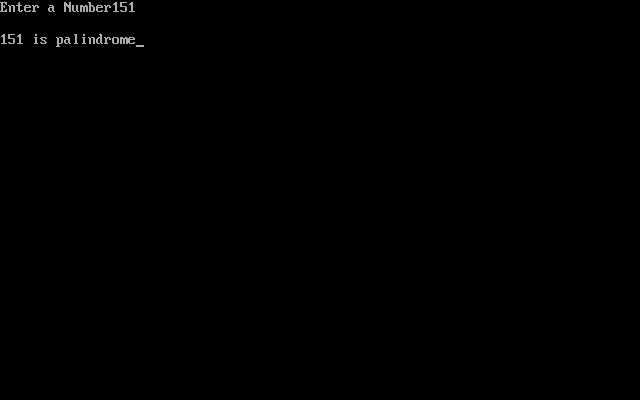




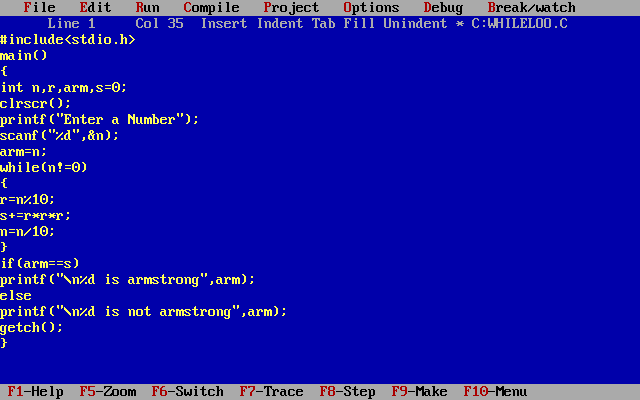




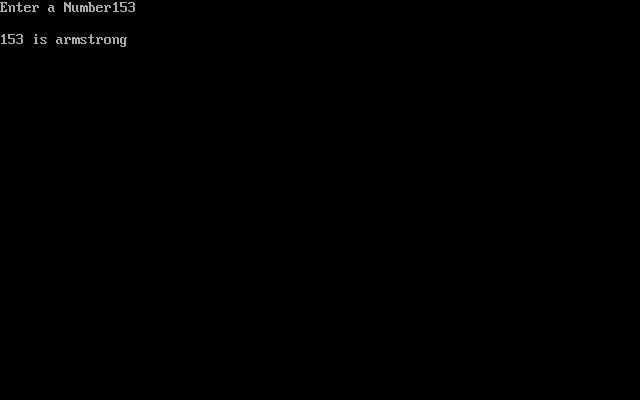




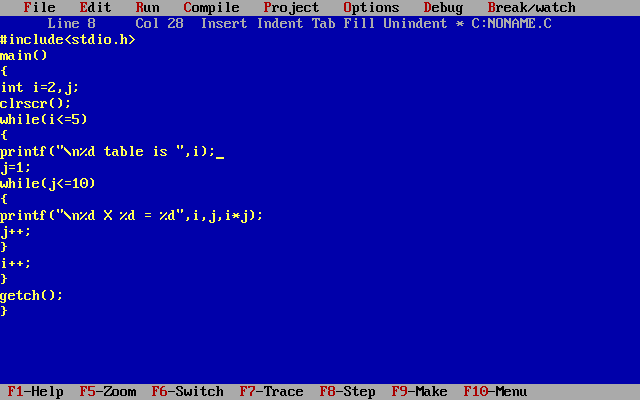


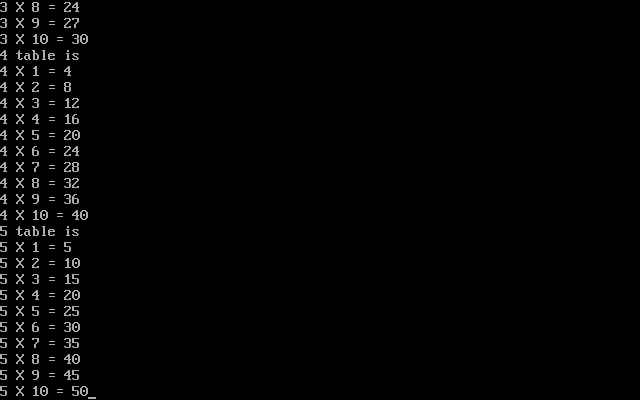


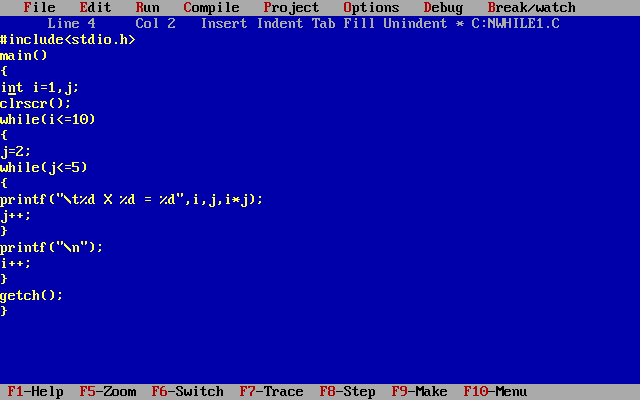


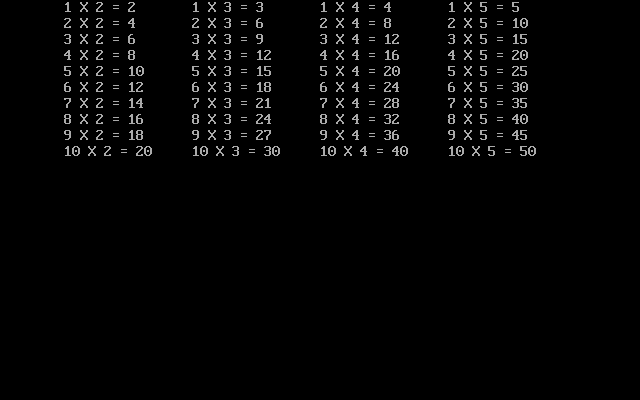


**Nested While loop:**

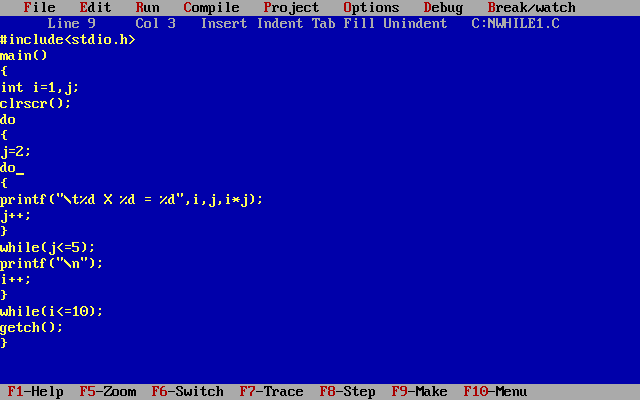


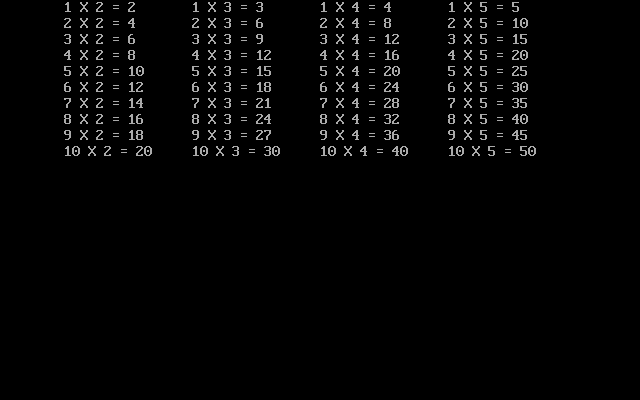






**Nested do-while loop**





**Nested For Loop:-**

Step 1:- First it enter into the initialization. After initialization it goes to the condition.

Step 2:- If the condition is true it enter into the 2nd for loop initialization.

Step 3:- After initialization it enter into the 2nd for loop condition.

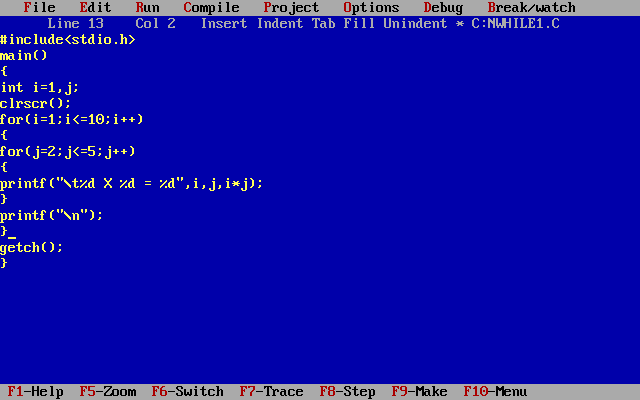
Step 4:- If the 2nd for loop condition is true it enter into the body of the loop.

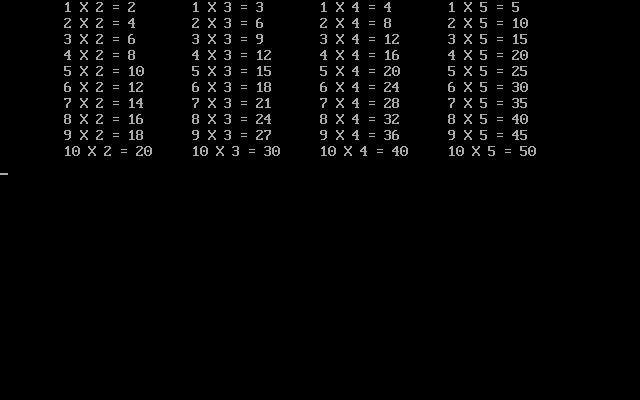
Step 5:-After executing the body of the loop it goes to the 2nd for loop increment / decrement.

Step 6:-After increment/ decrement it enter into the 2nd for loop condition. If the condition is true it executes the body of the loop till the 2nd for loop condition fail. If the condition fail it enter into the first for loop increment / decrement.

Step 7:- After increment / decrement it enter into first for loop condition.

Step 8:- If the 1st for loop condition is true it enter into the 2nd for loop initialization After initialization it enter into the 2nd for loop condition it repeats the step 4,step 5,step 6 till condition fail. After the 2nd for loop condition fail it enter into the first for loop increment/decrement after increment/decrement it enter into the first for loop condition, if it is true it repeats the above steps the till the condition fail.





**Array**

Array is a collection or group of elements (data). All the elements of c array are homogeneous (similar). It has contiguous memory location.

C array is beneficial if you have to store similar elements. Suppose you have to store marks of 50 students, one way to do this is allotting 50 variables. So it will be typical and hard to manage. For example we can not access the value of these variables with only 1 or 2 lines of code.

Another way to do this is array. By using array, we can access the elements easily. Only few lines of code is required to access the elements of array.

**Advantages**

1) **Code Optimization:** Less code to the access the data.

2) **Easy to traverse data:** By using the for loop, we can retrieve the elements of an array easily.

3) **Easy to sort data:** To sort the elements of array, we need a few lines of code only.

4) **Random Access:** We can access any element randomly using the array.

**Disadvantage**

1) **Fixed Size:** Whatever size, we define at the time of declaration of array, we can't exceed the limit. So, it doesn't grow the size dynamically like LinkedList.

**Declaration of Array**

data\_type array\_name[array\_size];

Ex:- int marks[5];

Here, int is the data\_type, marks is the array\_name and 5 is the array\_size.

**Initialization of Array**

A simple way to initialize array is by index. Notice that array index starts from 0 and ends with [SIZE - 1].

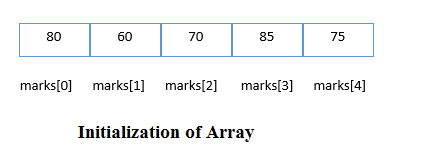
marks[0]=80;//initialization of array

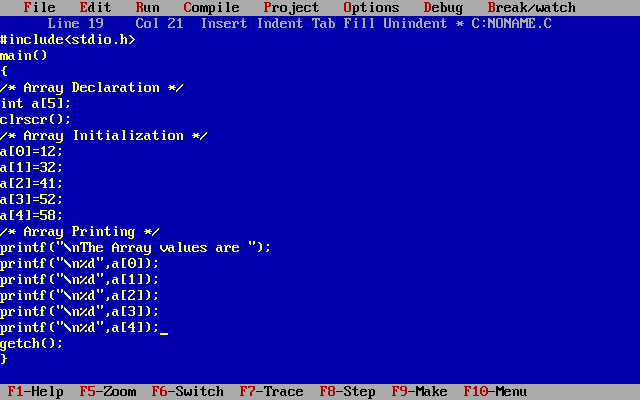
marks[1]=60;

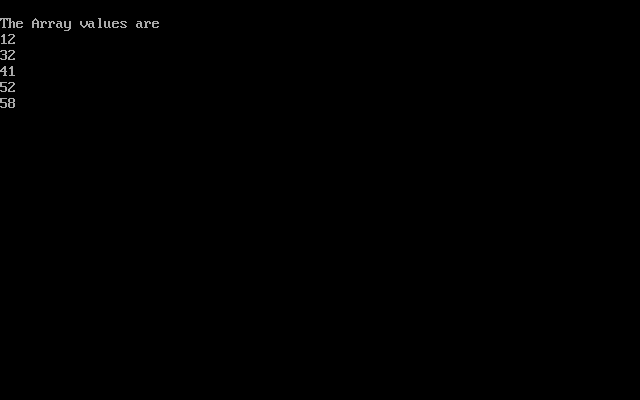
marks[2]=70;

marks[3]=85;

marks[4]=75;







**Declaration with Initialization**

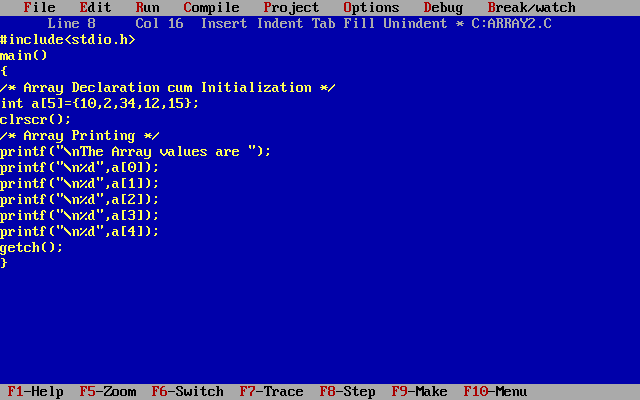
An array is declared it must be initialized. Otherwise, it will contain garbage value(any random value). An array can be initialized at either compile time or at runtime.

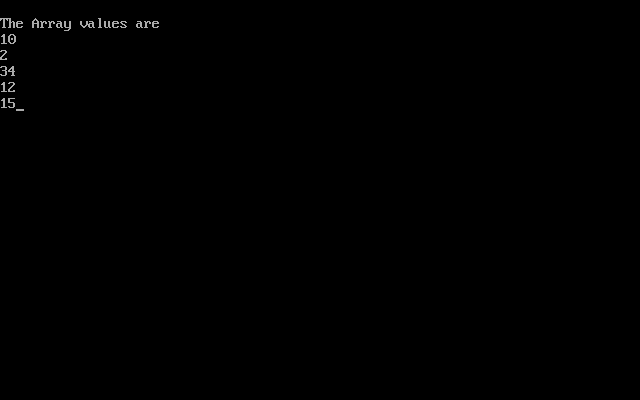
**Compile time Array initialization**

Compile time initialization of array elements is same as ordinary variable initialization. The general form of initialization of array is,

data-type array-name[size] = { list of values };

Ex:- int marks[5]={20,30,40,50,60};

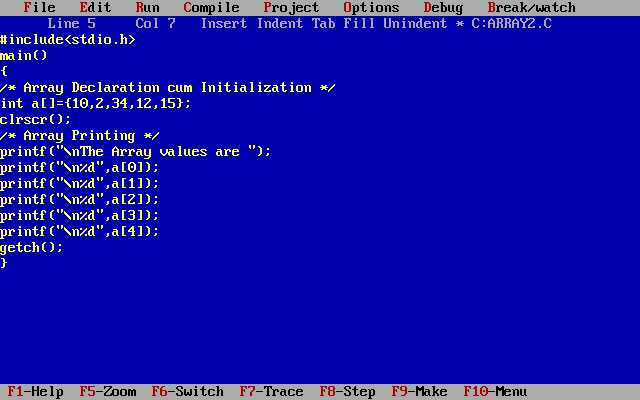


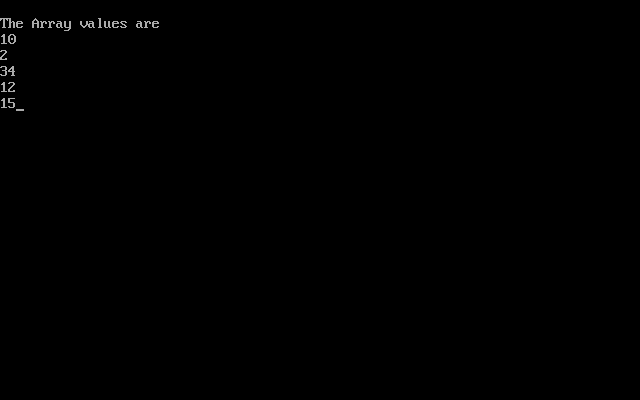


In such case, there is no requirement to define size. So it can also be written as the following code.

Ex:- int marks[]={20,30,40,50,60};

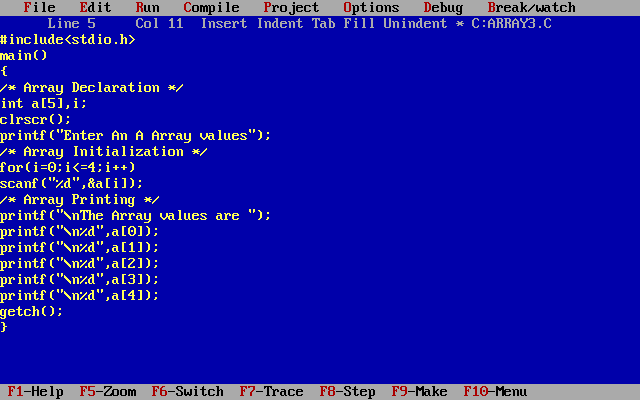
One important thing to remember is that when you will give more initializer (array elements) than the declared array size than the compiler will give an error.



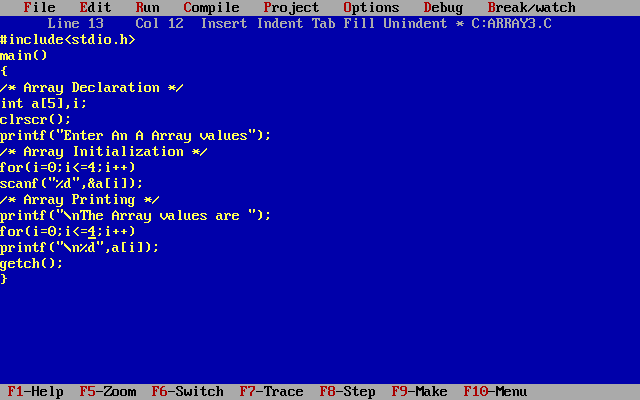


**Runtime Array initialization**

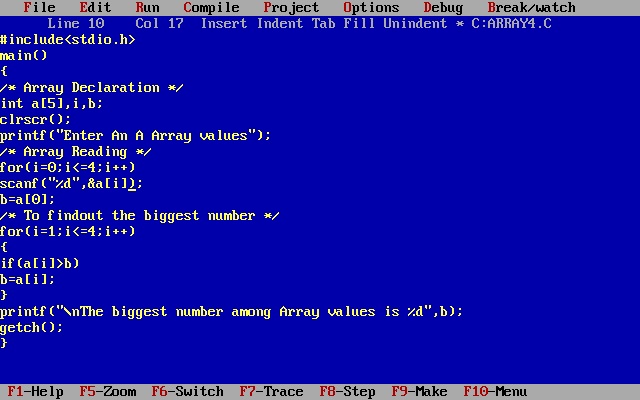
An array can also be initialized at runtime using scanf() function. This approach is usually used for initializing large arrays, or to initialize arrays with user specified values. Example,













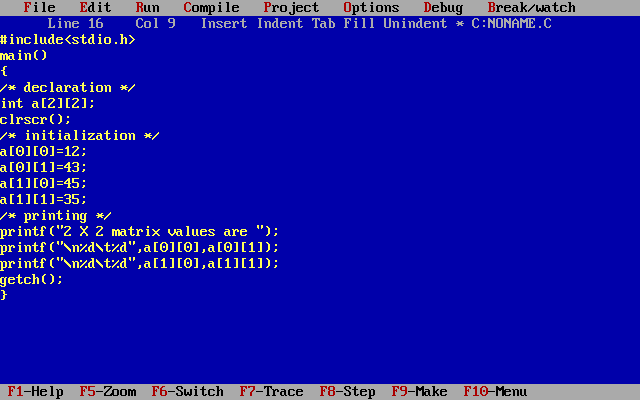
*Example of Array In C programming to find out the average of 4 integers*

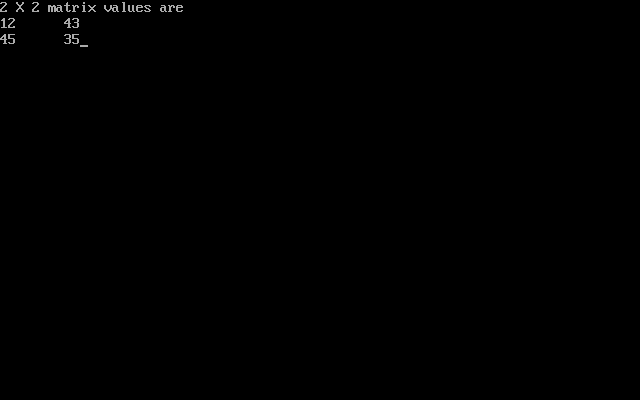
**TWO DIMENSIONAL ARRAY:**

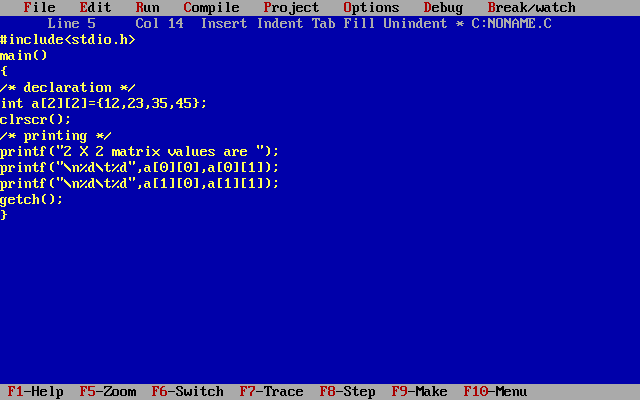
Two dimensional array is nothing but array of array.

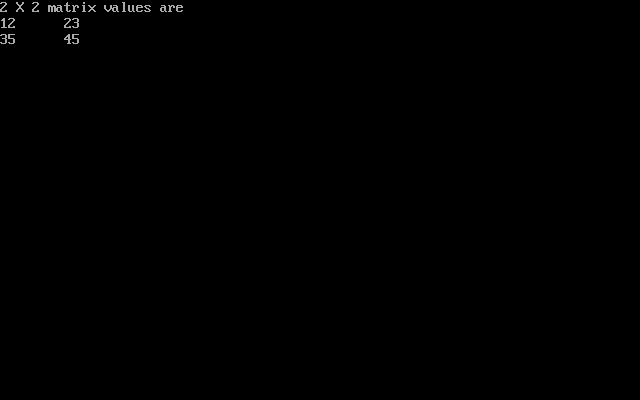
syntax : data\_type array\_name[num\_of\_rows][num\_of\_columns];

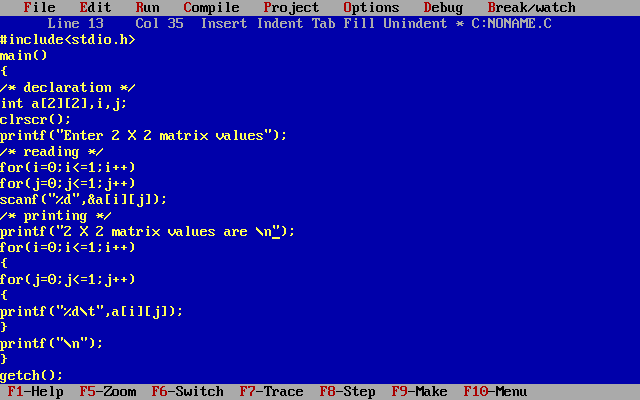
|  |  |
| --- | --- |
| Array declaration, initialization and accessing | Example |
| Array declaration syntax: data\_type arr\_name [num\_of\_rows][num\_of\_column];  Array initialization syntax:  arr\_name[rowindex][columnindex]=value; Array initialization syntax: data\_type arr\_name[2][2] = {{0,0},{0,1},{1,0},{1,1}}; | //Integer array declaration example:  int arr[2][2];  //Array initialization example arr [0] [0] = 1; arr [0] [1] = 2; arr [1][0]  = 3; arr [1] [1] = 4;  //declaration cum initialization example  int arr[2][2] = {1,2, 3, 4}; |

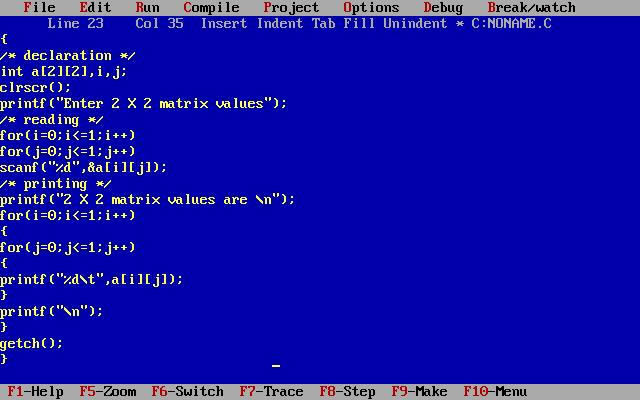




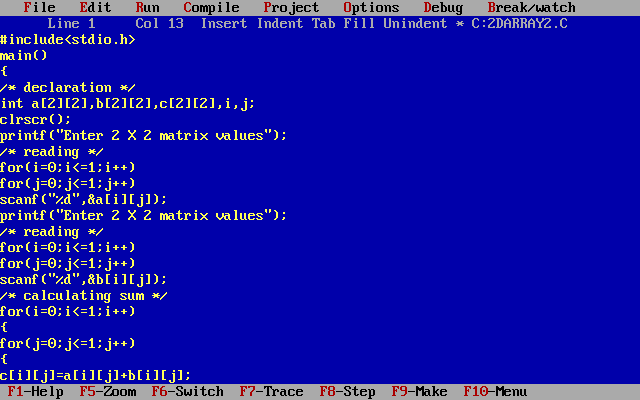


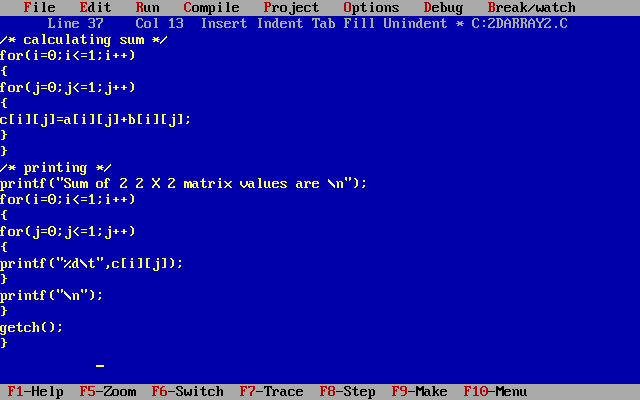




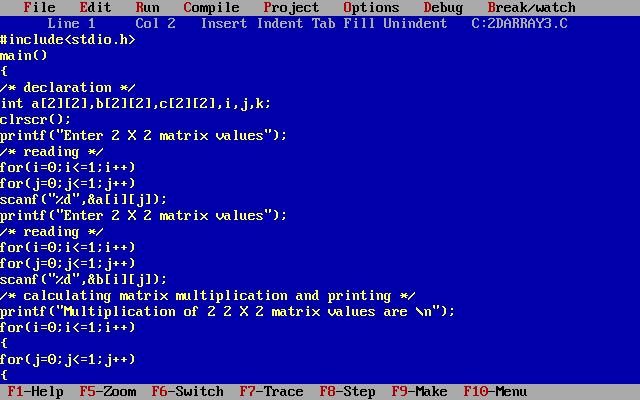


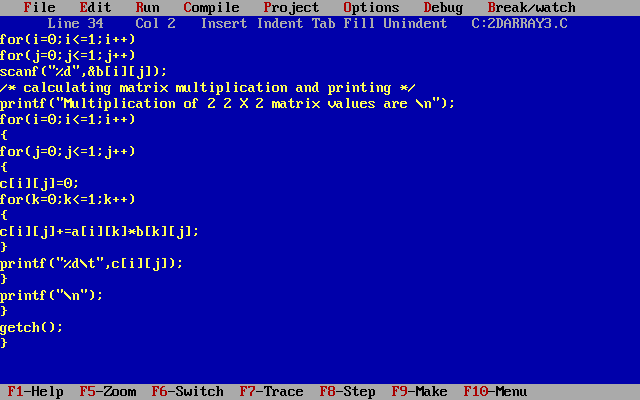


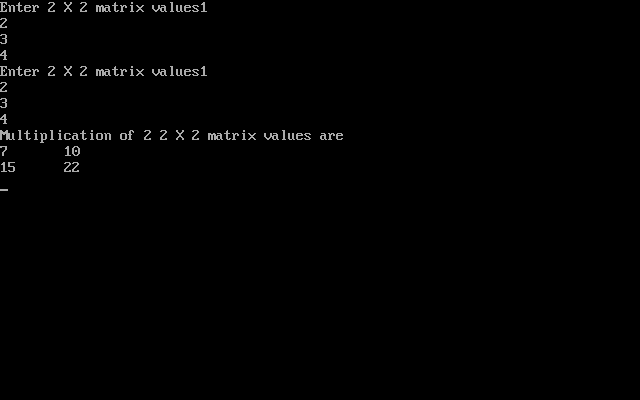






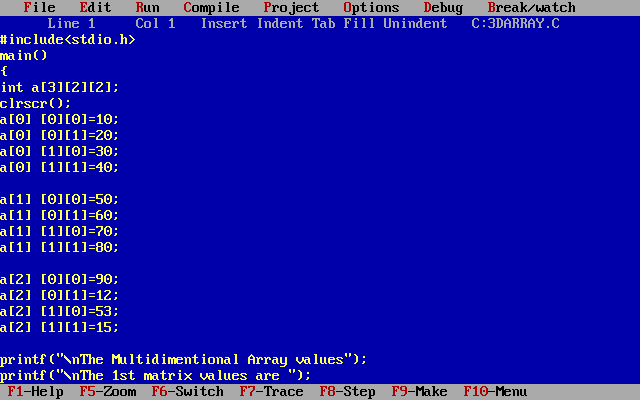


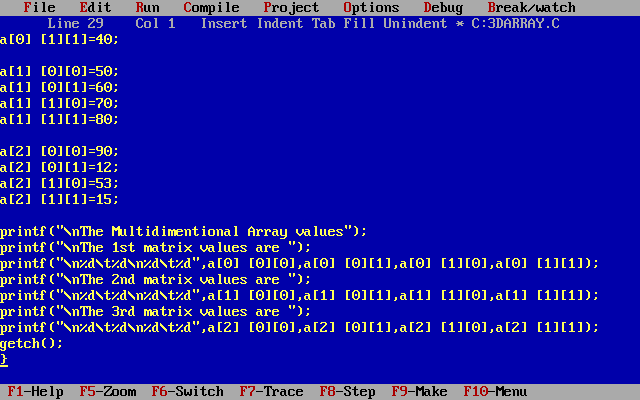


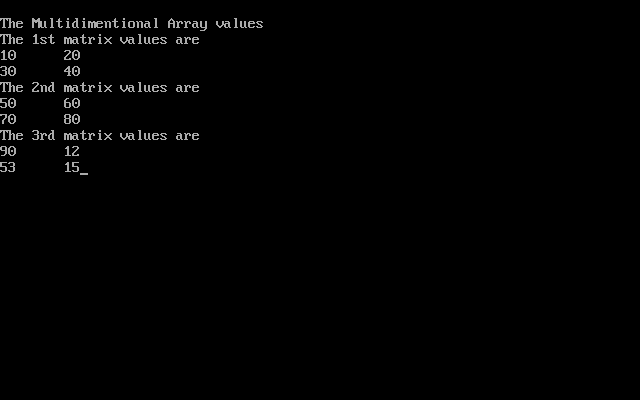


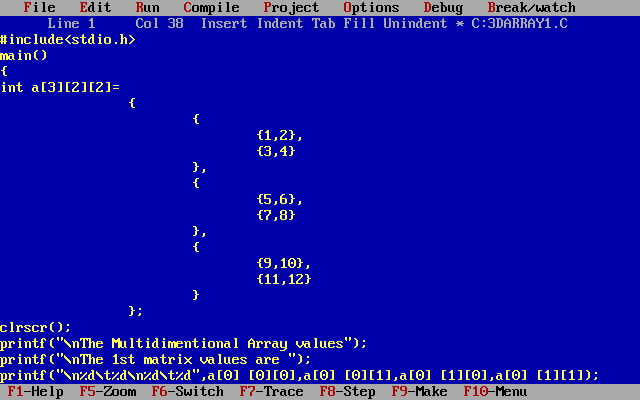
**Multidimentional Array:**

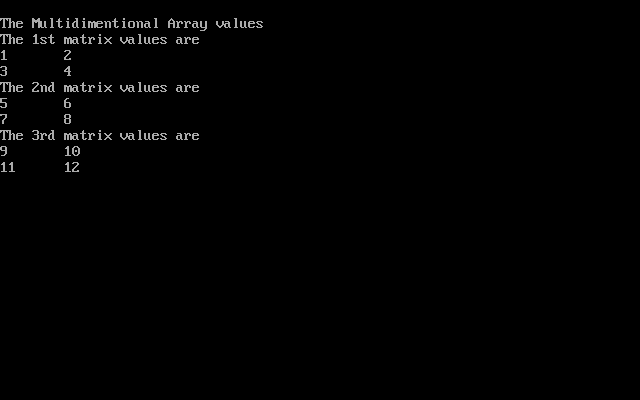
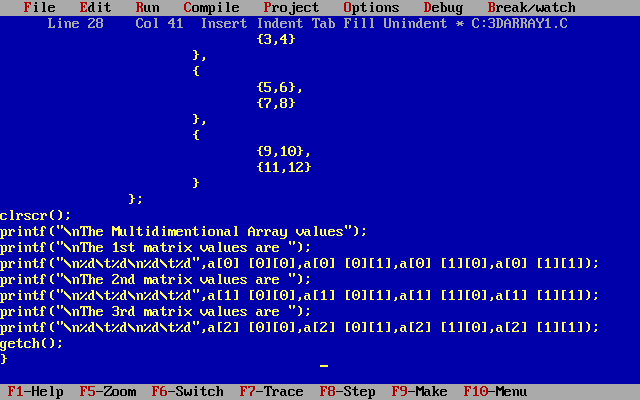
It is a memory allocation which enable you to store homogeneous mixture of elements in form of tables (or) in the form of array of arrays.



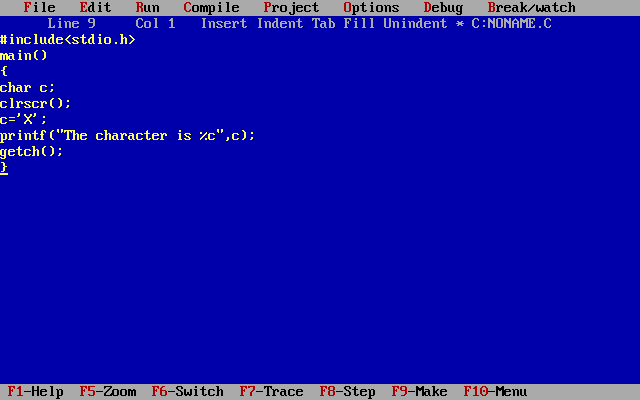


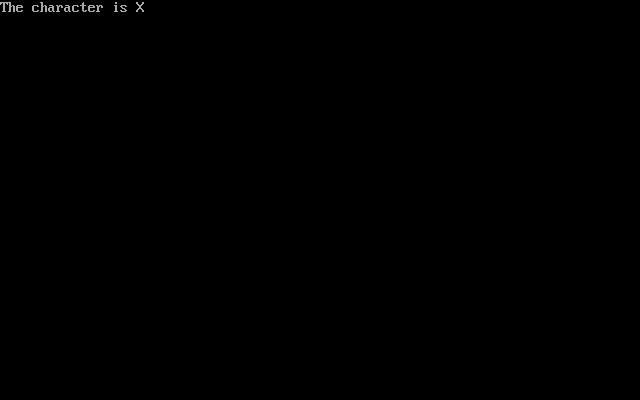


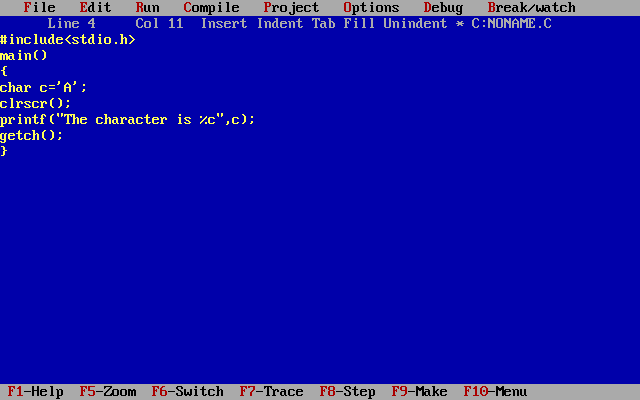


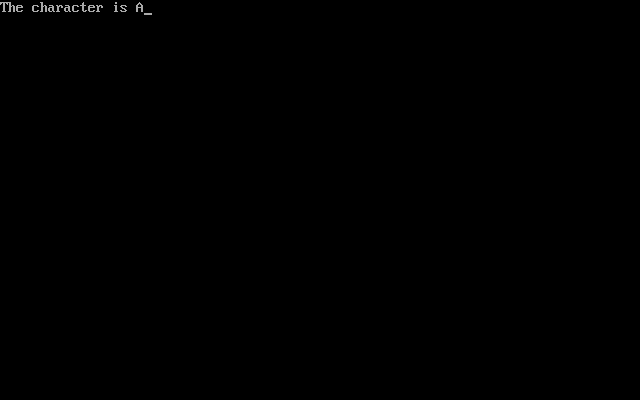


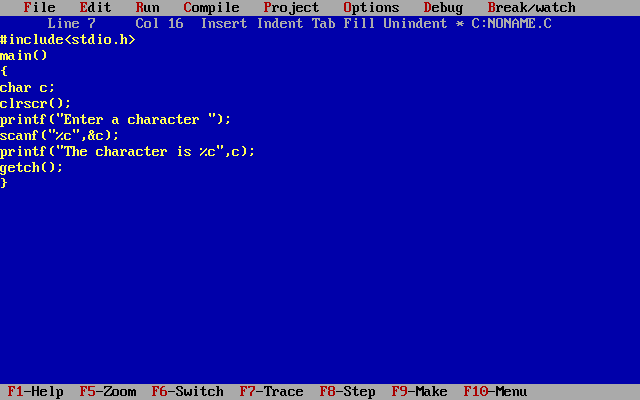
**Character**

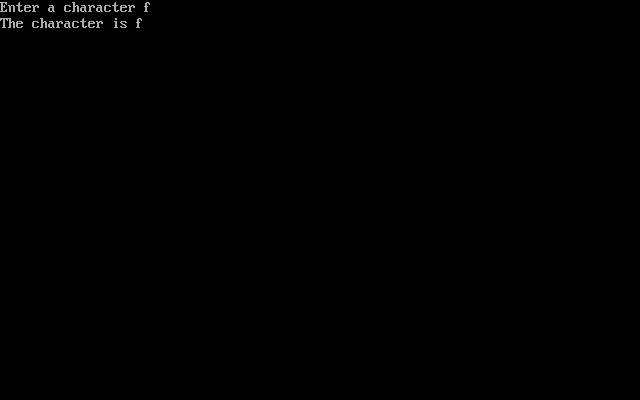








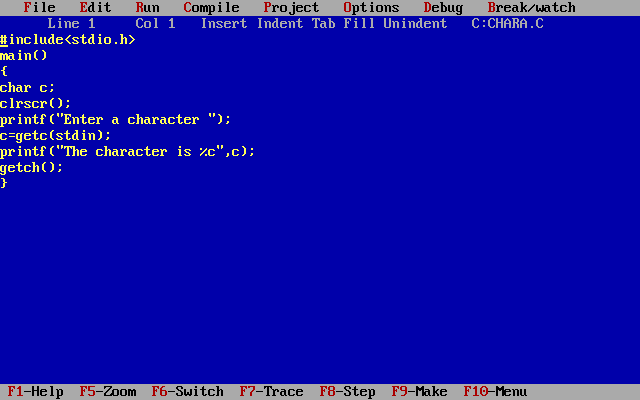


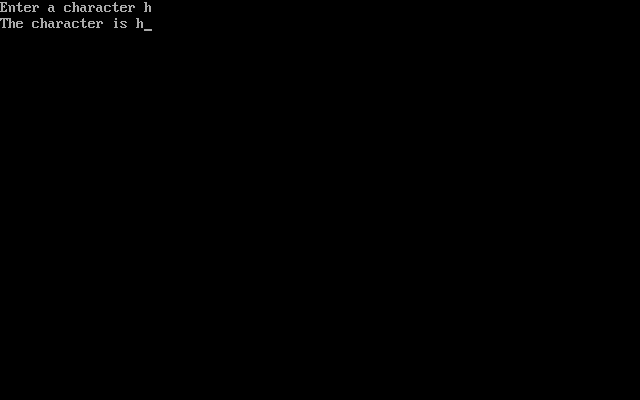


**Character Reading Functions:**

**Getc()**:- It accepts one character to be read from the device during program execution.

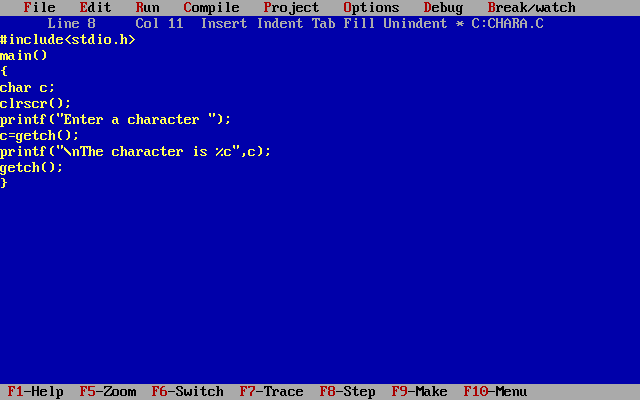
**Syntax**:- variable=getc(stdin);

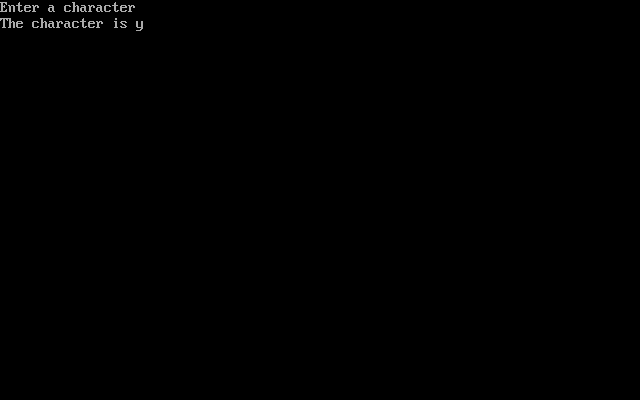




**getch()**:-getch() is used to read a single character from the user without echoing.

Syntax:- variable=getch();

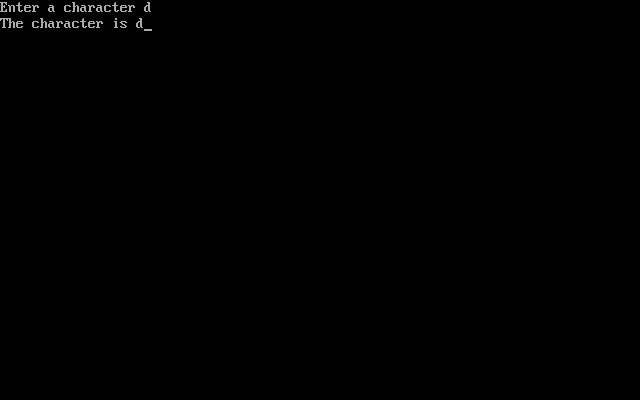




**getche()**:- getche() is used for reading a single character from the user with echoing and process it immediately.

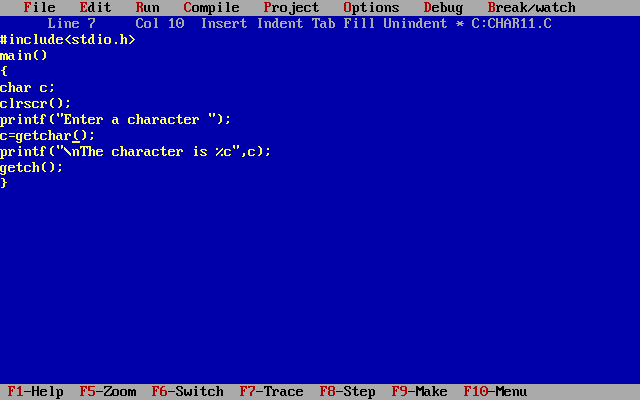
Syntax:- variable=getche();

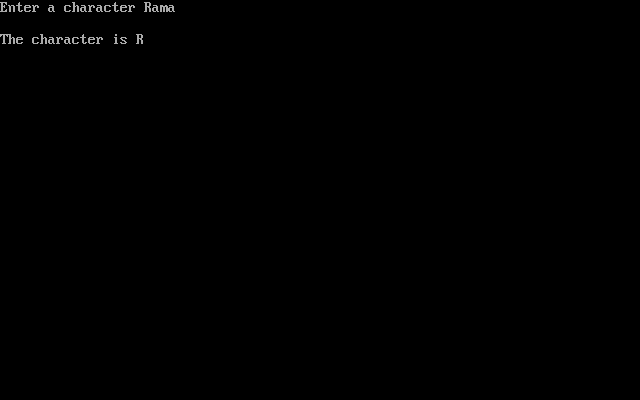




**getchar()**:- getchar() is used for reading a single character from the user with echoing and does not process it immediately.

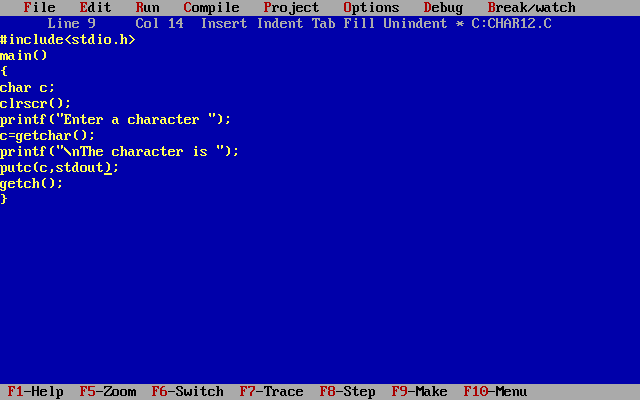
Syntax:- variable=getchar();

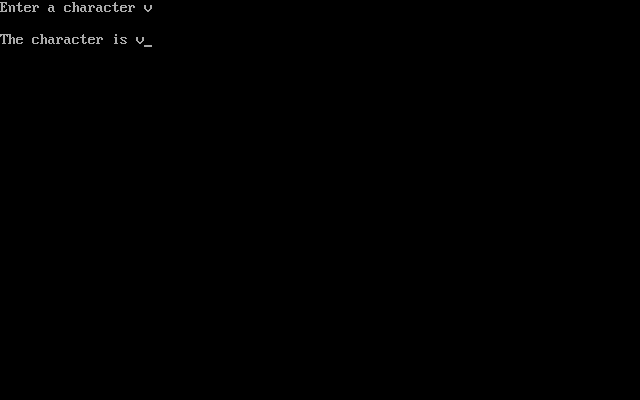


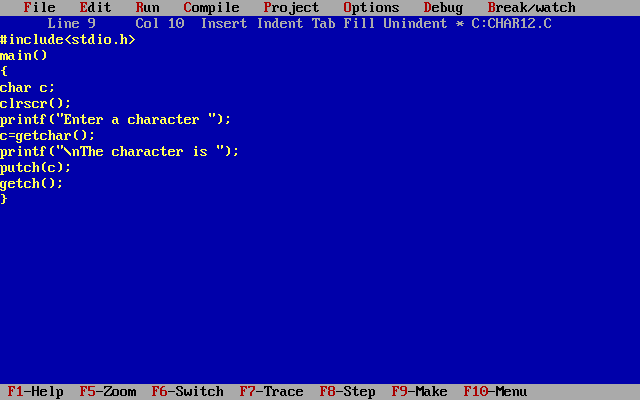


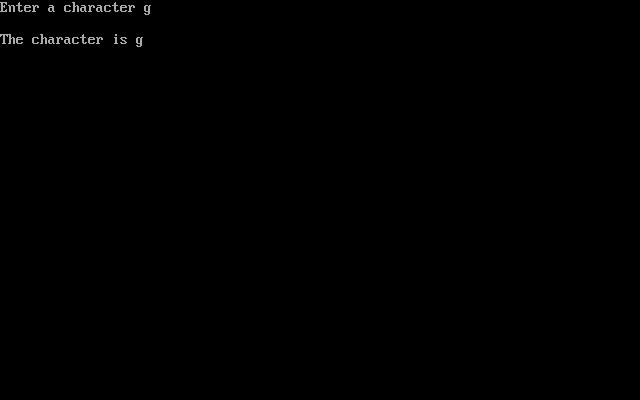
**Putc()**:- It prints one character during program execution.

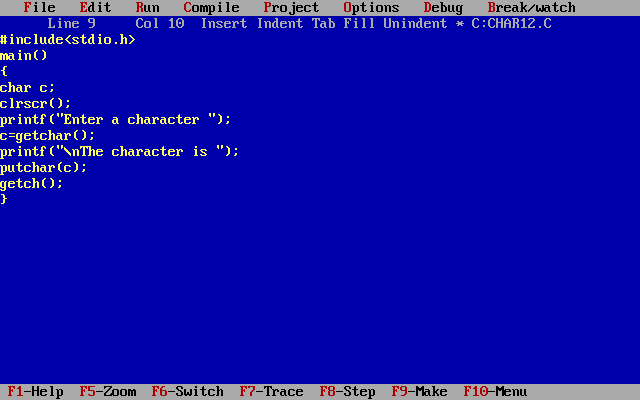
**Syntax**:- variable=putc(char,stdout);

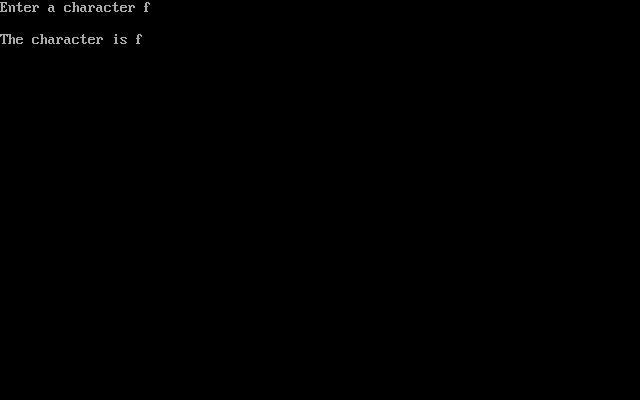












**Single Character handling functions:-**

**islower(character):-**only lowercase alphabetic.

**isupper(character):-**only uppercase alphabetic.

**isalnum(character):-**Only alphanumeric: any alphabetic or numeric character.

**ispunct(character):-** Any graphic character that is not alphanumeric.

**isalpha(character):-** Any alphabetic character, upper (or) lower case.

**isdigit(character):-** Decimal digits (0…9)

**isxdigit(character):-** Hexadecimal digits(0 -9,a-f,A-F)

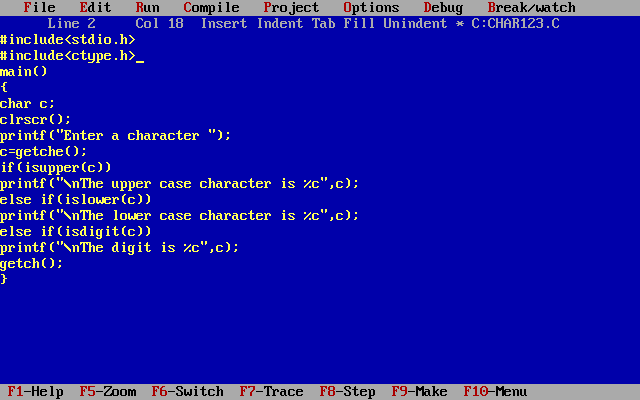
**isodigit(character):-** octal digits( 0 -7)

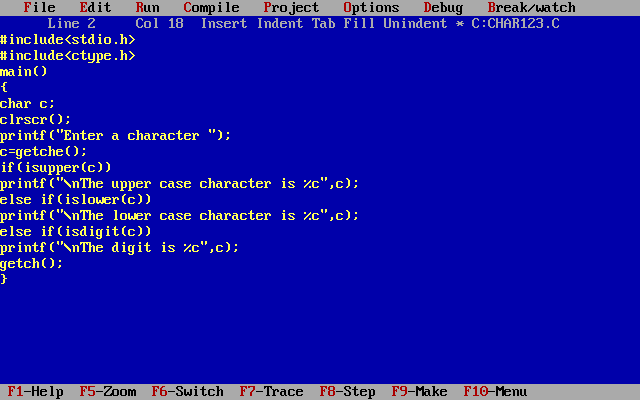
**isspace(character):-** Whitespace character: space character(32), horizontal tab(9),vertical tab(11), line feed(10), vertical tab(11),form feed(12), and carriage return (13).

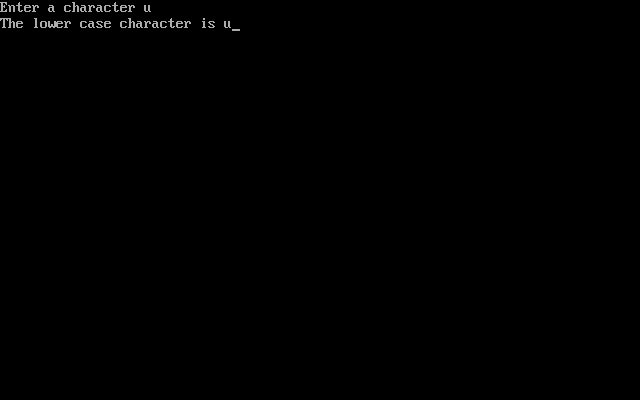
**isgraph(character):-** Character with printable graphic; all printable characters except space.

**isprint(character):-** Printable character, that is character with an assigned graphic.

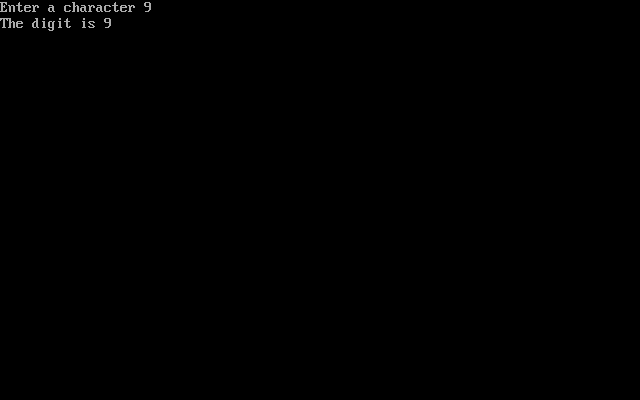
**Iscntrl(character):-** Printable character, that is character with an assigned graphic.











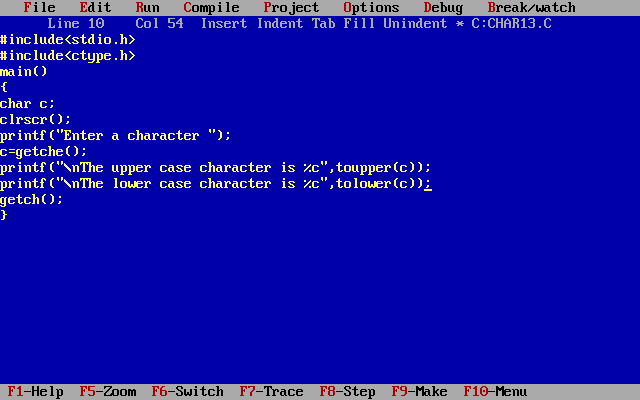
**Character Convertion functions:-**

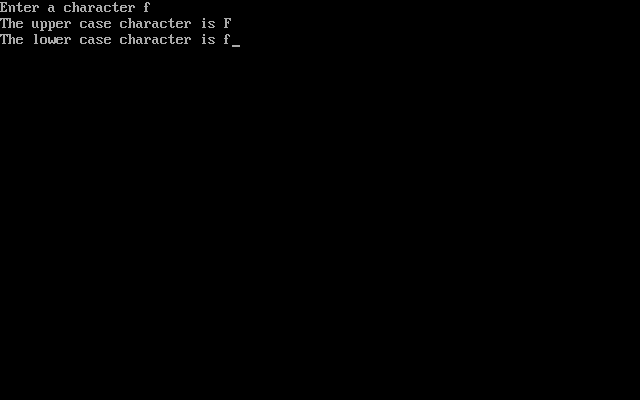
**1) toupper() :-** This function converts a single character to upper case.

**syntax:-**  toupper(character);

**2) tolower() :-** This function converts a single character to lower case.

**syntax:-** tolower(character);





**Note:-** when we are using single character functions we have to include a header file <ctype.h>

<ctype.h> supports the ASCII character set. <wctype.h> supports wide characters contains some functions not found in the ASCCII character library (ctype.h). Ex:- ASCII library(ctype.h) contains islower, wide-character library is iswlower follow a common naming format to test for lowercase alphabetic characters.

**Fflush():-** which is used to clear the non-printable characters which are occurring through standard input technically, flush as the stream. It is belongs to stdio.h header file.

**String**

String is a collection of characters. The compiler implicitly which plays the null character (\0) at the end of each string. Hence, Every string ends with a null value. Null is represented as \0. \0 is not a space. \0 is a non printable character.

**How to declare string?**

Syntax:-

<datatype> <variablename>[size];

Ex:- char a[10];

How to delcare and initialize a string in a single statement?

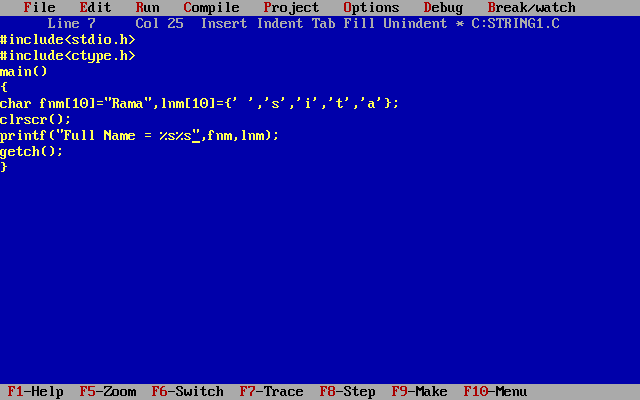
Syntax1:-

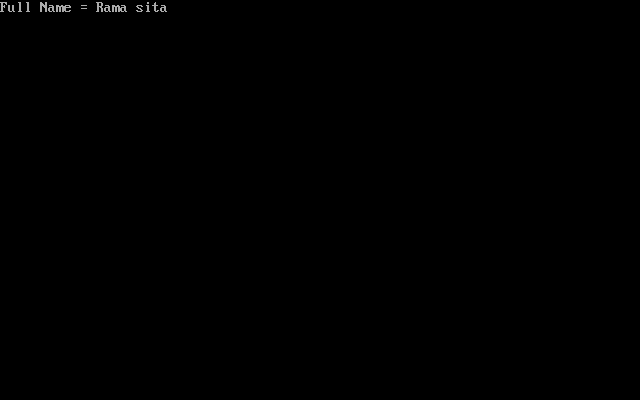
<datatype> <variablename>[size]=“type some string”;

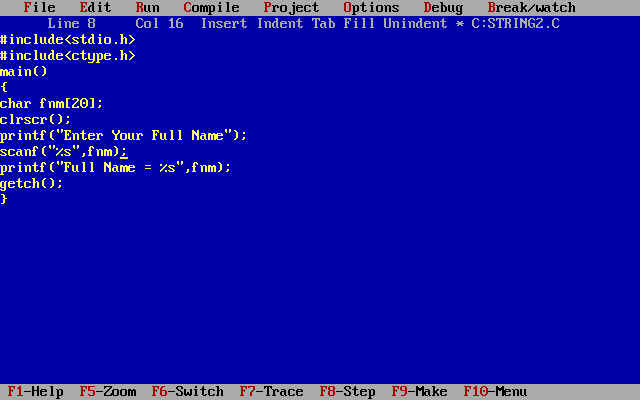
Note:- If the string is not properly terminated at the time of printing the string we will get some garbage value along with the string.

Syntax2:- <datatype> <variablename>[size]={‘character’,’character’,’character’,…};

String initialization is not possible. Declaration, declaration cum initialization is possible.







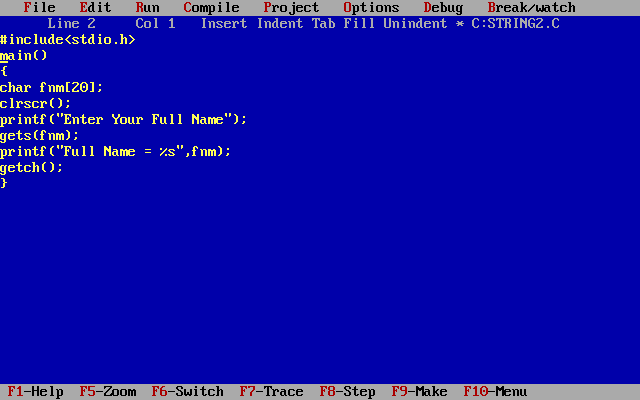


**gets()**:- gets() is used to read a string. Scanf statement is used to read any type of data but after the space if you type any string scanf doesnot consider so we use gets() for only strings.

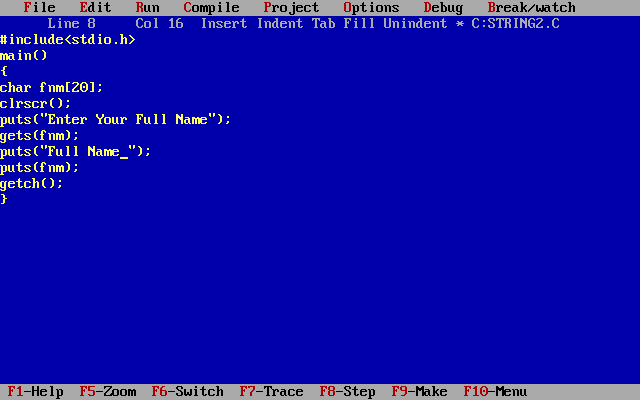
**puts()**:- puts() is used to print a string. Instead of printf statement we use puts().

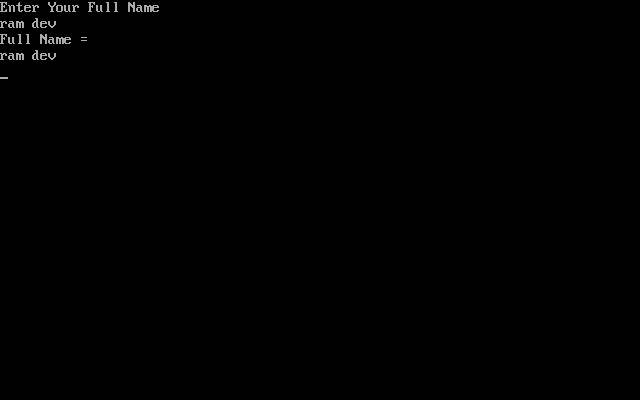
When you use any gets() or puts() it doesnot allow control strings.

When reading a string using scanf Address Operator(&) is optional in the scanner if & only if when your extracting the string from the user.







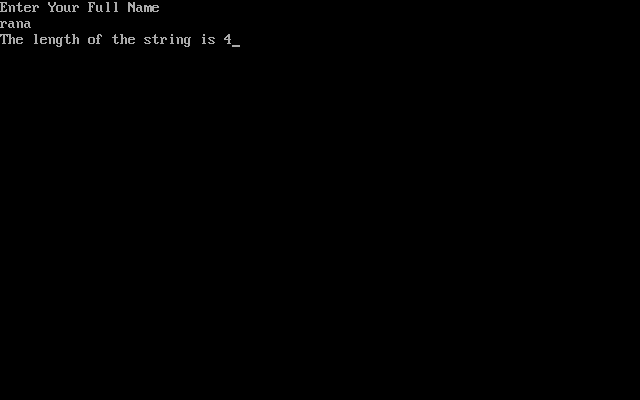


**String handling functions:-**

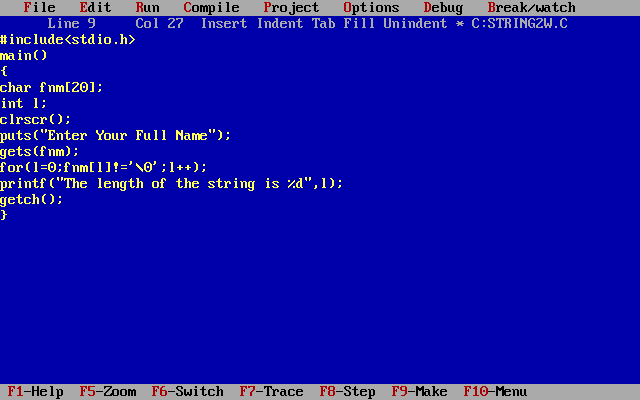
**1) strlen():-** This function is used to return the length of a string.

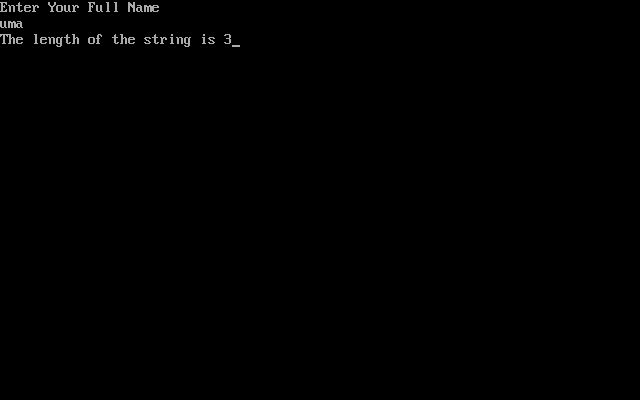
**syntax:-** strlen(string);





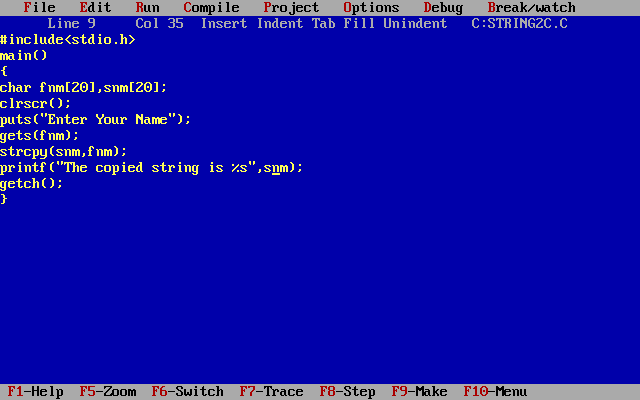
**Note:-** When we are using string functions we have to include a header file <string.h>

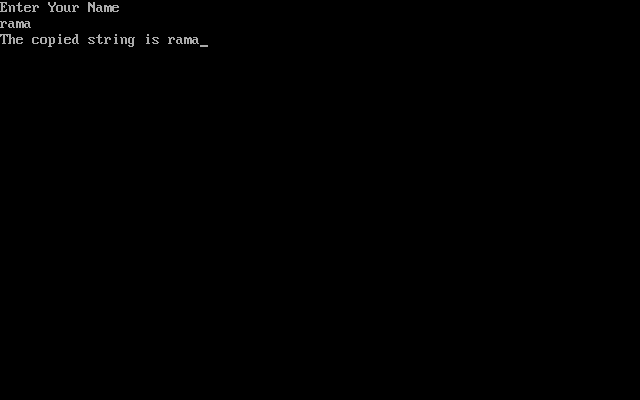


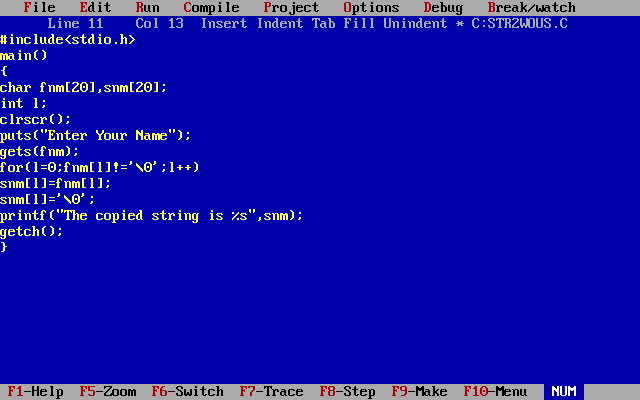


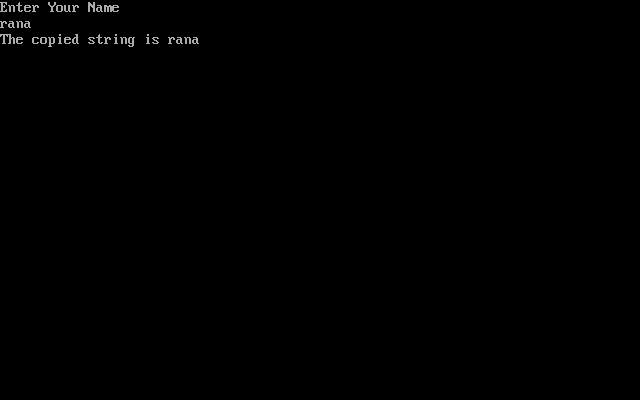
**2) strcpy():-** This function copies a string from one string variable to another string variable.

**syntax:-** strcpy(target\_string , source\_string);





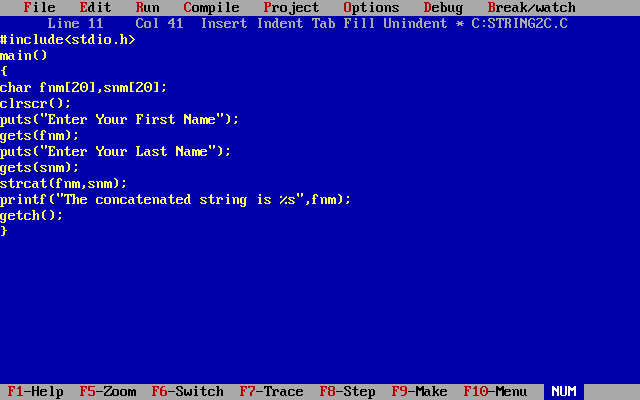




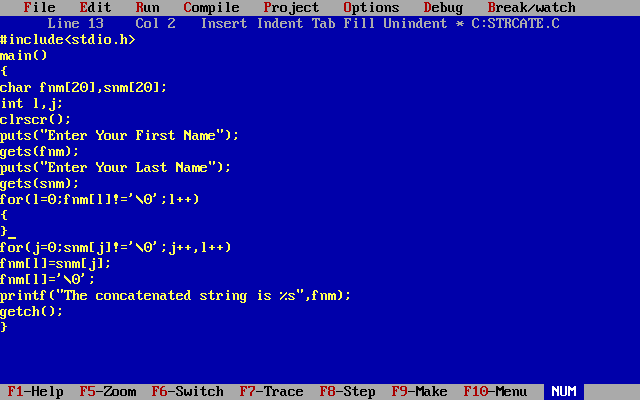
**3) strcat(): (String concatenation)**

This function adds a string at the end of another string

**syntax:-**  strcat(string1,stirng2);





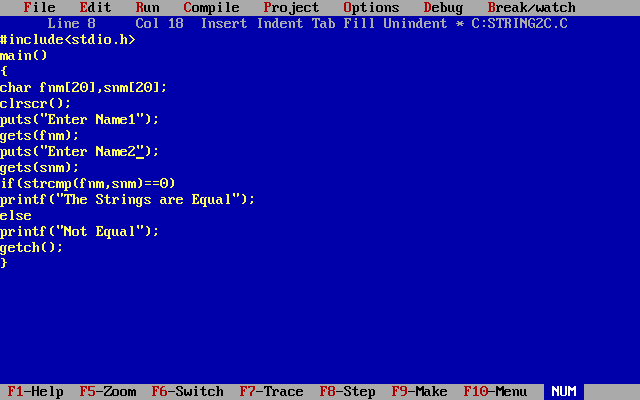


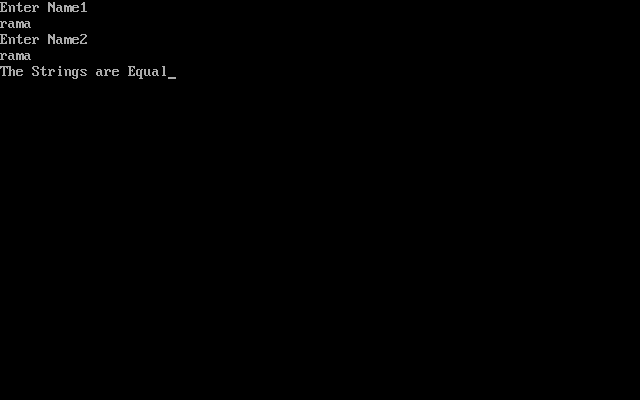


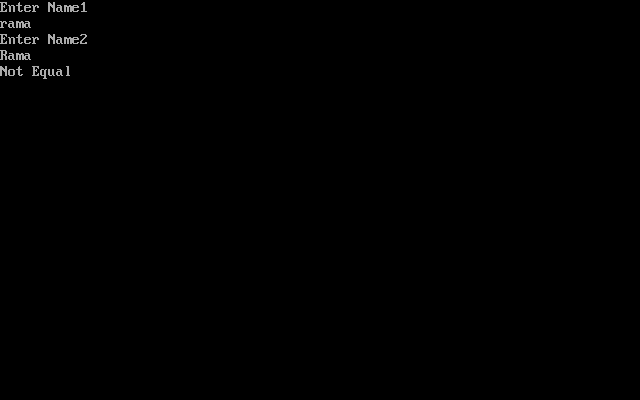
**4) strcmp(): (String comparison)**

This compares one string with another string, if two strings are equal this function returns 0, else it returns the numeric difference between the non-matching characters.

**Syntax:-** strcmp(string1, string2);







**Strrev():** This function reverses the given string

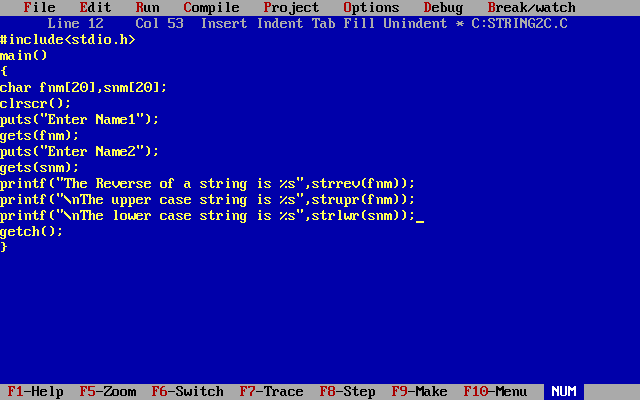
**Syntax:-** strrev(string);

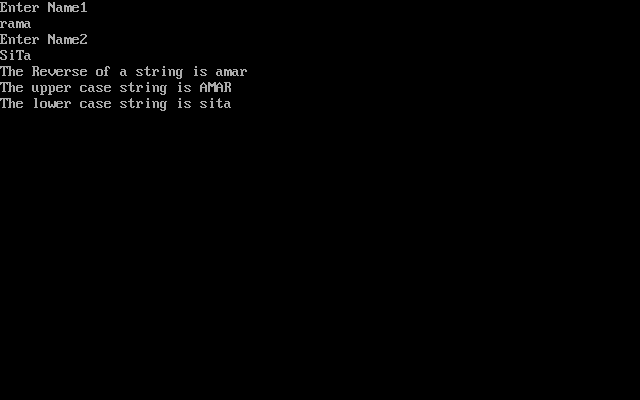
**5) strupr():** This function converts the given string into upper case(capitals)

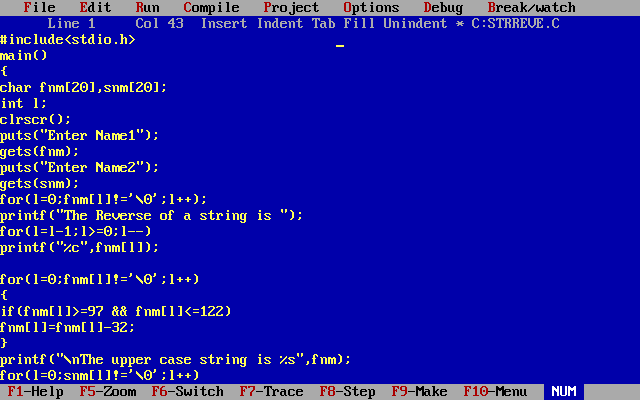
**syntax:-** strupr(string)

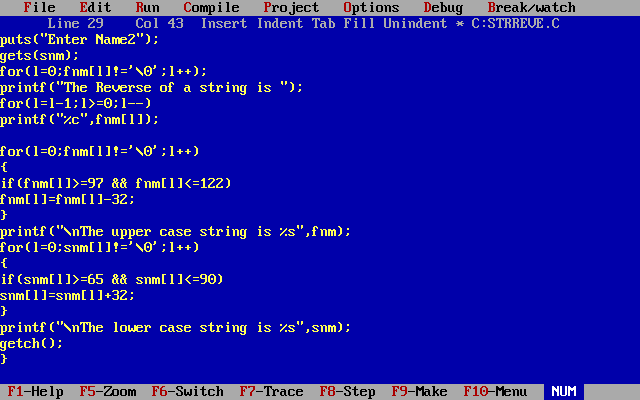
**6) strlwr():** This function converts the given string into lower case.

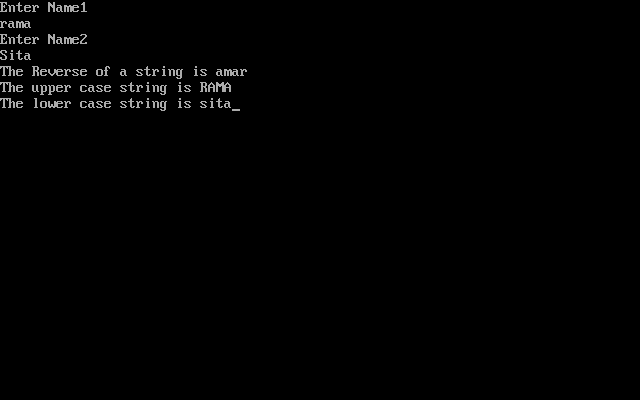
**syntax:-** strlwr(string);

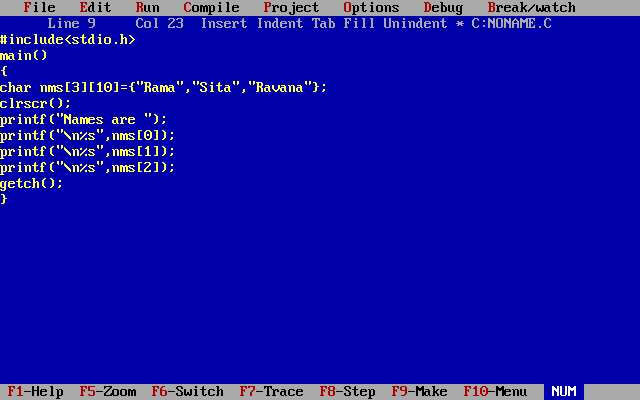


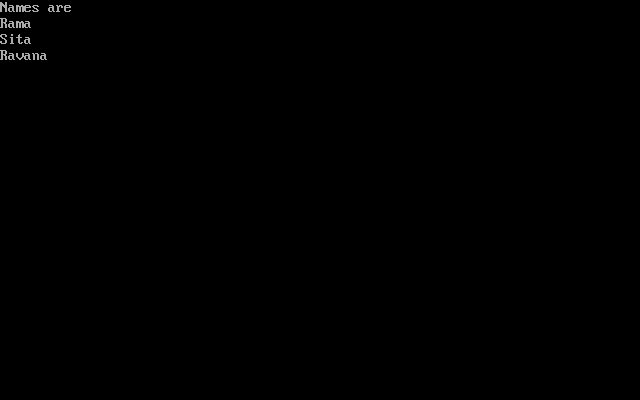


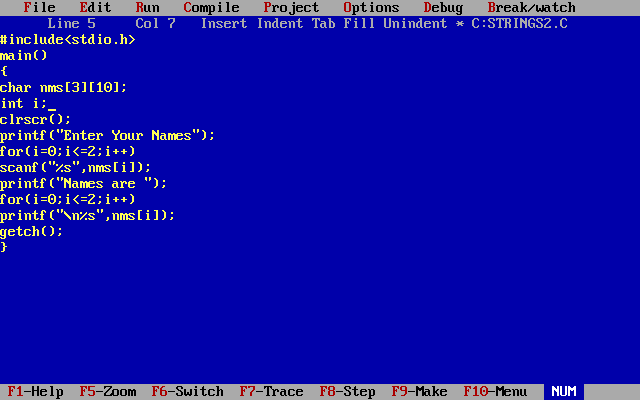


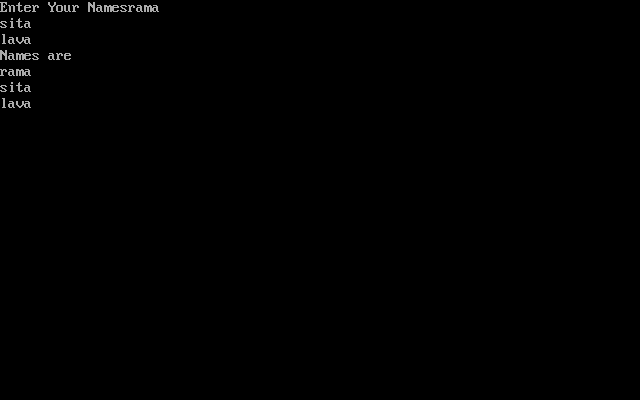












**Functions**

The function in C language is also known as procedure or subroutine in other programming languages. C functions are basic building blocks in a program. To perform any task, we can create function. A function can be called many times. All C programs are written using functions to improve re-usability, understandability and to keep track on them. C function contains set of instructions enclosed by “{  }” which performs specific operation in a C program. Actually, Collection of these functions creates a C program.

**Advantage of functions**

There are many advantages of functions.

1) Code Reusability

By creating functions in C, you can call it many times. So we don't need to write the same code again and again.

2) Code optimization

It makes the code optimized, we don't need to write much code. Suppose, you have to check 3 numbers (781, 883 and 531) whether it is prime number or not. Without using function, you need to write the prime number logic 3 times. So, there is repetition of code. But if you use functions, you need to write the logic only once and you can reuse it several times.

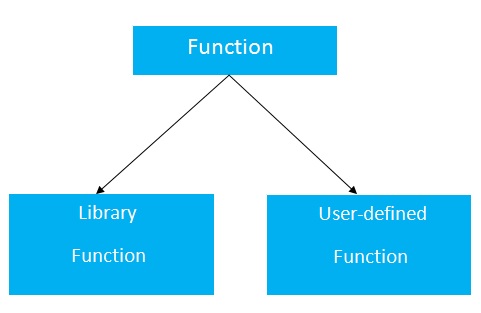
* C functions are used to avoid rewriting same logic/code again and again in a program.
* There is no limit in calling C functions to make use of same functionality wherever required.
* We can call functions any number of times in a program and from any place in a program.
* A large C program can easily be tracked when it is divided into functions.
* The core concept of C functions are re-usability, dividing a big task into small pieces to achieve the functionality and to improve understandability of very large C programs.

**Types of Functions**

There are two types of functions in C programming:

**Library Functions:** are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.

**User-defined functions:** are the functions which are created by the C programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code.



*Declaration of a function*

return\_type function\_name(data\_type parameter...){

//code to be executed

}

*Return Value*

A C function may or may not return a value from the function. If you don't have to return any value from the function, use void for the return type.

*Example without return value:*

void hello(){

printf("hello c");

}

If you want to return any value from the function, you need to use any data type such as int, long, char etc. The return type depends on the value to be returned from the function.

*Example with return value:*

int get(){

return 10;

}

In the above example, we have to return 10 as a value, so the return type is int. If you want to return floating-point value (e.g. 10.2, 3.1, 54.5 etc), you need to use float as the return type of the method.

float get(){

return 10.2;

}

Now, you need to call the function, to get the value of the function.

*Parameters*

A c function may have 0 or more parameters. You can have any type of parameter in C program such as int, float, char etc. The parameters are also known as formal arguments.

*Example of a function that has 0 parameter:*

void hello(){

printf("hello c");

}

*Example of a function that has 1 parameter:*

int cube(int n){

return n\*n\*n;

}

*Example of a function that has 2 parameters:*

int add(int a, int b){

return a+b;

}

**Argument, return value**

All C functions can be called either with arguments or without arguments in a C program. These functions may or may not return values to the calling function.

* C function with arguments (parameters) and with return value.
* C function with arguments (parameters) and without return value.
* C function without arguments (parameters) and with return value.
* C function without arguments (parameters) and without return value.

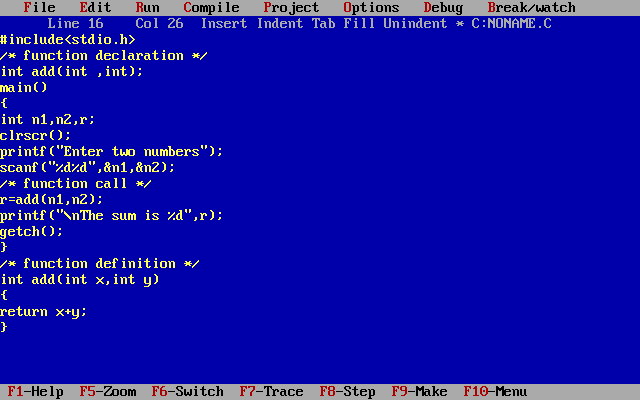
|  |  |
| --- | --- |
| C functions aspects | syntax |
| 1. With arguments and with return values | function declaration: int function ( int );  function call: function ( a );  function definition:        int function( int a ) { statements; return a; } |
| 2. With arguments and without return values | function declaration: void function ( int );  function call: function( a );  function definition: void function( int a ) { statements; } |
| 3. Without arguments and without return values | function declaration: void function();  function call: function();  function definition: void function() { statements; } |
| 4. Without arguments and with return values | function declaration: int function ( );  function call: function ( );  function definition: int function( ) { statements; return a; } |

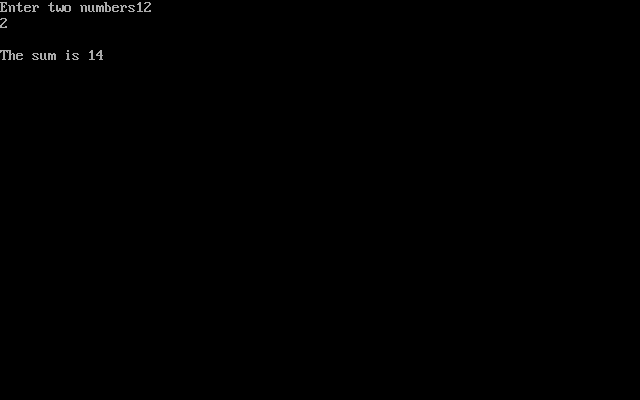
NOTE:

If the return data type of a function is “void”, then, it can’t return any values to the calling function. If the return data type of the function is other than void such as “int, float, double etc”, then, it can return values to the calling function.

*1. EXAMPLE PROGRAM FOR WITH ARGUMENTS & WITH RETURN VALUE:*

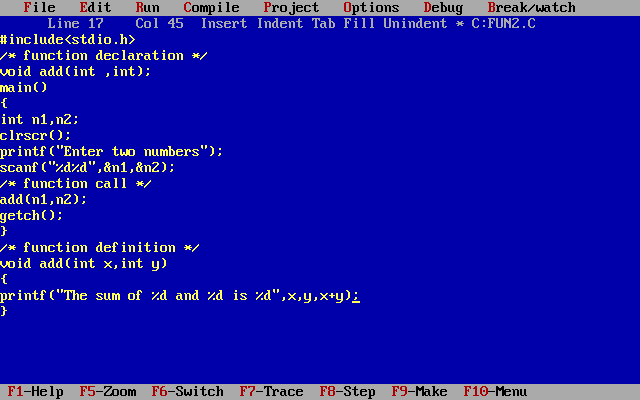
In this program, integer, array and string are passed as arguments to the function. The return type of this function is “int” and value of the variable “a” is returned from the function. The values for array and string are modified inside the function itself.

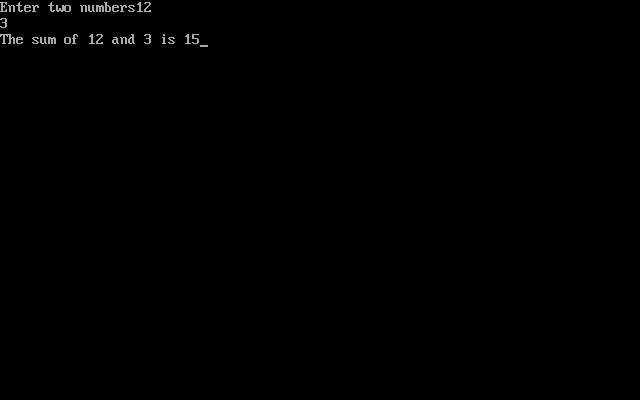




2. EXAMPLE PROGRAM FOR WITH ARGUMENTS & WITHOUT RETURN VALUE:

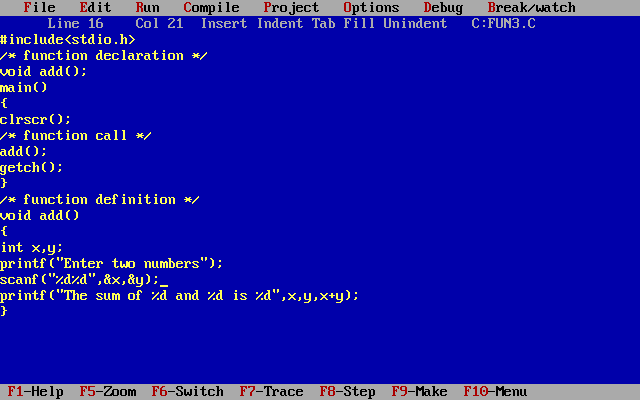
In this program, integer, array and string are passed as arguments to the function. The return type of this function is “void” and no values can be returned from the function. All the values of integer, array and string are manipulated and displayed inside the function itself.

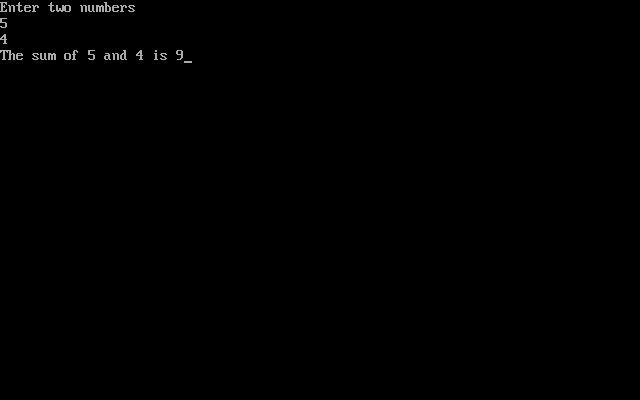




3. EXAMPLE PROGRAM FOR WITHOUT ARGUMENTS & WITHOUT RETURN VALUE:

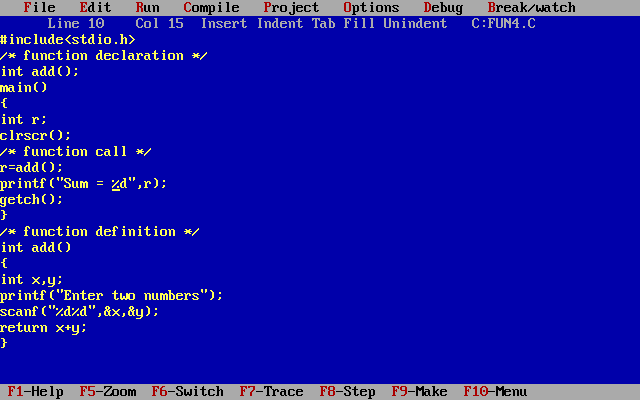
In this program, no values are passed to the function “test” and no values are returned from this function to main function.

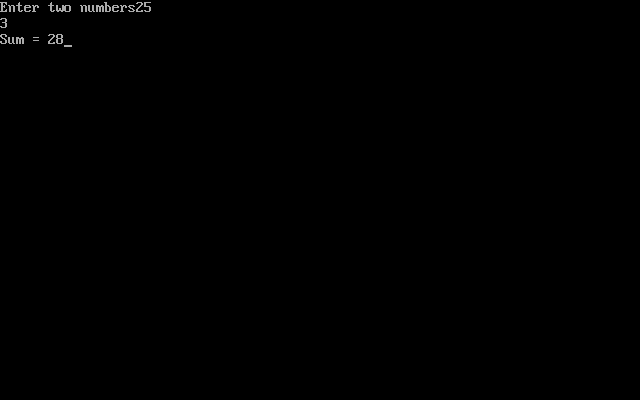


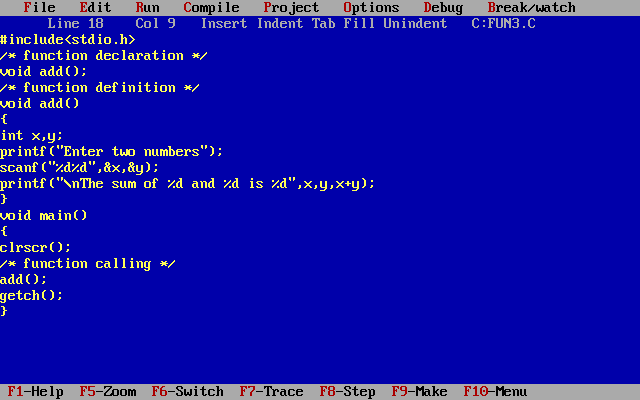


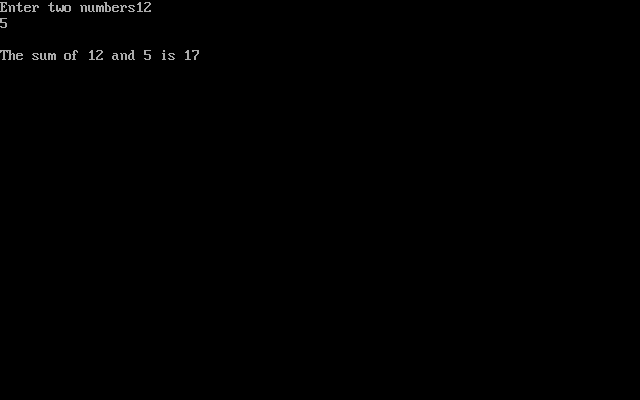
4. EXAMPLE PROGRAM FOR WITHOUT ARGUMENTS & WITH RETURN VALUE:

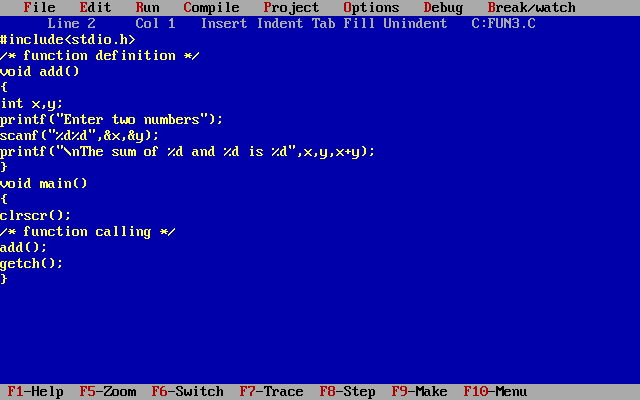
In this program, no arguments are passed to the function “sum”. But, values are returned from this function to main function. Values of the variable a and b are summed up in the function “sum” and the sum of these value is returned to the main function.



Implementation function we can write before the main also.







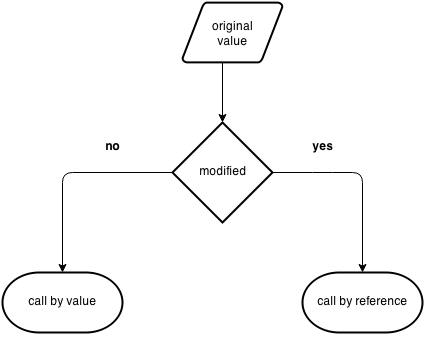


**Actual parameters**: The parameters that appear in function calls.  
**Formal parameters**: The parameters that appear in function declarations.

**Call by value and call by reference**

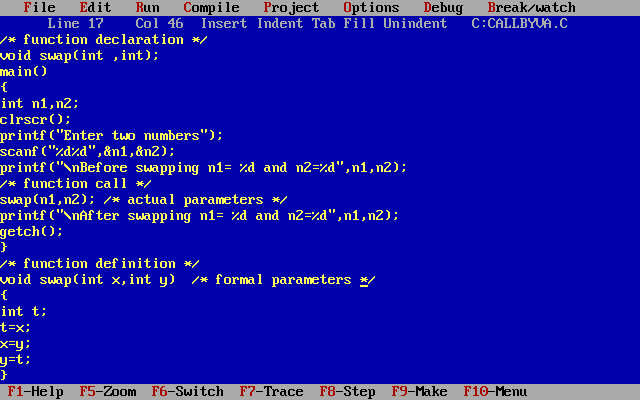
There are two ways to pass value or data to function in C language:

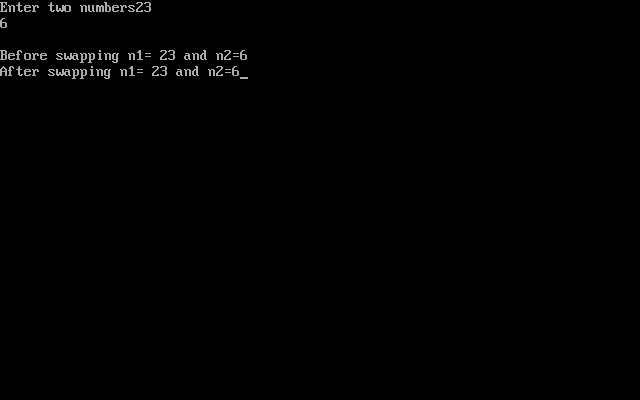
call by value and call by reference. Original value is not modified in call by value but it is modified in call by reference.



**Call by value**

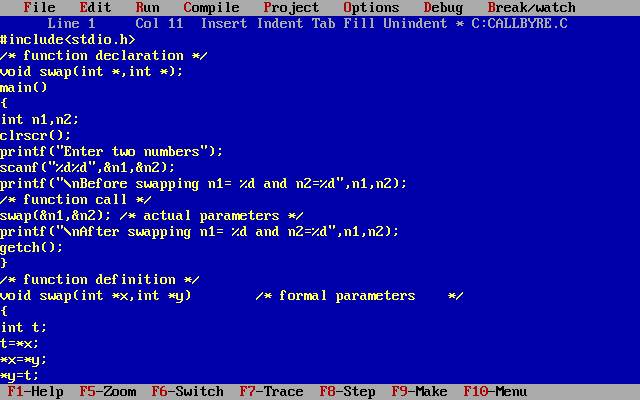
In call by value, original value is not modified. In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

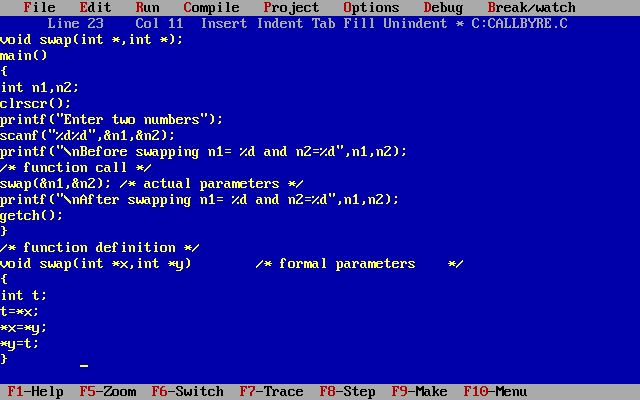


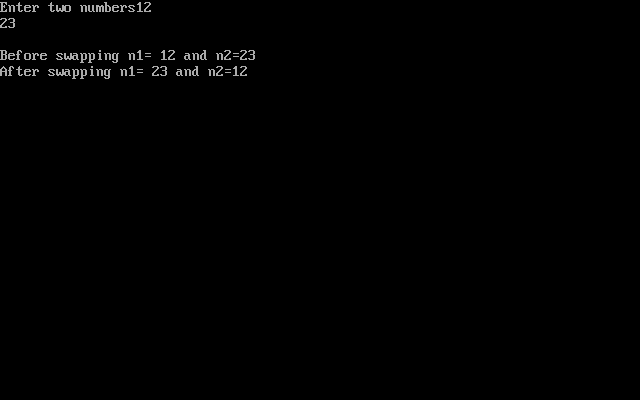


**Call by reference**

In call by reference, original value is modified because we pass reference (address). Here, address of the value is passed in the function, so actual and formal arguments shares the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.







**Difference between call by value and call by reference**

|  |  |  |
| --- | --- | --- |
| No. | Call by value | Call by reference |
| 1 | A copy of value is passed to the function | An address of value is passed to the function |
| 2 | Changes made inside the function is not reflected on other functions | Changes made inside the function is reflected outside the function also |
| 3 | Actual and formal arguments will be created in different memory location | Actual and formal arguments will be created in same memory location |

**DO YOU KNOW HOW MANY VALUES CAN BE RETURN FROM C FUNCTIONS?**

Always, Only one value can be returned from a function.

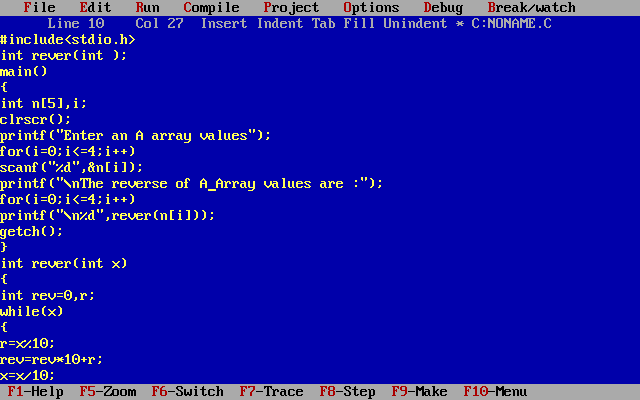
If you try to return more than one values from a function, only one value will be returned that appears at the right most place of the return statement.

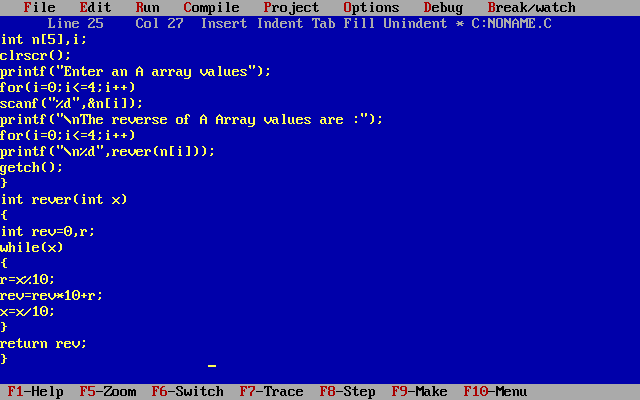
For example, if you use “return a,b,c” in your function, value for c only will be returned and values a, b won’t be returned to the program.

In case, if you want to return more than one values, pointers can be used to directly change the values in address instead of returning those values to the function.

**Passing One-dimensional Array In Function**

Single element of an array can be passed in similar manner as passing variable to a function. C program to pass a single element of an array to function.

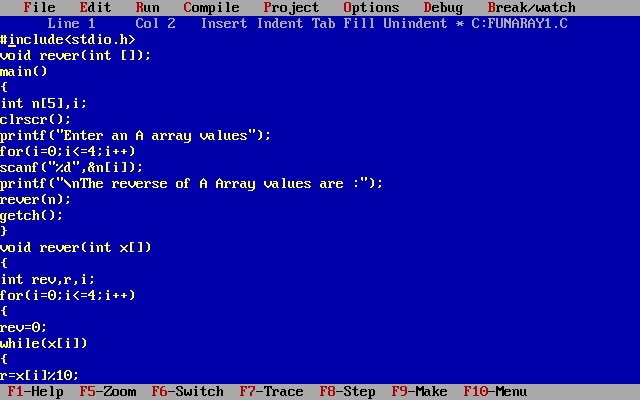


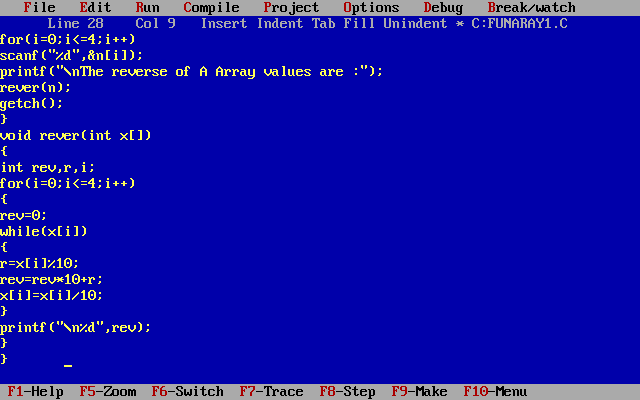




**Passing an entire one-dimensional array to a function**

While passing arrays as arguments to the function, only the name of the array is passed (,i.e, starting address of memory area is passed as argument). C program to pass an array containing age of person to a function. This function should find average age and display the average age in main function.

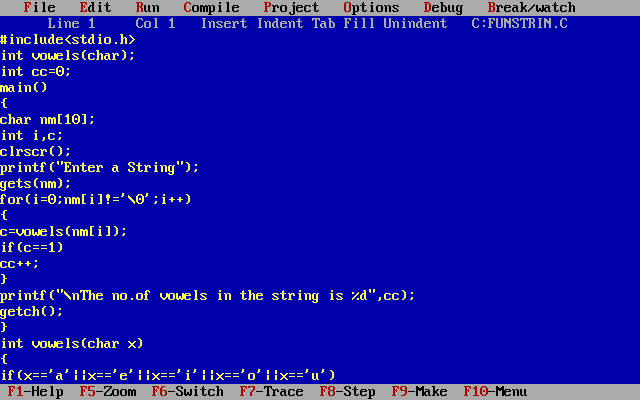


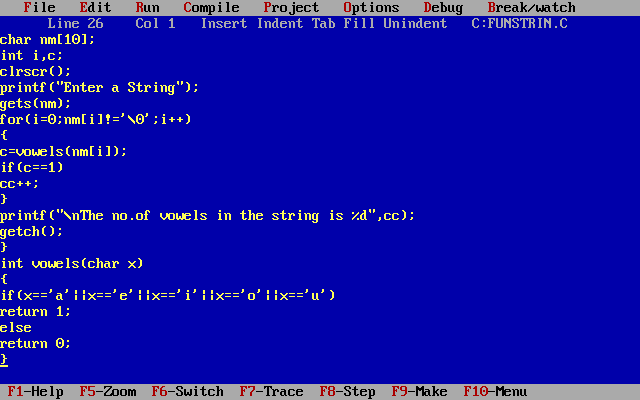


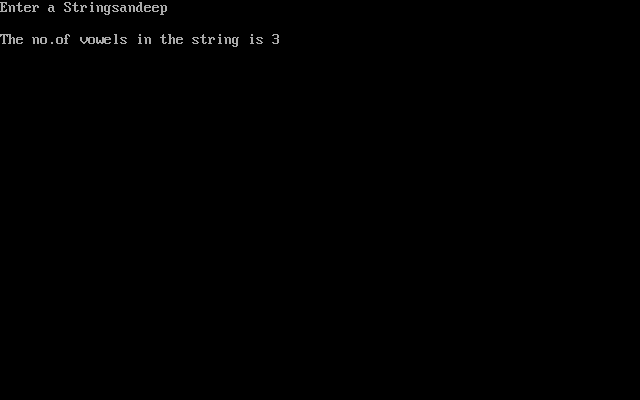


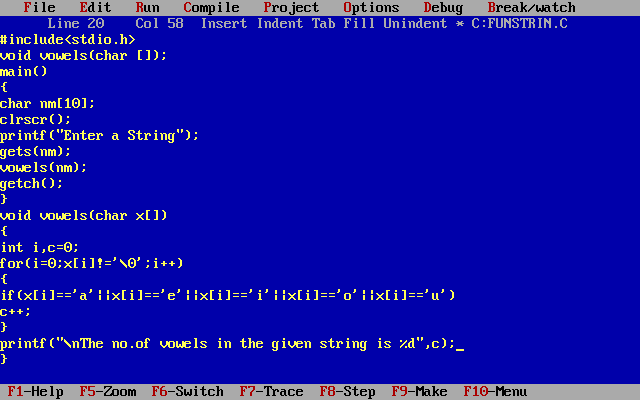
**Note**:-  a single array element or an entire [array](https://www.programiz.com/c-programming/c-arrays) can be passed to a [function](https://www.programiz.com/c-programming/c-functions). This can be done for both one-dimensional array or a multi-dimensional array.

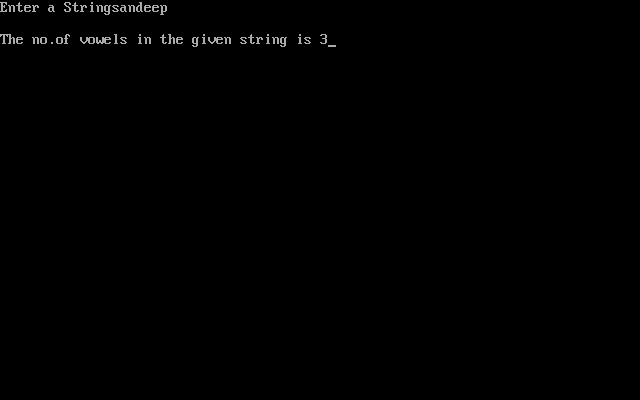
**Function with strings**



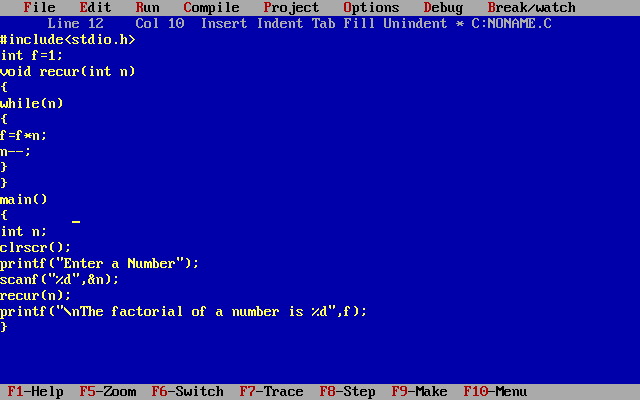


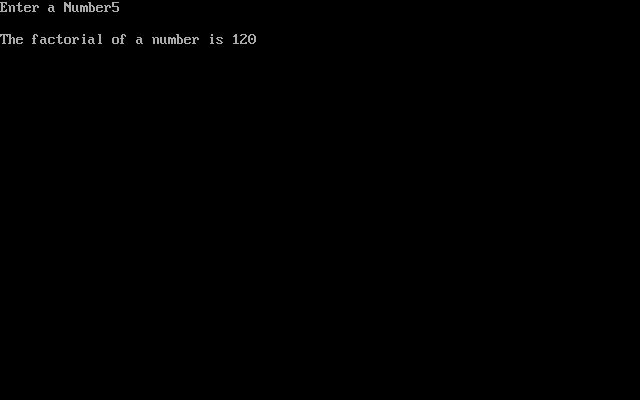






**Without Recursion**





**Recursion**

When function is called within the same function, it is known as recursion in C. The function which calls the same function, is known as recursive function. A function that calls itself, and doesn't perform any task after function call, is know as tail recursion. In tail recursion, we generally call the same function with return statement. An example of tail recursion is given below.

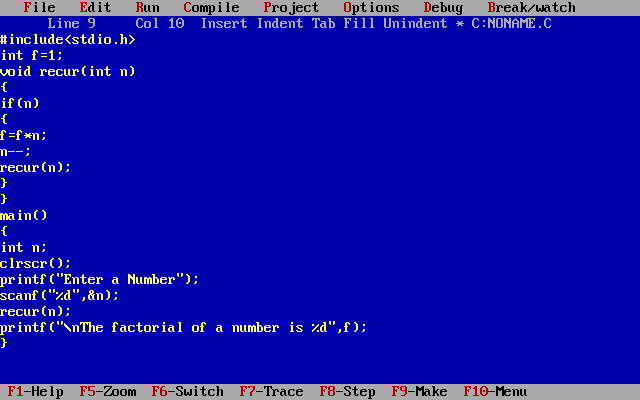
recursionfunction(){

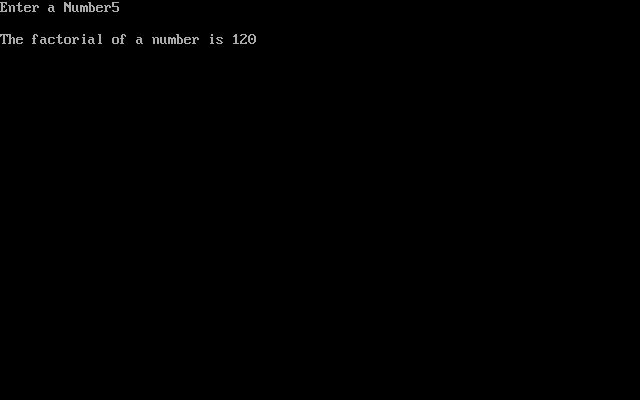
recursionfunction();//calling self function

}

*Example of tail recursion in C*

Let's see an example to print factorial number using tail recursion in C language.



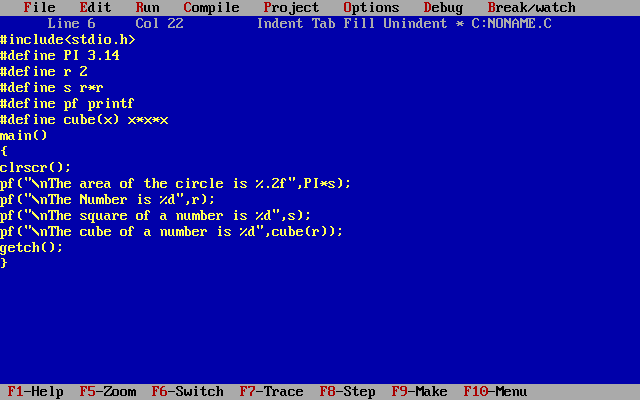


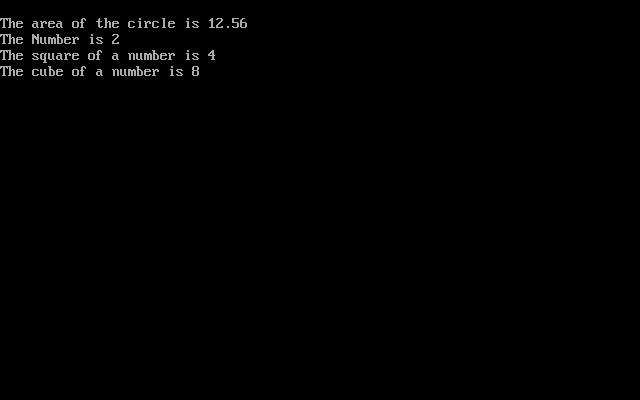
**C PREPROCESSOR DIRECTIVES**

Before a C program is compiled in a compiler, source code is processed by a program called preprocessor. This process is called preprocessing. Commands used in preprocessor are called preprocessor directives and they begin with “#” symbol. Below is the list of preprocessor directives that C programming language offers.

|  |  |
| --- | --- |
| Preprocessor | Syntax/Description |
| Macro | Syntax: #define This macro defines constant value and can be any of the basic data types. |
| Header file inclusion | Syntax: #include <file\_name> The source code of the file “file\_name” is included in the main program at the specified place. |

**Macro**





**Header File Inclusion**

**Library functions**

Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library. Each library function in C performs specific operation. We can make use of these library functions to get the pre-defined output instead of writing our own code to get those outputs. These library functions are created by the persons who designed and created C compilers. All C standard library functions are declared in many header files which are saved as file\_name.h. Actually, function declaration, definition for macros are given in all header files.

We are including these header files in our C program using “#include<file\_name.h>” command to make use of the functions those are declared in the header files.

When we include header files in our C program using “#include<filename.h>” command, all C code of the header files are included in C program. Then, this C program is compiled by compiler and executed.

If you want to check source code for all header files, you can check inside “include” directory after C compiler is installed in your machine.

*LIST OF MOST USED HEADER FILES:*

Check the below table to know all the C library functions and header files in which they are declared. Click on the each header file name below to know the list of inbuilt functions declared inside them.

|  |  |
| --- | --- |
| Header file | Description |
| [stdio.h](http://fresh2refresh.com/c/c-function/stdio-h-library-functions/) | This is standard input/output header file in which Input/Output functions are declared |
| [conio.h](http://fresh2refresh.com/c/c-function/conio-h-library-functions/) | This is console input/output header file |
| [string.h](http://fresh2refresh.com/c/c-function/string-h-library-functions/) | All string related functions are defined in this header file |
| [stdlib.h](http://fresh2refresh.com/c/c-function/c-stdlib-h-library-functions/) | This header file contains general functions used in C programs |
| [math.h](http://fresh2refresh.com/c/c-function/c-math-h-library-functions/) | All maths related functions are defined in this header file |
| [time.h](http://fresh2refresh.com/c/c-function/c-time-h-library-functions/) | This header file contains time and clock related functions |
| [ctype.h](http://fresh2refresh.com/c/c-function/c-ctype-h-library-functions/) | All character handling functions are defined in this header file |
| [stdarg.h](http://fresh2refresh.com/c/c-function/c-all-other-library-functions/) | Variable argument functions are declared in this header file |
| [signal.h](http://fresh2refresh.com/c/c-function/c-all-other-library-functions/) | Signal handling functions are declared in this file |
| [setjmp.h](http://fresh2refresh.com/c/c-function/c-all-other-library-functions/) | This file contains all jump functions |
| [locale.h](http://fresh2refresh.com/c/c-function/c-all-other-library-functions/) | This file contains locale functions |
| [errno.h](http://fresh2refresh.com/c/c-function/c-all-other-library-functions/) | Error handling functions are given in this file |
| [assert.h](http://fresh2refresh.com/c/c-function/c-all-other-library-functions/) | This contains diagnostics functions |

**Creating library functions**

*USER DEFINED FUNCTIONS*:

As you know, there are 2 types of functions in C. They are, library functions and user defined functions. Library functions are inbuilt functions which are available in common place called C library. Where as, User defined functions are the functions which are written by us for our own requirement.

*ADDING USER DEFINED FUNCTIONS:*

Do you know that we can add our own user defined functions in C library?

Yes. It is possible to add, delete, modify and access our own user defined function to or from C library.

The advantage of adding user defined function in C library is, this function will be available for all C programs once added to the C library.

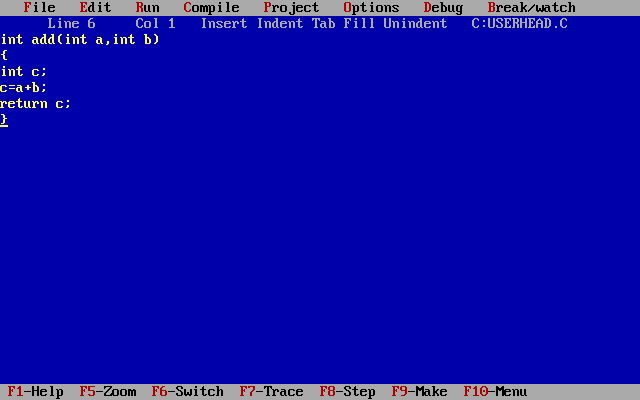
We can use this function in any C program as we use other C library functions.

In latest version of GCC compilers, compilation time can be saved since these functions are available in library in the compiled form. Normal header files are saved as “file\_name.h” in which all library functions are available. These header files contain source code and this source code is added in main C program file where we add this header file using “#include <file\_name.h>” command. Where as, precompiled version of header files are saved as “file\_name.h”.

*STEPS FOR ADDING OUR OWN FUNCTIONS IN C LIBRARY:*

STEP 1:

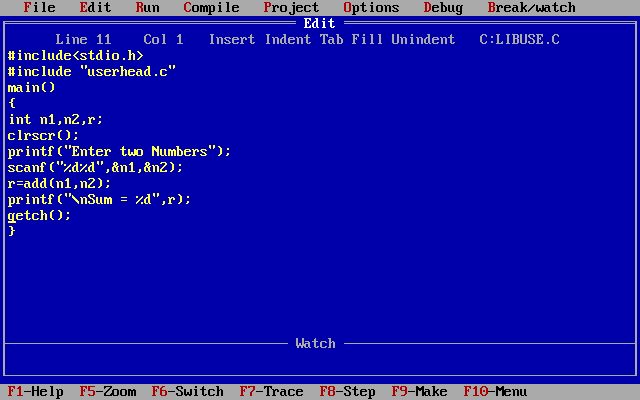
For example, below is a sample function that is going to be added in the C library. Write the below function in a file and save it as “userhead.c”

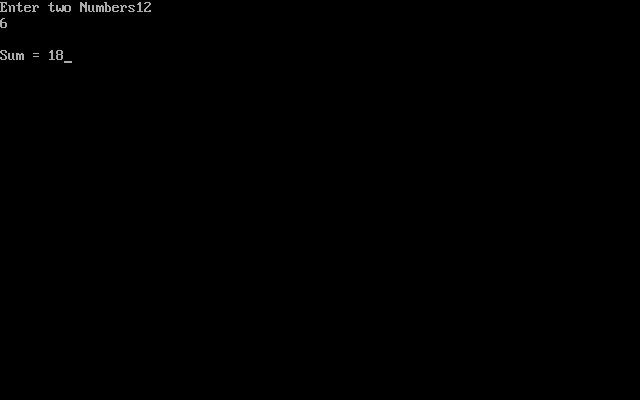
STEP 2:

Compile “userhead.c” file by using Alt + F9 keys (in turbo C).

STEP 3:

Let us see how to use our newly added library function in a C program.





**Command line arguments**

main() function of a C program accepts arguments from command line or from other shell scripts by following commands. They are,

argc

argv[]

where,

argc    – Number of arguments in the command line including program name  
argv[]   – This is carrying all the arguments

In real time application, it will happen to pass arguments to the main program itself.  These arguments are passed to the main () function while executing binary file from command line.

For example, when we compile a program (test.c), we get executable file in the name “test”.

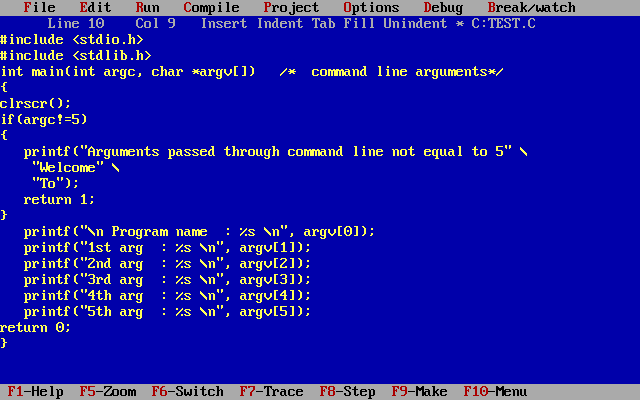
Now, we run the executable “test” along with 4 arguments in command line like below.

./test this is a program

Where,

argc             =       5  
argv[0]         =       “test”  
argv[1]         =       “this”  
argv[2]         =       “is”  
argv[3]         =       “a”  
argv[4]         =       “program”  
argv[5]         =       NULL

EXAMPLE PROGRAM FOR ARGC() AND ARGV() FUNCTIONS: (**Just for Knowledge**)





**THERE ARE TWO TYPES OF VARIABLES IN C PROGRAM THEY ARE,**

Local variable

Global variable

**DIFFERENCE BETWEEN VARIABLE DECLARATION & DEFINITION IN C:**

|  |  |
| --- | --- |
| Variable declaration | Variable definition |
| Declaration tells the compiler about data type and size of the variable. | Definition allocates memory for the variable. |
| Variable can be declared many times in a program. | It can happen only one time for a variable in a program. |
| The assignment of properties and identification to a variable. | Assignments of storage space to a variable. |

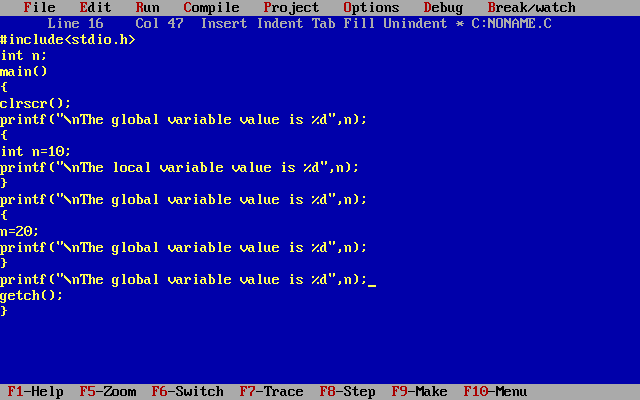
**1. EXAMPLE PROGRAM FOR LOCAL VARIABLE IN C:**

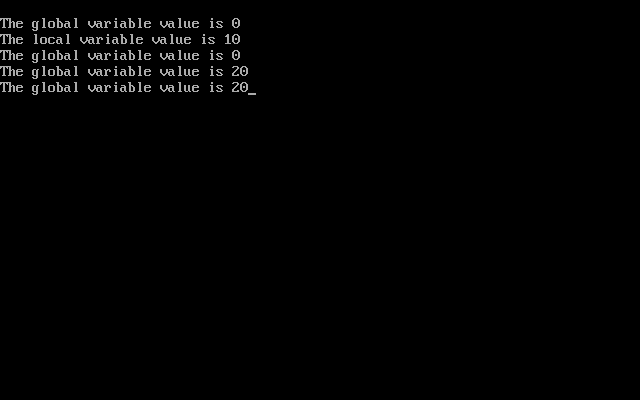
The scope of local variables will be within the function only. These variables are declared within the function and can’t be accessed outside the function. In the below example, m and n variables are having scope within the main function only. These are not visible to test function. Like wise, a and b variables are having scope within the test function only. These are not visible to main function.

**2. EXAMPLE PROGRAM FOR GLOBAL VARIABLE IN C:**

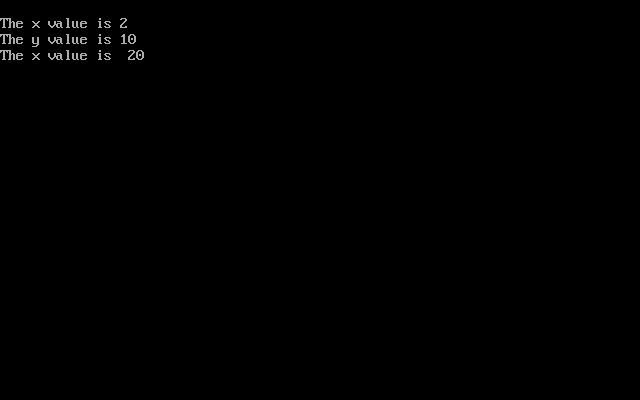
The scope of global variables will be throughout the program. These variables can be accessed from anywhere in the program.

This variable is defined outside the main function. So that, this variable is visible to main function and all other sub functions.









**Pointers**

A pointer is a variable it holds the value and address of  variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before you can use it.

Syntax:-

Datatype \*pointername;

to store any variable value and address.

 int \*a,x;

x=10;

a=&x;

x,\*a values and addresses are same. Because if you remove the \*  pointer holds the value and address of the variable.

&x means it sends the value and address also.

int \*b,y;

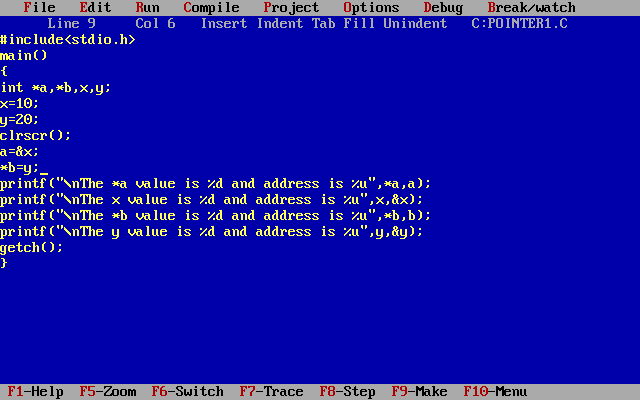
y=20;

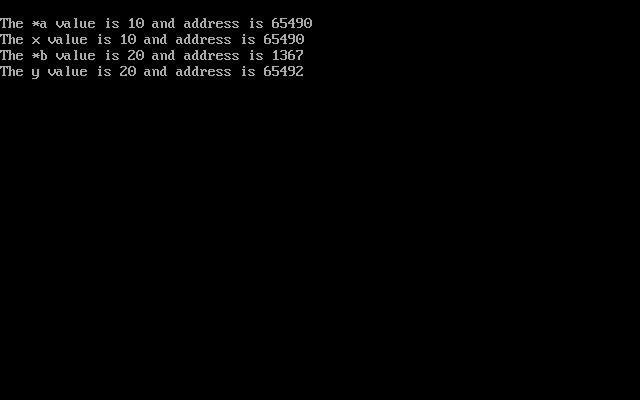
\*b=y;

y,\*b values are same. Because if you use \* for  pointer it holds the value but not the address.

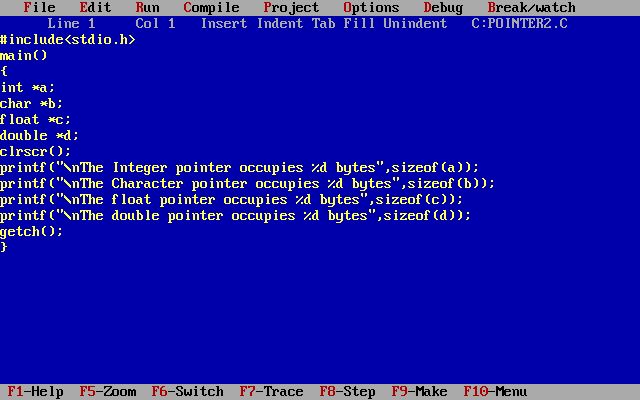
y means it sends the value it will not sends the address.

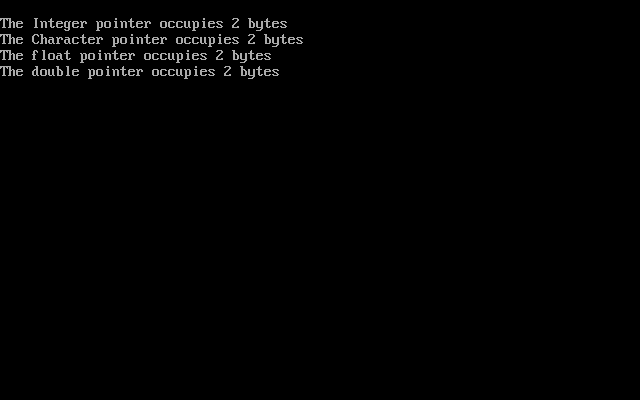
For printing the address we use %u.



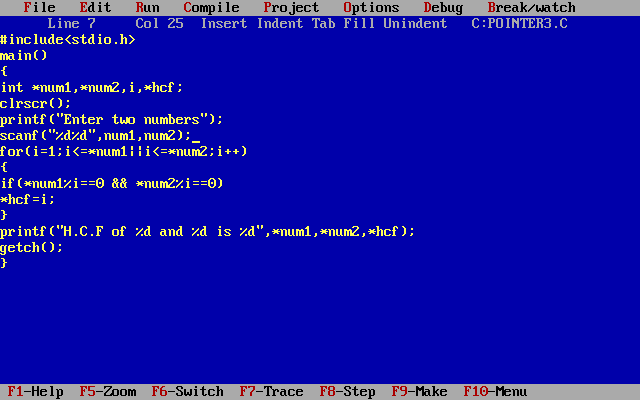


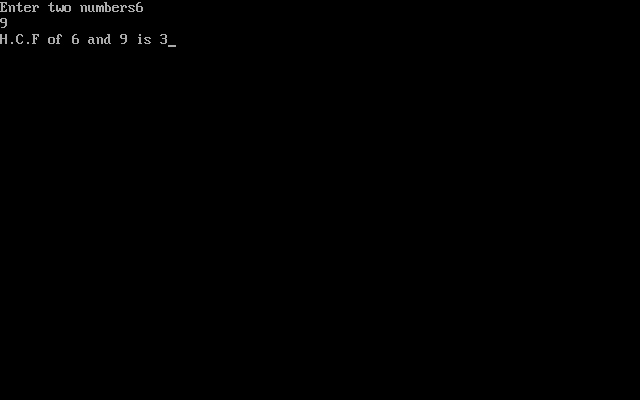
Every pointer occupies 2 bytes.



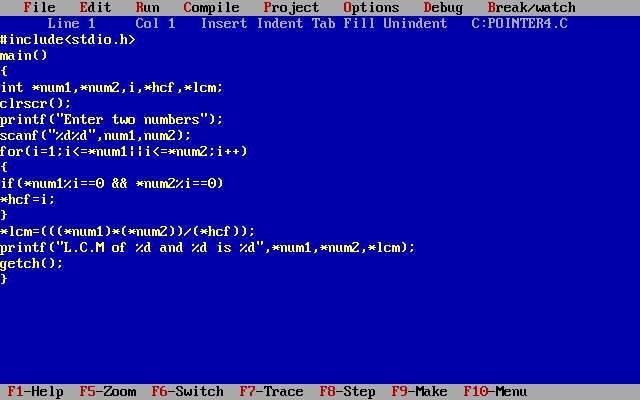


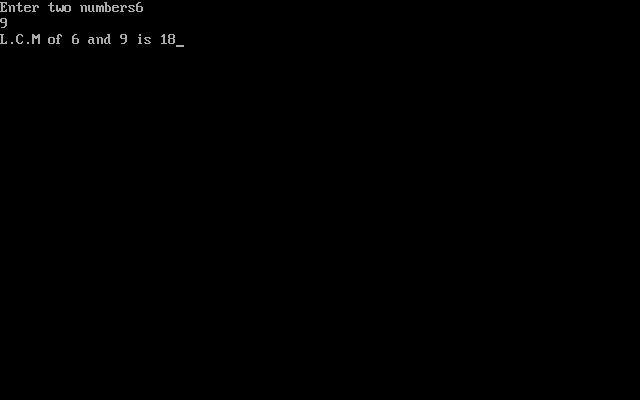
WAP findout hcf of two numbers with using pointers.



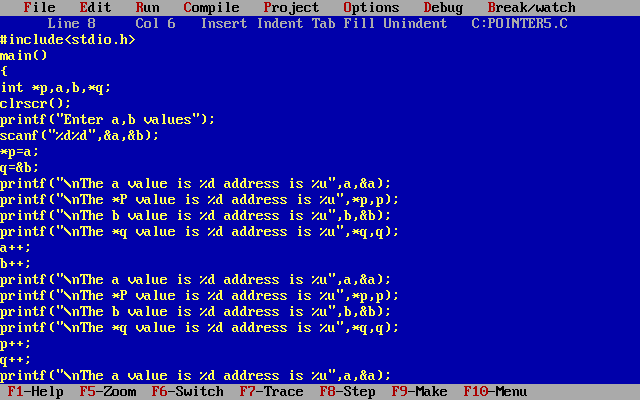


WAP findout lcm of two numbers with using pointers.

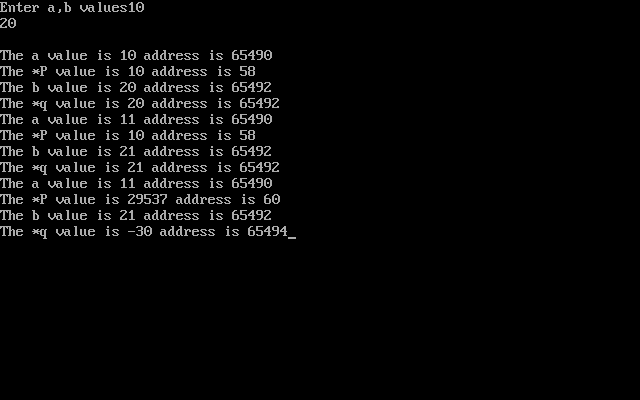


**Pointer Arithmetic**:-

There are four arithmetic operators that can be used on pointers: ++,--, +, -

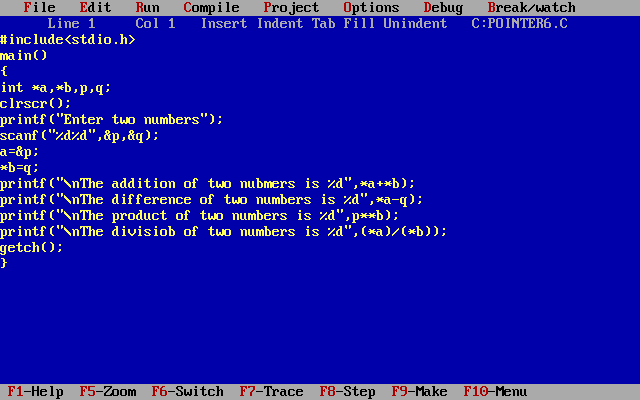


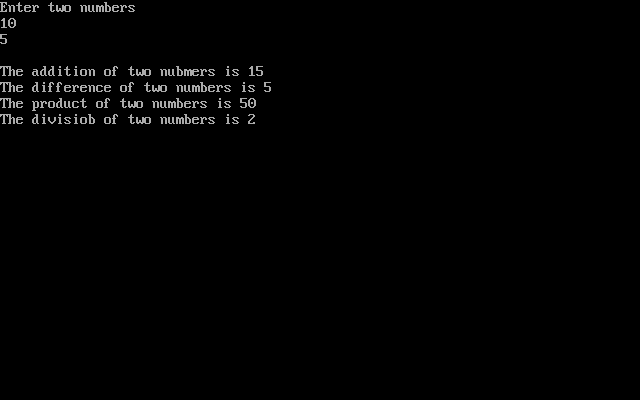




If you increment a variable it increments the value if you increment a pointer it increments the address.

Pointers using arithmetic operators

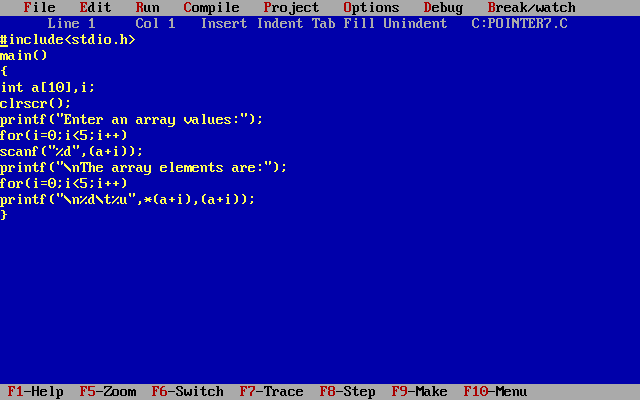




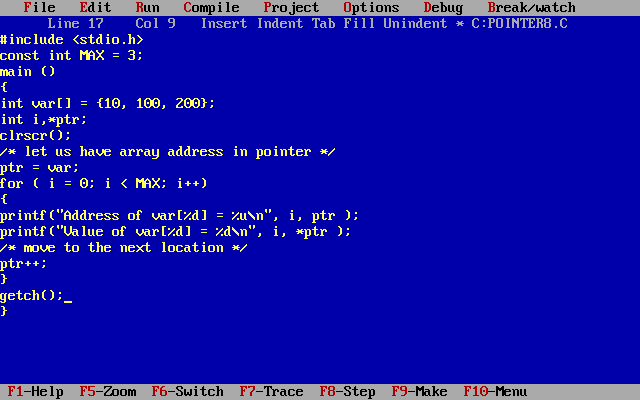
**Pointers with Arrays**:- Array itself is converted to a pointer.

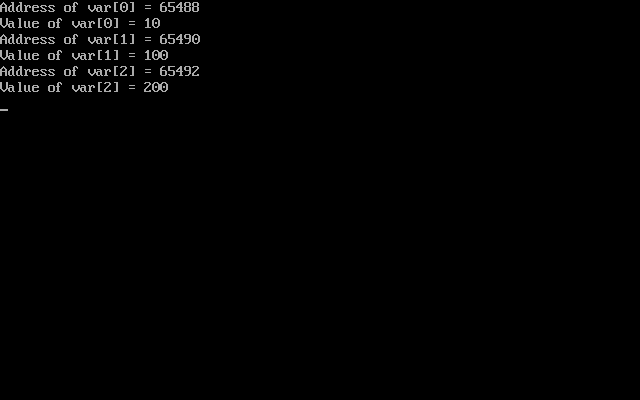
a[i] is represented as \*(a+i)

&a[i] is represented as (a+i)

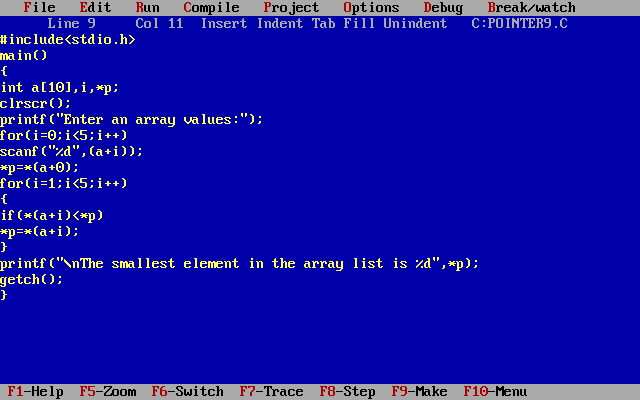








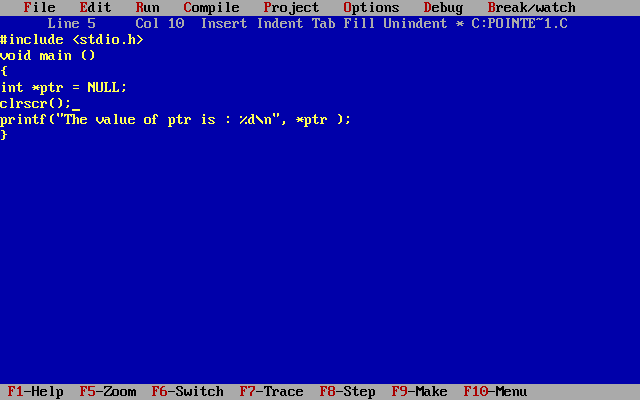
wap to findout the smallest number using pointers.

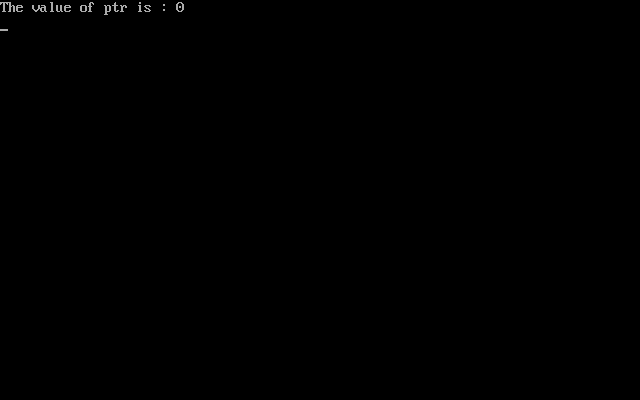




**NULL Pointers in C**:-

This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer.  The NULL pointer is a constant with a value of zero defined in several standard libraries.



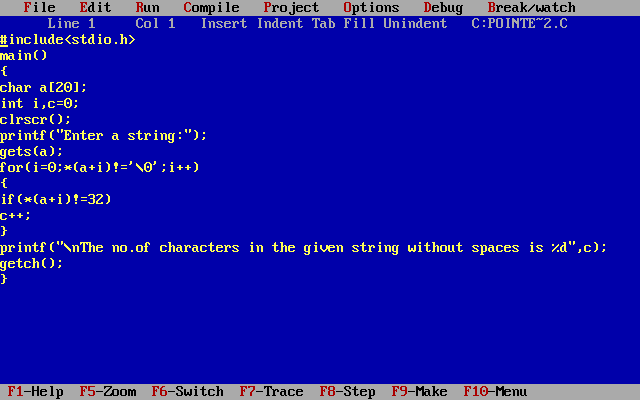
**Pointers with strings**:-

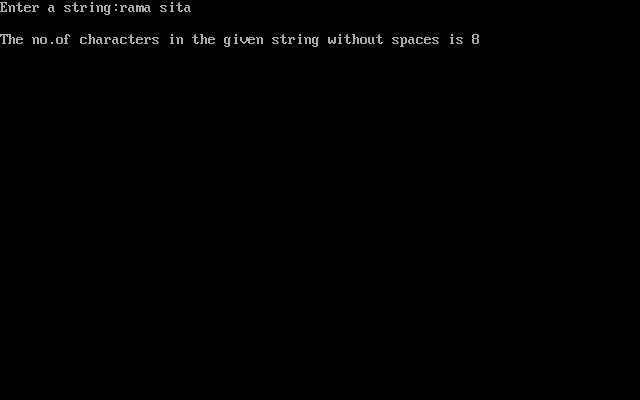
String itself is converted to a pointer.

a[i] is represented as \*(a+i)

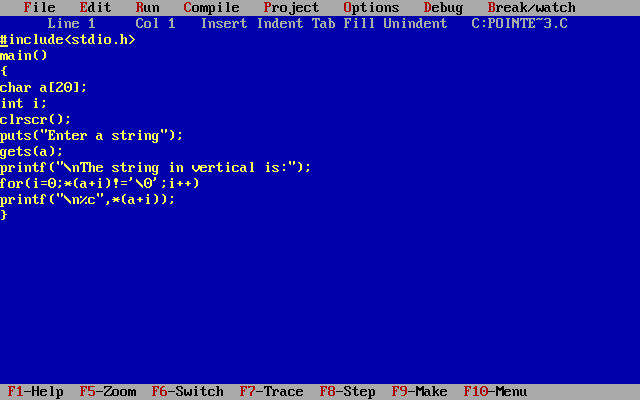
&a[i] is represented as (a+i)

Wap to count the no.of characters in the string without counting spaces.



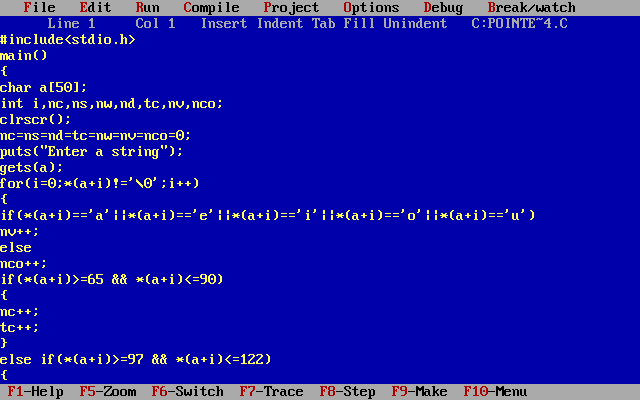


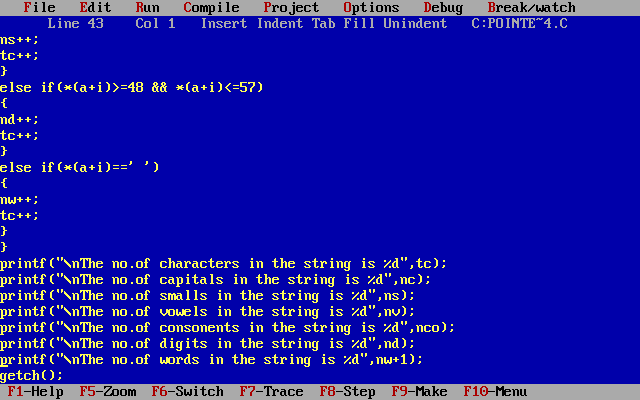
Wap to print the given string in vertical.

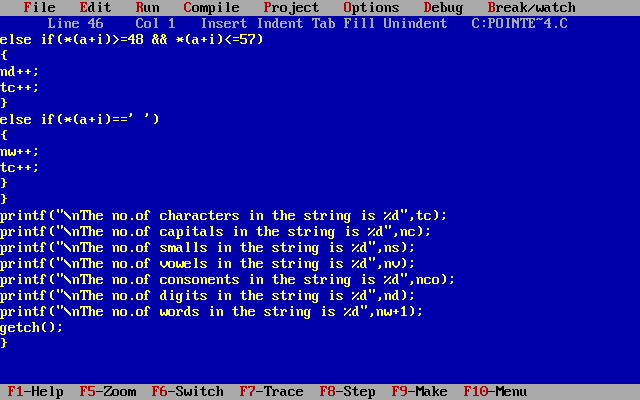


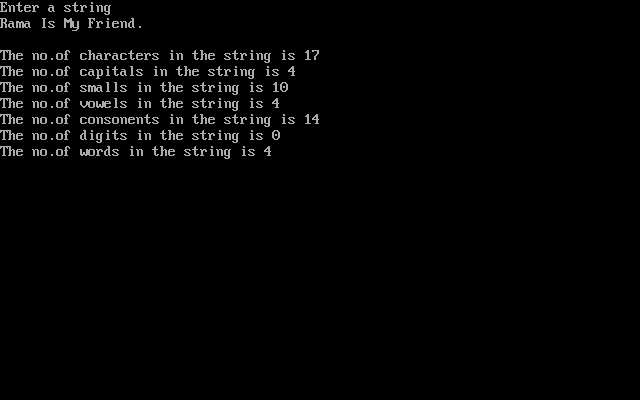


WAP read a string print the no.of characters, no.of capitals, no.of smalls, no.of digits, no.of vowels, no.of consonents, no.of words.









**Call by value**:- When formal parameters are modified actual parameters will not get any effect. That is nothing but call by value. Called function parameters are actual parameters. Implementation function parameters are formal parameters.

Wap to perform addition by adding 10.

#include<stdio.h>

void add(int x)

{

x=x+10;

}

main()

{

int p;

printf("Enter one value");

scanf("%d",&p);

printf(“Before calling p=%d”,p);

add(p);

printf(“After calling p=%d”,p);

getch();

}

**Pointer with Functions**:-

Call by reference:- When formal parameters are modified actual parameters will get any effect. That is nothing but call by Reference. Called function parameters are actual parameters. Implementation function parameters are formal parameters.

Wap to perform addition by adding 10.

#include<stdio.h>

void add(int \*x)

{

\*x=\*x+10;

}

main()

{

int p;

printf("Enter one value");

scanf("%d",&p);

printf(“Before calling p=%d”,p);

add(&p);

printf(“After calling p=%d”,p);

getch();

}

WAP to perform swapping

#include<stdio.h>

void swap(int \*x,int \*y)

{

\*x+=\*y;

\*y=\*x-\*y;

\*x-=\*y;

}

main()

{

int p,q;

printf("Enter two values");

scanf("%d%d",&p,&q);

printf("\nBefore swapping p=%d and q=%d",p,q);

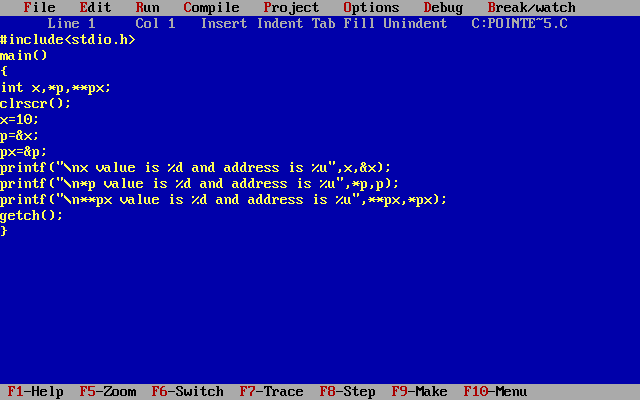
swap(&p,&q);

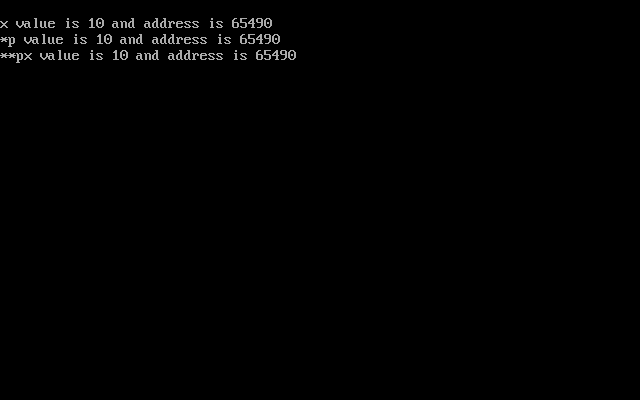
printf("\nAfter swapping p=%d and q=%d",p,q);

getch();

}

**Pointer with pointer**:- variable value and address can hold by pointer. Pointer value and address can hold by pointer with pointer.





**Unions**

Union is a keyword. Union is a user defined data type.  A union is a collection of variables that is referenced under a single name.  A union contains a number of data types grouped together.  These data types may or may not be of the same type. Union occupies max. no.of bytes. To know the size of the union and accessing the union variables we need union object. we use union keyword for declaring a union. Union always ends with semicolon. We can declare a union before a main() or within the main().

Declaring a union:-

Syntax:-

union <union-name>

{

union variable 1;

union variable 2;

 ---

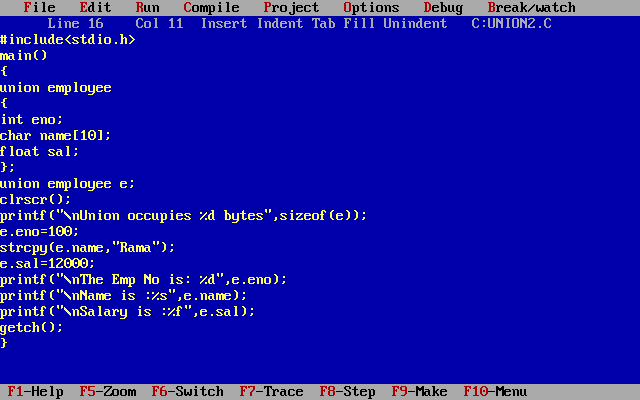
union variable n;

}union Objects;

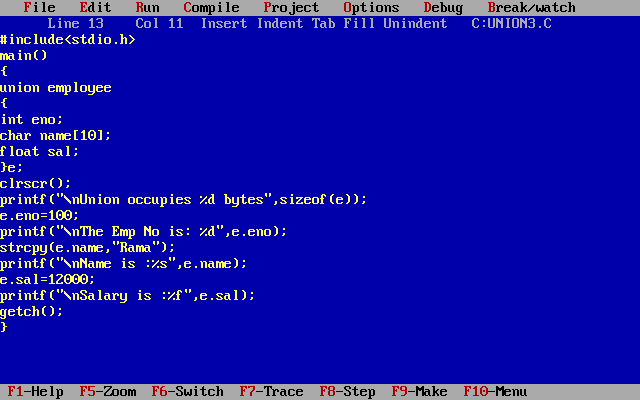
we can create no.of objects for a union. We can create a union object before union ends with semicolon or in main().

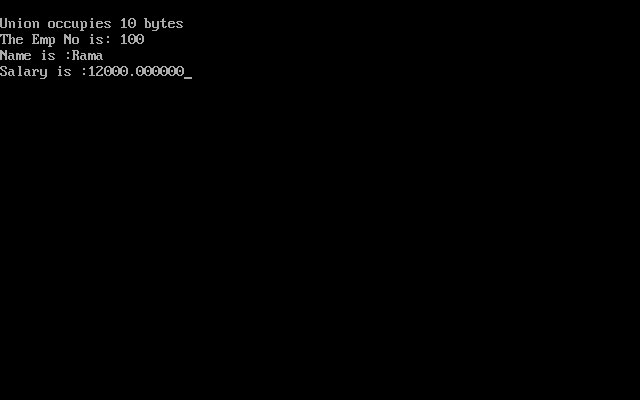
With in the main() if you want to create a union object use the following syntax.

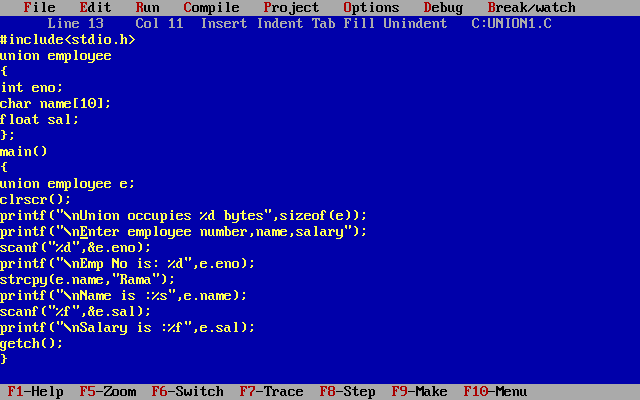
<union> <union-name> <union-object>;













**Structures**

Structure is a user defined data type.  A structure is a collection of variables that is referenced under a single name.  A structure contains a number of data types grouped together.  These data types may or may not be of the same type. Structure occupies total no.of bytes. To know the size of the structure and accessing the structure variables we need structure object. we use struct keyword for declaring a structure. Structure always ends with semicolon. We can declare a structure before a main() or within the main().

**Declaring a structure**:-

Syntax:-

struct <structure name>

{

 structure element 1;

 structure element 2;

 ---

 structure element n;

};

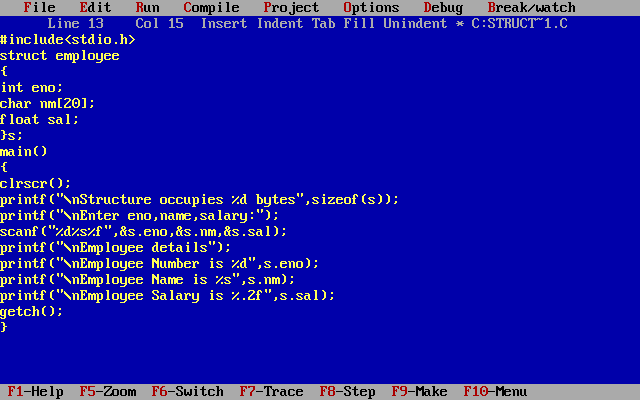
structure Objects;

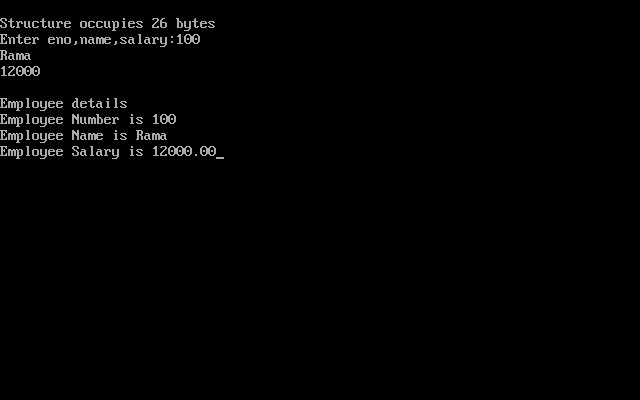
we can create no.of objects for a structure. We can create a structure object before structure ends with semicolon or in main().

With in the main() if you want to create a structure object use the following syntax.

<struct> <structure-name> <structure-object>;

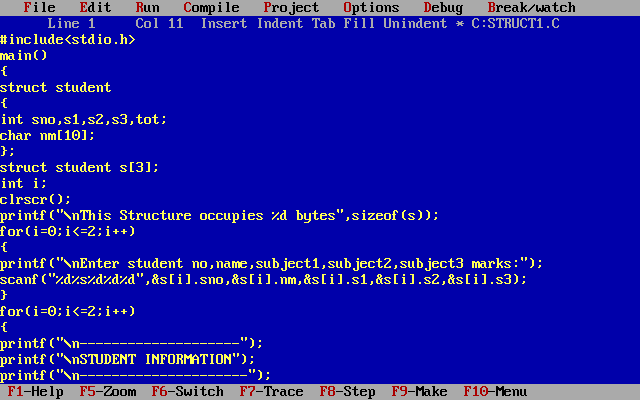
Wap read and print the employee information with using structures.

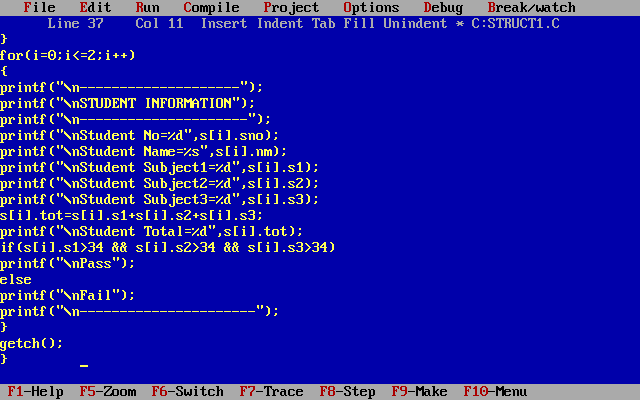


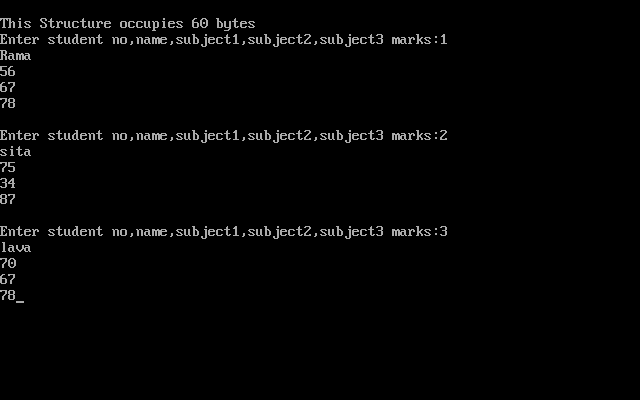


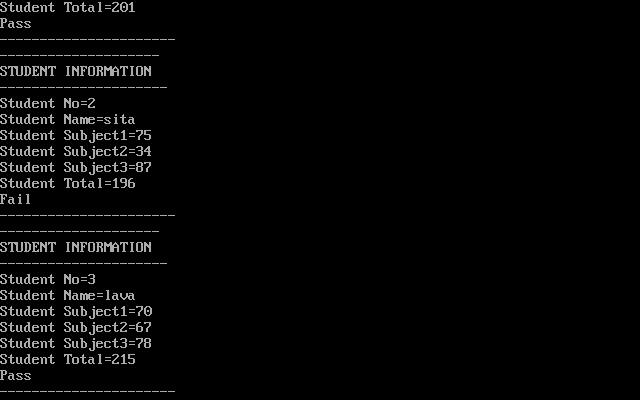
**Structures with Array**:- In the structure object if you specify the size is nothing but structure with array.

Wap read and print the 3 students information and print with using structures using arrays.



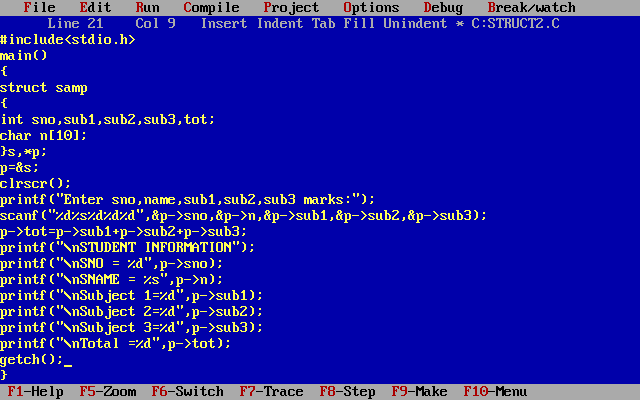






**structure with Pointers**:- In the structure object if you specify the pointer that is nothing but structure with pointer.

WAP to calculate the student total with using structure with pointer.

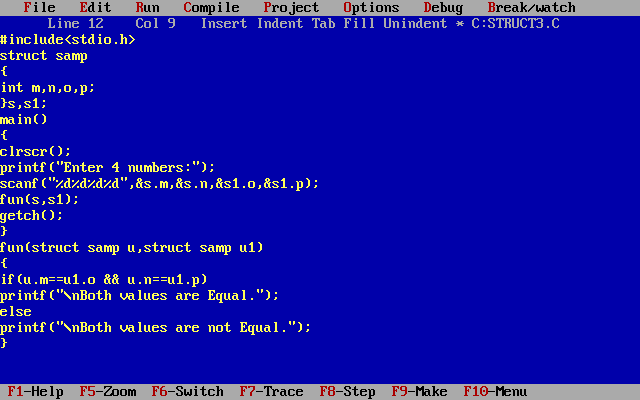


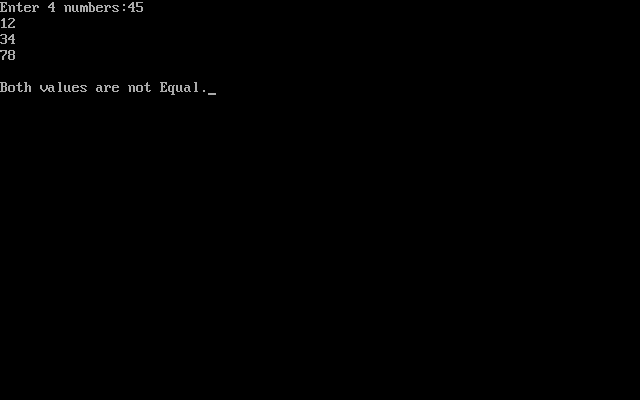


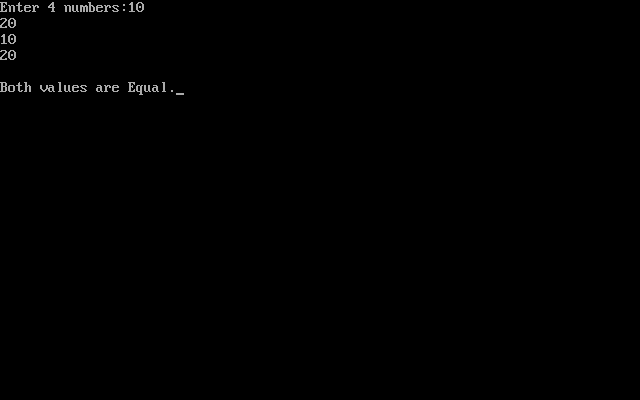
Note:- \* and . is replaced with -> operator.

**Structure with functions**:-In the function if you pass the structure object as the parameter that is nothing but structure with functions. For working with structure with functions structure is before the main() is compulsary. Calling function doesn’t required. Return type doesn’t required.

Wap read 4 numbers and check those are equal or not.



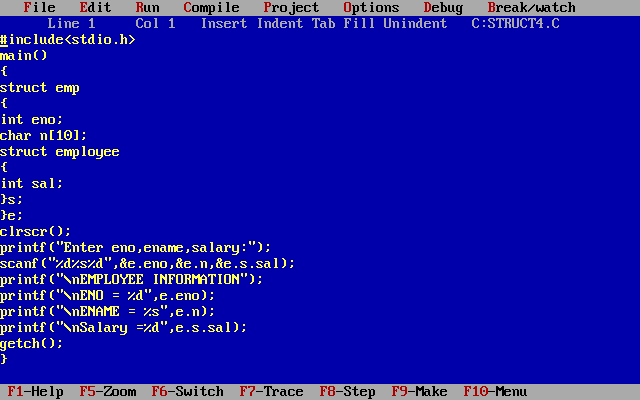


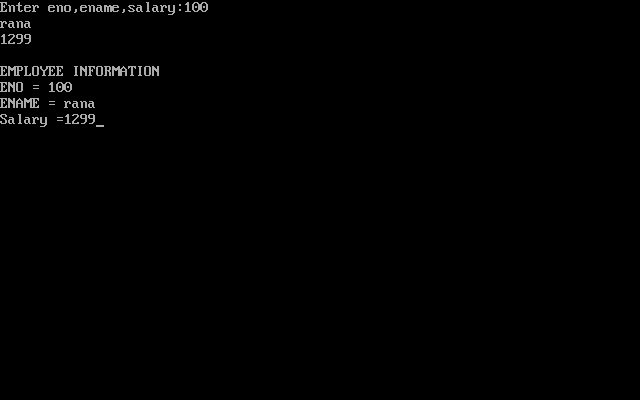


**Structure with structure**:- In the structure if u create another structure is called structure with structure.

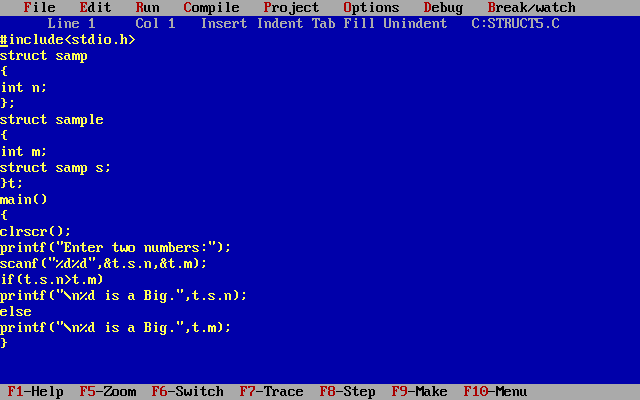
Or

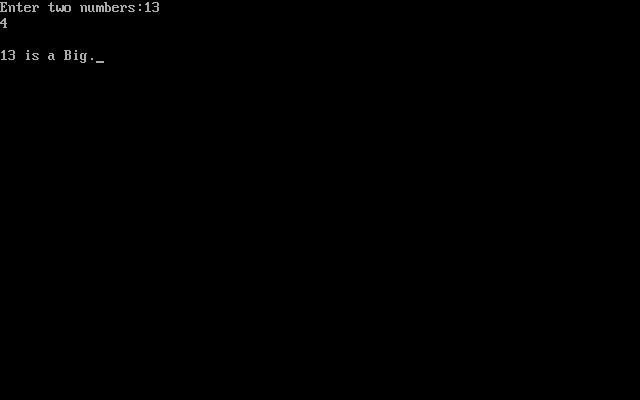
In one structure if you create object for another structure is called structure with structure.





Wap to findout the biggest number among two numbers.





**Files**

A file is a collection of information stored on a storage device with a particular file name.

**Uses of a file:**-

1) Stores the data in the form of a file and we can retrieve it whenever we require.

2) Using the data from the file in different programs.

**Opening a file**:- we can open a file by using

fopen().

**Syntax**:- fopen(file name, opening mode);

Ex: - FILE \*fp;

fp=fopen(“sssit.txt”,”w”);

**File opening modes**:-

1)w :- Writing the data into a file.  If the file already exists its contents will be over written.

2)r:-  Reading data from a file.

3)a:-  Adds data into an existing file.(Appending)

**closing a file**:-      When the file processing is over, the file must be closed. We can close a file by using fclose( ).

syntax:- fclose(file pointer);

ex:- fclose(fp);

**File functions**:-

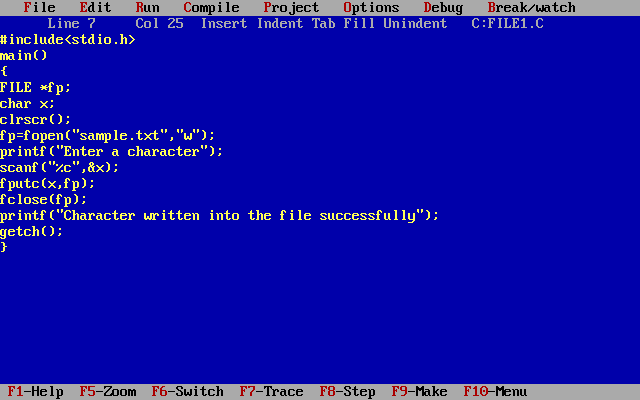
**1)fputc()**

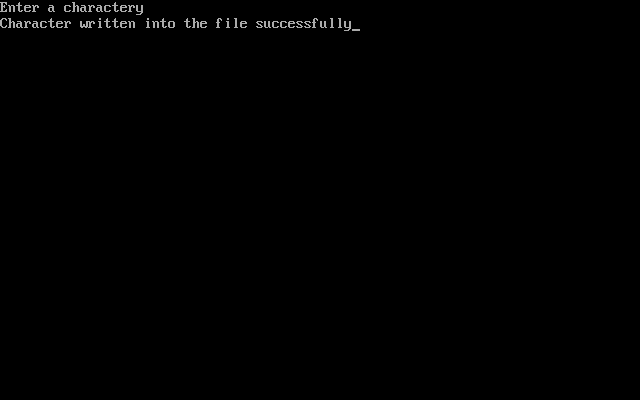
This function is used to write a character into a file.

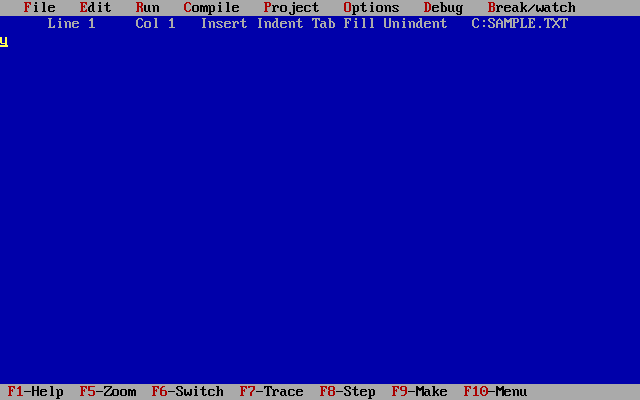
**syntax**:-    fputc(character, file pointer);

ex:- fputc(ch,fp);

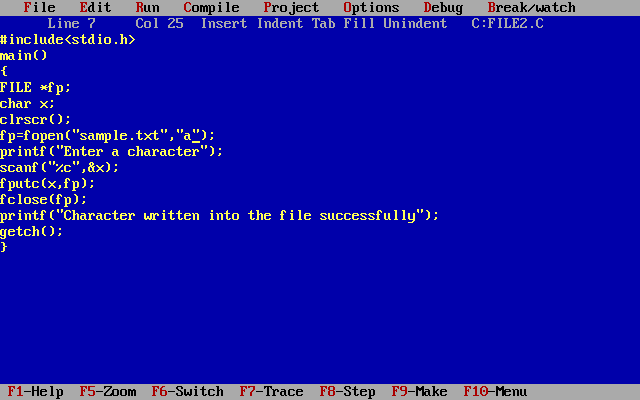
Write a program write a character to the file.



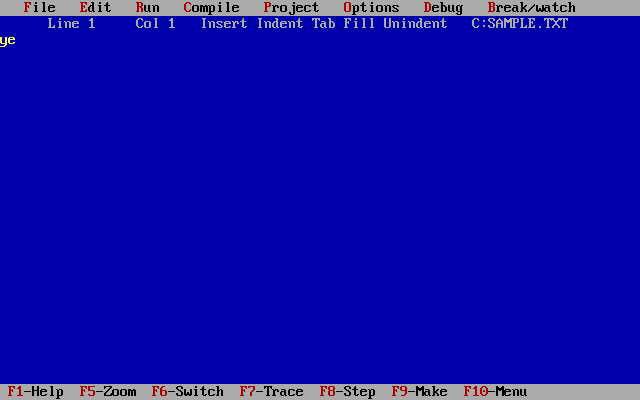




**Write a program add a character to the existing file.**







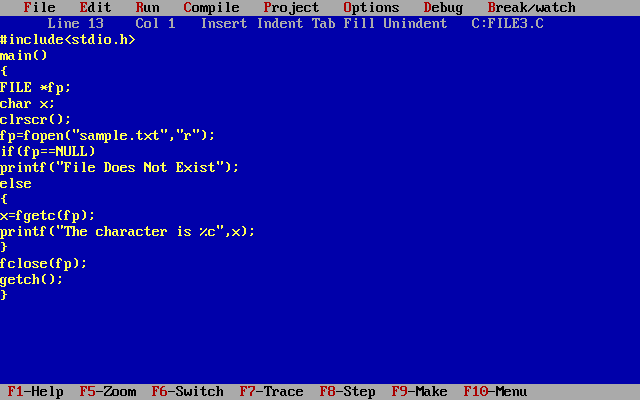
**2)fgetc()**

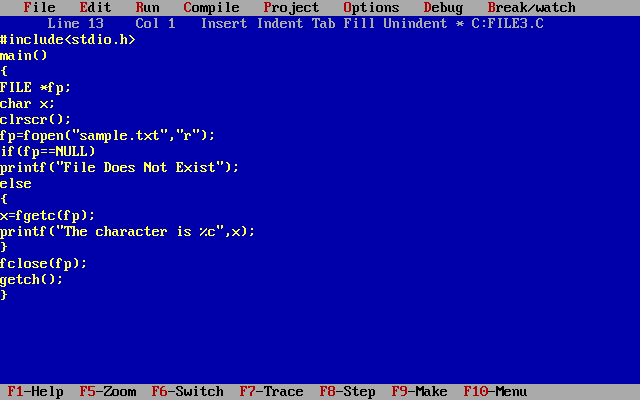
This function is used to read a character from a file.

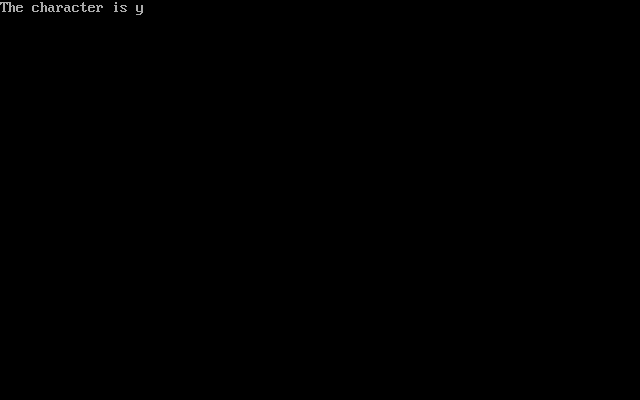
syntax :-  variable=fgetc(file pointer);

ex:- x=fgetc(fp);

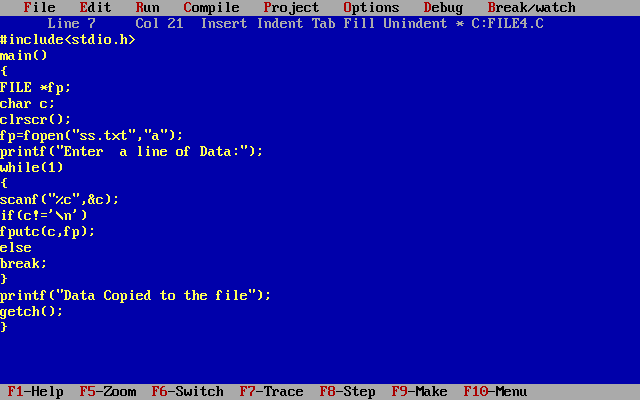
Wap read a character from the file.





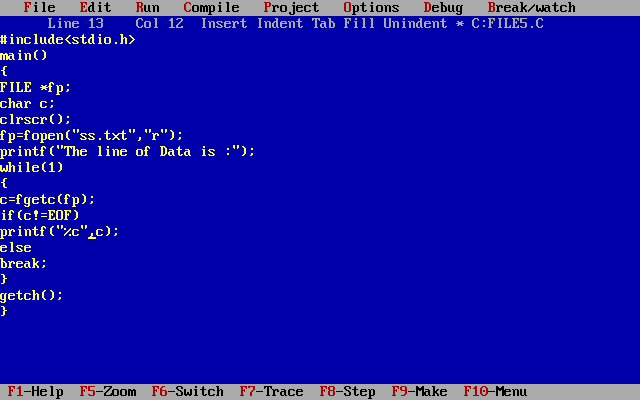


**WAP write a line of data to the file.**





WAP read line of data to the file.





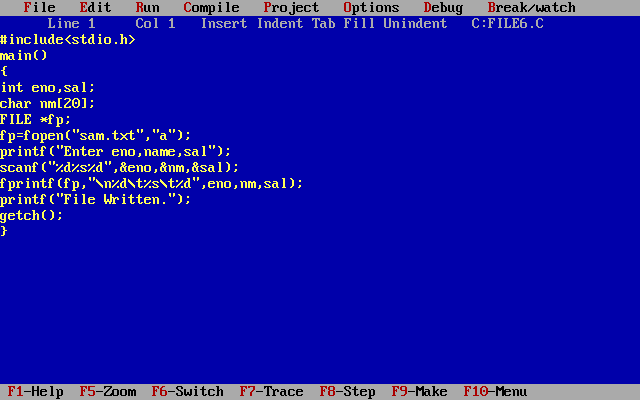
3)fprintf():-  This function writes formatted data into a file.

Syntax:-  fprintf(file pointer, “formatted string”, list of variables);

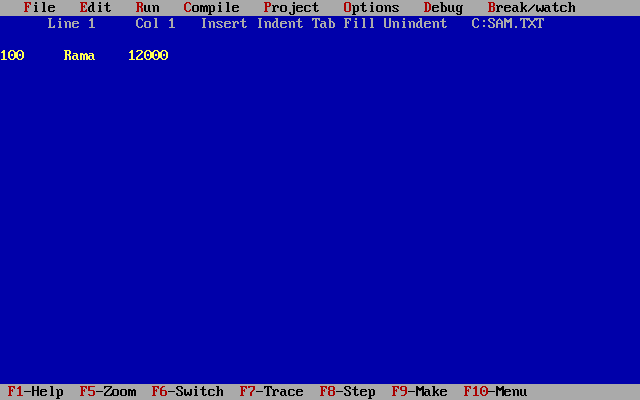
Ex:-      fp=fopen(“student.dat”,”w”);

   fprintf (fp , “%d %s %d”,sno ,name, marks);

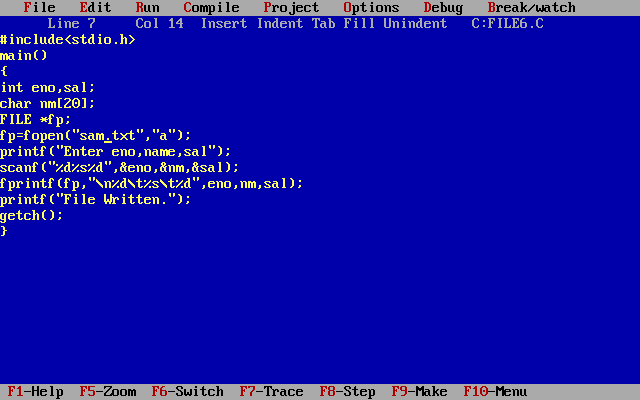
WAP write employee information(empno,name,sal) to the file.







WAP write employee information (empno,name,sal,bonus) to the file.





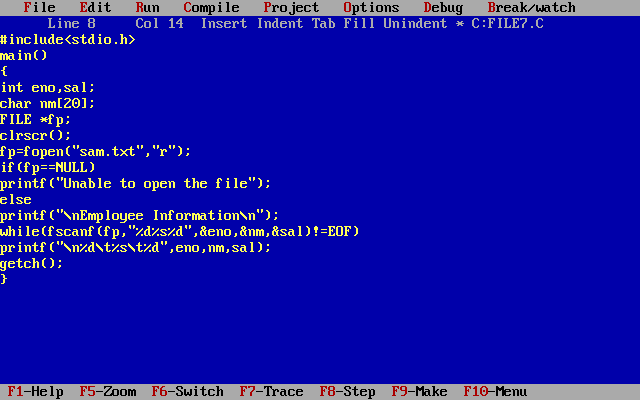
4)**fscanf()**:-  This function reads formatted data from a file.

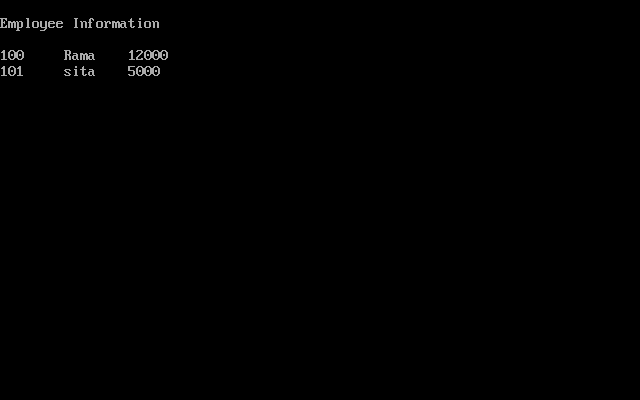
Syntax:-  fscanf(file pointer, “formatted string”, list of variables)

Ex:- fp=fopen(“student.dat”,”r”);

            fscanf(fp , “%d %s %d”,&sno ,name,&marks);

Wap to read the employee information with their net salaries from the file.





**WAP read employee information and printing.**

struct emp

{

            int eno;

            char ename[20];

            float esal;

}erec;

#include<stdio.h>

main()

{

            int temp,found,ch;

            float sal;

            FILE \*fp,\*fp1;

            clrscr();

            do

            {

clrscr();

gotoxy(30,1); printf("1. Add New Record");

gotoxy(30,2); printf("2. Enquiry on a Record");

gotoxy(30,3); printf("3. Modify a Record");

gotoxy(30,4); printf("4. Delete a Record");

gotoxy(30,5); printf("5. List all Records");

gotoxy(30,6); printf("6. Exit");

gotoxy(30,7); printf("Enter Your Choice :");

            scanf("%d", &ch);

                        switch(ch)

                        {

                        case 1:

            fp=fopen("emp.dat", "a+");

            printf("\n Enter Employee Number :");

            scanf("%d", &erec.eno);

            printf("\n Enter Employee Name :");

            fflush(stdin);

            gets(erec.ename);

            printf("\n Enter Employee Salary :");

            scanf("%f", &sal);

            erec.esal=sal;

            fwrite(&erec, sizeof(erec),1,fp);

            printf("\n Record Added");

            fclose(fp);

            getch();

            break;

            case 2:

            printf("\n Enter Employee Number to Display :");

            scanf("%d", &temp);

            fp=fopen("emp.dat", "r+");

if(!fp)

{

printf("\n Unable to Open Source File");

getch();

exit(0);

}

found=0;

fread(&erec, sizeof(erec),1,fp);

while(!feof(fp))

{

if(erec.eno==temp)

{

found=1;

printf("\n Employee Name :%s", erec.ename);

printf("\n Employee Salary :%.2f", erec.esal);

break;

}

else

fread(&erec,sizeof(erec),1,fp);

}

if(found==0)

printf("\n No such Employee");

getch();

break;

case 3:

printf("\n Enter Employee Number to Modify :");

scanf("%d", &temp);

fp=fopen("emp.dat","r+");

fp1=fopen("temp.dat", "a+");

if(!fp)

{

printf("\n Unable to Open Source File");

getch();

exit(0);

}

found=0;

fread(&erec, sizeof(erec),1,fp);

while(!feof(fp))

{

if(temp==erec.eno)

{

found=1;

printf("\n Employee Name     :%s", erec.ename);

printf("\n Employee Salary   :%.2f", erec.esal);

printf("\n Enter Employee Number :");

scanf("%d", &erec.eno);

printf("\n Enter Employee Name :");

fflush(stdin);

gets(erec.ename);

printf("\n Enter Employee Salary :");

scanf("%f", &sal);

erec.esal=sal;

fwrite(&erec,sizeof(erec),1,fp1);

printf("\n Record Modified");

}

else

fwrite(&erec,sizeof(erec),1,fp1);

fread(&erec,sizeof(erec),1,fp);

}

if(found==0)

printf("\n No such employee to modify");

fclose(fp);

fclose(fp1);

remove("emp.dat");

rename("temp.dat", "emp.dat");

break;

getch();

case 4:

printf("\n Enter Employee Number to Delete :");

scanf("%d", &temp);

fp=fopen("emp.dat","r+");

fp1=fopen("temp.dat", "a+");

if(!fp)

{

printf("\n Unable to Open Source File");

getch();

exit(0);

}

found=0;

fread(&erec, sizeof(erec),1,fp);

while(!feof(fp))

{

if(temp==erec.eno)

{

found=1;

printf("\n Employee Name     :%s", erec.ename);

printf("\n Employee Salary   :%.2f", erec.esal);

printf("\n Record Deleted");

}

else

fwrite(&erec,sizeof(erec),1,fp1);

fread(&erec,sizeof(erec),1,fp);

}

if(found==0)

printf("\n No such employee to delete");

fclose(fp);

fclose(fp1);

remove("emp.dat");

rename("temp.dat", "emp.dat");

getch();

break;

case 5:

fp=fopen("emp.dat", "r");

if(!fp)

{

printf("\n Unable to Open Source File");

getch();

exit(0);

}

fread(&erec, sizeof(erec),1,fp);

while(!feof(fp))

{

printf("%d\t%s\t%.2f\n", erec.eno,erec.ename,erec.esal);

fread(&erec,sizeof(erec),1,fp);

}

fclose(fp);

getch();

break;

case 6:

            exit(0);

default :

            printf("\n Invalid Chocie");

            getch();

            break;

            }

}

while(1);

getch();

}

**Storage Classes**

Storage classes are used to define scope and life time of a variable. There are four storage classes in C programming.

* auto
* extern
* static
* register

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Storage Classes | Storage Place | Default Value | Scope | Life-time |
| auto | RAM | Garbage Value | Local | Within function |
| extern | RAM | Zero | Global | Till the end of main program, May be declared anywhere in the program |
| static | RAM | Zero | Local | Till the end of main program, Retains value between multiple functions call |
| register | Register | Garbage Value | Local | Within function |

1) **auto**

The auto keyword is applied to all local variables automatically. It is the default storage class that is why it is known as automatic variable.

#include<stdio.h>

int main(){

int a=10;

auto int b=10;//same like above

printf("%d %d",a,b);

return 0;

}

2) **register**

The register variable allocates memory in register than RAM. Its size is same of register size. It has a faster access than other variables.

It is recommended to use register variable only for quick access such as in counter.

Note: We can't get the address of register variable.

register int counter=0;

3) **static**

The static variable is initialized only once and exists till the end of the program. It retains its value between multiple functions call.

The static variable has the default value 0 which is provided by compiler.

#include<stdio.h>

int  func(){

   static int i=0;//static variable

   int j=0;//local variable

   i++;

   j++;

   printf("i= %d and j= %d\n", i, j);

}

int main() {

  func();

  func();

  func();

return 0;

}

1. **extern**

The extern variable is visible to all the programs. It is used if two or more files are sharing same variable or function.

extern int counter=0;