**C introduction**

**A Brief History of C**

C was invented and first implemented by **Dennis Ritchie** on a DEC PDP-11 that used the UNIX operating system. C is the result of a development process that started with an older language called BCPL. BCPL was developed by Martin Richards, and it influenced a language called B, which was invented by Ken Thompson. B led to the development of C in the 1970s.

It was first described in The C Programming Language by Brian Kernighan and Dennis Ritchie (Englewood Cliffs, N.J.: Prentice-Hall, 1978). In the summer of 1983 a committee was established to create an ANSI (American National Standards Institute) standard that would define the C language.

The ANSI C standard was finally adopted in December 1989, the version of C defined by the 1989 standard is commonly referred to as C89.

**C Is a Middle-Level Language**

C is thought of as a middle-level language because it combines the best elements of high-level languages with the control and flexibility of assembly language. As a middle-level language, C allows the manipulation of bits, bytes, and addresses— the basic elements with which the computer functions. Despite this fact, C code is also very portable.

Portability means that it is easy to adapt software written for one type of computer or operating system to another type.

Different Languages:

Ada, Modula-2 , Pascal , COBOL, FORTRAN, BASIC, Java , C++, C, Assembly.

C is special in that it allows the direct manipulation of bits, bytes, words, and pointers. This makes it well suited for system-level programming, where these operations are common.

**C Is a Structured Language:**

C is commonly referred as a structured language. The distinguishing feature of a structured language is compartmentalization of code and data. If you develop compartmentalized functions, you need to know only what a function does, not how it does it.

A structured language offers a variety of programming possibilities. For example, structured languages typically support several loop constructs, such as while, do-while, and for. No structured programming languages: FORTRAN, BASIC, and COBOL

Structured programming languages: Pascal, Ada, C++, C, Java, Modula-2

C's main structural component is the function. In C, functions are the building blocks in which all program activity occurs. They allow you to define and code individually the separate tasks in a program, thus allowing your programs to be modular.

**Compilers vs. Interpreters:**

It is important to understand that a computer language defines the nature of a program and not the way that the program will be executed. There are two general methods by which a program can be executed. It can be compiled, or it can be interpreted.

An **interpreter** reads the source code of your program one line at a time, performing the specific instructions contained in that line. A **compiler** reads the entire program and converts it into object code, which is a translation of the program's source code into a form that the computer can execute directly. Object code is also referred to as binary code or machine code.

In general, an **interpreted** program runs **slower** than a **compiled** program. Remember, a compiler converts a program's source code into object code that a computer can execute directly. Therefore, compilation is a one-time cost, while interpretation incurs an overhead each time a program is run.

**Structure of a C program**

**Documentation statements:**

Documentation statements

Library statements

Definition statements

Global Declarations

int main(void)

{

Declaration statements;

Executable part statements;

}

ret\_type funName(parameters)

{

Statements;

}

ret\_type funName(parameters)

{

Statements;

}

The statements are used to describe the program. These statements are not included in the execution of the program. For that, we need to specify them as comments.

Comments in C written using following syntax:

/\* used to indicate comment starting

\*/ used to indicate ending of comment

Example:

/\* Program to print some message \*/

Comments/Document statements usage:

* Documentation of variables and functions and their usage
* Explaining complex sections of code
* Describes the program, author, date, modification changes, revisions…

**Library Statements:**

Here we write pre processor statements to include the library files (Header Files). Header files contain declarations of functions and variables which can be incorporated into any C program by using the pre-processor #include statement.

Standard header files are provided with each compiler, and cover a range of areas: string handling, mathematics, data conversion, printing and reading of variables, etc.

To use any of the standard functions, the appropriate header file should be included. This is done at the beginning of to the C program file.

Syntax: #include<name of the library>

For example, to use the function printf() in a program, the line

#include <stdio.h>

should be at the beginning of the source file, because the declaration for printf() is found in the file stdio.h. All header files have the extension .h and generally reside in the /include subdirectory.

Some examples:

#include <string.h>

#include <math.h>

#include "mylib.h"

• The use of angle brackets <> informs the compiler to search the compiler’s include directories for the specified file. The use of the double quotes "" around the filename informs the compiler to start the search in the current directory for the specified file.

**Definition Section:**

Here we define constants called as Symbolic Constants which are names given to values that cannot be changed, Implemented with the #define preprocessor directive.

Example:

#define N 3000

#define FALSE 0

#define PI 3.14159

#define FIGURE "triangle"

* Note that preprocessor statements begin with a # symbol, and are NOT terminated by a semicolon. Traditionally, preprocessor statements are listed at the beginning of the source file. By convention, preprocessor constants are written in UPPERCASE.
* Preprocessor statements are handled by the compiler (or preprocessor) before the program is actually compiled. All # statements are processed first, and the symbols (like N) which occur in the C program are replaced by their value (3000).

**main function:**

All C programs consist of one or more functions. As a general rule, the only function that must be present is called main( ), which is the first function called when program execution begins. “main( )” contains what is, in essence, an outline of what the program does. The outline is composed of function calls.

**An Example C Program**

/\* This program prints a one-line message \*/

#include <stdio.h>

int main()

{

printf("Hello World\n");

return 0;

}

**The Library and Linking:**

All C compilers come with a standard library of functions that perform most commonly needed tasks for your program. Standard C specifies a minimal set of functions that will be supported by all compilers.

When you call a library function, the C compiler ''remembers" its name. Later, the **linker** combines the code you wrote with the object code already found in the standard library. This process is called **linking**. Some compilers have their own linker, while others use the standard linker supplied by your operating system.

.obj file

.exe file

**Compilation and Running:**

C allows a program to be spread across two or more files, and it lets you compile each file separately. Once you have compiled all files, they are linked, along with any library functions, to form the complete object code.

Creating an executable form of your C program consists of these three steps:

1. Creating your program

2. Compiling your program

3. Linking your program with whatever functions are needed from the library

Running programs using compiler:

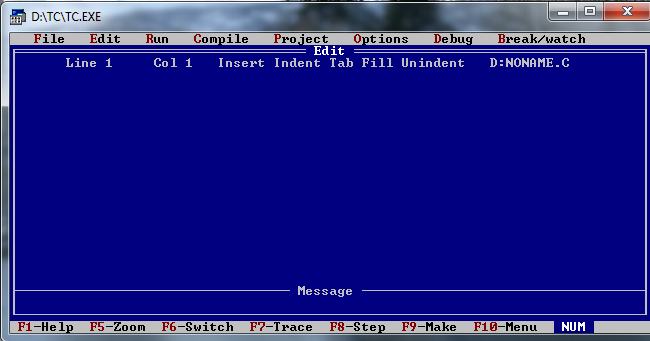
In windows environment we use different IDE (Integrated Development Environment) s that provide compilers to run our programs. There are different types of compilers available in the world.

Example:

Microsoft, Boreland (Turbo C2, Turbo C3)

We work on Boreland Turbo C2 compiler to develop c programs.

Turbo C2:

 Using Boreland Turbo C2 we get an Integrated Development Environment (IDE) where we can edit, compile and run C programs. Usually, Turbo C2 inserted in C drive. There is an application with the name TC, which provides an IDE. To run that we use the following path: C:\Turboc2\Tc.exe

If you run the TC.exe we will get an IDE as shown above.

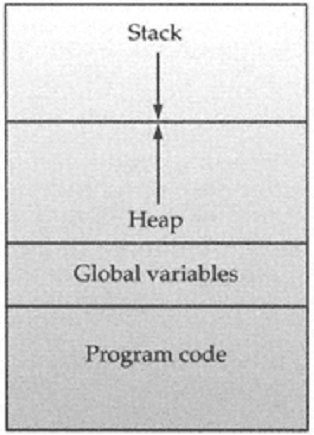
To make it a full screen window we use **Alt+Enter**.

In IDE, we develop C Source programs and we can run those programs.

Compilation: Alt + F9 or Alt + C + enter

Running: Ctrl + F9 or Alt + R + enter

Result: Alt + F5

**C's Memory Map:**

A compiled C program creates and uses four logically distinct regions of memory. The first region is the memory that actually holds the program's executable code. The next region is memory where global variables are stored. The remaining two regions are the stack and the heap. The stack is used for a great many things while your program executes. It holds the return addresses of function calls, arguments to functions, and local variables. It will also save the current state of the CPU. The heap is a region of free memory that your program can use via C's dynamic memory allocation functions.

**Terms:**

The terms that follow will be used frequently throughout the remainder of this reference. You should be completely familiar with them.

• **Source code**: The text of a program that a user can read commonly thought of as the program. The source code is input into the C compiler.

• **Object code:** Translation of the source code of a program into machine code, which the computer can read and execute directly. Object code is the input to the linker.

• **Linker:** A program that links separately compiled modules into one program. It also combines the functions in the Standard C library with the code that you wrote. The output of the linker is an executable program.

• **Library**: The file containing the standard functions that your program can use. These functions include all I/O operations as well as other useful routines.

• **Compile time**: The time during which your program is being compiled.

• **Run time**: The time during which your program is executing

**C fundamentals**

**Character set:**

Whenever we write any C program then it consists of different statements. Each C Program is set of statements and each statement is set of different characters as:

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

**Valid C Characters: Special Characters are listed below -**

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

**Expressions**

The most fundamental element of the C language: the expression. Expressions in C are substantially more flexible and powerful than in many other computer languages.

Expressions are formed from these atomic elements: **data and operators**. **Data** may be **represented by variables, constants, or values returned by functions**. C supports several different types of data. It also provides a wide variety of **operators**.

**The Basic Data Types**

A data type defines a set of values that a variable can store along with a set of operations that can be performed on that variable.

C89 standard defines five foundational data types: character, integer, floating-point, double floating-point, and valueless. These are declared using **char, int, float, double, and void**, respectively.

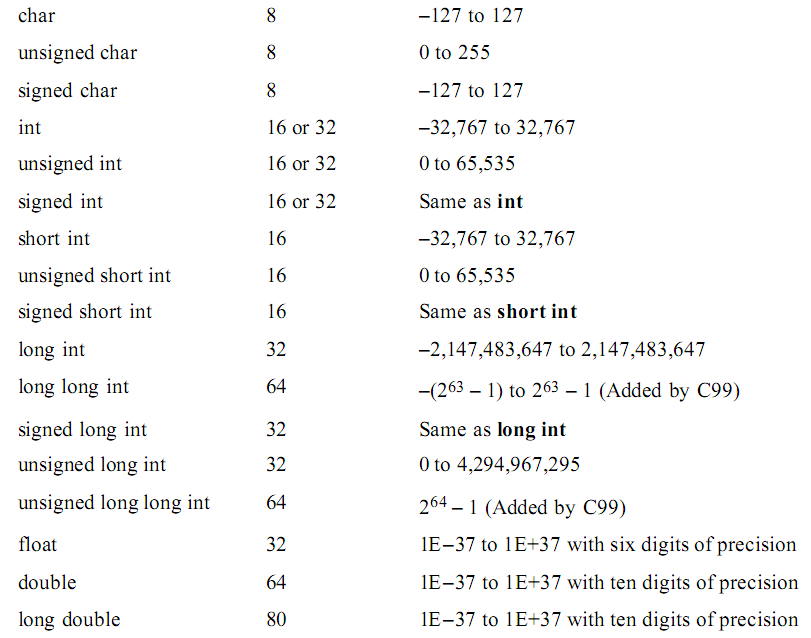
These types form the basis for several other types. Every value of particular type can have some storage and the range of values (stored by the Variable). The size and range of these data types may vary among processor types and compilers.

Variables of type char are generally used to hold values defined by the ASCII character set.

**Modifying the Basic Types**

Except type void, the basic data types may have various modifiers preceding them. A type modifier alters the meaning of the base type to more precisely fit a specific need. The list of modifiers is shown here: **signed, unsigned, long,** and **short.**

The int base type can be modified by signed, short, long, and unsigned. The char type can be modified by unsigned and signed. You may also apply long to double. (C99 also allows long to modify long, thus creating long long)



The following sets of type specifiers are equivalent:

**Specifier Same As**

signed signed int

unsigned unsigned int

long long int

short short int

**Identifier Names:**

In C, the names of variables, functions, labels, and various other user-defined items are called identifiers. The length of these identifiers can vary from one to several characters.

Rules to define a valid Identifier:

* The **first character** must be a letter(a-z, A-Z) or an underscore(\_),
* The Subsequent characters must be letters, digits, or underscores.

Here are some correct and incorrect identifier names:

**Correct Incorrect**

count 1count

test23 hi!there

high\_balance high . . . balance

In an identifier, upper-case and lower-case are treated as distinct. Hence, count, Count, and COUNT are three separate identifiers why because C Language is a CASE Sensitive Language.

An identifier cannot be the same as a C keyword and should not have the same name as functions that are in the C library.

**Key Words**

It is a word that has a specified/Defined Meaning in any Programming language which defines your language.

C89 standard defines 32 Keywords, which are as following…

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

**Variables**

A variable is a named location in memory that is used to hold a value that can be modified by the program time to time. All variables must be declared before they can be used. The general form of a declaration is

Syntax:

Data\_type variable list;

Here, type must be a valid data type plus any modifiers, and variable\_list may consist of one or more identifier names separated by commas. Here are some declarations:

int i, j, k;

short int si;

unsigned int ui;

double balance, profit, loss;

**Where Variables Are Declared?**

Variables can be declared in three places: inside functions, in the definition of function parameters, and outside of all functions. These positions correspond to **local variables, formal parameters, and global variables**, respectively.

**Constants**

Constants refer to fixed values that the program may not alter. Constants can be of any of the basic data types. The way each constant is represented depends upon its type. Constants are also called literals.

**Character constants** are enclosed between single quotes. For example, **'a' and '%'** are both character constants. However, "a" is a string containing only one letter.

**Integer constants** are specified as numbers without fractional components. For example, 10 and – 100 are integer constants.

**Floating-point constants** require the decimal point followed by the number's fractional component. For example, 11.123 is a floating-point constant. C also allows you to use scientific notation for floating-point numbers.

**Hexadecimal and Octal Constants:** It is sometimes easier to use a number system based on 8 or 16 rather than 10. The number system based on 8 is called octal and uses the digits 0 through 7. In octal, the number 10 is the same as 8 in decimal. The base 16 number system is called hexadecimal and uses the digits 0 through 9 plus the letters A through F, which stand for 10, 11, 12, 13, 14, and 15, respectively. For example, the hexadecimal number 10 is 16 in decimal. Because these two number systems are used frequently, C allows you to specify integer constants in hexadecimal or octal instead of decimal. A hexadecimal constant must consist of a 0x followed by the constant in hexadecimal form. An octal constant begins with a 0. Here are some examples:

int hex = 0x80; /\* 128 in decimal \*/

int oct = 012; /\* 10 in decimal \*/

**String Constants:** C supports another type of constant: the string. A string is a set of characters enclosed in double quotes. For example, ''this is a test" is a string. You have seen examples of strings in some of the printf( ) statements in the sample programs.

**Example:**

**/\* program that reads an integer constant and print on the screen \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

char cvar;

clrscr();

cvar = 'c';

printf("%c", cvar);

}

**/\* program that reads an integer constant and print on the screen \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int ivar;

clrscr();

ivar = 9;

printf("%d", ivar);

}

**/\* program that reads an integer constant and print on the screen \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

float fvar;

clrscr();

fvar = 996.215;

printf("%f", fvar);

}

**Backslash Character Constants:**

C includes the special backslash character constants; these are also referred to as escape sequences.

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Meaning | Code | Meaning |
| \b | **Backspace** | **\\** | **Backslash** |
| \f | **Form feed** | **\v** | **Vertical tab** |
| \n | **New line** | **\a** | **Alert** |
| \r | **Carriage return** | **\?** | **Question mark** |
| \t | **Horizontal tab** | **\N** | **Octal constant (where N is an octal constant)** |
| \" | **Double quote** | **\xN** | **Hexadecimal constant (where N is a hexadecimal constant)** |
| \' | **Single quote** |  |  |

**Data Reading/Writing:**

we can read/write information in two different ways.

1. Using Console

2. Using File

**Console Input / Output:**

Console is a User screen where you can enter data or you can read. If you want to perform Input / Output operations using Console we need to use some functions that are related to the Standard Input / Output.

**Functions to perform Character Input / Output:**

Input Functions:

**a. getch()**

It reads single character from the standard input device (keyboard) without echoing the character.

usage: char a;

a = getch();

**b. getchar()**

It reads single character from the standard input device with echoing the character.

usage: char a;

a = getchar();

**c. gets(const\* char)**

It reads a single/sequence of characters as a string untill a new line character encounters.

usage: char \*a;

gets(a);

Output Functions:

**a. putch(char)**

It prints a character on to the standard output.

usage: char a;

a = getch();

putch(a);

**b. putchar(char)**

It prints a character on to the standard output.

usage: char a;

a = getchar();

putchar(a);

**c. puts(char \*string)**

It prints a string to the standard output, with appending a new line to the end.

usage: char \*ch;

gets(ch);

puts("This is a string you entered...");

puts(ch);

**/\* program to use getch(), and putch(), getchar(), and putchar() \*/**

void main()

{

char ch;

clrscr();

/\* Reading and printing a character using getch() and putch() \*/

puts("Enter a character: ");

ch = getch();

puts("The character you entered is: ");

putch(ch);

/\* Reading and printing a character using getchar() and putchar() \*/

puts("\nEnter a character: ");

ch = getchar();

puts("The character you entered is : ");

putchar(ch);

getch();

}

**/\* program to read and write a string using gets() and puts() \*/**

void main()

{

char str[15];

clrscr();

puts("Enter a string: ");

gets(str);

/\* scanf("%s",&str); \*/

puts("\nThe string you entered is: ");

puts(str);

/\* printf("%s", str);\*/

getch();

}

**Formatted Input:**

a. scanf()

It reads formatted input from the standard input(stdin) device by default keyboard.

syntax: int scanf(format string, list of variables);

format string:

it is a string, which consists of sequence of characters / format specifiers.

format specifiers:

These indicate the format of specific type, which Preceded by a percentage (%) symbol.

|  |  |  |  |
| --- | --- | --- | --- |
| %c | character | %o | octal |
| %d | decimal ( integral ) | %s | string |
| %e | exponential ( float ) | %u | unsigned |
| %f | float | %x | hexa decimal |
| %h | short (short int ) | %lf | double |
| %i | integer ( decimal, octal, hexa ) | %Lf | Long Double |
| %l | long |  |  |

List of Variables:

These are list of variables separated by commas, which specify the address of the variable in which the scanned formatted data can be stored. These variables can be preceded by an address of (&) operator.

usage: int a;

char c;

scanf("%d", &a);

scanf("%c", &c);

**Formatted Output:**

**a. printf()**

It prints the formatted output to the standard output (stdout) which is Console by default.

syntax: int printf(format string, list of variables);

**format string:**

it is a string, which consists of sequence of characters / format specifiers.

**format specifiers:**

These indicate the format of specific type, which started with a percentage (%) symbol.

|  |  |  |  |
| --- | --- | --- | --- |
| %c | character | %o | octal |
| %d | decimal ( integral ) | %s | string |
| %e | exponential ( float ) | %u | unsigned |
| %f | float | %x | hexa decimal |
| %h | short (short int ) | %lf | double |
| %i | integer ( decimal, octal, hexa ) | %Lf | Long Double |
| %l | long |  |  |

**List of Variables:**

These are list of variables separated by commas from which the formatted data can be printed.

usage:

int a;

char c;

printf("%d", a);

printf("%c", c);

/\* program to read a character from the screen, and print on to the screen \*/

#include<stdio.h>

#include<conio.h>

void main()

{

char cvar;

clrscr();

printf("Enter a character: ");

scanf("%c", &cvar);

printf("%c The character you entered is: ", cvar);

getch();

}

/\* program that read an integer using scanf, and prints using printf \*/

#include<stdio.h>

#include<conio.h>

void main()

{

int ivar;

clrscr();

printf("Enter an integer: ");

scanf("%i", &ivar);

printf("\nThe value you entered is: ");

printf("%i", ivar);

getch();

}

/\* program to declare a float variable and assign a value, then print on to the screen \*/

#include<stdio.h>

#include<conio.h>

void main()

{

float fvar;

clrscr();

printf("Enter a float value: ");

scanf("%f", &fvar);

printf("The float value you entered is : %f", fvar);

getch();

}

/\* program to read multiple integer variables \*/

void main()

{

int i1, i2, i3;

clrscr();

/\* printf("Enter three values:\n");

scanf("%d", &i1);

scanf("%d", &i2);

scanf("%d", &i3);

printf("You entered :\n");

printf("%3d", i1);

printf("%3d", i2);

printf("%3d", i3);

\*/

printf("Enter three values:\n");

scanf("%d%d%d", &i1, &i2, &i3);

printf("You entered :");

printf("%d %d %d", i1, i2, i3);

getch();

}

C Program contains the statements which are of set of Expressions. The Expression comprises of two parts: 1. Data, and 2. Operations

Expressions:

An Expression is evaluated for a value of the specific type, which performs an operation consists of Operands (constant, variable), or Operators.

**Operators**

Operator is a symbol to perform an operation. The operator can be used in anywhere as it is applicable.

There are three types of operators:

**1. Unary Operators**

The Operator used with only one operand.

Ex:

unary minus:

-a, -234

**2. Binary Operators**

The Operators can be used with at-least two operands.

Ex:

Binary Plus or Minus: a + b, a - 2

**3. Ternary Operator**

This Operator require three operands to perform particular operation.

Operators Categories based on Type of Operation:

**1. Arithmetic Operators**

These are used to perfom Arithmetic operations like Addtion, subtraction, Multiplication, Division, Modular Division.

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Operation | Expression | value |
| + | Addition | 2 + 5 = | 7 |
| - | Subtraction | 5 - 4 = | 1 |
| \* | Multiplication | 4 \* 6 = | 24 |
| / | Division | 7 / 3 = | 2 |
| % | Modular Division | 7 % 3 =(for reminder) | 1 |

/\* program to use Arithmetic operators( +, -, \*, /, %) \*/

#include<stdio.h>

#include<conio.h>

void main()

{

int var1, var2, result;

clrscr();

/\* printing addition of two numbers \*/

printf("Enter two values: \n");

scanf("%d%d", &var1, &var2);

result = var1 + var2;

printf("The Addition of the two values( %d & %d ) is: %d", var1, var2, result);

/\* printing subtraction of two numbers \*/

printf("\nEnter two values: \n");

scanf("%d%d", &var1, &var2);

result = var1 - var2;

printf("The Subtraction of the two values( %d & %d ) is: %d", var1, var2, result);

/\* printing multiplication of two numbers \*/

printf("\nEnter two values: \n");

scanf("%d%d", &var1, &var2);

result = var1 \* var2;

printf("The Multipllication of the two values( %d & %d ) is: %d", var1, var2, result);

/\* printing division of two numbers \*/

printf("\nEnter two values: \n");

scanf("%d%d", &var1, &var2);

result = var1 / var2;

printf("The Division of the two values( %d & %d ) is: %d", var1, var2, result);

/\* printing Modular Division of two numbers \*/

printf("\nEnter two values: \n");

scanf("%d%d", &var1, &var2);

result = var1 % var2;

printf("The Modular Division of the two values( %d & %d ) is: %d", var1, var2, result);

getch();

}

**2. Relational Operators:**

These operators are used to check the relation among the operands/expressions.

Expressions evaluated with these operators have only two values. they are 0 (zero) or any Non Zero( By default 1) which represent the false and true state of the expression.

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Operation | Expression | value |
| < | Less than | 1 < 5 = | 1 |
| > | Greater than | 2 > 5 = | 0 |
| == | Equals to | 2 == 7 | 0 |
| <= | Less than or | 2 <= 2 | 1 |
|  | equals to | 2 <= 4 | 1 |
| >= | greater than or | 2 >= 2 | 1 |
|  | equals to | 2 >= 4 | 0 |
| != | Not equals to | 4 != 5 | 1 |

**/\* program to understand the relational operators \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int a = 4, b = 6;

clrscr();

printf("Consider the Result as:\n1. for True\n0. for False\n");

printf("%d is greater than %d , Result is: %d\n", a, b, a>b);

printf("%d is Less than %d , Result is: %d\n", a, b, a<b);

printf("%d is greater than or equal to %d , Result is: %d\n", a, b, a>=b);

printf("%d is Less than or equal to %d , Result is: %d\n", a, b, a<=b);

printf("%d is Equal to %d , Result is: %d\n", a, b, a==b);

printf("%d is Not equal to %d , Result is: %d\n", a, b, a!=b);

getch();

}

**Logical Operators:**

These are used to operate more than one condition ( relational Expression).

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Operation | Expression | Value |
| && | Logical AND | (1>5) && (4<5) | 0 |
| || | Logical OR | (12>10)||(20<5) | 1 |
| ! | Logical Not | !(3>7) | 1 |

c1 c2 c1 && c2

1 1 1

1 0 0

0 1 0

0 0 0

c1 c2 c1 || c2

1 1 1

1 0 1

0 1 1

0 0 0

**/\*program to understand logical operators (&&,||,!) \*/**

#include <stdio.h>

#include <conio.h>

void main()

{

int a1,a2,a3;

a1 = 3;

a2 = 2;

a3 = 4;

clrscr();

/\* Working with Logical And(&&) \*/

printf("Result:\n1. For True,\n0. For False.\n");

printf("%d is greater than %d and %d --> Result: %d\n",a1,a2,a3,(a1>a2 && a1>a3));

printf("%d is greater than %d and %d --> Result: %d\n",a2,a1,a3,(a2>a1 && a2>a3));

printf("%d is greater than %d and %d --> Result: %d\n",a3,a1,a2,(a3>a1 && a3>a2));

/\* Working with Logical OR(||) \*/

printf("%d is greater than any of %d or %d --> Result: %d\n",a1,a2,a3,(a1>a2 || a1>a3));

/\* Using Logical NOT (!) \*/

printf("%d is Not greater than %d --> Result: %d\n",a1,a2,!(a1>a2));

getch();

}

**Assignment Operators:**

The Assignment operator (=) is used to assign a value of an expression to its right to the variable of its left.

Ex: a = 20; Here value 20 is assigned to the variable 'a'.

**Short Hand Assignment Operators**:

These are a combination of Assignment operators with the Arithmetic. They are,

where a = 20;

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Meaning | Expression | value |
| += | a = a + exp | a += 20; | 40 |
| -= | a = a - exp | a -= 20; | 0 |
| \*= | a = a \* exp | a \*= 20; | 400 |
| /= | a = a / exp | a /= 20; | 1 |
| %= | a = a % exp | a %= 20; | 0 |

/\* Program to perform operations using Assignment operators(=, +=, -=, \*=, /=, %=) \*/

#include<stdio.h>

#include<conio.h>

void main()

{

int a = 20;

clrscr();

printf("Working on ASSIGNMENT OPERATORS ....\n");

/\* Using Assignment operator \*/

a = 35;

printf("After Assigning the value to \'a\' is : %d\n", a);

/\* Using += Operator \*/

a += 10; /\* This is equals to the expression a = a + 10; \*/

printf("After using += 10 on \'a\' is : %d\n", a);

/\* Using -= Operator \*/

a -= 10; /\* This is equals to the expression a = a -10; \*/

printf("After using -= 10 on \'a\' is : %d\n", a);

/\* Using \*= Operator \*/

a \*= 10; /\* This is equals to the expression a = a \* 10; \*/

printf("After using \*= 10 on \'a\' is : %d\n", a);

/\* Using /= Operator \*/

a /= 10; /\* This is equals to the expression a = a / 10; \*/

printf("After using /= 10 on \'a\' is : %d\n", a);

/\* Using %= Operator \*/

a %= 10; /\* This is equals to the expression a = a % 10; \*/

printf("After using modular division = 10 on \'a\' is : %d\n", a);

getch();

}

**Bitwise Operators:**

These operators dealt with the bits representation of the data.

For ex: number 2 can be represented in bits as 0000 0010

number 3 can be as 0000 0011

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Meaning | Expression | value |
| & | Bitwise AND | 2 & 3 = | 2 |
| | | Bitwise OR | 2 | 3 = | 3 |
| ~ | Complement |  |  |
| ^ | Bitwise XOR |  |  |
| << | Left Shift | 2 << 2 = | 8 |
| >> | Right Shift | 2 >> 2 = | 0 |

**/\* program: To Understand the Bitwise operators \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int a = 3, b = 4;

clrscr();

printf("Using BITWISE OPERATORS...\n");

printf("Values in a and b are: \na = %d, b = %d\n", a, b);

/\* Using Bitwise AND \*/

printf("After performing a & b : %d\n", (a & b));

/\* Using Bitwise OR \*/

printf("After Performing a | b : %d\n", (a | b));

/\* Using Bitwise XOR \*/

printf("After Performing a ^ b : %d\n", (a ^ b));

/\* Using Bitwise LEFT SHIFT \*/

printf("After Performing a << 2 : %d\n", (a << 2));

/\* Using Bitwise RIGHT SHIFT \*/

printf("After Performing a >> 2 : %d\n", (a >> 2));

getch();

}

**Increment/Decrement Operators:**

These operators used to increment or decrement a value by 1(one).

There are two types:

Pre Increment/ Pre Decrement

Post Increment/ Post Decrement

where a = 10;

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Meaning | Expression | value |
| ++ | Increment | a++( a= a +1) | 11 |
| -- | Decrement | a-- ( a = a -1) | 9 |

**/\* program to understand Increment and decrement operators \*/**

**/\* pre : ++(increment), --(decrement), post: (increment)++, (decrement) -- \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int value = 5, element = 10;

int result = 0;

clrscr();

printf("Initial Values of value : %d, and element : %d", value, element);

result = value++ + --element;

printf("\nResult of post incre and pre decre of value and element is: %d", result);

printf("\nValues after post incre of value : %d, and pre decre of element : %d", value, element);

result = ++value + element--;

printf("\nResult of pre incre and post decre of value and element is: %d", result);

printf("\nValues after pre incre of value : %d, and post decre of element : %d", value, element);

getch();

}

**Conditional Operator (?:) :**

It is the Ternary Operator which requires three operands in order to use it.

Syntax:

**(Expression 1) ? (Expression 2) : (Expression 3);**

Here Exp1 will be a condition, that can be evaluated for a value, If it is true then the Exp2 will be executed otherwise Exp3 will be executed.

Example:

(a > b) ? printf("%d", a) : printf("%d", b);

**/\* program to understand the conditional operator (?:) to find the greatest among two numbers \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int num1 = 4, num2 = 6, greater;

clrscr();

printf("Checking the two numbers(%d and %d) for greatest:\n",num1,num2);

greater = (num1 > num2) ? num1 : num2;

printf("%d is greater...",greater);

/\* (num1 > num2) ? printf("%d is greater",num1) : printf("%d is greater",num2); \*/

getch();

}

Special Operators:

These are special operators, that are used in different ways.

**comma(,) :** This is to specify the separation

**sizeof()** : This function is to find the memory size(in bytes) of the given variable/ type. It will return a number which specify the size of the variable/type.

ex:

sizeof(int); ---> 2

**/\* program to find sizes of int , float, char, double, short, long signed, unsigned \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

char ch;

int i;

clrscr();

/\* printing the size of the different types \*/

printf("Size of un signed character is: %d \n", sizeof(unsigned char));

printf("Size of signed character is: %d \n", sizeof(signed char));

printf("Size of un signed int is: %d \n", sizeof(unsigned int));

printf("Size of signed int is: %d \n", sizeof(signed int));

printf("Size of short int is: %d \n", sizeof(short int));

printf("Size of long int is: %d \n", sizeof(long int));

printf("Size of float is: %d \n", sizeof(float));

printf("Size of double is: %d \n", sizeof(double));

printf("Size of long double is: %d \n", sizeof(long double));

i = 30293;

printf("Size of integer variable i is: %d\n", sizeof(i));

ch = 'r';

printf("Size of character variable ch is: %d", sizeof(ch));

getch();

}

Pointer Operators:

& (Address of), \* (Indirection)

Member Accessing Operators:

. (Dot), -> (Arrow)

**Conditional/Control Statements**

These statements will control the execution of the code / program based on a condition.

**1. Selection Statements:**

Here using these statements we can skip some/block of statements executing sequentially.

The selection can be done based on a condition; condition may have two possible values:

**true --- any non zero value,**

**false --- zero**

a. simple if

b. nested if

c. if-else

d. nested if-else

e. else-if laddar

f. switch

**a. Simple if**

syntax:

if(condition )

{

...True Statements...

}

this selection statement executed as if the condition is true

than only the "True Statements" can be executed.

Eg:

if( a>b )

{

printf("a is greater value...");

}

**/\*program for understanding the simple if by finding the max among four values\*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int a, b,c,d;

clrscr();

{

printf("enter four numberd to find greatest among them...\n");

scanf("%d%D%D%D",&a,&b,&c,&d);

}

/\*simple if checking a condition...\*/

if((a>b)&&(a>c)&&(a>d))

{

printf("greatest is:%d",a);

}

if ((b>a)&&(b>c)&&(b>d))

{

printf("greatest is;%d",b);

}

if((c>a)&&(c>b)&&(c>d))

{

printf("greatest is:%d",c);

}

if((d>a)&&(d>b)&&(d>c))

{

printf("greatest is:%d",d);

}

if((a==b)&&(b==c)&&(c==d))

printf("they are equal...");

getch();

}

**b. nested if**

syntax:

if( condition1 )

{

...True statements for condition1...

if( condition2 )

{

...True statements for condition2...

}

}

**/\* program for understanding the Nested IF \*/**

#include <stdio.h>

#include <conio.h>

void main()

{

int a,b,c;

clrscr();

printf("Enter three numbers to find greatest among them...\n");

scanf("%d%d%d",&a,&b,&c);

/\* nested if checking a condition...\*/

if( a>b )

{

if( a>c )

{

printf("Greatest is: %d",a);

}

}

if( b>a )

{

if( b>c )

{

printf("Greatest is: %d",b);

}

}

if( c>a )

{

if( c>b )

{

printf("Greatest is: %d",c);

}

}

getch();

**c. if-else**

syntax:

if( condition )

{

...true statements...

}

else

{

...false statements...

}

In this control statement if the condition is true then the true statements will be executed otherwise the statements in the else block will be executed.

Example:

if( a>b )

{

printf("a is gretest...");

}

else

{

printf("b is gretest...");

}

**/\* program for understanding if-else where it is used to find the minimum among the two\*/**

void main()

{

int num1,num2;

clrscr();

printf("Enter two numbers:\n");

scanf("%d%d",&num1,&num2);

/\* if-else staement \*/

if( num1<num2 )

{

printf("Minimum is : %d",num1);

}

else

{

printf("Minimum is : %d",num2);

}

getch();

}

**d. nested if-else**

syntax:

if( condition 1 )

{

true statements for condition 1

if( condition2 )

{

true statements for condition2

}

else

{

false statements for condition2

}

}

else

{

false statements for condition 1

if( condition3 )

{

true statements for condition 3

}

else

{

false statements for condition3

}

}

This is the nested if-else where there are multiple number of conditions to check, if a condition1 is true then the true statements of the condition will be executed, again condition2 will be checked then statements according to that will be executed.

/\* program to find the greatest among the three numbers using Nested IF-ELSE \*/

#include<stdio.h>

#include<conio.h>

void main()

{

int a1, a2, a3;

clrscr();

printf("Enter three numbers to find greatest: \n");

scanf("%d%d%d",&a1, &a2, &a3);

/\* Finding greatest using nested if-else statements \*/

if( a1 > a2 )

{

if( a1 > a3 )

{

printf("%d is greater value...\n", a1);

}

else

{

printf("%d is greater value...\n", a3);

}

}

else

{

if( a2 > a3 )

{

printf("%d is greater value...\n", a2);

}

else

{

printf("%d is greater value...\n", a3);

}

}

getch();

}

**e. else-if laddar:**

syntax:

if( condition1 )

{

...true statements of condition1...

}

else if( condition2 )

{

...true statements of condition2...

}

else if( condition3 )

{

...true statements of condition3...

}

:

:

:

else if (condition N)

{

...true statemetns of conditionN

}

else

{

...false statements of conditionN...

}

Here the condition1 will be checked first and if it is true, then the true statements of it will be executed, otherwise condition2 will be checked then if it is true the true statements of it will be executed, this process continued untill the last condition...

**/\* program to understand the usage of else-if laddar statement \*/**

#include <stdio.h>

#include <conio.h>

void main()

{

int color;

clrscr();

printf("Enter a number between 1 to 5...\n");

scanf("%d",&color);

if(color==1)

{

printf("1 for GREEN...");

}

else if(color==2)

{

printf("2 for BLUE...");

}

else if(color==3)

{

printf("3 for VOILET...");

}

else if(color==4)

{

printf("4 for BLACK...");

}

else if(color==5)

{

printf("5 for WHITE...");

}

else

{

printf("you entered a WRONG number...");

}

getch();

}

**f. switch statement**

This statement is used to check the choice, make the selection based on the case that we have declared.

syntax:

switch( choice )

{

case number1: statements1...

break;

case number2: statemetns2...

break;

:

:

:

default: statements...

break;

}

Example:

int ch=3;

switch(ch)

{

case 1:printf("you entered 1");

break;

case 2: printf("you entered 2");

break;

case 3: printf("you entered 3");

break;

default: printf("you entered other than 1, 2, and 3...");

break;

}

**/\* program for understanding switch statement where we take one choice as input, based on input we perform different operations, output value according to the operation performed\*/**

void main()

{

int ch,a,b,c;

clrscr();

printf("Enter \n 1. for Addition\n 2. for Subtraction\n 3.for multiplication\n 4. for division\n");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("Enter two numbers:\n");

scanf("%d%d",&a,&b);

c = a+b;

printf("Addition of %d & %d is: %d",a,b,c);

break;

case 2: printf("Enter two numbers:\n");

scanf("%d%d",&a,&b);

c = a-b;

printf("Subtraction of %d & %d is: %d",a,b,c);

break;

case 3: printf("Enter two numbers:\n");

scanf("%d%d",&a,&b);

c = a\*b;

printf("Multiplication of %d & %d is: %d",a,b,c);

break;

case 4: printf("Enter two numbers:\n");

scanf("%d%d",&a,&b);

c = a/b;

printf("Division of %d & %d is: %d",a,b,c);

break;

default:

printf("INVALID Choice...Try Again...!!!");

break;

}

getch();

}

**/\* finding the given character is vowel or consonant using SWITCH \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

char ch;

clrscr();

printf("Enter a character to find whether it is vowel or consonant:");

ch = getchar();

if( ((ch >= 65) && ( ch <= 90)) || ((ch >= 97) && (ch <= 122)) )

{

ch = toupper(ch);

switch( ch )

{

case 'A':

case 'E':

case 'I':

case 'O':

case 'U':

printf("Entered character is VOWEL...");

break;

default:

printf("Entered character is CONSONENT...");

break;

}

}

else

printf("the character you entered is NOT a VALID character...!!!");

getch();

}

**/\* program to find whether the given number is Even or Odd \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int num;

clrscr(); /\* Reading a number to check for even or odd \*/

printf("Enter a number to check for even or odd: ");

scanf("%d", &num);

/\* checking for Even or Odd \*/

if( (num % 2) == 0 )

{

textcolor(YELLOW);

cprintf("It is EVEN Number...");

}

else

{

textcolor(GREEN);

cprintf("It is ODD Number...");

}

getch();

}

**/\* Program to check whether a given number is positive or not \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int num;

clrscr();

/\* Reading a number \*/

printf("Enter a number to find whether it is +ve or not: ");

scanf("%d", &num);

if( num > 0 )

{

textcolor(GREEN);

cprintf("It is a POSITIVE Number...");

}

else

{

textcolor(RED);

cprintf("It is NOT a Positive Number ...!!!");

}

getch();

}

**/\* Program to find whther given number is divisible by a number or not \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int num, div;

clrscr();

/\* Reading a number to check for divisibility \*/

printf("Enter a number: ");

scanf("%d", &num);

/\* Reading a number which divides the number you entered to check \*/

printf("Enter number to check for divisibility of no. you entered: ");

scanf("%d", &div);

if( (num % div) == 0 )

{

textcolor(GREEN);

cprintf("%d is divisible by %d...", num, div);

}

else

{

textcolor(RED);

cprintf("%d is NOT divisible by %d...", num, div);

}

getch();

}

**2. Iteration/looping Statements:**

Iteration Statements are used to repeat the block of statements until the condition gets false.

Iteration statements are of two types:

i. pre condition

The iteration statements will check the condition before executing the block of statements.

ii. Post condition

The iteration statements will check the condition after executing the block of statements once.

**Different Looping Statements:**

a. While

b. do-while

c. For

a. While

It is a pre conditional Iteration statement, where a condition will be checked first and then the repetition of the block of statements (should contain the termination condition) continued until the condition is false.

Syntax:

Initialization;

while( condition )

{

statements to be executed;

increment/decrement;

}

Example:

Without using iteration statements

int a=1;

printf("%d",a); 1

a = a+1;

printf("%d",a); 2

a = a+1;

printf("%d",a); 3

:

:

with iteration statement

int a=1;

while( a<=10 )

{

printf("%d ",a);

a = a+1;

}

**/\* program to print even numbers up to the given number \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int i, num;

clrscr();

printf("Enter the number to print even numbers up to that no: ");

scanf("%4d", &num);

/\* printing even numbers from 1 to the given size \*/

i = 1;

while( i <= num )

{

if( i%2 == 0 )

printf("%5d", i);

i++;

}

getch();

}

**b. do-while**

this looping statement is the post condition checking looping statement that executes the block of statements once before going to check the condition.

syntax:

do

{

block of statements;

}while( condition );

here the block of statements will be executed once before checking condition, then if the condition is true the block of statements will be executed until it gets false.

Eg:

printf("Enter a character...(enter 'n' for end):");

ch = getchar();

do

{

printf("you entered : ");

putc(ch);

printf("Enter character...: ");

ch = getchar();

}while( ch != 'n');

**c. For Statement**

This is a pre-condition checking statement, where it has an Initialization, condition checking, and Increment/Decrement in a single statement.

Syntax:

for( initialization; condition ; increment / decrement)

{

block of statements;

}

In for statement we can initialize, and check the condition, and decrement or increment a single line.

Where the execution starts from initialization, then it checks the condition, if it is true then executes the block of statements in for loop, after that it executes the increment/decrement statement provided in the for statement then it comes to check condition, the process repeats until the condition is false.

Eg:

for(a=1; a<=10; a++)

{

printf("%d",a);

}

Variations in For Statement:

a. for(initialization ; Condition ; I/D ment)

b. initialization;

for( ; Condition ; I/D ment )

c. initialization;

for(; ; I/D ment )

{ Condition; }

d. initialization;

for( ; ; )

{ condition;

I/D ment; }

**/\* program to print 1 to 10 numbers using for looping Statement \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int a;

clrscr();

printf("Values from 1 to 10: \n");

for( a = 1; a <= 10; a = a + 1)

{

printf("%d ", a);

}

getch();

}

**/\* printing the characters from a to z using for loop \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

char ch = 'a';

clrscr();

/\* printing the characters \*/

for( ; ch <= 'z'; ch++)

printf("%c ", ch);

getch();

}

**/\* program to understand the for looping statement \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int i;

clrscr();

printf("The ASCII numbers for the characters are:\n");

for(i=0;i<128;i++)

{

printf("%3d ->%1c ", i, i);

}

printf("\nTo continue enter any key...");

getch();

clrscr();

printf("\n");

for(i=128;i<=255;i++)

{

printf("%4d -> %c ",i,i);

}

getch();

}

**/\* program to find the sum of individual digits of a given number \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

long rem, sum = 0, num, temp;

clrscr();

/\* enter a number for which we need to find the sum of digits \*/

printf("Enter a number to find sum of its digits(max 6 digits) : ");

scanf("%6ld", &num);

temp = num;

/\* finding the sum of the individual digits of the given number \*/

while( num != 0 )

{

rem = num % 10;

sum = sum + rem; num = num / 10;

}

/\* printing the result \*/

printf("The sum of the individual digits of the num %ld is: %ld", temp, sum);

getch();

}

**/\* program to find the sum of numbers in the given range \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int rfrom, rto, sum = 0, i;

clrscr();

/\* Reading range from and to values \*/

printf("Enter the range (from,to): ");

scanf("%d%d", &rfrom, &rto );

/\* Checking the rfrom and rto values \*/

if( rfrom > rto )

{

textcolor(RED);

cprintf("You Entered INVALID Range...!");

}

else

{

for( i = rfrom; i <= rto ; i++ )

{

sum += i;

}

/\* Writing the summation on to the standard output \*/

printf("\nThe sum of values in the given range: %d", sum);

}

getch();

}

**/\* program to find the reverse of a given number \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

long digit, rev = 0, num, temp;

clrscr();

/\* Enter a number for which we need to find the reverse of it \*/

printf("Enter a number to find its reverse(max 8 digits): ");

scanf("%8ld", &num);

temp = num;

/\* finding the reverse of the given number \*/

while( num != 0 )

{

digit = num % 10;

rev = (rev\*10) + digit ;

num = num / 10;

}

/\* printing the result \*/

printf("The Reverse of the given num %ld is: %ld", temp, rev);

getch();

}

**/\* Program to check whether the given number is prime or not \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int num, i, flag;

clrscr();

/\* Reading the number to check for Prime \*/

printf("Enter The number to check for PRIME : ");

scanf("%4d", &num);

/\* Finding count \*/

flag = 0;

for( i = 1; i <= num; i++ )

{

if( (num % i) == 0 )

{

flag++;

}

}

/\* Checking for Prime \*/

if( flag == 2 )

{

textcolor(GREEN);

cprintf("It is a PRIME NUMBER...");

}

else

{

textcolor(RED);

cprintf("It is NOT a PRIME NUMBER...!");

}

getch();

}

**Loops Inside another loops(Nested Loops):**

Here we use nesting of loops to perform some operations.

eg:

while(Condition1)

{

while(condition2)

{

}

}

In the above example if untill the condition1 is true the condition2 will be executed untill it is ture(Condition 2).

**/\* program that makes use of nested loops\*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int a,b;

clrscr();

printf("Printing Values from 10 to 1 (10 times) :\n");

a=1;

while(a<=10)

{

b=10;

printf("time %2d -> ",a);

while(b>=1)

{

printf("%-3d",b);

b=b-1;

}

a=a+1;

printf("\n");

}

getch();

}

**/\* Program to print the following structure**

**1 2 3 4 5**

**1 2 3 4 5**

**1 2 3 4 5**

**1 2 3 4 5**

**1 2 3 4 5**

**\*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int i, j;

clrscr();

/\* Printing the Structure \*/

for( i = 1; i <= 5; i++ )

{

for( j = 1; j <= 5; j++ )

{

textcolor(YELLOW);

cprintf("%2d", j);

}

printf("\n");

}

getch();

}

**/\* Program to print the following structure**

5 4 3 2 1

5 4 3 2 1

5 4 3 2 1

5 4 3 2 1

5 4 3 2 1

\*/

#include<stdio.h>

#include<conio.h>

void main()

{

int i, j;

clrscr();

/\* Printing the Structure \*/

for( i = 1; i <= 5; i++ )

{

for( j = 5; j >= 1; j-- )

{

textcolor(YELLOW);

cprintf("%2d", j);

}

printf("\n");

}

getch();

}

/\* Program to print the following structure

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

\*/

#include<stdio.h>

#include<conio.h>

void main()

{

int i, j;

clrscr();

/\* Printing the Structure \*/

for( i = 1; i <= 5; i++ )

{

for( j = 1; j <= i; j++ )

{

textcolor(YELLOW);

cprintf("%2d", j);

}

printf("\n");

}

getch();

}

Structure 4:

5

5 4

5 4 3

5 4 3 2

5 4 3 2 1

Structure 5:

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

Structure 6:

5

4 4

3 3 3

2 2 2 2

1 1 1 1 1

Structure 7:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

Structure 8:

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

Structure 9:

5

5 4

5 4 3

5 4 3 2

5 4 3 2 1

Structure 10:

5

4 4

3 3 3

2 2 2 2

1 1 1 1 1

Structure 11:

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

Structure 12:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

Structure 13:

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

Structure 14:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

**/\* program to print the prime numbers from 1 to the given range \*/**

#include<stdio.h>

#include<conio.h>

void main()

{

int range, i, j, count;

clrscr();

/\* Reading the range \*/

printf("Enter the range to print prime no's: ");

scanf("%4d", &range);

/\* printing the prime numbers \*/

for( i=1; i <= range; i++)

{

count = 0;

for( j=1; j <= (i/2); j++)

{

if( i%j == 0 )

{

count++;

if(count>1)

break;

}

}

if( count == 1 )

printf("%4d", i);

}

getch();

}

**3. jump statements:**

These statements are used to skip the block of/ lines of code.

a. break

b. continue

c. return

d. goto

a. break

this statement is used in the block of statements to transfer the control to the end of the block.

syntax:

break;

for(i=0;i<10;i++)

{

if(i==5)

break;

printf("%d",i);

}

**/\* program to understand break statement \*/**

void main()

{

int i;

clrscr();

printf("the values from 1 to 10:\n");

for(i=1;i<=10;i++)

{

if( i == 5 )

break;

printf("%-3d",i);

}

getch();

}

**b. continue**

this statement is used in the loops to continue with the current iteration by skipping the statement after it.

syntax:

continue;

Eg:

for(i=0;i<10;i++)

{

if(i==5)

continue;

printf("%d",i);

}

0 1 2 3 4 6 7 8 9

**c. return**

this statement is used in functions which indicates the returning of the function(any value/any expression).

syntax:

return (expression);

Eg:

return 3;

return (a+5);

return (a+8/9\*r);

d. goto

this statement is used to make the control jump to the specified label.

label: it is the name of the code set.

syntax:

label1:

statements....;

if(condition)

goto label2;

goto label1;

label2:

statements...;

/\* program to understand the usage of goto \*/

#include<stdio.h>

#include<conio.h>

void main()

{

int i = 4;

clrscr();

L1:

printf("From lable One...\n");

printf("....");

if(i>2)

{

i--;

goto L1;

}

else

{

printf("Going to Lable Two...\n");

goto L2;

}

printf("This will not be executed...\n");

L2:

printf("From Lable Two...\n");

getch();

}