PROGRAMMNING IN C

- Programming Language
- Types of Programming Languages
- History of C Language
- Features of C Lang. (why prefer C)
- General Steps of Programming
- How to write & execute a Program in C
- Various Parts / Sections of a C Program

Programming Language

- We can not communicate with a computer by using any of the human language like English, Hindi, Urdu etc.
- For communicating with a computer we have to speak a language that the computer can understand
- To describe the notation of a problem to a computer we have to describe the problem into a set of commands that the computer can understand and execute.

- To process a particular set of data computer must be given an appropriate set of instructions which is known as a program.
- This program must be written in a language that computer can understand.
- Thus a programming language is a systematic notation by which we describe a computational processes to a computer.

Types of the Programming Language

There are four types of Programming language.

1. Machine Level or 1st Generation
2. Assembly Level or 2nd Generation
3. High Level or 3rd Generation
4. Fourth Generation Language

Machine Level or 1st Generation Language

- This language is the language of binary numbers i.e. 0 & 1.
- Native Language to the Computer
- Execute very fast
- Writing program is a very tedious task
- Machine Dependent
- Non Portable
- Do not require any conversion software

Assembly or 2nd Generation Language

- This language is the language of mnemonics.
- Certain commonly performed operations of the computer can be represented by mnemonics.
- For example we have the ADD, SUBTRACT,DIV,MUL mnemonics for performing the Addition, subtraction, multiplication & division operation.

- For example when we want to perform the addition of two numbers we just use the ADD mnemonics between two variables A & B just like as shown below

  ADD A, B
- Writing program is a little bit easier than machine level language
- Require a conversion software known as Assembler.

- Assembler is a program which converts a program written in assembly language to machine language.
- Machine Dependent
- Non Portable
- Execute slower than machine language program
- Examples are assembly language of 8085,8086 etc.

High Level or 3rd Generation Lang.

- These languages are the language of English like structure
- Writing program is easy (Problem Oriented)
- Machine independent
- Portable
- Require conversion software known as compiler or interpreter.
- Examples are Fortran, Pascal, Cobol ,C etc.

- **Compiler & interpreter** are the programs which convert a program written in high level language to the machine language.

Difference between compiler & Interpreter

- **Compiler** converts the entire program at a time in the machine language and if there is any error in writing the program then report it to the programmer.

- **Interpreter** converts the program **line by line** in to the machine language and If there is any error in that line then report it to the user.
- For example the programming language like **C, Pascal use Compiler.**
- Programming language like **Visual Basic, DBASE use Interpreter** to convert their written program into the machine language

4th Fourth Generation Language

- These languages are the language of menus & graphics.
- Very easy to make programs
- Languages for the beginner
- Machine independent
- Portable
- Require conversion software known as compiler or interpreter.
- Examples are FoxPro, DBASE etc.

History of the C Language

- C was created by Dennis Ritchie at the Bell Telephone Laboratories in New Jersey, USA in 1972.
- The language was written to write the UNIIX operating system (A Multi-user Operating System developed by Ken Thompson at Bell Lab, U.S.A.).
- C is equally good for system as well as application programming.

- C is such a powerful and flexible language, its use quickly spread beyond Bell Labs. Programmers everywhere began using it to write all sorts of programs.
- Different organizations began utilizing their own versions of C, and subtle differences between implementations started to pose a serious problem in front of the system developers.
- In response to this problem, the American National Standard Institute (ANSI) formed a committee in 1983 to establish a standard version of C

- This committee approved a version of C in 1989 which was known as ANSI C89.
- This version was approved by the International Standard Organization (ISO) in 1990.
- This standard was further updated in 1999 and became C99.

Now, what about the name?

- The C language is so named because its predecessor was called B.
- The B language was developed by Ken Thompson of Bell Labs in 1970.
- The major limitation of B was that it was too specific and could deal with only specific problems.
- Dennis Ritchie modified and improved this language 'B' inheriting all the features of 'B' and added some new features and named the newly formed language C.

Why Prefer C?

- In today's world of computer programming, there are many high-level languages to choose from, such as C, Pascal, BASIC, and Cobol. These are all excellent languages suited for most programming tasks. Even so, there are several reasons why many computer professionals feel that C is at the top of the list. Some of them are

1. C is a powerful and flexible language
2. C is a middle level language
3. C is a portable language.
4. C is a language of few words, containing only a handful of terms, known as keywords.
5. C is a modular language. Its code can be written in routines called functions

General Steps of Programming

- When we solve a problem we should take certain steps . These are
  1. **Define** the problem.
  2. **Devise a plan** to fix it.
  3. **Create** the plan
  4. **Implement** the plan.
  5. **Test** the plan.
  6. **Debug** the plan

This same logic can be applied to many other areas, including programming.

Program Development Cycle in C

The Program Development Cycle is the cycle we have to follow for creating a program. It has the following steps.

1. Creation of the Source Code using an editor.
2. Compilation of the source code to create an object file.
3. Linking of the compiled code to create an executable file.
4. Running the program
5. Testing & Debugging of the program.

How to Write & Execute a C Program

For writing and executing a C program we follow the following steps.

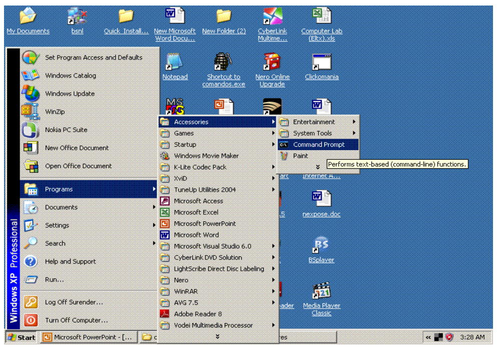
1. First of all we go to the dos prompt.
2. Type CD\TC on the DOS prompt for going to the Turbo C directory i.e. C:\TC
3. Again type TC to open Turbo C IDE.
4. Select File | New option from the IDE to create a new program and then type the source program

5. Save the program by selecting File | Save option from the IDE or press F2 key.
6. Compile the program by selecting Compile | Compile from the IDE or press Alt + F9 key .
7. Execute program by selecting Run | Run option from IDE or press Ctrl + F9 key.
8. For seeing the result of a program select Window | User Screen option from the IDE or press F5 key.

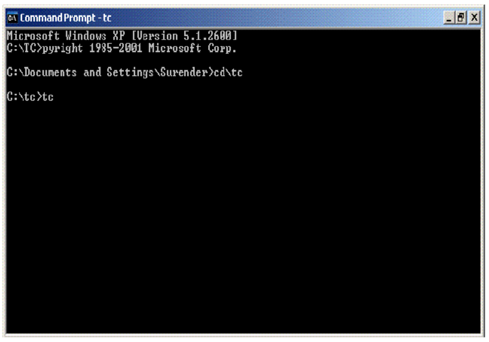
9. For opening an already saved program select FILE | OPEN option from the IDE or press F3 key.
10. For cut, copy and paste in the IDE use the following key combination

Cut	-	Shift + Del
Copy	-	Ctrl + Insert
Paste	-	Shift + Insert
Undo	-	Alt + Backspace

11. For Exiting from the IDE either select FILE | Quit option from the IDE or press Alt + X key.

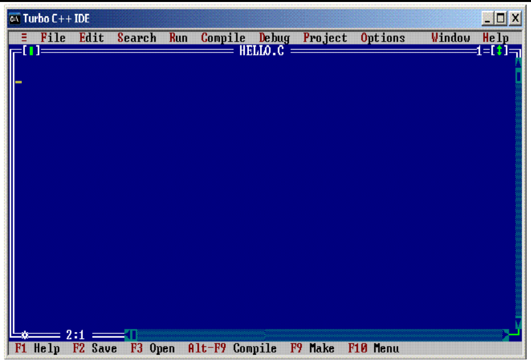


Steps for opening the dos prompt in Windows XP



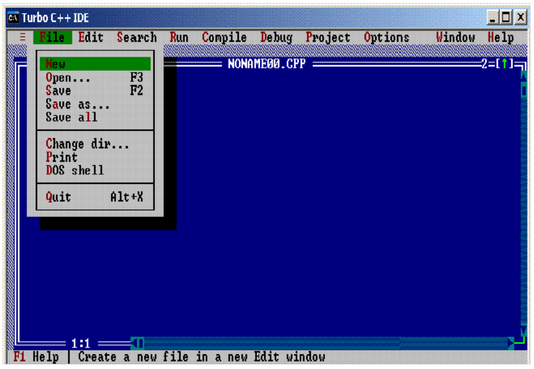
```
Microsoft Windows XP [Version 5.1.2600]
C:\TC>pyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Surender>cd\tc

C:\tc>tc
```

Steps for Starting Turbo C


Turbo C++ IDE
≡ File Edit Search Run Compile Debug Project Options Window Help
HELLO.C
1
2:1

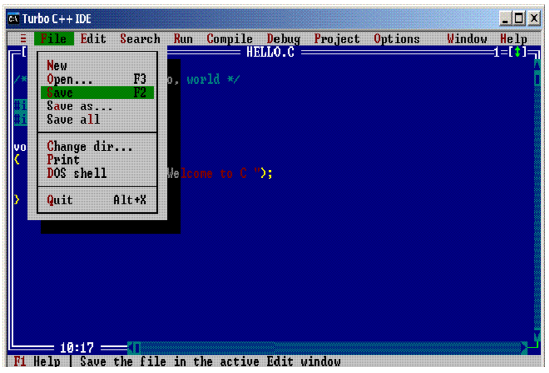
```
F1 Help  F2 Save  F3 Open  Alt-F9 Compile  F9 Make  F10 Menu
```

IDE of C Language



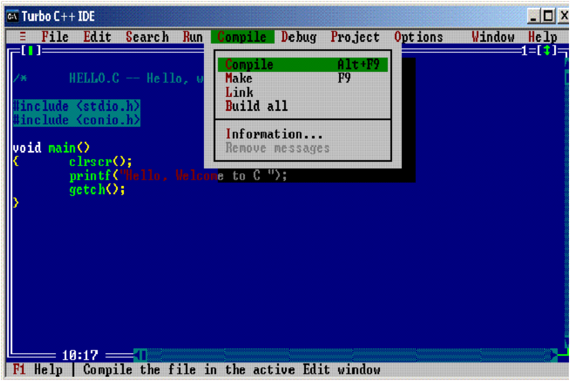
```
F1 Help | Create a new file in a new Edit window
```

Writing a New program in C



```
F1 Help | Save the file in the active Edit window
```

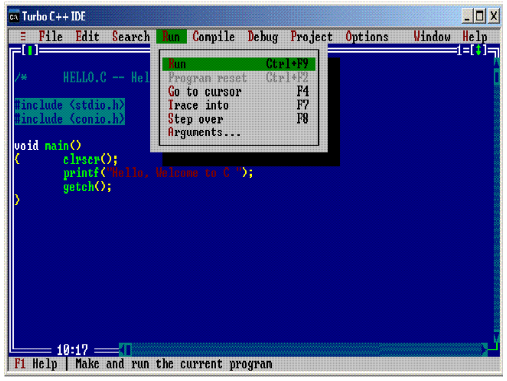
Saving a C program



```
/*      HELLO.C -- Hello, w

#include <stdio.h>
#include <conio.h>

void main()
{       clrscr();
        printf("Hello, Welcome to C ");
        getch();
}
```

```
F1 Help | Compile the file in the active Edit window
```

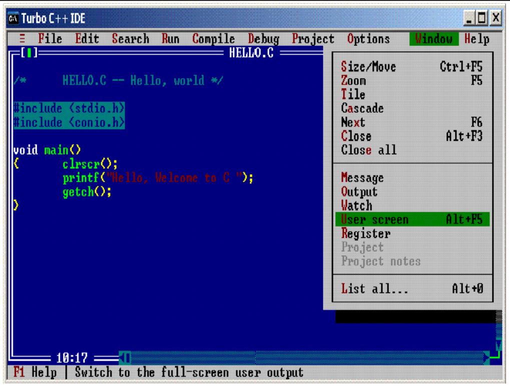
Compiling a C program



```
/*      HELLO.C -- Hel

#include <stdio.h>
#include <conio.h>

void main()
{       clrscr();
        printf("Hello, Welcome to C ");
        getch();
}
```

F1 Help | Make and run the current program

Running a C program



```
/*      HELLO.C -- Hello, world */

#include <stdio.h>
#include <conio.h>

void main()
{       clrscr();
        printf("Hello, Welcome to C ");
        getch();
}
```

F1 Help | Switch to the full-screen user output

Seeing the result of a C program

Documentation Section
Link Section
Definition Section
Global Declaration Section

main() function
{
Local Variable Declaration;
Executable Statements;
}

User Defined Functions

Basic Structure of A C Program

Documentation Section

- Documentation section consists of a set of comment lines giving name of the program, name of the author and other details which the user would like to use latter to debug the program easily.
- For giving the comments in a C program we will use the “/* comment   */” symbol.
- For example for writing the program name we may give the following comment line.

  /* Program to add two number */

Link section

- Link section provides instructions to the compiler for linking functions from the system library. For this we will use the #include preprocessor directive. e.g.

  #include <stdio.h>

Definition section

- Definition section defines all symbolic constants used in the program.
- For this we will use #define preprocessor directive e.g.

  #define PI 3.142

Global Declaration

- Global declaration section defines the variables that are used in more than one functions .This section also declares all the user defined functions.

main() function

- Every C program must have a main() function section.
- This section contains two parts : declaration part and executable part.

User defined

- User defined section contains all the user defined functions

Library Functions

- Library functions are those functions that perform the commonly used operations or calculations.
- Library functions are grouped together according to their functionality and stored in different files known as header files.
- For example all the standard I/O operations are defined in the <stdio.h> header file.
- Similarly all the mathematical functions are defined in the <math.h> header file.

- A library function is accessed simply by writing the function name followed by a list of arguments that represent information being passed to the function.
- If we want to access the functions stored in the C library then it is necessary to tell the Compiler about the files in which they are defined.
- This can be achieved by using the following commands

  #include <filename.h>   or

  #include " filename.h"

Commonly Used Library Functions

Function	Type	Purpose
abs(i)	int	Return the abs. value of i
fabs(d)	double	Return the abs. value of d
exp(d)	double	Raise e to power d
log(d)	double	Return natural log of d
pow(d1,d2)	double	Return d1 raised to d2
sin(d)	double	Return the Sine of D
getch()	int	Enter a character from standard input device

Rules for writing a C Program

- C is a procedural programming language in which problem is divided into modules or functions. Therefore to write a C program first of all we create functions and then put them together.
- One of the function must be called main(). This function will provide an environment for calling all other functions from a single place.
- Some of the functions are written by users and many others are already stored in the C library

---

- Functions in C Consists of one or more block of statements which are enclosed in a pair of curly braces ' { ' and ' } '.
- To write a C program we should use small letters.
- Blank space may be inserted between two words to improve the readability of the program. However no blank spaces are allowed with in a variable, constant & keyword.
- C has no specific rules for the position at which a statement is to be written. That' why it is called a free form language.

- Mostly statements in C are ended with a semicolon ' ; ' . However there are some exceptions to this rule.

A Sample C program

```
/*   Sample.C  */
 #include <stdio.h>
 #include <conio.h>
main()
{
clrscr();
printf(“ Welcome to C  Programming");
getch();
}
```

Run :   Welcome to C Programming

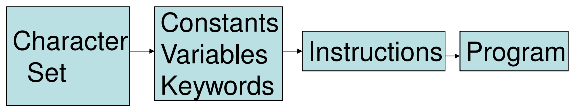
Exercise

1. Programming exercise on executing a C program.
2. Programming exercise on editing a C program.
3. Write a program in C to print the word “Hello”.
4. Write a program in C to print the sentence “Computer Programming and Applications”.

Data Types & Operators

Getting Started with C

- To learn the C language first of all we have to learn the character set used in the language
- Then we use these symbols to construct various program elements like constants , variables and keywords.



Steps In Learning C Language

C Character Set

- Character set denotes any alphabet, digit or special symbol used to represent the information. The characters in C are grouped into the following categories:

1. Letters	Uppercase	A…..Z
	Lowercase	a…...z
2. Digits	0,1,2,3,4,5,6,7,8,9	

3. Special Characters like

   ( ) { } [] : ; " ' < > , . ? / + - / * & ^ % @ !

4. White Spaces Like

- Blank Spaces
- Horizontal Tab
- Vertical Tab
- Carriage Return
- New Line
- Form Feed

C Tokens

- In a passage of text individual words and punctuation marks are called tokens.
- Similarly in a C program the smallest individual units are known as C tokens.
- C has six types of tokens
  1. Keywords or Reserve words
  2. Identifiers.
  3. Constants
  4. Strings
  5. Operators
  6. Special Symbols like [] {} etc.

Keywords or Reserve words

- Keywords are those words whose meaning is already explained to the C compiler.
- The keywords can not be used to define other program elements like variables, constants etc.
- If we do so we are trying to assign a new meaning to the keyword which is not allowed by the compiler
- There are only 32 keywords available in C.
- All keywords must be written in lowercase. e.g. auto,break,case etc.

Identifier

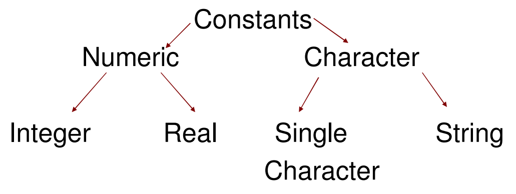
- Identifier refers to the various program elements like variables, functions and arrays.
- These are user-defined names and consist of a sequence of letters and digits, with letter as a first character.
- Both uppercase and lowercase letters are permitted although lowercase letters are commonly used.
- The underscore (_) letter is also permitted.

Rules for Constructing Identifier

1. First character must be an alphabet or underscore.
2. Must consist of only letters, digits or underscore.
3. Only first 32 characters are significant.
4. Can not use a keyword.
5. No commas or blank spaces are allowed with in an identifier.
6. No special symbol other than an underscore can be used in an identifier.

Constant

- Constants in C refer to fixed values that do not change during the execution of a program .
- C has four basic types of constants which are further divided into two categories as shown below



Integer Constant

- An integer constants refer to a sequence of digits.
- There are three types of integers constant namely **decimal, octal and hexadecimal** integer constant.
- A **decimal integer constant** consists of a set of digits 0 through 9, preceded by an optional – or + sign.
- Valid examples of decimal integers are : 123 , -321 , +6534.

- Embedded spaces, commas and non-digit characters are not permitted between digits.
- For example

  16 789 , 30.567 , $2345

  are illegal numbers.
- An **octal integer constant** consists of any combination of digits from the set 0 through 7, with a leading 0.
- For example

  067, 01, 04356 , 0567

- An **hexadecimal integer** constant must begin with either 0x or 0X.
- It is then followed by any combination of digits taken from the sets 0 through 9 and alphabets a through f (either uppercase or lowercase).
- The letters a through f (or A through F) represent the decimal quantities from 10 to 15.
- For example

  0x1 , 0x1235, 0xacd, 0x2345,0X1235.

- The magnitude of the integer constant is machine dependent.
- A typical maximum value for **16** bit machine is **32767** decimal (equivalent to **77777** octal or **7FFF** hexadecimal) which is **2^{15} - 1**.
- For **32** bit machine this value is **2147483647** (which is $2^{31} - 1$).
- Thus if a particular computer uses a w bit word then an ordinary integer quantity may fall within the range **-2^{w-1}** to +**2^{w-1} -1**

Real Constant

- Integer numbers are inadequate to represent quantities that very continuously, such as distances, heights , temperatures, prices.
- These quantities are represented by fractional parts like 15.789. Such numbers are called real or floating point constants.
- A real number or a floating point number can be represented in two notations.
- Decimal notation & Exponential or Scientific Constant.

- In Decimal notation a whole number is followed by a decimal notation and the fractional part. It is possible to omit digits before the decimal point or digits after the decimal point.
- Examples of this type are

  215.23, 324., -0.97 , +0.8
- A real number can also be represented in exponential or scientific notation.
- The general form of this notation is

  mantissa e exponent

- The mantissa is either a real number expressed in decimal notation or an integer.
- The exponent is an integer number with an optional plus or minus sign.
- The letter e separating mantissa and exponent can be written in either lowercase or uppercase.
- Examples of legal floating point constants are 0.65e4 , 12e-2 , 1.5e+5, -1.2E-1

Single Character Constant

- A single character constant or simply a character constant contains a single character enclosed within a pair of single quote marks.
- Examples are '5' 'x' 'C'

String Constant

- A string constant is a sequence of characters enclosed in double quotes.
- The characters may be letters, numbers or any special characters and blank spaces.
- Examples are

  "Hello Friend" , "2007 " , "X"

Backslash Character Constant

- Certain non printing characters such as new line , bell alert and tabs are represented by special backslash character constants.
- The character combination used for representing these non printing characters are known as escape sequences.
- For example the symbol '\n' stands for new line character.
- Examples of other commonly used escape sequences are as follows

Constant	Meaning
\a	Alarm
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab
\'	Single Quotes
\""	Double Quotes

Variable

- A variable is an entity that may change during the execution of a program.

  or

- A variable is a named location in memory that is used to hold a value that can be modified by the program.
- All variables must be declared before they are used in the program
- The general form of a variable declaration is
- Datatype var name1,… var nameN;

- Here, data type must be a valid data type and variable name is the name of the variables separated by commas.
- Some valid declarations are :

  int length;

  float area;

  char ch;

Invalid examples of variables

123

(area)

@25th

Data Type

- Data type defined what is the type of the variables or constants which we are using in our program.
- In C there are two types of data type available.
  1. Basic Data Type
  2. Abstract Data Type

Basic Data Type

- The data types that are built into the C language are known as basic data type.
- C defines five basic data types:
  1. Character
  2. Integer
  3. Floating-point (Single precision)
  4. Double floating-point (Double precision)
  5. Valueless or Void
- These data types are declared by using char, int, float, double, and void.

- Except type **void** all other basic data types may have various modifiers preceding them.
- A type modifier alters the meaning of the basic data type more precisely to fit a specific need. The list of modifiers are
  1. Short
  2. Long
  3. Signed
  4. Unsigned
- The **int** type can be modified by **short**, **long**, **signed**, and **unsigned**.

- The **char** type can be modified by **signed** and **unsigned.**
- Double can be modified by **long.**
- The size of the data types depends on the type of computer.
- Size, memory requirements and range of the various data types is as follows

Type	Size(Bytes)	Range
Int	2	-32768 to 32767
Short Int	2	-32768 to 32767

Type	Size(Bytes)	Range
Unsigned Int	2	0 to 65,535
Long signed Int	4	-2147483648 to +2147483648
Long unsigned Int	4	0 to 4294967295
Char	1	–128 to 127
Signed char	1	-128 to 127
Float	4	3.4e-38 to 3.4e+38
Double	8	1.7e-308 to 1.7e+308
Long Double	10	3.4e-4932 to 1.1e+4932

Abstract Data Type

- The data types based on the basic data types are known as abstract data type.
- Examples of abstract data types are
  1. Array
  2. String
  3. Structure
  4. Union
  5. Pointer

Operator

- An operator is a symbol that tells the computer to perform certain mathematical or logical operations.
- Operators are used to manipulate data and variables.
- C operators can be classified into the following eight categories.
  1. Arithmetic Operators
  2. Relational Operators
  3. Logical Operators
  4. Assignment Operators
  5. Compound Assignment
  6. Increment & Decrement
  7. Conditional Operators
  8. Special Operators

Arithmetic Operators

- Arithmetic Operators are used to perform the arithmetic operations.
- C has the five arithmetic operators
- They are +, - , *, / , %.

- The table below shows the various arithmetic operators and their meaning in C.

Operator	Meaning
+	Addition or Unary Plus
-	Subtraction or Unary minus
*	Multiplication
/	Division
%	Remainder after integer Division

- The operators +, –, *, and / work as they do in most other computer languages. You can apply them to almost any built-in data type. i.e. integer, character, float, double.
- When you apply ‘ / ’ to an integer or character, any remainder will be truncated. For example, 5/2 will equal 2 in integer division.
- The modulus operator % gives the the remainder after division. However, we can use it only with integer type values.

- Examples of use of arithmetic operators are
  a-b , a + b , a*b , a/b , a%b
- Here a and b are variables and are known as operands.

Integer arithmetic

When both the operands in an arithmetic operation are integers then the expression is called an integer expression and the operation is called integer arithmetic

- If a = 14, and b = 12 then
  a + b = 26 , a –b = 2, a*b = 168
  a/b = 1 , a %b = 2.

- During Integer division if both the operands are of the same sign the result is truncated towards zero. For example
  6 / 7 = 0 or -6 /-7 = 0

but -6 / 7 may be zero or -1 (Machine dependent)

Real Arithmetic

- An arithmetic operation involving only real operands is called real arithmetic.
- A real operand may assume values either in decimal or exponential notation. For example
  x = 6.0/7.0 = 0.857243
  y = 1.0/3.0 = 0.33333

Relational Operators

- Relational operators are used to compare two quantities.
- C supports six relational operators.

Operator	Action
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
= =	Equal
!=	Not equal

- An expression using a relational operator is known as relational expressions. Such as
  a < b or 1 <20
- The value of a relation expression is either one or zero.
- If the relation is true then it is 1 and 0 if the relation is false.

Logical operators

- Logical operators are used to perform the logical operations.
- C supports three logical operators : AND , OR & NOT.

Logical Operators	Operator Action
&&	AND
| |	OR
!	NOT

- Logical operators also have true or false value which is represented by 1 & 0.
- Logical AND operator results in 1 if both the operands have non zero values
- Logical OR operator have 1 value if either of the operands have 1 value.
- Not operator negates the value of the operand.

Assignment Operators

- Assignment operator are used to assign the result of an expression to a variable
- The symbol ' = " is used to represent the assignment operator.
- Assignment operator assigned the right hand side value to the variable on the left hand side of the operator. For example

```
x = 56;
x = x +1
d = (a*b)/45 ;
```

Compound Assignments

- There is a variation of the assignment operator called compound assignment operator which simplifies the coding of a certain type of assignment operations.
- Compound assignment operators exist for all the binary operators (those that require two operands). For example + , - , * , / , % etc.
- In general, statements like

  var = var operator expression can be written as

  var operator = expression

- For example,

  x = x+10;   can be written as

  x += 10;
- The operator **+=** tells the compiler to assign to **x** the value of **x** plus 10.
- For another example,

  x = x-100;   is the same as

  x -= 100;

  And similarly   x = x * 123 is the same as

  x *= 123   and

  x = x /123 is same as

  x /= 123

Multiple Assignment

- We can assign many variables the same value by using multiple assignments in a single statement.
- For example, this program fragment assigns **x**, **y**, and **z** the value 0:

  x = y = z = 0;

Increment and Decrement Operators

- C includes two operators that simplify two common operations. These are increment (++) and decrement (--)operators.
- The operator **++** adds 1 to its operand, and

  -- subtracts 1 from its operand. In other words:

  x = x+1   is the same as   ++x   and

  x = x–1   is the same as   x--
- Increment and decrement operators have two forms **prefix** and **postfix.**

- In prefix form the operator precedes the operand i.e. ++x , --x .
- In postfix form the operator follows the operand i.e. x++ , x--.
- When we use ++ and -- operator with a single operand then there is no difference between prefix and postfix form.

  i.e.   ++x

  and   x++

  both equal to x = x+1

- However, there is, a difference between the prefix and postfix forms when we use these forms in a expression.
- In prefix form the operand will be altered in value before it is utilized. For instance

```
x = 10;
y = ++x;
```

Sets **y** and x to 11

- In postfix form the operand will be altered in value after it is utilized. For instance

```
x = 10;
y = x++;
```

Sets **y** to 10 and x to 11

Conditional Operator

- C contains a powerful and convenient operator that replaces certain statements of the if-then-else form. This is conditional or ternary operator.
- The conditional or ternary operator ? : have the following general form
- Exp1 ? Exp2: Exp3;

  where Exp1, Exp2, and Exp3 are expressions.
- The **? :** operator works like this: Exp1 is evaluated. If it is true, Exp2 is evaluated and becomes the value of the expression. If Exp1 is false, Exp3 is evaluated, and its value becomes the value of the expression.

- For example, if x = 10 and

  y = x>9 ? 100 : 200;

  y is assigned the value 100. If x is greater than 9 otherwise **y** will have the value 200.

Comma Operator

- Comma operator strings together several expressions.
- The left side of the comma operator is always evaluated as **void**. This means that the expression on the right side becomes the value of the total comma-separated expression.

For example, x = (y=3, y+1);

- first assigns **y** the value 3 and then assigns **x** the value 4. The parentheses are necessary

Sizeof Operator

- **sizeof** is a unary compile-time operator that returns the length, in bytes, of the variable or parenthesized type specifier which it precedes. For example

  int d;

  sizeof(d) it will return 2 because integer will take 2 bytes space in memory.

Precedence and Associativity

- The priority according to which various operations are performed in an expression is called precedence of operators.
- The order of evaluation of the operators when they have equal priority is called associativity of the operators.
- The parentheses () has the highest priority among various operators and comma operator has the lowest priority.
- Only Unary, Assignment and Compound assignment operators have associativity from right to left all others have left to right.

Implicit Type Conversion

- When we perform an operation between two different data types constants and variables C automatically convert them to the proper type so that the expression can be evaluated without any difficulty.
- If the operands are of different data types the lower data type is automatically converted to the higher type before the operation precedes.
- If one of the operand is int and other is double than int operand is automatically converted to the float type to evaluate expression.

Type Conversion in Assignment

- In an assignment statement, the type conversion rule is easy: The value of the right hand side of the assignment is converted to the type of the left side (target variable), as illustrated here:

```
int x = 20, y;
float f , z = 34.5
f = x      y = z;
```

Now here f is 20.0 and y is 34.

Explicit Type Conversion

- We can force an expression to be of a specific data type by using a type cast or explicit conversion. The general form of a type cast is

  (type) expression

  where *type* is a valid data type.
- For example
  1. x = (int) 8.5   8.5 is converted to integer
  2. (float) x/2 will be evaluated as float exp.
  3. a = (int) 21.3/(int)4.5 will be evaluated as 21/4 and the result would be 5.

C Instruction

- An instruction is a combination of the variables, constants,keywords,operators according to the rules of the language.
- An instruction tells the computer what operations should be performed by the computer.
- C has basically four types of Instructions.
  1. Type Declaration Instruction
  2. Input/Output Instruction
  3. Arithmetic Instruction
  4. Control Instruction

Exercise

1. Write a program which defines various type of variables and assign values to them.
2. Write a program which shows the use of arithmetic operators.
3. Write a program to find the sum and average of three numbers.
4. Write a program to find the simple interest of an amount.
5. Write a program to find the area of a circle.

Input & Output Functions

Input & Output Functions

- Input and Output functions are used to accept values for various variables and displaying them after processing is over.
- Input and Output of data can be made through standard Input Output devices.
- The C language does not define any keywords that perform I/O.
- Input and output are accomplished through standard library functions.
- The header files for the I/O functions is <stdio.h >

- The library functions available for standard Input/Output are of two types
  1. Formatted I/O
  2. Unformatted I/O

Formatted Input

- Formatted input can read data in various formats that are under our control.
- For example consider the following data
  12.56 123 Sunil
- Formatted input is completed by using scanf function of the stdio.h header file.

- Scanf can read all the built-in data types and automatically convert numbers into the proper internal format.
- The general format of scanf is
  scanf(“Control string”,arg1,arg2……argn);
- The Control String refers to a string containing certain required formatting information for the input data.
- The arguments arg1,arg2,….. argn represent the individual input data item.

- The control string consists of individual groups of characters with one character group for each input data item.
- In the simplest form each character group consists of the percent sign (%) followed by a conversion character which indicates the type of the corresponding data item.
- In the control string multiple character groups can be continuous or they can be separated by blank spaces, tabs or new line characters.

- In control string argument names must be preceded with an ampersand sign.
- Frequently used conversion characters for scanf function is given below.

Character	Meaning
%c	Character.
%d	Signed decimal integers.
%u	Unsigned Decimal Integers
%ld	Signed long integer
%lu	Unsigned Long Integer
%i	Signed decimal integers
%e	Scientific notation (lowercase e)

Character	**Meaning**
%E	Scientific notation (uppercase E)
%f	Decimal floating point
%s	String of characters.
%x	Unsigned hexadecimal (lowercase letters).
%X	Unsigned hexadecimal (uppercase letters).
%o	Unsigned octal.
%g	Decimal floating point.
%G	Decimal floating point.

Inputting Integer Numbers

- The field specifications for reading an integer number is % wd.
- The percentage sign (%)indicates a conversion specifications.
- W is an integer number that defines the field width of the number to be read.
- D denotes the data type character that shows number to be read is the integer number.
- For example

  scanf (%3d%5d", &num1,&num2);

- if data line is 45 30567

  Then the value 45 will be assigned to num1 and 30567 will be assigned to num2.
- If data line is 234 233765
- Then variable num1 will be assigned value 23 because of %2d and num2 will be assigned value 4 which is not as per our expectations.
- To get rid of this type of problem we specify the field specifications without any field width.

- Thus if we write the previous statement in the form i.e.

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

- Then if data line is 453 30562

  Then the value 453 will be assigned to num1 and 30562 will be assigned to num2.

Inputting Real Numbers

- Scanf can read the real numbers using the simple specification %f for both the notations namely decimal point notation and exponential notation.
- For real numbers the field width need not to specify.
- For example

  scanf(“ %f%f%f”,&x,&y,&z)

  with the input data

  47.45 ,43.32e-1 , 678 will assign the value

  x = 47.45 , y = 4.332 and z = 678.0

Inputting Character & String

- General format for reading a character from the keyboard by using scanf function is %c
- For example

  scanf(“%c%c”, &name1,&name2)

  with input data A B then A will be assigned to name1 and B to name2.
- General format for reading a string from the keyboard by using scanf function is %s
- For reading strings through scanf function the string variables need not to be preceded by ampersand sign (&).

- For example

  scanf(“%s%s”,name1, name2)

  with input data line sunil mohan

  then sunil will be assigned to name1 and mohan will be assigned to name2.

Reading Mixed Mode Data

- One scanf statement can be used to input a data line containing a mixed mode data.
- For reading mixed mode data the input data items must match the control specifications in order and type.
- When an attempt is made to read an item that does not match the expected type scanf function does not read any further and immediately returns the values read.
- For example consider the following scanf

  scanf(“%d%c%f%s”,&a,&b,&ratio,&name);

will read an integer, a character , a floating point and a string variable from the keyboard.

- If the input data is

  15 e 12.45 sunil

  then 15 will be assigned to a ,

  e will be assigned to b,

  12.45 will be assigned to ratio

  and sunil will be assigned to name.

Rules for Scanf Functions

- Each variable to be read must have a character group.
- For each character group there must be a variable address of proper type.
- Any blank space, tab and new line in the format string must have a matching character in the user input.

Formatted Output

- Formatted output can display data in various formats that are under our control.
- For example consider the following data

  15 123.34 Sunil

- Formatted output is completed by using printf function of the <stdio.h> header file.
- printf function can display all the built-in data types and automatically convert numbers in to the proper internal format.

- The general format of printf function is

  printf(“Control string”,arg1,arg2……argn);

- The Control String refers to a string containing certain required formatting information.
- The arguments arg1,arg2,….. argn represent the individual output data item.
- The control string consists of individual groups of characters with one character group for each output data item.

- In the simplest form each character group consists of a percent sign (%) followed by a conversion character which indicates the type of the corresponding data item.
- In the control string multiple character groups can be continuous or they can be separated by blank spaces, tabs or new line characters.
- In control string of printf function argument names need not be preceded with an ampersand sign (&).

- Frequently used conversion characters for printf function is given below.

Character	**Meaning**
%c	Character.
%d	Signed decimal integers.
%u	Unsigned Decimal Integers
%ld	Signed long integer
%lu	Unsigned Long Integer
%i	Signed decimal integers
%e or %E	Scientific floating point
%f	Decimal floating point

Character	**Meaning**
%s	String of characters.
%x or %X	Hexadecimal numbers
%o	Unsigned octal.
%g or %G	Decimal floating point

Outputting Integer Numbers

- The field specifications for displaying an integer number is % wd.
- The percentage sign (%)indicates a conversion specifications.
- W is an integer number that defines the field width of the number to be displayed.
- D denotes the data type character that the number to be display is the integer number.
- For example

  printf (%3d%5d", num1,num2);

- If data line is 45 30567

  Then the value _45 will be displayed for num1 and 30567 will be displayed for num2.
- If data line is 234 3045
- Then variable num1 will be assigned value 234 because of %3d and num2 will be assigned value _3045 because of %5d.

Outputting Real Numbers

- General format for displaying real number is % wf.
- W is the field width of the data item.
- Printf can read the real numbers using the simple specification %f for decimal notations and %e for exponent notation.
- For example

  printf(" %f%e%f",&x,&y)

  with the display data

  47.45 ,43.32

  x = 47.450000 , y = 4.332000e+01

- Similarly if we want to define the number of decimal places & width of the data item we can do this in the way as shown below

  if data item to display is

  x = 125.12 and y = 3465.78 then

  printf(" x = %5.2f \n y = 6.2%",x ,y) will display

  x = 123.12 instead of 123.120000

  y = 3465.78 instead of 3465.780000

Outputting Character & Strings

- General format for displaying a character by using printf function

  is   %c
- For example

  printf("%c%c", name1,name2)

  with output data A & B then A will be displayed for name1 and B for name2.
- General format for displaying a string by using the printf function

  is   %s
- For example

  printf("%s%s", name1,name2)

- with output data sunil and mohan

  then sunil will be displayed for name1

  and mohan will be displayed for name2.

Displaying Mixed Mode Data

- One printf statement can be used to output a data line containing a mixed mode data.
- For displaying mixed mode data the output data items must match the control specifications in order and type.
- When an attempt is made to display an item that does not match the expected type printf function does not display the data item in the proper way.
- For example consider the following printf

  printf("%d%c%f%s",a,b,ratio,name);

will display an integer, a character , a floating point and a string variable.

- If the output data is

  15 12.45 and

  printf("a =%d\n b =%.2f);

  then it will be displayed as

  a = 15

  b = 12.45

Unformatted Input

- Unformatted input functions data can not be formatted by the user. Instead it will display the data in the standard format.
- There are four types of unformatted input functions.

1. Getch()	For entering a charaacter
2. Getche()	For entering a charaacter
3. Getchar()	For entering a charaacter
4. Gets()	For entering a string

Getch()

- The **getch( )** function waits for a key press after which it returns immediately.
- Need not to press the enter key for digesting the character.
- It does not echo the character on the screen.
- For example ch = getch() or getch()

Getche()

- The **getche( )** function is the same as **getch** but the key is echoed on the screen.
- Need not to press the enter key for digesting the character.
- For example ch = getche()

Getchar()

- The **getchar( )** function waits until a key is pressed and then returns its value.
- The key pressed is also automatically echoed on the screen
- We have to press the enter key after the character so that it will be digested by the corresponding character variable.
- For example ch = getchar()

Gets()

- The **gets( )** function reads a string of characters entered at the keyboard and stores them at the address pointed to by its argument.
- You can type characters from the keyboard until you strike a carriage return.
- The carriage return does not become part of the string. Instead, a null terminator is placed at the end.
- Its general format is

  gets(character string constant or var);

- For example

```
char str[80];
gets(str);
```

Unformatted Output

- Unformatted output functions display the data but data can not be formatted by the user. Instead it will display the data in the standard format
- There are three types of unformatted output functions.

1. putchar() For displaying a charaacter
2. putch() For displaying a charaacter
3. Puts() For displaying a string

Putchar()

- The **putchar( )** function writes a character to the screen at the current cursor position.
- The **putchar( )** function returns the character written or **EOF** if an error occurs.
- For example

  putchar(ch);

Putch()

- From the functioning point there is no difference between the putchar() and putch().
- For example putch(ch).

Puts

- The **puts()** function writes its string argument to the screen followed by a new line.
- Its general format is

  puts(character string constant or var);
- For example

  puts("hello");

  puts(str)

Exercise

1. Write a program to find the roots of a quadratic equation.
2. Write a program to find the compound interest of an amount.
3. Write a program which shows the reading and writing of character & string by using unformatted input output functions.
4. Write a program which find the area of a circle and a sphere.

Decision Making

True and False in C

- Many C statements rely upon a conditional expression that determines what course of action is to be taken in the program.
- A conditional expression evaluates to either a true or false value.
- In C, true is any nonzero value, including negative numbers and false is 0.
- This approach to true and false allows a wide range of routines to be coded extremely efficiently.

Selection Statement

- C supports two selection statements:
  if and switch.
- The conditional operator (? :) is an alternative to (if) in certain circumstances.

if Statement

- The general form of the if statement is

```
if (Condition)
 {
Statement Block;
 }
```

- Where a *statement* may consist of a single statement , a block of statements, or nothing (in the case of empty statement).
- If *expression* evaluates to true (anything other than 0) then the statement or block that forms the target of **if** will be executed.
- If *expression* evaluates to false then the first statement after the target will be executed.
- The conditional statement controlling **if** must produce a scalar result.

- A scalar is either an integer, character, pointer, or a floating-point type value.
- It is rare to use a floating-point number to control a conditional statement because this slows the execution time considerably.
- It takes several instructions to perform a floating point operation.
- It takes relatively few instructions to perform an integer or character operation.

- For example the following program contains an example of **if**. The program finds out a big number from two numbers.

```
#include <stdio.h>
#include <conio.h>
main()
{
  float a, b;
  clrscr();
printf(“Enter two number = “);
scanf(“%f%f”,&a,&b);
```

```
        if(a > b)
        printf("Big number is : %f",a);
        if (b >a)
         printf("Big number is : %f",b);
         getch();
      }
```

Run :

```
        Enter two numbers = 12.5 45.0
        Big number is : 45.00
```

- Write a program to calculate the HRA according to the following condition.

Basic Pay	HRA
<= 2000	200
> 2000 and <= 4000	350
> 4000	500

```
/*  Program to calculate HRA */
    #include <stdio.h>
    #include <conio.h>
    main()
    {
      int bp,hra;
```

```
    clrscr();
    printf("Enter the basic pay =");
    scanf("%d",&bp);
    If(bp <=2000)
    hra   = 200;
    if(bp >=2000 && bp <= 4000)
    hra = 350;
    if(bp > 4000)
    hra  = 500;
    printf("HRA = %d" , hra);
    }
```

Run : Enter the basic pay = 4400

```
         HRA  = 500
```

If…….Else statement

- The general form of If...Else statement is

```
    If ( Condition )
    {
      Statement Block 1 ;
    }
   else
    {
     Statement Block 2;
    }
```

- If the given condition is true then statement block1 will be executed otherwise statement block 2 will be executed.
- For example consider the previous program which finds out the largest no from two nos.

```
/*  Program to find division  */
    #include <stdio.h>
    #include <conio.h>
    main()
    {
       float a, b;
       clrscr();
```

```
    main()
    {
    float a, b;
    clrscr();
    printf("Enter two number = ");
    scanf("%f%f",&a,&b);
    if(a > b)
    printf("Big number is : %f",a);
    else
    printf("Big number is : %f",b);
    getch();
    }
```

```
Run :
    Enter two numbers = 12.5 45.0
    Big number is : 45.00
```

Nesting of If….Else Statement

- When a series of decisions are involved then we can use more than one if...else statement.
- The general form of nested if..else is as shown below

```
If (test condition -1)
 {
 Statement(s) – 1;
   if (test condition – 2)
        {
         Statement(s) – 2;
        }
        else
         {
         Statement(s) – 3;
         }
 }
     else
     {
     Statement(s) – 4;
     }
```

- Write a program to find the division obtained by a student in an examination.The student gets a division as per them following rules.

1. Marks above or equal to 60 – First
2. Marks between 50 and 59 – Second
3. Marks between 40 and 49 – Third
4. Marks less than 40 – Fail

```
/* Program to find division    */
   #include <stdio.h>
  #include <conio.h>
```

```
main()
  {
  int m1,m2,m3,m4,m5,per;
  clrscr();
  printf("Enter marks of five subjects : ");
  scanf("%d%d%d%d%d",&m1,&m2,&m3,
       &m4,&m5);
 per = ( m1 + m2 +m3 +m4 + m5) /5 ;
 if ( per >= 60)
  printf("First Division ");
  else
     {
```

```
  if (per >= 50 && per < = 59)
     printf(" Second Division ");
     else
      {
       if (per >=40 && per < = 49)
       printf(" Third Division ");
       else
          printf("Fail");
      }
    }
   getch();
   }
Run : Enter marks of five subjects :  65 56 78 90 65
         First Division
```

The Else if ladder

- Else if ladder is an another way of putting ifs together when multi path decisions are to be evaluated.
- A multi path is a chain of ifs in which the statement associated with each else is an if.
- The general format of else if ladder is

```
  if (condition 1)
     statement – 1;
  else if (condition 2)
     statement – 2;
```

```
else if (condition 3)
     statement – 3;
    else if (condition n)
     statement – n;
      else
      default statement;
      statement x.
```

- The statements are evaluated from the top to downwards.
- As soon as a true condition is found the statement associated with it is executed and after that control is transferred to the statement x.

```
/* Program to calculate division */
#include <stdio.h>
#include <conio.h>
main()
  {
  int m1,m2,m3,m4,m5,per;
  clrscr();
  printf("Enter marks of five subjects : ");
  scanf("%d%d%d%d%d",&m1,&m2,&m3,
        &m4,&m5);
 per = ( m1 + m2 +m3 +m4 + m5) /5 ;
```

```
if ( per >= 60)
printf("First Division ");
else if (per >= 40 && per <= 59)
printf(" Second Division ");
else if (per >=40 && per <= 49)
printf(" Third Division ");
else
printf("Fail");
getch();
}
```

Run: Enter marks of 5 subjects : 61 62 65 65 65
First Division

The Switch Statement

- Control statements, such as if, were limited to evaluating an expression that could have only two values: true or false.
- If we want to control a program flow based on more than two values by using "if" then we have to use multiple nested if statements.
- C's another flexible program control statement is the switch statement, which lets your program execute different statements based on an expression that can have more than two values

- The general format of the switch statement is as follows:
- switch (expression)

```
  {
 case  value 1:
 statement block 1;
 break;
case value 2 :
statement block 2;
 break;
 ……
 …….
```

```
 ….
….
case value n:
 statement block n;
 break;
 default:
 default block;
  }
```

- The *expression* must evaluate to an integer type. Thus we can use character or integer type values, but floating-point expressions are not allowed.

- The value of *expression* is tested against the values of the constants specified in the **case** statements.
- When a match is found, the statement sequence associated with that **case** is executed until the break statement or the end of the **switch** statement is reached.
- The **default** statement is executed if no matches are found.
- The **default** is optional, and if it is not present, no action takes place if all matches fail.

- C89 specifies that a **switch** can have at most 257 **case** statements.
- C99 specifies that at most 1,023 **case** statements can be included.
- For example following program shows the use of switch statement.

```
/* Demonstrates the switch statement. */
        #include <stdio.h>
        #include <conio.h>
        main()
        {
          int reply;
```

```
puts("Enter a number between 1 and 3:");
scanf("%d", &reply);
switch (reply)
{
 case 1:
 puts("You entered 1.");
 break;
case 2:
 puts("You entered 2.");
 break;
case 3:
 puts("You entered 3.");
```

```
   break;
   default:
   puts("Out of range, try again.");
   }
   getch();
  }
      Run :
      Enter a number between 1 and 5: 3
      You entered 3
```

Important things about the switch

- The **switch** differs from the **if** in that **switch** can only test for equality, whereas **if** can evaluate any type of relational or logical expression.
- No two **case** constants in the **switch** can have identical values.
- However a **switch** statement enclosed by an outer **switch** may have **case** constants that are in common.
- If character constants are used in the **switch** statement then they are automatically converted to integers

The goto Statement

- The goto statement is one of C's unconditional jump, or branching, statements .
- When a program execution reaches the goto statement, execution immediately jumps, or branches, to the other location in the program specified by the goto statement.
- This statement is unconditional because execution always branches when a goto statement is encountered.
- The branch doesn't depend on any program conditions like if statement.

- The general form of the **goto** statement is

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

- For example following program shows the use of goto statement.

```
/* Demonstrates the goto statement */
    #include <stdio.h>
    #include <conio.h>
    main()
    {
        int n;
            start: ;
puts("Enter a number between 0 and 10 ");
    scanf("%d", &n);
    if (n > 0 || n < 5 )
    goto start;
    else if (n == 0)
    goto location0;
    else if (n == 1)
    goto location1;
    else
    goto location2;
location0:
puts("You entered 0.\n");
goto end;
location1:
puts("You entered 1.\n");
goto end;
location2:
puts("You entered something between 2 and 4\n");
end: ;
}
```

Run:

```
Enter a number between 0 and 10: 1
You entered 1.
Enter a number between 0 and 10: 3
You entered something between 2 and 4.
```

Exercise

1. Write a program to check that whether a number is an even number or not.
2. Write a program to check that whether a number is a prime number or not.
3. Write a program to check that whether an year is a leap year or not.
4. Write a program to check that whether a number is a positive number, negative number or zero.
5. Write a program which will receive a character from the keyboard and check whether it is a vowel or not.

Decision Making and Looping

Iteration or Loop Statement

- In C iteration statement allows a set of particular instructions to be executed again and again until a specific condition is true.
- Iteration statement is also known as loop.
- The condition may be checked at the beginning of the loop (as in the while & for loop) or may be checked at the end of the loop (as in the do-while loop).
- There are three types of loop or iteration statements in C.

1. **While Loop**
2. **Do – while Loop**
3. **For Loop**

The while loop

- The while statement, also called the while loop, executes a block of statements as long as a specified condition is true

- The while statement has the following general form :

```
 loop Initialization ;
while (conditional statement)
   {
   statement block;
   Increment or Decrement statement ;
   }
```

- Before executing the loop a variable is initialized by using a loop initialization statement.
- Conditional statement is any C expression that can be evaluated to either true or false.

- Statement block is either a single statement or a compound C statement .
- When a program execution reaches a while statement then the following events occur :

1. First of all the loop is initialized & this can be done by assigning a value to a variable.
2. Then this assigned value is tested with a conditional statement. If it is false then the loop terminates, and control passes to the first statement after the loop.
3. If condition evaluates to true (that is, nonzero), then C statement(s) in statement block are executed.
4. With in the statement block the loop control variable is again incremented or decremented and at the end of the loop it is again tested with the conditional statement.
5. If the condition is true then the C statement(s) in statement block are executed. i.e. second iteration will start.
6. If the condition is false then the loop terminates and control passes to the first statement after the loop.

- For example Consider the following simple program that uses a while statement to print the numbers from 1 to 20.

```
/* Demonstrates a simple while statement */
    #include <stdio.h>
    #include <conio.h>
    main()
    {
     int count;
     count = 1;
```

```
while (count <= 20)
   {
    printf("%d\n", count);
     count++;
    }
  getch();
   }
```

Run :

```
        1
        2
        ..
       20
```

Nesting While Statement

- A while statement can be inserted within another while statement. This is called *nesting of while statement .*
- The general format of nested while is as follows

```
    While( condition 1)
    {
      while( condition 2)
       {
     statement(s);
    }
  }
```

- Example Demonstrate use of nested while.

```
/* Multiplication Table from 1 to 5 */
 #include <stdio.h>
#include<conio.h>
 void main()
 {
      int row  =1, column,  y;
      clrscr();
printf(" Multiplication Table from 1 to 5     \n");
printf("---------------------------------------------\n");
```

```
while(row <= 5)   /* Outer Loop Begins */
     {
 column = 1;
 while(column <= 10) /*   Inner loop begins */
     {
  y = row * column;
  printf("%4d", y);
  column = column + 1;
  }      /*... Inner Loop Ends ...*/
 printf("\n");
  row = row + 1;
  }     /*... Outer Loop Ends ...*/
```

```
printf("---------------------------------------------\n");
getch();
 }
```

Run :

```
 Multiplication Table From 1 to 5
-------------------------------------------------------
1   2    3    4    5    6    7    8    9   10
2   4    6    8   10   12   14   16   18   20
3   6    9   12   15   18   21   24   27   30
4   8   12   16   20   24   28   32   36   40
5  10   15   20   25   30   35   40   45   50
-------------------------------------------------------
```

The do-while Loop

- Unlike **while** loop, which test the loop condition at the top of the loop, the **do-while** loop checks its condition at the bottom of the loop
- The general form of the **do-while** loop is

```
loop Initialization
do
 {
statement block;
Increment or Decrement statement ;
 } while(condition);
```

- A **do-while** loop always executes at least once whether a condition is true or false.
- In do while loop the curly braces are not necessary when only one statement is present however they are commonly used to avoid confusion.
- When a program execution reaches a do...while statement, then the following events occur:

1. First of all the loop is initialized & this can be done by assigning a value to a variable.
2. Then the statements with in the statement block will be executed.
3. One of the statement with in the statement block may increment or decrement the initialized value.
4. Then this value is again tested with the conditional statement.
5. If the condition is true then the C statement(s) in statement block are executed.
6. If the condition is false then the loop terminates and the control passes to the first statement after the loop.

- The do...while loop is used less frequently than while and for loops. It is most appropriate when the statement(s) associated with the loop must be executed at least once.
- Program below shows the use of do...while loop. This program will display a menu for the user selection.

```
/* Demonstrates a  do...while loop */
     #include <stdio.h>
     #include <conio.h>
     main()
     {
int choice = 0;
clrscr();
printf (“Choose Menu Option : ”);
do
{
 printf("\n" );
 printf("\n1 - Add a Record" );
 printf("\n2 - Change a record");
 printf("\n3 - Delete a record");
 printf("\n4 - Quit");
 printf("\n" );
printf("\n Enter a choice : " );
scanf("%d", &choice);
}  while ( choice >= 1 && choice <= 4 );
  getch();
 }
```

Run :

```
Choose Menu Option :
1.  Add a Record
2.  Change a record
3.  Delete a record
4.  Quit
Enter a choice : 1
```

Nesting Do....While Statement

- A do while statement can be inserted within another do….while statement. This is called *nesting of while statement .*
- The general form of nested do….while is as follows

```
   do
   {
     do
      {
     statement(s);
      } while( condition 2);
   }   while( condition 1);
```

- Example Demonstrate use of nested do..while.

```
/* Multiplication Table from 5 to 10 */
 #include <stdio.h>
#include<conio.h>
 void main()
 {
      int row  =6, column,  y;
      clrscr();
printf(" Multiplication Table from 6 to 10  \n");
printf("---------------------------------------------\n");
```

```
  do /* Outer Loop Begins */
      {
  column = 1;
  do /*   Inner loop begins */
      {
   y = row * column;
   printf("%4d", y);
   column = column + 1;
} while(column <= 10); /*. Inner Loop Ends..*/
  printf("\n");
   row = row + 1;
 } while(row <= 10); /*... Outer Loop Ends ...*/
```

```
printf("---------------------------------------------\n");
 getch();
 }
Run :
 Multiplication Table From 6 to 10
--------------------------------------------------------
6    12  18  24  30  36  42  48  54  60
7    14  21  28  35  42  49  56  63  70
8    16  24  32  40  48  56  64  72  80
9    18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90  100
--------------------------------------------------------
```

The for Loop

- The for loop is another entry controlled loop that provides a more concise loop structure.
- The general form of the for loop is

```
for (initialization; test-condition; increment)
{
statement block;
}
```

- Here Initialization, condition, and increment all are C expressions, and statement block is a single or a compound C statement.

- First of all the initialization of the control variable is done by using an assignment statement. The variable used in this process is known as loop control variable.
- The value of control variable is tested by using the test condition.
- If it is false then the loop terminates and control is transferred to the first statement after the loop
- If it is true then the statements with in the body of the for loop is evaluated.

- After all the statements with in the loop are executed then the value of the control variable is incremented or decremented and it is again tested with the test condition.
- If it is true then the second iteration of the loop will be started and if it is false then the control is transferred to the first statement after the loop.

- For example consider the following simple program which calculate the sum of first 10 natural numbers by using for loop.

```
/* Sum of first 10 natural numbers. */
    # include <stdio.h>
    #include<conio.h>
   main()
   {
      int  sum = 0, i;
      clrscr();
```

```
for( i = 0; i<=10;++i)
    sum = sum + i ;
printf(“\n”);
printf(“ The sum of first 10 no is : %d”, sum);
getch();
 }

 Run :
 The sum of first 10 no is : 55
```

- **Nested For loop**

- When a for loop is embedded with in another for loop then this structure is known as nested for loop.
- The inner and outer loops need not be controlled by the same control variable.
- Each loop must be controlled by a different index .

General form of nested - for loop

- The general form of nested for loop is

```
for (initialization; test-condition; increment)
 {
 statement block 1;
 for (initialization; test-condition; increment)
 {
 statement block 2;
  }
 }
```

Example of Nested for loop

```
/* Calculate average of list of numbers */
    #include<stdio.h>
    #include<conio.h>
   main()
   {
    int n, i, loops, loopcount;
    float x, avg, sum ;
    clrscr();
    printf(“How many lists ?”);
    scanf(“%d”, &loops);
```

```
 for(lpcount = 1; lpcount <= loops;++lpcount)
  {
   sum = 0;
printf("\n List number %d\n How many no ? ",
   lpcount );
   scanf("%d", &n);
   for(count =1;count < = n;++count)
   {
   printf("x = ");
   scanf("%f", &x);
   sum = sum + x;
    }
```

```
   avg = sum / n ;
   printf("\n The average is %.2f \n", avg);
   }
 }
Run :
       How many lists ? 1
       How many numbers ? 3
       x = 4
       x = 3.5
       x =  6.7
     The average is 4.73
```

Jumps in the loops

- Loops perform a set of operations again and again until the control variable satisfies a particular test condition.
- Sometimes in a loop it becomes desirable to skip a part of the loop or to leave the loop as soon as a certain condition occurs.
- An early exiting from a loop or skipping a part of loop can be achieved by using the break , continue, exit and goto statement.

Break

- When a break statement is encountered with in a loop then the loop is immediately exited and the program continues with the statement immediately following the loop.

Continue

- During the loop operation if we want to skip a portion of the loop when a certain condition occurs with in the loop then we use the continue statement in the loop.

Exit

- To permanently get out of a program we use the exit() statement.

```
/* Demonstration for break ,continue & exit() */
    # include <stdio.h>
    #include <conio.h>
    main()
   {
     int count;
     clrscr();
     for( count = 0; count <= 5; ++count)
     {
        if (count == 3)
        break;   or continue; exit();
      printf(" \nCount = %d",count);
      getch();
      }
```

```
Run:  When break  or exit() is used
         The output will be
              Count = 0
              Count = 1
              Count = 2
  Run:  When continue is used
             The output will be
              Count = 0
              Count = 1
              Count = 3
              Count = 4
              Count = 5
```

Exercise

1. Write a program to find the sum of the first N natural numbers.
2. Write a program that reads N numbers and find the smallest number among them.
3. Write a program to evaluate the equation $Y = X^{n.}$
4. Write a program to find the average of n numbers.

Arrays and File

Why we use an array

- When we have to process large amounts of data it is not easy to handle this data with the fundamental data types of C like int, char, float and double.
- Fundamental data types are constrained by the fact that a variable of these data types can store only one value at a time.
- To process large amount of data we need a powerful data type that would facilitate efficient storing, accessing and manipulation of data items.
- C supports a data type known as array that can be used for this purpose.

What Is an Array?

- An array is a collection of variables of the same data type that are referred by a common name.
- A specific element in an array is accessed by an index which is known as subscript of the array
- In C, all array consists of continuous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

- Each array element (i.e. each individual data item) is referred by specifying the array name followed by one or more subscripts.
- Each subscript is enclosed in square brackets .Thus in the n element array X, the array elements are x[0],x[1],x[2]…..x[n].
- The number of subscripts of the array determines the dimension of the array. For example a x[i] refers to one dimensional array x[i][j] refers to two dimensional array.

Defining a one dimensional array

- Arrays are defined in much the same manner as ordinary variables except that each array name must be accomapnied by a size specifications (i.e. the number of elements in the array).
- For a one dimensional array the size is specified by a positive integer expression enclosed in square brackets.
- In general terms a one dimensional array may be expressed as

  data-type array-name[size];

- Several typical one dimensional arrays are shown below

```
int x[100];
char text[80];
float n[12];
```

Accessing Elements of an array

- To access the individual array elements we use the subscript which is the array name followed by a number in brackets. for example to access the second element of an array we use x[1].
- All the array elements are numbered from zero not from one.

Entering data into a one dimension array

- For entering data into a one dimensional array we have to use one loop statement.
- For example following code displays how to enter data into an integer array marks[30].

```
for(i=0;i<=29;++i)
{
 printf("\nEnter marks :");
 scanf("%d", &marks[i]);
}
```

- For example following code shows how to display data of a one dimensional integer array marks[30].

```
for(i=0;i<=29;++i)
{
 printf("\n I = ");
 printf("%d", &marks[i]);
}
```

Array initialization

- Array can also be initialized where we declare it. Following examples show how we can do this.

```
int num[6] = { 0,1,2,3,4,5 };
int n[ ] = {2,3,4,5,6,7,8}
float[ ] = {12.4,23.5,67.9,-90.8}
```

Bound Checking

- In C there is no check to see if the subscript used for an array exceeds the size of the array or not.
- If the data entered with a subscript exceeds the array size then it will simply be placed in memory outside the array or simply on the top of the other data.

Two Dimensional Array

- The array with two subscripts is known as two dimensional array.
- The general form of a two dimensional array is as follows :

  data type array-name[size][size];

- For example

```
int matrix[3][3];
float cost[4][5];
double[3][2];
```

Entering data into a two dimension array

- For entering data into a two dimensional array we have to use two loop statements.
- For example following code displays how to enter data into an array matrix[3][3]

```
for(i=0;i<=2;++i)
{
for(j =0;j<=2;++j)
{
printf("\nEnter value [%d,%d] :",i,j);
scanf("%d", &matrix[i][j]);
}
}
```

- For example following code shows how to display data of a two dimensional array matrix[3][3].

```
for(i=0;i<=2;++i)
{
printf("\n");
for(j =0;j<=2;++j)
printf("%3d\n", matrix[i][j]);
}
```

- Write a program to read and display a matrix

```
#include <stdio.h>
main()
{
int i,j,a[3][3];
clrscr();
printf("\nEnter the matrix");
for(i=0;i<=2;++i)
{
for(j =0;j<=2;++j)
{
printf("\nEnter a[%d,%d]: ",i,j);
scanf("%d",&a[i][j]);
```

```
}
}
printf("\nThe matrix is  :\n");
for(i=0;i<=2;++i)
{
printf("\n");
for(j =0;j<=2;++j)
printf("%3d", a[i][j]);
}
getch();
}
```

Files

- Many applications require that information should be written to or read from an auxiliary storage device. Such information is stored on a device in the form a data file.
- Data files allow us to store information permanently and to access and alter that information whenever necessary.
- C has an extensive set of library functions for creating and processing files.

- When working with a file in C the first step is to establish a buffer area where information can be stored temporarily while information is being transferred between the computer's memory and the data file.
- The buffer area is establish by writing

  FILE *ptvar; where
- FILE is a structure defined in the <stdio.h> header file.
- ptvar is a pointer variable to this structure FILE.

Commonly Used C File-System Functions

- Important file handling functions that are necessary for performing the basic operations on the file are as follows :

Name	Operation
fopen()	*Created a new file
fclose	*Close a file
getc()	*reads a character from a file
putc()	*writes a character to a file
fscanf()	*reads a set of data values
fprintf()	*writes a set of data values

Name	Operation
getw()	*Reads an integer
putw()	*Writes an integer
fgets()	*Reads a string
fputs()	*Writes a string
fseek()	*Sets the postion to a desired point in the file
ftell()	*Gives the current postion in the file
rewind()	*Sets the position to the beginning of the file

Defining and Opening a file

- A data file must be opened before it can be created or processed.
- This associates the file name with the buffer area i.e. with the stream.
- It also defines how the data file will be utilized that is as a read only file, a write only file or both.
- The library function fopen is used to open a file.The function is typically written as

  ptvar = fopen(file-name, file-type);

- The name chosen for the file name must be consistent with the rules of file naming.
- The file type must be one of the strings as shown below.

  File Type

  1. “r’
  2. “w”
  3. “a”
  4. “r+”
  5. “w+”
  6. “a+”

- A simple C program showing the procedure of opening a file

```
/* Demonstration of Opening a file */
 #include <stdio.h>
main()
  {
     FILE *fp;
    fp = fopen(“skm.c”,”r”);
    if (fp == NULL)
    {
    puts(“ Sorry ! Can not open file”);
    exit();
    }
  }
```

Closing a File

- When we have finished reading from a file we have to close that file. For this we will use the function fclose .
- The general form of this function is as

  fclose(fp)

  Here fp is a pointer to the file which is used for closing the file.

Input/Output Operations on files

- Once a file is opened the reading out of the file or writing to the file is accomplished by using standard I/O routines.

The getc() and putc() functions

- The simplest file I/O functions are getc() and putc().
- These are similar to the getchar() and putchar() functions.
- The general form of getc()

  char variable = getc(file-pointer)

  For example c = getc(fp2)

- The general form of putc() functions

  putc(char-var, file pointer);

  For example putc(c, fp2);
- Write a program to read data from the keyboard and then write it to a data file.

```
/* File sample program */
 #include  <stdio.h>
 main()
   {
     FILE *f1;
     char c;
```

```
printf("Data Input\n\n");
/* Open the file INPUT */
f1 = fopen("INPUT", "w");
/* Get a character from keyboard   */
while((c=getchar()) != EOF)
/* Write a character to INPUT  */
putc(c,f1);
/* Close the file INPUT   */
fclose(f1);
```

```
printf("\nData Output\n\n");
/* Reopen the file INPUT    */
f1 = fopen("INPUT","r");
/* Read a character from INPUT*/
while((c=getc(f1)) != EOF)
/* Display a character on screen */
printf("%c",c);
/* Close the file INPUT       */
 fclose(f1);
}
```

Run :

This is a program to test the file handling features on this system.

Data Output :

This is a program to test the file handling features on this system

The getw and putw functions

- The getw and putw functions are integer oriented. They are similar to the getc and putc functions.
- Getw and putw are used to read and write integer values.These functions are useful when we deal with only integer data.
- The general form of getw function is

  integer variable = getw(file-pointer);

  For example n = getw(fp);

- The general form of putw function is

  putw(integer , file-pointer);

  For example n = putw(int-var ,fp);
- Write a program to read integer data from the keyboard and then write it to a data file.

```
#include  <stdio.h>
main()
{
   FILE  *f1;
   int   number, i;
```

```
printf("Contents of Data file\n");
f1 = fopen("DATA", "w ");
for(i = 1; i <= 30; i++)
{
scanf("%d", &number);
if(number == -1)
break;
putw(number,f1);
}
 fclose(f1);
```

```
    /* Read from DATA file */
    f1 = fopen("DATA", “r ");
    printf(“\nThe data file is “);
    while((number = getw(f1)) != EOF)
    printf( “% d” , number);
   /* close the file */
    fclose(f1);
    getch();
   }
Run  : Contents of Data file
        1 2 3 4 -1
        The data file is
        1 2 3 4
```

The fgets and fputs Function

- Reading from files and writing to files is as easy as reading and writing of individual character string.
- C provides two main functions that deal with string I/O in files : fgets and fputs.
- The general form of fets is

  fgets(str , n , fptr);

  For example fgets(s , 79 , fp);
- The general form of fputs is

  fputs (str, fptr)

  For example fputs(s,fp);

- Write a program to read a string from the keyboard and then write it to a file.

```
/* Demonstrate program for a string */
  #include<stdio.h>
  #include<string.h>
  #include<conio.h>
  void main()
  {
    FILE *fp;
    char str[80];
    clrscr();
```

```
fp = fopen("Players.txt","w");
if(fp ==NULL)
{
puts("Sorry ! Can not open the file");
exit(0);
}
printf("\n Enter string or just press enter
to stop :");
gets(str);
while(strlen(str) >0)
{
```

```
fputs(str,fp);
fputs("\n" , fp);
printf("\n Enter string or just press enter
to stop :");
gets(str);
}
fclose(fp);
fp = fopen("Players.txt", "r");
if(fp ==NULL)
{
```

```
puts("Sorry ! Can not open the file");
exit(0);
}
puts("\n File Name : Players.txt");
while((fgets(str,80,fp) != NULL))
puts(str);
fclose(fp);
getch();
}
```

The fprintf and fscanf functions

- The functions fscanf and fprintf can handle a group of mixed data simultaneously.
- The fscanf and fprintf functions are identical to scanf and printf function except that they work on files.
- The first argument of fscanf and fprintf function is a file pointer which specifies the file to be used.
- The general form of fscanf is

  fscanf(fp , “control string” , list);

  For example

  fscanf(fp, “%c%d”,&item,&quantity);

- The general form of fprintf is

  fprintf(fp , “control string” , list);

  For example

  fprintf(fp, “%c%d”,item,quantity);

- Write a program to open a file named INVENTORY and store in it the following format :

  Item name Number Price Quantity

  Extend the program to read this data from the file INVENTORY and display the inventory table with the value of each item

```
/* Example of fprintf and fscanf  */
   #include  <stdio.h>
   #include <conio.h>
   main()
   {
      FILE  *fp;
      int    number, quantity, i;
      float  price, value;
      char   item[10], filename[10];
      printf("Input file name\n");
      scanf("%s", filename); f
       fp = fopen(filename, "w");
      printf("Input inventory data\n\n");
```

```
printf("Item Name  Number  Price Quantity\n");
   for(i = 1; i <= 3; i++)
     {
    fscanf(stdin, "%s %d %f %d",
     item, &number, &price, &quantity);
     fprintf(fp, "%s %d %.2f %d", item,number,
               price,quantity);
     }
      fclose(fp);
      fprintf(stdout, "\n\n");
      fp = fopen(filename,”r”);
```

```
printf("Item Name   Number   Price
Quantity     Value\n");
for(i = 1; i <= 3; i++)
{
fscanf(fp, "%s%d%f”,item,&number,&price,
         &quantity);
value = price * quantity;
fprintf(stdout, "%-8s %7d %8.2f %8d11.2f\n",
          item, number, price, quantity, value);
 }
 fclose(fp);
 }
```

Exercise

1. Write a program to sort a one dimensional integer array.
2. Write a program to find the subtraction of two matrices.
3. Write a program to find the multiplication of two matrices.
4. Write a program to copy the contents of a text file into another text file.

Structure and Union

Data Types

C programming language which has the ability to divide the data into different types. The type of a variable determine the what kind of values it may take on. The various data types are

- Simple Data type

  → Integer, Real, Void, Char

- Structured Data type

  →Array, Strings

- User Defined Data type

  →Enum, Structures, Unions

Structure Data Type

A structure is a user defined data type that groups logically related data items of different data types into a single unit. All the elements of a structure are stored at contiguous memory locations. A variable of structure type can store multiple data items of different data types under the one name. As the data of employee in company that is name, Employee ID, salary, address, phone number is stored in structure data type.

Defining of Structure

A structure has to defined, before it can used. The syntax of defining a structure is

```
struct <struct_name>

{

<data_type> <variable_name>;

<data_type> <variable_name>;
……..

<data_type> <variable_name>;

};
```

Example of Structure

The structure of Employee is declared as

```
struct employee

{

int  emp_id;

char name[20];

float salary;

char address[50];

int dept_no;

int age;

};
```

Declaring a Structure Variable

A structure has to declared, after the body of structure has defined. The syntax of declaring a structure is

```
struct <struct_name> <variable_name>;
```

The example to declare the variable for defined structure "employee"

```
struct employee e1;
```

Here e1 variable contains 6 members that are defined in structure.

Initializing a Structure Members

The members of individual structure variable is initialize one by one or in a single statement. The example to initialize a structure variable is

1) struct employee e1 = {1, “Hemant”,12000, “3 vikas colony new delhi”,10, 35);

2) e1.emp_id=1;   e1.dept_no=1   e1.name=“Hemant”;

    e1.age=35;

    e1.salary=12000;

    e1.address=“3 vikas colony new delhi”;

Accessing a Structure Members

The structure members cannot be directly accessed in the expression. They are accessed by using the name of structure variable followed by a dot and then the name of member variable. The method used to access the structure variables are e1.emp_id, e1.name, e1.salary, e1.address, e1.dept_no, e1.age. The data with in the structure is stored and printed by this method using scanf and printf statement in c program.

Structure Assignment

The value of one structure variable is assigned to another variable of same type using assignment statement. If the e1 and e2 are structure variables of type employee then the statement

e1 = e2;

assign value of structure variable e2 to e1. The value of each member of e2 is assigned to corresponding members of e1.

Program to implement the Structure

```
#include <stdio.h>
```

```
#include <conio.h>

struct employee

{

int  emp_id;

char name[20];

float salary;

char address[50];

int dept_no;

int age;

};
```

<u>Program to implement the Structure</u>

```
void main ( )

   { struct employee e1,e2;

      printf (“Enter the employee id of employee”);

      scanf(“%d”,&e1.emp_id);

      printf (“Enter the name of employee”);

      scanf(“%s”,e1.name);

      printf (“Enter the salary of employee”);

      scanf(“%f”,&e1.salary);

      printf (“Enter the address of employee”);
```

```
scanf("%s",e1.address);

printf ("Enter the department of employee");

scanf("%d",&e1.dept_no);

printf ("Enter the age of employee");
```

Program to implement the Structure

```
scanf("%d",&e1.age);

printf ("Enter the employee id of employee");

scanf("%d",&e2.emp_id);

printf ("Enter the name of employee");

scanf("%s",e2.name);

printf ("Enter the salary of employee");

scanf("%f",&e2.salary);

printf ("Enter the address of employee");

scanf("%s",e2.address);

printf ("Enter the department of employee");

scanf("%d",&e2.dept_no);

printf ("Enter the age of employee");

scanf("%d",&e2.age);
```

Program to implement the Structure

```
printf ("The employee id of employee is : %d",                    e1.emp_id);
printf ("The name of employee is : %s",                    e1.name);

printf ("The salary of employee is : %f",                  e1.salary);

printf ("The address of employee is : %s",                 e1.address);

printf ("The department of employee is : %d",              e1.dept_no);

printf ("The age of employee is : %d",                     e1.age);
```

Program to implement the Structure

```
printf ("The employee id of employee is : %d",         e2.emp_id);

printf ("The name of employee is : %s",                    e2.name);

printf ("The salary of employee is : %f",                  e2.salary);

printf ("The address of employee is : %s",                 e2.address);

printf ("The department of employee is : %d",              e2.dept_no);

printf ("The age of employee is : %d",e2.age);

getch();
```

Output of Program

```
Enter the employee id of employee 1

Enter the name of employee Rahul

Enter the salary of employee 15000

Enter the address of employee 4,villa area, Delhi

Enter the department of employee 3
```

```
   Enter the age of employee 35

   Enter the employee id of employee 2

   Enter the name of employee Rajeev

   Enter the salary of employee 14500

   Enter the address of employee flat 56H, Mumbai

   Enter the department of employee 5

   Enter the age of employee  30
```

Output of Program

```
  The employee id of employee is : 1

     The name of employee is : Rahul

     The salary of employee is : 15000

     The address of employee is : 4, villa area, Delhi

     The department of employee is : 3

     The age of employee is : 35

     The employee id of employee is : 2

     The name of employee is : Rajeev

     The salary of employee is : 14500

     The address of employee is : flat 56H, Mumbai

     The department of employee is : 5
```

The age of employee is : 30

Array of Structure

C language allows to create an array of variables of structure. The array of structure is used to store the large number of similar records. For example to store the record of 100 employees then array of structure is used. The method to define and access the array element of array of structure is similar to other array. The syntax to define the array of structure is

Struct <struct_name> <var_name> <array_name> [<value>];

For Example:-

Struct employee e1[100];

Program to implement the Array of Structure

```
#include <stdio.h>

#include <conio.h>

struct employee

{

int  emp_id;

char name[20];

float salary;

char address[50];

int dept_no;

int age;

};
```

Program to implement the Array of Structure

```
void main ( )
   {
     struct employee e1[5];
     int i;
     for (i=1; i<=100; i++)
     {
     printf ("Enter the employee id of employee");
     scanf ("%d",&e[i].emp_id);
     printf ("Enter the name of employee");
     scanf ("%s",e[i].name);
     printf ("Enter the salary of employee");
     scanf ("%f",&e[i].salary);
```

Program to implement the Array of Structure

```
    printf ("Enter the address of employee");
    scanf ("%s", e[i].address);
    printf ("Enter the department of employee");
    scanf ("%d",&e[i].dept_no);
    printf ("Enter the age of employee");
     scanf ("%d",&e[i].age);
```

```
 }

for (i=1; i<=100; i++)

{

 printf ("The employee id of employee is : %d",                    e[i].emp_id);

 printf ("The name of employee is: %s",e[i].name);
```

Program to implement the Array of Structure

```
printf ("The salary of employee is: %f",                                          e[i].salary);

printf ("The address of employee is : %s",                                          e[i].address);

 printf ("The department of employee is : %d",                                     e[i].dept_no);

 printf ("The age of employee is : %d", e[i].age);

 }

getch();

}
```

Structures within Structures

C language define a variable of structure type as a member of other structure type. The syntax to define the structure within structure is struct <struct_name>{

<data_type> <variable_name>;

```
                    struct <struct_name>

                       { <data_type> <variable_name>;

                      ……..}<struct_variable>;
```

<data_type> <variable_name>;

Example of Structure within Structure

The structure of Employee is declared as

```
 struct employee
  {  int   emp_id;
     char name[20];
     float salary;
     int dept_no;
     struct date
           {    int day;
                int month;
                int year;
           }doj;
};
```

Accessing Structures within Structures

The data member of structure within structure is accessed by using two period (.) symbol. The syntax to access the structure within structure is

```
struct _var. nested_struct_var. struct_member;
```

For Example:-

```
e1.doj.day;
```

e1.doj.month;

e1.doj.year;

Pointers and Structures

C language can define a pointer variable of structure type. The pointer variable to structure variable is declared by using same syntax to define a pointer variable of data type. The syntax to define the pointer to structure

```
 struct <struct_name> *<pointer_var_name>;
```

For Example:

```
 struct employee *emp;
```

It declare a pointer variable “emp” of employee type.

Access the Pointer in Structures

The member of structure variable is accessed by using the pointer variable with arrow operator(→) instead of period operator(.). The syntax to access the pointer to structure.

```
pointer_var_name→structure_member;
```

For Example:

```
 emp→name;
```

Here “name” structure member is accessed through pointer variable emp.

Passing Structure to Function

The structure variable can be passed to a function as a parameter. The program to pass a structure variable to a function.

```
#include <stdio.h>

#include <conio.h>

struct employee

{

int  emp_id;

char name[20];

float salary;

};
```

Passing Structure to Function

```
void main ( )

  {

     struct employee e1;

     printf (“Enter the employee id of employee”);

     scanf(“%d”,&e1.emp_id);

     printf (“Enter the name of employee”);

     scanf(“%s”,e1.name);

     printf (“Enter the salary of employee”);

     scanf(“%f”,&e1.salary);

     printdata (struct employee e1);

     getch();
  }
```

Passing Structure to Function

```
 void printdata( struct employee emp)
  {
     printf ("\nThe employee id of employee is :                    %d", emp.emp_id);
     printf ("\nThe name of employee is : %s",                           emp.name);
     printf ("\nThe salary of employee is : %f",                         emp.salary);
   }
```

Function Returning Structure

The function can return a variable of structure type like a integer and float variable. The program to return a structure from function.

```
#include <stdio.h>
#include <conio.h>
struct employee
{
int  emp_id;
char name[20];
float salary;
};
```

Function Returning Structure

```
void main ( )
  {
     struct employee emp;
     emp=getdata();
     printf ("\nThe employee id of employee is :                  %d", emp.emp_id);
     printf ("\nThe name of employee is : %s",                          emp.name);
     printf ("\nThe salary of employee is : %f",                        emp.salary);
     getch();
  }
```

Function Returning Structure

```
 struct employee getdata( )
 {
     struct employee e1;
     printf ("Enter the employee id of employee");
     scanf("%d",&e1.emp_id);
     printf ("Enter the name of employee");
     scanf("%s",e1.name);
```

```
    printf ("Enter the salary of employee");

    scanf("%f",&e1.salary);

    return(e1);

  }
```

Union Data Type

A union is a user defined data type like structure. The union groups logically related variables into a single unit. The union data type allocate the space equal to space need to hold the largest data member of union. The union allows different types of variable to share same space in memory. There is no other difference between structure and union than internal difference. The method to declare, use and access the union is same as structure.

Defining of Union

A union has to defined, before it can used. The syntax of defining a structure is

```
 union <union_name>

 {

   <data_type> <variable_name>;

   <data_type> <variable_name>;

    ……..

   <data_type> <variable_name>;

 };
```

Example of Union

The union of Employee is declared as

```
union employee

{

int  emp_id;

char name[20];

float salary;

char address[50];

int dept_no;

int age;

};
```

Difference between Structures & Union

1) The memory occupied by structure variable is the sum of sizes of all the members but memory occupied by union variable is equal to space hold by the largest data member of a union.

2) In the structure all the members are accessed at any point of time but in union only one of union member can be accessed at any given time.

Application of Structures

Structure is used in database management to maintain data about books in library, items in store, employees in an organization, financial accounting transaction in company. Beside that other application are

1) Changing the size of cursor.

2) Clearing the contents of screen.

3) Drawing any graphics shape on screen.

4) Receiving the key from the keyboard.

5) **<u>Application of Structures</u>**

6) 5) Placing cursor at defined position on screen.

7) 6) Checking the memory size of the computer.

8) 7) Finding out the list of equipments attach to computer.

9) 8) Hiding a file from the directory.

10) 9) Sending the output to printer.

11) 10) Interacting with the mouse.

12) 11) Formatting a floppy.

13) 12) Displaying the directory of a disk.

<u>Summary</u>

- A structure is a user defined data type that groups logically related data items of different data types into a single unit.
- The elements of a structure are stored at contiguous memory locations.
- The value of one structure variable is assigned to another variable of same type using assignment statement.
- An array of variables of structure is created.
- A variable of structure type is defined as a member of other structure type called nested structure.

<u>Summary</u>

- The member of structure variable is accessed by pointer variable with arrow operator (→).
- The structure variable can be passed to a function as a parameter.

- The function can return a variable of structure type.
- A union is like structure that group logically related variables into a single unit. The union allocate the space equal to space need to hold the largest data member of union.
- Structure used in database management and many more applications.