

**INTRODUCTION TO C LANGUAGE**

C is a programming language developed at AT & T‟s Bell laboratories of USA in 1972. It

was developed by **Dennis Ritchie.**

**Characteristics of C:**

* C is a general purpose programming language.
* It is a structured programming language (It allows modularization process of dividing problems into sub-problems and C has structured statements like sequence, selection

and iteration)

* C is a middle level language (C is a high level language but also incorporates features of low-level languages like manipulation of bits, bytes, words and addresses.)
* Helps in development of system software ( C is used for writing application programs and since it has low level languages features it can be used in development of system

software)

* Has rich set of operators and data types and built-in functions.
* Provides compact representation for expressions. This makes the language simple, orderly and easy to learn.
* No rigid format. Any number of statements can be typed in a single line.
* Portability: any C program can be run on different machines with little or no modifications.
* Less number of reserved words
* Pointer arithmetic and pointer manipulation
* Ability to extend itself by adding functions to its library.

**History of C:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **year** | **Language** | **Developed by** | **Remarks** |  |
|  |  |  |  |  |  |
|  | 1960 | ALGOL | International committee | Too general, to abstract |  |
|  |  |  |  |  |  |
|  | 1963 | CPL | Cambridge | Difficult to learn and implement |  |
|  |  |  | University(CU) |  |  |
|  |  |  |  |  |  |
|  | 1967 | BCPL | Martin Richards at CU | Could deal with only specific problems |  |
|  |  |  |  |  |  |
|  | 1970 | B | Ken Thomson at AT & T | Could deal with only specific problems |  |
|  |  |  |  |  |  |
|  | 1972 | C | Dennis Ritchie at | Restored the lost generality of BCPL |  |
|  |  |  | AT & T | and B |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  | | |  |  |



**Applications of C:** Because of its portability and efficiency, C is used to develop thesystem as well as application software. Some important system programs include operating systems, compilers, interpreters, assemblers, editors, linkers and loaders.

Being a structured language, it is useful in developing application programs like database systems, word processors, spreadsheets, graphics packages, CAD/CAM applications, office automation tools, scientific and engineering applications, animation and games.

**Basic structure of a C program or Syntax:**

Every C program includes the following components

* Pre-processor directives
* Main()
* A pair of flower brackets { }
* Declaration and statements
* User created sub-programs or functions.

Pre-processor directives

Global declarations;

Main ()

{

Declarations;

Statements;

}

User created sub-programs or functions

**Pre-processor directives:** these statements begin with # symbol. They direct the C pre-processor to include header files and symbolic constants into a C program.

**E.g.:** #include<stdio.h> : For the standard input, output function

#include<conio.h> : For the Console input, output function

# define NULL 0 : For defining symbolic constant, NULL=0





**Global declarations:** Variables or functions whose existence is known in the main ()function and other user defined functions, are called the global variables or functions. Their declarations should be made before main ().

**Main ():** This is the main function of every C program. Execution of a C program starts withMain (). No C program is executed without the main () function. It should be written in lowercase letters and should not be terminated with semicolon. It calls other library functions and user defined functions. There must be one and only one main () function in any C program.

**A pair of flower brackets:** The flower brackets indicate body of the program, whichincludes instructions to perform the required task. { indicates beginning of the main () function and } indicates end of the main() function.

**Declarations:** It is the part where all the variables, arrays, functions etc., used in the Cprogram are declared and may be initialized with their basic data types.

**Statements:** These are instructions to perform certain operations like input, output,assignment, arithmetic statements, control statements and other statements. They also include comments. Comments are explanatory notes on some instructions. The statements to be commented must be enclosed within /\* and \*/. The comment lines are not compiled and executed.

**User created sub-programs or functions:** These are sub-programs created by the user toperform certain common functions. They may be written before or after main ().

**Example:**

**/\* C program to display hello world \*/**

#include<stdio.h>

#include<conio.h> void main()

{

printf(“\n HELLO WORLD”); getch();

}

**Output:**

HELLO WORLD





**C PROGRAMMING PRELIMINARIES**

**CHARACTER SET:**

|  |  |  |
| --- | --- | --- |
| Alphabets | Uppercase letters | A to Z |
|  | Lowercase letters | a to z |
| Digits | 0 to 9 |  |
| Special Characters | blank , ; : “ + - \* / { } ? ! # \_= | ^ ~ < > \ () []@...etc. | |
|  |  |  |

**IDENTIFIERS:** These are names given to program elements like variables, arrays, functionsand structures. Basically identifiers are the sequence of alphabets and digits.

Rules for forming identifier names:

1. The first character must be an alphabet (upper or lowercase) or an underscore.
2. All succeeding characters must be either letters or digits but no other special character or punctuation symbols allowed.
3. Uppercase and lowercase identifiers are different in C. (E.g.: SUM, sum and sum are different identifiers.)
4. No two successive underscores are allowed.
5. Keywords should not be used as identifiers.
6. Identifiers should be single words. That is, blank spaces cannot be included in identifiers.

E.g.: valid identifiers: - sum, emp\_name, stu\_address, number, age

Invalid identifiers: - father‟s-name, while, @total, do-it, 3by2.

**VARIABLES:**

An identifier/ quantity that change its value during program execution are called

„variable‟. A variable is an object or element that may take on any value of a specified type.

Variables represent named storage locations, whose value can be manipulated during program execution.

E.g.:- sum, area, m1, length, sname, age, city etc.

Rules for naming a variable – same as rules for forming an identifier name

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Valid variables** | **Invalid variables** |  |
|  | Marks | 8ab |  |
|  | TOTAL\_MARK | TOTAL-MARK |  |
|  | Gross\_salary\_2004 | Gross-salary-2004 |  |
|  | Area\_of\_circle | Area-of-circle |  |
|  | Num[20] | Num(20) |  |
|  |  |  |  |
|  |  |  |  |
|  | |  |  |



**CONSTANTS:** It refers to fixed values that do not change during the execution of theprogram. C supports several types of constants.

* 1. Numeric Constants
     1. Integer constant
     2. Float (real) constant
  2. Character constants
     1. Single character constants
     2. String constants
  3. Symbolic constants
  4. Backslash constants

1. **Numeric Constants**
2. **Integer constants:** All whole numbers are integer constants. They aresequence of digits without decimal point

|  |  |
| --- | --- |
| **E.g.:** Decimal constants | : 134, 9000, 3567, -987 |
| Octal constant | : 06, 034, 0677, -07867 |
| Hexadecimal constant | : 0x20, 0xA, 0x1AB |

* 1. **Float (Real) constant:** Numbers having decimal point are floating pointconstants. They can contain both an integer part and a fractional part in the number. They are stored differently from integers. They may be represented either in decimal form or in the exponent form.

**E.g.:** Valid decimal constants**: -** 18.5, .18, 5. , -25.67Valid exponent constants: - 5.3E-5, -6.79E3, 34e-8

Here the number before „E‟ / „e‟ is called mantissa and mantissa can be either real or integer, the number after „E‟/‟e‟ is called exponent and exponent should be an integer.

1. **Character constants** 
   * 1. **Single character:** It is a letter or any symbol enclosed in single quotes.Character constants can be used in assignment statements or in symbolic constant definitions.

**E.g.:** ch=‟h‟ ;

#define PERIOD „.‟

* + 1. **String constant:** A sequence of characters enclosed between doublequotes is called string constant.

**E.g.:** “C programming is fun”“Enter two numbers”

1. **Symbolic constants**

It is a name that substitutes a numeric, character or a string constant. **E.g.:** #define PI 3.1415 (Here we substitute PI for numeric constant)

#define PLACE “MYSORE”

#define FOUND 1

#define NUM 023 /\* indicates octal number \*/ #define VALUE 0x20 /\*0x indicates hexadecimal \*/





1. **Backslash constant**

They are also called „escape sequences‟ and are used in the output statements. A backslash constant is a combination of two characters in which the first character is always the backslash (\) and the second character can be any one of a, b, f, n, r, t, v.

|  |  |
| --- | --- |
| **Backslash constant** | **Meaning** |
| „\a‟ | System alarm |
| „\b‟ | Back space |
| „\f‟ | Form feed |
| „\n‟ | New line |
| „\r‟ | Carriage return |
| „\t‟ | Horizontal tab |
| „\v‟ | Vertical tab |
| „\”‟ | Double quote |
| „\‟‟ | Single quote |
| „\0‟ | Null character |
| „\\‟ | Back slash character |

**KEYWORDS (Reserved words):** These are predefined words that have special meaningand these meanings cannot be changed. They can be used only for the purpose for which they have been defined. All keywords must be written in lowercase letters.

Example:

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

**DATA TYPES:**

Data type indicates the type of data a variable can have. Data type defines set of values, which can be stored in the variable along with certain operations that may be performed on those values.





There are three classes of data types

* Primary (or fundamental) data types:- int, float, char, double
* Derived data types:- arrays, functions, structures, pointers
* User-defined data types:- typedef, enum

**Primary Data types:**

**int:** Means that the variable is an integer. An integer needs two bytes of space. Range ofvalues that can be stored in two bytes is -32768 to +32767. The operations that can be performed on an integer value are addition, subtraction, multiplication, division and remainder.

**char:** Means that the variable is a character. A character needs one byte of storage space. Ifunsigned, the range of values is 0 to 255. The possible operations are input and output.

**float:** Means that the variable is a real number. Four bytes or 32 bits are allocated in memory,this includes 7 significant digits after the decimal point. The range of values is 3.4E-38 to 3.4E+38.

**double**: Means that that variable is a double precision float type. Eight bytes or 64 bits ofstorage space and range of values is 1.7E-308 to 1.7E+308.

**Void**: the void type has no values. This is usually used to specify the type of functions. Thetype of a function is said to be void when it does not return any value to the calling function.

**Size and Range of data types**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Size (bits)** | | **Range** |  |
| Char or signed char | 8 (1 byte) | | -128 to 127 |  |
| Unsigned char | 8 (1 byte) | | 0 to 255 |  |
| int or signed int | 16 | (2 bytes) | -32,768 to 32,767 |  |
| Unsigned int | 16 | (2 bytes) | 0 to 65535 |  |
| Short int or signed short int | 8 (1 byte) | | -128 to 127 |  |
| Unsigned short int | 8 (1 byte) | | 0 to 255 |  |
| Long int or signed long int | 32 | (4 bytes) | -2,147,483,648 | to |
|  |  |  | 2,147,483,647 |  |
| Unsigned long int | 32 | (4 bytes) | 0 to 4,294,967,295 |  |
| float | 32 | (4 bytes) | 3.4E-38 to 3.4E+38 |  |
| double | 64 | (8 bytes) | 1.7E-308 to 1.7E+308 |  |
| Long double | 80 | (10 bytes) | 3.4E-4932 to 1.1E+4932 |  |

**Declaration (definition) of variables:**

**Syntax:** data\_type variable list;

Variables must be separated by comma and declaration statement must end with a semicolon.

Example: int a, b, c; float sum, average;





**Assigning values to variables:**

**Method 1:** values can be assigned to variables at the time of declarationE.g.: int sum=0;

float val=3.14159; char ans=‟y‟;

**Method 2:** values can be assigned using an assignment statementE.g.: sum=0;

Val=3.14159;

Ans=‟y‟;

**Example:** C program to compute area and circumference of circle#include<stdio.h>

#include<conio.h> #define PI 3.1415 void main()

{

int r; float a, c; clrscr();

printf(“enter the radius of the circle\n”); scanf(“%d”,&r);

a=PI\*r\*r;

c=2\*PI\*r;

printf(“area of the circle=%f\n”,a); printf(“circumference=%f\n”,c); getch();

}

**Output:**



**INPUT-OUTPUT STATEMENTS**

C is a functional programming language. To perform input-output operations C provides a library of functions. This library is called a standard input-output library. It is denoted by **stdio**. The header file containing such library functions is called **stdio.h**

Some of the standard I-O functions in C are

|  |  |  |
| --- | --- | --- |
| 1. | scanf() | 5. gets() |
| 2. | printf() | 6. puts() |
| 3. | getchar() | 7. getch() |
| 4. | putchar() | 8. getche() |
|  |  |  |

There are two types of input-output statements

1. Formatted I/O statements
2. Unformatted I/O statements

|  |  |  |
| --- | --- | --- |
|  | Input statements | Output statements |
| Formatted | scanf() | printf() |
| Unformatted | getchar() getche() getch() | putchar() puts() |
|  | gets() |  |

**FORMATTED I/O STATEMENTS:**

They allow the input read from the keyboard or the output displayed on VDU to be formatted as per our requirements i.e., where this output will appear on screen, how many spaces between two values, number of decimal places etc. Can be controlled using formatted I/O statements.

**FORMATTED INPUT:**

scanf() statement is used to input the numeric, character and string type of data.

**Syntax:-** scanf(“control string”, address\_list);

**Example:-** scanf(“%d%d”,&a,&b);scanf(“%c%d%f”,&ch,&a,&r);

% indicates that a conversion specification.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | **Control String** |  | **Meaning** |  |
|  | %c |  | Read a single character |  |
|  | %d |  | Read a decimal integer |  |
|  | %e or %f or %g |  | Read a floating-point number |  |
|  | %h |  | Read a short int |  |
|  | %I |  | Read a decimal or hexadecimal or octal number |  |
|  | %o |  | Read on octal number |  |
|  | %p |  | Read a pointer |  |
|  | %s |  | Read a string |  |
|  | %u |  | Read an unsigned integer |  |
|  | %x |  | Read a hexadecimal number |  |

**FORMATTED OUTPUT:**

C provides the printf() function to display the data on the monitor.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Syntax**: | printf(“control string”, varlist); | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **E.g.:** | printf(“%d”,num); | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | printf(“ result=%f\n”,res); | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | printf(“ area=%6.2f, circumference=%6.2f\n “,ar,ci); | | | | | | | | | | | | | | | | | | | | |  |  |  |
| **Format specifiers for integers:** | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| printf(“%d”,6789); | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | 6 | | 7 | |  | 8 | | 9 | |  |  |  |  |  |  |  |
| printf(“%8d”,1239); | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | | 2 | | 3 | | 9 | |  |  |  |
| printf(“%-8d”,1239); | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 1 | | 2 | | 3 | | 9 | |  |  |  |  |  |  |  |  |  |  |  |  |
| printf(“%08d”,1239); | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 0 | | 0 | | 0 | | 0 | |  | 1 | | 2 | | 3 | | 9 | |  |  |  |
| **Format specifiers for floating point numbers:** | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |
| printf(“%f”,num); | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | |  | |  | |  |  |  |  | |  | |  | |  | |  | |  |  |
|  |  | 1 | | 2 | | 3 | | 4 | | . |  |  | 6 | | 7 | | 8 | | 0 | | 0 | | 0 | |
| printf(“%10.2f”,num); | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | |  | |  | | |  | | |  |  | |  | |  |
|  |  |  |  |  |  |  |  |  | 1 | | 2 | |  | 3 | | 4 | | . |  | 6 | | 8 | |  |
| printf(“%-10.2f”,num); | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | | | | | |  | |  | |  |  |  | | |  | |  |  |  |  |  |  |  |
|  |  | | 1 | | 2 | | 3 | | 4 | | . |  |  | 6 | | 8 | |  |  |  |  |  |  |  |
| printf(“%10.2e”,num); | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | | |  |  | |  | | |  | |  | |  | |  | |  |
|  |  |  |  |  |  |  | 1 | | . |  | 2 | |  | 3 | | E | | + | | 0 | | 3 | |  |
| **Format specifiers for characters:** | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

printf(“%c”,”A”);

A

printf(“%4c”,”A”);

A

printf(“%-4c”,”A”);

A





**Example program 1:**

#include<stdio.h>

#include<conio.h> void main()

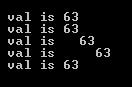
{

int val=63; clrscr();

printf(“\n val is %d”,val); printf(“\n val is %2d”,val); printf(“\n val is %4d”,val); printf(“\n val is %6d”,val); printf(“\n val is %-6d”,val); getch();

}

**Output:**



**Example Program 2:**

#include<stdio.h>

#include<conio.h> void main()

{

int num=126; clrscr();

printf(“ number is=%d\n”,num);

printf(“ octal equivalent of %d=%o\n”,num,num); printf(“ hexadecimal equivalent of %d=%x\n”,num,num); printf(“hexadecimal equivalent of %d=%X\n”,num,num); getch();

}

**Output:**



**UNFORMATTED INPUT:**

These statements are primarily concerned with reading the character type data from keyboard.

**getchar() function :** This function reads a single character from the standard input

device.

**Syntax:** ch\_var=getchar();





**gets() function:** This function reads in everything you enter from the keyboard until theENTER key is pressed i.e., it reads a string of all printable ASCII characters.

**Syntax:** gets(string);

**UNFORMATTED OUTPUT:**

These statements are mainly concerned with displaying or printing the character type data on the monitor.

**putchar() function:-** This function prints a single character on the screen.

**Syntax:** putchar(ch\_var);

**puts() function:-** This function prints a string of characters on screen.

**Syntax:** puts(string);

Example 1: #include<stdio.h> #include<conio.h> void main()

{

char letter; clrscr(); letter=getchar(); putchar(letter); getch();

}

Output:



**Example 2:**

#include<stdio.h>

#include<conio.h> void main()

{

char mesg[20]; clrscr();

printf(“enter the message to be displayed\n”); gets(mesg);

puts(mesg);

getch();

}





Output:



All data transfer operations in C are carried out using functions available in the standard library. These functions are known as the standard I/O library. These operations can be classified into two types.

* 1. Buffered Input/output
  2. Unbuffered Input/output

**Buffered Input:**

* Buffer is a temporary storage in the memory. The characters entered at the keyboard are stored in the buffer until the ENTER key is pressed and then made available to the program as a block of characters.
* Input-output operations take less time on a buffered system.
* If any mistake is done while entering the data before ENTER key is pressed, can be corrected.

E.g.: scanf() and getchar() are buffered input function.

**Unbuffered Input:**

Here, the character which is entered at the keyboard is made available to the program immediately. It is useful in some interactive programs and to read and write binary data.

E.g.: getch() and getche() are unbuffered input function.

**STANDARD MATHEMATICAL FUNCTIONS**

Mathematical functions such as cos, sqrt, log, etc, are frequently used in analysis of real life problems. Most of the C compilers support these basic math functions[#include<math.h>].

Note : x and y should be declared as double.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Function** | | **Meaning** |  |
|  | acos(x) | | Arc cosine of x |  |
|  | asin(x) | | Arc sine of x |  |
|  | atan(x) | | Arc tangent of x |  |
|  | atan2(x,y) | | Arc tangent of x/y |  |
|  | cos(x) | | Cosine of x |  |
|  | sin(x) | | Sine of x |  |
|  | tan(x) | | Tangent of x |  |
|  | cosh(x) | | Hyperbolic cosine of x |  |
|  | sinh(x) | | Hyperbolic sine of x |  |
|  | tanh(x) | | Hyperbolic tangent of x |  |
|  | ceil(x) | | X rounded up to the nearest integer |  |
|  | exp(x) | | E to the x power (ex) |  |
|  | fabs() | | Absolute value of x |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  |  | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | floor(x) |  | X rounded down to the nearest integer |  |
|  | fmod() |  | Remainder of x/y |  |
|  | log(x) |  | Natural log of x, x>0 |  |
|  | log 10(x) |  | Base 10 log of x, x>0 |  |
|  | pow(x,y) |  | X to the power y (xy) |  |
|  | sqrt(x) |  | Square root of x, x>0 |  |

**Example:**

#include<stdio.h>

#include<conio.h>

#include<math.h> void main()

{

int x,y,p,z; clrscr();

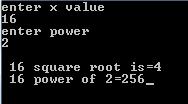
printf("enter x value\n"); scanf("%d",&x); printf(“enter power\n”); scanf(“%d”,&p) y=sqrt(x);

z=pow(x,p);

printf("\n %d square root is=%d",x,y); printf("\n %d power of %d=%d",x,p,z); getch();

}

Output:





**OPERATORS AND EXPRESSIONS**

**OPERATORS:**

Operators are tools for manipulating data. Data items on which operators act are called operands. Some operators act on only one operand. Such operators are called unary operators. Similarly binary and ternary operators are there depending on the number of operands on which they act.

Operators in C

* + Arithmetic operators
  + Assignment operator
  + Relational operators
  + Logical operators
  + Conditional operators
  + Bitwise operators
  + Special operators

**a. Arithmetic operators:**

They are used to perform numerical manipulations in C.

Binary arithmetic operators

|  |  |  |
| --- | --- | --- |
|  | OPERATOR | PURPOSE |
|  | + | Addition |
|  | - | Subtraction |
|  | \* | Multiplication |
|  | / | Division |
|  | % | Remainder of division |
| E.g.: sum=num1+num2; | |  |
| Val=a-b-c; | |  |
| Unary arithmetic operators | |  |
|  |  |  |
|  | OPERATOR | PURPOSE |
|  | - | Unary minus |
|  | ++ | Increment |
|  | -- | Decrement |
| E.g.: value=-6; | |  |
| Z=-x+y; | |  |
| Num=++i; | |  |

++int\_var indicates pre increment. That is, the value of int\_var must be incremented before it is used (increment and use)

int\_var++ indicates post increment. That is, use the value of int\_var first and then increment it (use and increment).

|  |  |  |
| --- | --- | --- |
| For example, a=++b; | |  |
|  | This is equal to |  |
|  | b=b+1; |  |
|  | a=b; |  |
| Similarly, | a=b++; |  |
|  | |  |
|  | |  |
|  | |  |



This is equal to a=b;

b=b+1; **b. Assignment operator:**

It is used to assign value to an identifier. **Syntax**: variable=value or expression; **E.g.:** num=20;

C allows users to combine assignment operators with arithmetic operators. Such operators are called as compound assignment operators/ shorthand assignment operator.

Let num1=10, num2=13, f1=7.8 and f2=4.2

|  |  |  |
| --- | --- | --- |
| **Assignment statement** | **Equivalent** | **Result** |
| Num1=num1+50; | Num1+=50; | 60 |
| Num1=num1\*2; | num1\*=2; | 20 |
| num2=num2\*10+20; | num2\*=10+20; | 150 |
| f1=f1/1.3; | f1/=1.3; | 6.0 |
| num2=num2%5; | num2%=5; | 3 |
| f1=f2+f1; | f2+=f1; | 12.0 |

**c. Relational operators:**

They are used to relate two quantities in conditional statements (if -statement)

|  |  |
| --- | --- |
| **Relational operator** | **Meaning** |
| == | Equal to |
| != | Not equal to |
| < | Less than |
| <= | Less than or equal to |
| > | Greater than |
| >= | Greater than or equal to |

E.g.: if x=5, y=2 and z=0;

Then x > y + z evaluates to 5>2+0=1(true)

x + z <= y evaluates to 5+0<=2=0(false) Differences between = and == signs

== is a relational operator

= is an assignment operator

== Sign does not change the value of the identifier, it only checks if one value is equal to another value and gives the answer true or false

* + sign assigns the value of the identifier on its left. **d. Logical operators:**

The logical operators && and || are used when we want to test more than one condition and make decisions.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Operator** | **Meaning** |  |
|  | && | Logical AND |  |
|  | || | Logical OR |  |
|  | ! | Logical NOT |  |
| E.g.: if (age>55 && salary<1000) | |  |  |
|  |  |  |  |
|  |  |  |  |



if (number<0|| number>100) if(num!=0)

**e. Conditional operators:**

It is also called **ternary operator,** as it operates on three operands. Syntax: result=exp1? exp2:exp3;

E.g.1: c=a<b? a: b; That is,

If a<b c=a; else c=b;

E.g.2: C program to find biggest of two numbers. #include<stdio.h>

#include<conio.h> void main()

{

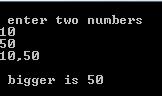
int a,b,big; clrscr();

printf(“\n enter two numbers”); scanf(“ %d%d”,&a,&b); printf(“%d,%d\n”,a,b); big=a>b?a:b;

printf(“\n bigger is %d\n”,big); getch();

}

Output:



**f. Bitwise operators:**

All data are stored in the computer as a sequence of bits (0‟s and 1‟s), and some applications need the manipulation of these bits. Manipulation of individual bits is possible in machine and assembly languages. But c provides operators to manipulate bits. Bitwise operators can be applied only on integers and characters but not on float.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Operator** | **Symbol name** | **Meaning** |  |
|  | & | Ampersand | Bitwise AND |  |
|  | | | Pipeline | Bitwise OR |  |
|  | ^ | Caret | Exclusive-OR (XOR) |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  | | |  |



|  |  |  |
| --- | --- | --- |
| ~ | Tilde | 1‟s complement |
| << | Double less than | Left shifting of bits |
| >> | Double greater than | Right shifting of bits |

For an 4-bit binary representation of numbers

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Decimal** |  | **Binary** | |  |  |
|  | **Number** |  |  |  |  |  |
|  | 0 |  | 0000 | |  |  |
|  | 1 |  | 0001 | |  |  |
|  | 2 |  | 0010 | |  |  |
|  | 3 |  | 0011 | |  |  |
|  | 4 |  | 0100 | |  |  |
|  | 5 |  | 0101 | |  |  |
|  | 6 |  | 0110 | |  |  |
|  | 7 |  | 0111 | |  |  |
|  | 8 |  | 1000 | |  |  |
|  | 9 |  | 1001 | |  |  |
|  | 10 |  | 1010 | |  |  |
| For example consider a=4, b=3 | |  |  |  |  |  |
| 4(10)=0100(2) and 3(10)=0011(2) | |  |  |  |  |  |
| The result of bitwise AND of a and b | | The result of bitwise OR of a and b | | | | |
| a=0100 | |  |  | a=0100 | | |
| b=0011 | |  |  | b=0011 | |  |
| a&b =0000 | |  |  | a|b=0111 | | |
| The result of bitwise Exclusive-OR | |  |  |  |  |  |
| a=0100 | |  |  |  |  |  |
| b=0011 | |  |  |  |  |  |



a^b =0111 ( XOR gives output 1 if bits are different (1 and 0), if bits are same i.e., either 0 and 0 or 1 and 1 it gives output 0.)

The result of 1‟s complement

C = ~a

C = ~ (0100) =1011

Left and right shift operators.

Syntax: <variable><shift operator><number of bits to be shifted>

|  |  |  |
| --- | --- | --- |
| Ex: Right shift operator | | Left shift operator |
| num=110011110 then | | ch=11010111 then |
| num>>1 gives 011001111 | | ch<<1 gives 10101110 |
| num>>3 gives | 000110011 | ch<<3 gives 10111000 |
| num>>5 gives | 000001100 | ch<<5 gives 11100000 |





**Example: C program to illustrate bitwise operators.**

#include<stdio.h>

#include<conio.h> void main()

{

unsigned int x,y; clrscr();

x=128; y=32; x=x>>1;

printf(“\n After right shifting by 1,x=%d”,x); y=y<<2;

printf(“\n After left shifting by 2,y=%d”,y); getch();

}

Output:



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
|  | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| X= | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Y= | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | The result after shifting the bits | | | | |  |  |  |
| X= | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Y= | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**g. Special operators:**

C supports the following special operators.

1. Comma operator
2. Sizeof() operator
3. Pointer operators-> \* dereferencing operator, & address operator
4. Member selection operators-> dot (.) operator, -> arrow operator

**Comma operator**: It is basically associated with for statement. It is used to link therelated expressions together, it is used as statement separator to simplify some code.

|  |  |
| --- | --- |
| **Ex 1**: sum=(a=10, b=15, a+b); | 2) b=(a=20, a+10); |
| Equal to: a=10; | equal to: |
| b=15; | a=20; |
| sum=a+b; | b=a+10; |
|  |  |
|  |  |
|  |  |



**sizeof() operator:** It returns the size i.e., number of bytes of the operand. The operandmay be a constant, a variable or a datatype.

**Syntax:** sizeof(operand);

|  |  |  |
| --- | --- | --- |
| **Variable declaration** | **Expression** | **Value** |
| int num | sizeof(num) | 2 |
| long num | sizeof(num) | 4 |
| float num | sizeof(num) | 4 |
| char ch | sizeof(ch) | 1 |
| double num | sizeof(num) | 8 |
| int a,b | sizeof(a+2\*b) | 2 |

Example: C program to illustrate sizeof() operator. #include<stdio.h>

#include<conio.h> void main()

{

int a,e; float b,f; double c,g; char d,h; clrscr(); e=sizeof(a); f=sizeof(b); g=sizeof(c); h=sizeof(d);

printf(“\nsize of a=%d”,e); printf(“\nsize of b=%f”,f); printf(“\nsize of c=%f”,g); printf(“\nsize of d=%d”,h); getch();

}

Output:





**EXPRESSIONS:**

An expression in c consists of a syntactically valid combination of operators and operands that computes a value.

E.g.: 2+5\*10 here + and \* are operators and 2, 5 and 10 are operands.

An expression in c can be of the following types.

1. Arithmetic expression
2. Relational expression
3. Logical expression

**Arithmetic expression:** An expression that use arithmetic operators along with numeric orcharacter operands is called arithmetic expression.

1. **Integer mode expressions:** These are formed using integer type constants, variables,functions and operators.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Expression** | **Result** |
| + | 10+3 | 13 |
| - | 10-3 | 7 |
| \* | 10\*3 | 30 |
| / | 10/3 | 3 |
| % | 10%3 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ii.** | **Real mode expressions:** These | | are formed using real mode quantities constants, | | |
|  | variables functions and operators. | |  |  |  |
|  |  |  |  |  |  |
|  | **Operators** |  | **Expression** | | **Result** |
|  | + |  | 1.5+6.7 | | 8.2 |
|  | - |  | 6.7-1.5 | | 5.2 |
|  | \* |  | 1.5\*6.3 | | 9.45 |
|  | / |  | 6.4/3.2 | | 2 |
| **iii.** | **Mixed mode expressions:** C permits mixing of integer and real quantities in an | | | | |
|  | expression. | |  |  |  |
|  |  | |  |  |  |
|  | **Expression** | |  |  | **Result** |
|  | **1.5\*10+6.7** |  |  |  | **21.7** |
|  | **7%5+10.0\*10/4** | |  |  | **27.0** |

**Relational Expression:** A meaningful combination of operands and relational operators.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Expression** | **Result** |
| == | 7==8 | false |
| != | 4/3!=7-5 | True |
| < | 8<19 | True |
| <= | 3\*3<=3\*4 | True |
| > | 9>23 | False |
| >= | 3\*2>=3\*4 | False |





**Logical Expression:** The expression that result into 1 (true) or 0 (false) are called logicalexpressions they use logical operators.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Meaning** | **Value** | **Expression** | **Result** |
| ! | negation | Num=3 | !num | False |
| && | AND | m=69 | (m>=35)&&(m<=100) | True |
| || | OR | ch=‟h‟ | (ch>=‟a‟ || ch<=‟z‟) | true |

**TYPE CONVERSIONS:**

C can perform conversion between different data types, but a programmer should take care with these conversions.

1. Implicit type conversion
2. Explicit type conversion
3. **Implicit type conversion:**

When we perform arithmetic manipulation on operands belonging to different data types, they may undergo type conversion automatically before evaluating the final value.

**Example 1:**

#include<stdio.h>

#include<conio.h> void main()

{

int a=5; float b; clrscr(); b=a;

printf("\n the value of a=%f",b); getch();

}

**Output:**



**Example 2:**

#include<stdio.h>

#include<conio.h> void main()

{

int a;

float b=45.67; clrscr();

a=b;

printf("\n the value of a=%d",a); getch();





}

**Output:**



1. **Explicit Conversion (Type casting):**

An explicit type conversion is user-defined that forces an expression to be of specific type. Variable may be converted explicitly by type casting.

**Example:**

#include<stdio.h>

#include<conio.h> void main()

{

int a=5; float x; clrscr(); x=(float)a;

printf("\n the value of x=%f",x); getch();

}

**Output:**



**Evaluation of Arithmetic expression:**

The operators in C are evaluated in a specific order. This is known as precedence (hierarchy) of operators.

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedence Level** | **Operators** | | **Associativity** |
| 1 | ( | ) | Left to right |
| 2 | ++ | -- | Left to right |
| 3 | % | \* / | Left to right |
| 4 | + | - | Left to right |

**BODMAS Rule:**

* Brackets will have the first precedence
* Of operator will have the second precedence
* Division operator will have the third precedence
* Multiplication operator will have the fourth precedence
* Addition operator will have the fifth precedence
* Subtraction operator will have the sixth precedence

**Example 1**: 6\*(2+3)/5=6\*5/5 =6\*1 =6





**Example 2**: x=a-b/3+c\*2-1 let a=9, b=12 and c=3X=9-12/3+3\*2-1

=9-4+3\*2-1 =9-4+6-1 =5+6-1 =11-1 =10

**Example 3:** C program to illustrate the use of variables in expressions and theirevaluation.

#include<stdio.h>

#include<conio.h> void main()

{

float a,b,c,x,y,z; clrscr();

a=9; b=12; c=3; x=a-b/3+c\*2-1; y=a-b/(3+c)\*(2-1); z=a-(b/(3+c)\*2)-1; printf(“x=%f\n”,x); printf(“y=%f\n”,y); printf(“z=%f\n”,z); getch();

}

**Output:**





|  |  |
| --- | --- |
|  | **CONTROL STRUCTURE** |
| Structured high level languages such as C contain statements that can alter the flow of | |
| a sequence of instructions. These statements are called as control statements. | |
| There 3 types of statements |  |
| A. Selection statement |  |
| B. Looping statement | Conditional control statements |
| C. Jumping statement | Unconditional control statements |



**Selection statement:**

It is the ability of a programming language to allow the user to decide on the execution of a certain set of statements based on a requirement or condition. These constructs are also called conditional constructs.

They are used to indicate decision in a program

Selection structure can be used to implement the following statements 1) Simple-if 2) if-else 3) Nested-if 4) if- else if 5) switch

1. **Simple-if:**

The if statement is the simplest form of selection statement. It is very frequently used

in decision making and altering the flow of program execution. **Syntax:** if (test condition)

Statement;

**Action:** 1. Test condition is checked.

2. if test condition true, statement is executed **Example:** C program to print bigger of two numbers

#include<stdio.h>

#include<conio.h> void main()

**{**

int a,b,big; clrscr();

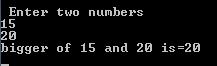
printf(“\n Enter two numbers”); scanf(“%d%d”,&a,&b);

big=a; if(b>big) big=b;

printf(“bigger of %d and %d is=%d\n”,a,b,big); getch();

}

**Output:**





1. **If-else:**

**Syntax:** if (test condition)Statement A;

else

Statement B;

**Action**: 1. Test condition is checked.

* 1. If test condition is true, statement A is executed.
  2. If test condition is false A is skipped, statement B is executed. **Example**: C program to check the given number is even or Odd.

#include<stdio.h>

#include<conio.h> void main()

{

int num; clrscr();

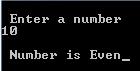
printf(“\n Enter a number”); scanf(“%d”,&num); if(num%2==0)

printf(“\n Number is Even”); else

printf(“\n Number is Odd”); getch();

}

**Output:**



**3. Nested-if:**

The if statement may itself contain another if statement is known as nested if. **Syntax:** if (test condition 1)

if (test condition 2) Statement A;

else

Statement B;

else

if (test condition 3) Statement C;

else

Statement D;





**Action:** 1. Test condition 1 is checked.

1. If condition 1 is true, condition 2 is checked. If condition 2 is true, statement A is executed if condition 2 is false, statement B is executed.
2. If condition 1 is false, condition 3 is checked. If condition 3 is true, statement C is executed if condition 3 is false, statement D is executed.

**Example:** C program to find the biggest of three numbers.#include<stdio.h>

#include<conio.h> void main()

{

int a,b,c,big; clrscr();

printf(“\n Enter 3 numbers”); scanf(“%d%d%d”,&a,&b,&c); if(a>b)

if(a>c)

big=a;

else

big=c;

else

if(b>c)

big=b;

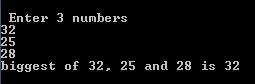
else

big=c;

printf(“biggest of %d, %d and %d is %d\n”,a,b,c,big); getch();

}

**Output:**



1. **If else-if:**

**Syntax:** if (test condition 1)Statement A;

else if (test condition 2) Statement B;

else if (test condition 3) Statement C;

else

Statement D;





**Action:** 1. Test condition 1 is checked, if it is true A is executed.

* 1. if condition 1 is false, condition 2 is checked, if condition 2 is true B is

executed.

* 1. if condition 2 is false, condition 3 is checked, if condition 3 is true C is

executed.

1. if all the conditions are false, statement D is executed.

**Example:** C Program to find the grade of the student marks.#include<stdio.h>

#include<conio.h> void main()

**{**

int marks; clrscr();

printf(“\n Enter marks”); scanf(“%d”,&marks); if(marks<35)

printf(“fail”); else if(marks<50)

printf(“pass class”); else if(marks<60)

printf(“second class”); else if(marks<70)

printf(“first class”);

else

printf(“Distinction”); getch();

}

**Output:**



1. **Switch: (Multiple-selection statement) Syntax:**

Switch (control expression)

{

Case value 1: {statement A;

Break;

}

Case value 2: {Statement B; Break;

}

----------

----------





Case value n :{ statement C; Break;

}

Default: {default statement;

}

}

**Action:** 1**.** Control expression may be integer or character. This is compared with thecase value 1, case value 2 etc in the given order.

1. If control expression=case value 1 then statement A is executed.
2. If control expression= case value 2 then statement B is executed.
3. If control expression is not equal to any of the case values 1,2,3,...n then default statement is executed.

**Example:** C program to perform arithmetic operations.#include<stdio.h>

#include<conio.h>

#include<stdlib.h> void main()

{

int a,b,c,ch; clrscr();

printf("\n enter two numbers"); scanf("%d%d",&a,&b); printf("1.add\n2.sub\n3.mul\n4.div\n5.exit"); printf("\n enter u r choice"); scanf("%d",&ch);

switch(ch)

{

case 1: c=a+b;

printf("\n sum=%d",c); break;

case 2: c=a-b;

printf("\n sub=%d",c); break;

case 3: c=a\*b;

printf("\n mul=%d",c); break;

case 4: c=a/b;

printf("\n div=%d",c); break;

case 5: exit(0);

default: printf("\n enter proper choice");

}

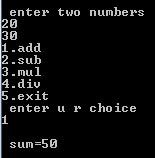
getch();

}





**Output:**



**Looping Statement:**

The concept of loops is fundamental to structured programming. A loop is a program construct that causes a statement to be executed again and again. The process of repeating the execution of a certain set of statements again and again is termed as looping.

C has several such statements that can be used to form loops they are.

1. **while** statement
2. **do-while** statement
3. **for** statement
   1. **while statement:**

This structure is also called as the “pre-tested” looping statement.

**Syntax: while (**test condition**)**

{

Statement 1;

Statement 2;

...........

}

Statement n+1;

**Example**: C program to find sum of the digits using while loop.#include<stdio.h>

#include<conio.h> void main()

{

int sum=0,count=1; clrscr(); while(count<=10)

{

sum=sum+count;

count++;

}

printf(“Total sum=%d”,sum); getch();

}

Output:





1. **do-while statement**

This structure is also called as the “post-tested” looping statement.

**Syntax: do**

**{**

Statements 1;

Statements 2;

..........

**} while(**test condition**);** Statements n+1;

**Example:** C program to find product of the digits using do-while loop.#include<stdio.h>

#include<conio.h> void main()

{

int prod=1,count=1; clrscr();

do

{

prod=prod\*count;

count++;

} while(count<=10) printf(“Total product=%d”,prod); getch();

}

**Output:**



1. **for statement**

**for** statement is also called as the “fixed execution” looping structure.

**Syntax: for(**Expression\_1;Expression\_2;Expression\_3)

{

Statements\_1;

Statements\_2;

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

}

Statement\_n+1;

1. **Expression\_1** represents the initialization expression.
2. **Expression\_2** represents the expression for the final condition.
3. **Expression\_3** represents the increment of decrement expression.

**Example:** C program to print the n elements using for loop.





#include<stdio.h>

#include<conio.h> void main()

{

int i,n; clrscr();

printf(“\n Enter n value”); scanf(“%d”,&n); for(i=0;i<=n;i++) printf(“%d\n”,i);

getch();

}

**Output:**



**Jumping statement:**

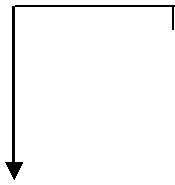
The jump statements unconditionally transfer program control within a function. There are 5 types of jumping statement

* goto
* break
* continue
* exit()
* return

**1. goto statement:**

The goto statement is a simple statement, used to transfer control from one point in a program to any other point in that program.

**Example:**



goto **label**;

.................

.................

**label**: statement;





**Example:** C program to demonstrate the working of goto statement.

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

{

float num,average,sum; int i,n;

clrscr();

printf("Maximum no. of inputs:"); scanf("%d",&n); for(i=1;i<=n;++i)

{

printf("Enter n%d: ",i); scanf("%f",&num); if(num<0.0)

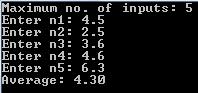
goto jump; sum=sum+num;

}

jump: average=sum/(i-1); printf("Average: %.2f",average); getch();

}

**Output:**



**2. break statement:**

The break statement is used to terminate loops. It can be used within a while, do-while or for statement.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Example:** | while(.....) | | |  |
|  | { |  |  |  |
|  | .......... |  |  |  |
|  | .......... |  |  |  |
|  | **break**; |  |  |  |
|  |  |  |  |
|  | .......... |  |  |  |
|  | } |  |  |  |
|  | Next statement; | |  |  |
|  |  |  |



**Example:** C program to demonstrate the working of break statement.





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

{

float num,average,sum; int i,n;

clrscr();

printf("Maximum no. of inputs: "); scanf("%d",&n); for(i=1;i<=n;++i)

{

printf("Enter n%d: ",i); scanf("%f",&num); if(num<0.0)

break;

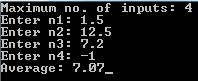
sum=sum+num;

}

average=sum/(i-1); printf("Average: %.2f",average); getch();

}

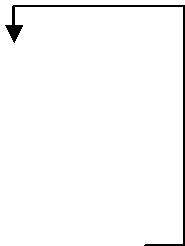
**Output:**



**3. continue statement:**

In some programming situation we want to take the control back to the beginning of the loop, bypassing the statements inside the loop, which have not yet been executed.

**Example:**



while(......)

{

...........

...........

Continue;

...........

}

Next statement;

**Example:** C program to find the product of 4 integers entered by a user. If user enters0 skip it.





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

{

int i,num,product; for(i=1,product=1;i<=4;++i)

{

printf(“Enter num%d:”,i); scanf(“%d”,&num); if(num==0)

continue;

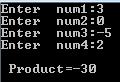
product\*=num;

}

printf(“\n Product=%d”,product); getch();

}

**Output:**



**4. exit() function:**

exit() is a standard library function that comes ready-made with the C compiler. Its purpose is to terminate the execution of the program.

**5. return statement:**

The return statement is used to return from a function.





**ARRAYS**

An array is defined as an ordered list of homogenous data elements stored in consecutive memory locations.

OR

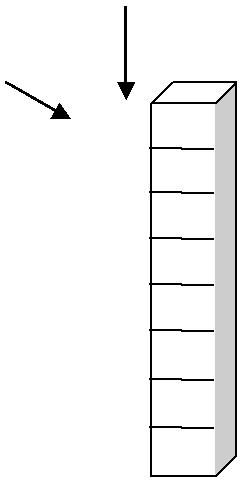
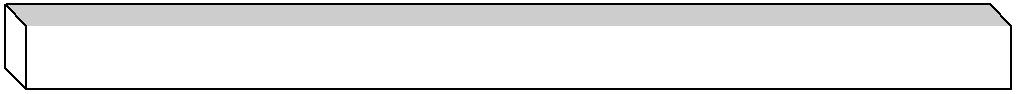
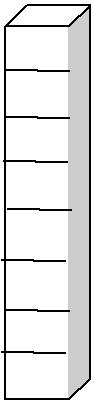
An array is a collection of similar type of elements stored in consecutive memory locations.

**Need for Arrays:**

If a large amount of data is to be stored, it would be a long and tedious task to think of a separate variable name for each datum, and then to type the declaration for each one of these variables. **Arrays** are the solution to this problem. They are such an important part of the art of programming that they are included in every high level language.

* To reduce program length.
* Easy to access.
* Make faster execution.
* To store large volume of data for similar type.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Index |  |  |  |  |
| **Array Representation:** | | |  |  |  |  |  |  |  |
| Array name |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **A[0]** | 2 |  |  |  | **A[7]** | 5 |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| **A[1]** | 3 |  |  |  | **A[6]** | 3 |  |  |  |
| **A[2]** | 4 |  |  |  | **A[5]** | 4 |  |  |  |
| **A[3]** | 5 |  |  |  | **A[4]** | 6 |  |  |  |
| **A[4]** | 6 |  |  |  | **A[3]** | 8 |  |  |  |
| **A[5]** | 7 |  |  |  | **A[2]** | 9 |  |  |  |
| **A[6]** | 8 |  |  |  | **A[1]** | 1 |  |  |  |
| **A[7]** | 9 |  |  |  | **A[0]** | 8 |  |  |  |
| **a) Vertical representation** | | | **b) Building representation** | | | | | |  |



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 23 | 45 | 90 | 35 | 40 | 22 | 85 | 65 | 48 |  | 50 | 59 | 60 | 65 |  |
|  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |
| **A[0] a[1] a[2]** | | | | **a[3]** | **a[4] a[5]** | | **a[6]** | **a[7]** | **a[8] a[9] a[10] a[11] a[12]** | | | | | | |
|  |  |  |  | **c) Horizontal representation** | | | | | | |  |  |  |  |  |





**Types of Arrays:**

Arrays are of different types-

1. **One-dimensional array:** A structured collection of elements, all of the same typethat is given a single name. Each element is accessed by an index that indicates the components position within the collection.
2. **Two-dimensional array:** A two dimensional array is a collection of elements, allof the same type, structured in two dimensions. Each element is accessed by a pair of indices that represent the elements position in each dimension.
3. **Multi-dimensional array:** A collection of elements, all of the same type, orderedon N dimensions (N>=1). Each element is accessed by N indices, each of which represents the elements position within that dimension.

**One-dimensional array (1D):**

A **one-dimensional array** is a structured collection of elements that can be accessed individually by specifying the position of an element with a single index value.

**Declaration of array (1D):**

Data\_type Array\_name[size];

**Example:** int a[20];

float height[5]; char name[30];

**Array Initialization:**

There are two types of array initialization.

1. Compile time initialization
2. Run time initialization

**Compile time initialization:**

We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.

**Syntax:** type array\_name [size] = {list of values}; **Example**: int num[5]={2,5,6,4,6};

**Run time initialization:**

An array can be explicitly initialized at run time. This approach is usually applied for initializing large arrays**.**

**Example:** int x[3]; scanf(“%d%d%d”,&x[0],&x[1],&x[2]);

Will initialize array elements with the values entered through the keyboard.





**Example 1:** C program to read and print one-dimensional array.#include<stdio.h>

#include<conio.h> void main()

{

int a[50],i,n; clrscr();

printf(“\n Enter size of elements”); scanf(“%d”,&n);

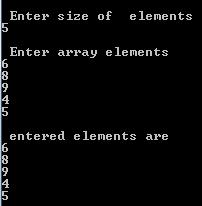
printf(“\n Enter array elements”); for(i=0;i<n;i++) scanf(“%d”,&a[i]);

printf(“\n entered elements are\n”); for(i=0;i<n;i++) printf(“%d\n”,a[i]);

getch();

}

**Output:**



**Example 2:** C program to find the largest element in an array and position of its occurrence.

#include<stdio.h>

#include<conio.h> void main()

{

int a[100];

int large,pos,num,i; clrscr();

printf(“\n enter number of elements”); scanf(“%d”,&num); for(i=0;i<num;i++) scanf(“%d”,&a[i]);

large=a[0];

pos=0;





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

{

if(large<a[i])

{

large=a[i];

pos=i;

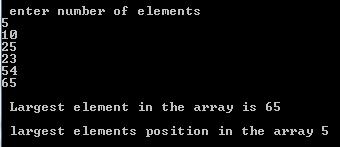
}

}

printf(“\n Largest element in the array is %d\n”,large); printf(“\n largest elements position in the array %d\n”,pos+1); getch();

}

**Output:**



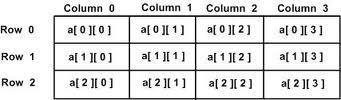
**Two-dimensional array (2D):**

A two dimensional array is used to represent elements in a table with rows and columns, provided each element in the table is of the same type.

**Two-dimensional array declaration:**

Data\_type array\_name[row size][column size]; **Example:** int A[10][10];

A is a two dimensional array which contains 10 rows and 10 columns.



**Two-dimensional array initialization**

Similar to the one dimensional array, we can initialize a two-dimensional array either by initializing it in its declaration or by using assignment statements.





**Example:**

int matrix[2][3]={1,2,3,4,5,6};

int a[3][3]={{1,2,3},{4,5,6},{7,8,9}}; **Unsized**: int sales[][4]={1,2,3,4,6,1,4,5,3,4,5,6};

**Accessing elements of an array:**

Initializing an array in its declaration is impractical if the array is large. For a 100-row by 100-column array, it is very difficult to list 10,000 values. If the values are all different, input them into array at runtime.

Consider the following declaration: int a[3][3];

**Read the 2D array and store them in the array a.** for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

scanf(“%d”,&a[i][j]);

}

}

**Print the 2D array and store them in the array a.** for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

printf(“%d”,a[i][j]);

}

}

**Example 1:** C program to read and display a square matrix.#include<stdio.h>

#include<conio.h> void main()

{

int a[10][10],m,n,i,j; clrscr();

printf(“\n Enter order of the matrix”); scanf(“%d%d”,&m,&n);

printf(“\n enter matrix elements”); for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

scanf(“%d”,&a[i][j]);

}

}





printf(“matrix A is \n”); for(i=0;i<m;i++)

{

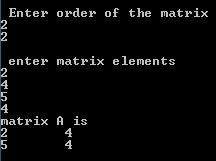
for(j=0;j<n;j++) printf(“%d\t”,a[i][j]); printf(“\n”);

}

getch();

}

**Output:**



**Example 2:** Program to add and subtract two MxN matrices#include<stdio.h>

#include<conio.h> void main()

{

int a[10][10],b[10][10],c[10][10],i,j,m,n; clrscr();

printf("Enter m and n for matrix size\n"); scanf("%d%d",&m,&n);

printf("Enter the First Matrix :"); for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

scanf("%d",&a[i][j]);

}

printf("Enter the Second Matrix : "); for(i=0;i<m;i++)

{

for(j=0;j<n;j++)





scanf("%d",&b[i][j]);

}

printf("\nThe Sum of Matrices is : \n"); for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

c[i][j]=a[i][j]+b[i][j]; printf("%d\t ",c[i][j]);

}

printf("\n");

}

printf("\n\nThe Difference of Matrices is : \n"); for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

c[i][j]=a[i][j]-b[i][j]; printf("%d\t ",c[i][j]);

}

printf("\n");

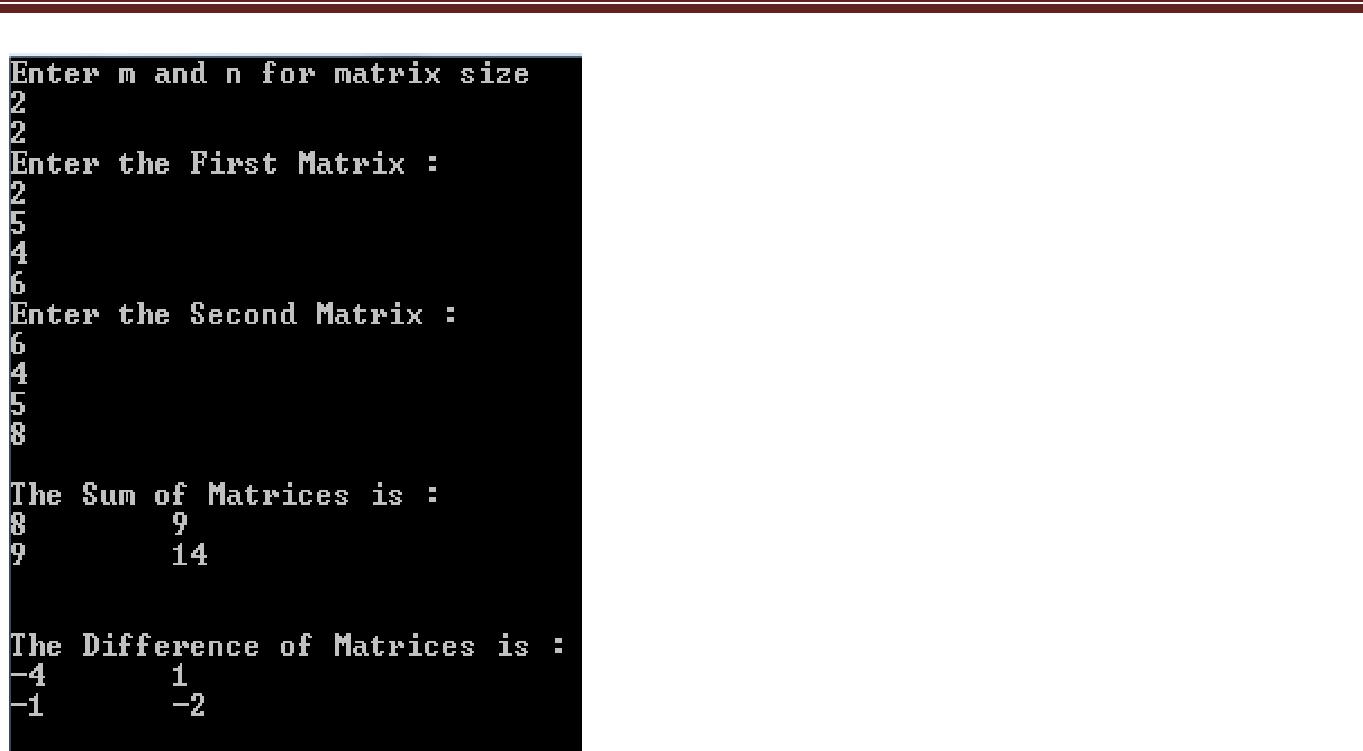
}

getch();

}

**Output:**





**Example 2:** Program to multiply two MxN matrices#include<stdio.h>

#include<conio.h> void main()

{

int a[10][10],b[10][10],c[10][10],i,j,k,m,n; clrscr();

printf("Enter the size of the matrix \n"); printf("as m\*n\n");

scanf("%d",&m);

scanf("%d",&n);

printf("Enter the First matrix\n"); for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

scanf("%d",&a[i][j]);

}

}

printf("Enter the Second matrix\n"); for(i=0;i<m;i++)





{

for(j=0;j<n;j++)

{

scanf("%d",&b[i][j]);

}

}

printf("The First matrix is\n"); for(i=0;i<m;i++)

{

printf("\n");

for(j=0;j<n;j++)

{

printf("%d\t",a[i][j]);

}

}

printf("\nThe Second matrix is\n"); for(i=0;i<m;i++)

{

printf("\n");

for(j=0;j<n;j++)

{

printf("%d\t",b[i][j]);

}

}

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

{

for(j=0;j<n;j++)

{

c[i][j]=0;

for(k=0;k<n;k++)

{

c[i][j]=c[i][j]+a[i][k]\*b[k][j];

}





}

}

printf("\nThe multiplication of two matrix is\n"); for(i=0;i<m;i++)

{

printf("\n");

for(j=0;j<n;j++)

{

printf("%d\t",c[i][j]);

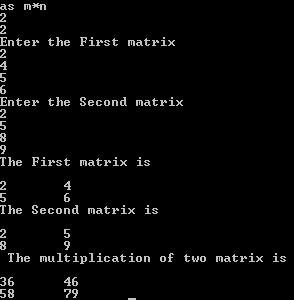
}

}

getch();

}

**Output:**



**Multi-dimensional array:**

**Syntax:** type array\_name[s1][s2][s3]......[sm];Where si is the size of the ith dimension.

**Example:**

int p[2][3][3][5];





**STRING HANDLING**

**String:**

A string is a sequence of characters.

**Declaration of string arrays:**

**Syntax**: char variable\_name [size];

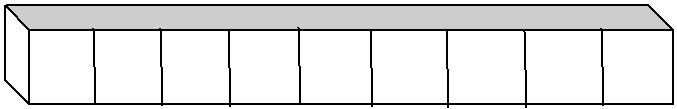
**Example**: char name [20];

**STRING ARRAY INITIALIZATION**

Once you declare an array, you can enter its value using **scanf ()** or **gets ().** You can also assign the array initial values when you declare it.

**Example:** consider the following string initialization.

**Case 1**:char name [9] =”COMPUTER”;



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | O | M | P | U | T | E | R | \0 |
| name [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] |
| **Case 2:** | char name [] =”Computer”; | | | | |  |  |  |

**OR**

**Case 3:** char name [9] = {‘c’,’o’,’m’,’p’,’u’,’t’,’e’,’r’,’\0’};

**READING A STRING:**

To read a string, we use a standard I-O library function **scanf ().** Scanf () reads formatted input from the standard input device, which is usually the keyboard. We can read values into a string with scanf () function using **%s** formatting code:

scanf (“%s”,name);

There is a problem with **scanf ()** function while reading a string data. Consider the following program.

**Program 1:** A C program to read a string using **scanf ()** function.#include <stdio.h>

#include<conio.h> void main()

{

char name[20]; clrscr();

printf(“Enter a string\n”); scanf(“%s”,name);

printf(“\n Entered string was %s”, name);





getch();

}

**Output:**



The input string was **sri nagar** but the output result is **sri**, this is because of the blank space. The scanf () function accepts characters one at a time until it encounters a blank space or newline character. To avoid such error, c provides another function called **gets()**. The **gets()** function accepts an entire line of input from the standard input device.

**Program 2:** A C program to read a string using **gets()** function.#include <stdio.h>

#include<conio.h> void main()

{

char name[20]; clrscr();

printf(“Enter a string\n”); gets(name);

printf(“\n Entered string was %s”, name); getch();

}

**Output:**



**WRITING (PRINTING) A STRING:**

puts() and printf() functions are used to display a string on the standard output device.

**Program 2:** A C program to read and write a string using **gets()** and **puts()** function.#include <stdio.h>

#include<conio.h> void main()

{

char name[20]; clrscr();

puts(“Enter a string\n”); gets(name);

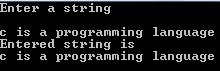
puts(“\n Entered string is:”); puts(name);

getch();



}

**Output:**



**STRING OPERATIONS:**

The C library supports a large number of string handling functions that can be used to carry out many of the string manipulations such as

* Length (number of characters in a string)
* Concatenation (Adding two or more strings)
* Comparing two strings
* Substring (Extract substring from the given string)
* Copy (copies one string over another)

**Note:** To use all the string operations in C you must include **string.h** library header file inthe program

**STRING HANDLING FUNCTIONS:**

|  |  |
| --- | --- |
| **FUNCTION** | **DESCRIPTION** |
| Strcat() | Adds the characters of one string to another. |
| Strcmp() | Compares two strings considering case. |
| Strcmpi() | Compare two strings; not case sensitive. |
| Strcpy() | Copies one string or string literal, to another. |
| Strlen() | Calculates the string length. |
| Strlwr() | Converts a string to lowercase. |
| Strupr() | Converts a string to uppercase. |
| Strchr() | Returns the position of a specified character in the string. |
| Strcspn() | Returns the position of a character in the string from a specified character |
|  | set. |
| Strncat() | Appends specified characters from one string to another. |
| Strncmp() | Compares specified characters of two strings. |
| Strncpy() | Copies specified characters from one string to another. |
| Strnset() | Changes specified characters in a string to another character. |
| Strrev() | Reverses the characters in a string. |
| Strstr() | Finds one string within another. |

**1. strlen() FUNCTION**

This function counts and returns the number of characters in a string. The length does not include the NULL character.





**Syntax:** n=strlen(string);

Where n is an integer variable which receives the value of the length of

the string.

**Example1:** write a c program to find the length of a string.

#include<stdio.h>

#include<string.h>

#include<conio.h> void main()

{

char name[100]; int length; clrscr();

printf(“enter a string\n”); gets(name); length=strlen(name);

printf(“\n Number of characters in the string is %d”,length); getch();

}

**Output:**



**Example2:** write a c program to find the frequency of occurrence of a specified character.

#include<stdio.h>

#include<string.h>

#include<conio.h> void main()

{

char text[100],ch; int length,i,count=0; clrscr();

printf(“enter a string\n”); gets(text);

puts(“Enter a character to be searched”); scanf(“%c”,&ch);

length=strlen(text);

for(i=0;i<length;i++) if(text[i]==ch) count++;

if(count)





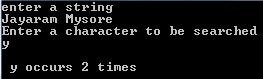
printf(“\n %c occurs %d times”,ch,count);

else

printf(“\n %c does not exist in the text”,ch); getch();

}

**Output:**



**2. strcpy() FUNCTION:**

C does not allow you to assign characters to a string directly, as in the statement

char name[30]; name=”NIE”;

Instead, use the strcpy() function found in most compilers. The syntax of the function is illustrated below:

**Syntax:** strcpy(string1,string2);

Assign the contents of string2 to string1. String2 may be a character array variable or a string constant.

**Example:** strcpy(name,”NIE”);

**Example1:** write a c program to copy one string to another.

#include<stdio.h>

#include<string.h>

#include<conio.h> void main()

{

char name[100], city[100]; clrscr();

printf(“enter a string\n”); gets(name); strcpy(city,name);

printf(“\n copied string is %s”,city); getch();

}

**Output:**



strcat(string1,string2);

strcpy(st1,”sri”); strcpy(st2,”nagar”); printf(“%s”,strcat(st1,st2));



1. **strcat() FUNCTION:**

When you combine two strings, you add the characters of one string to the end of

another string. The process is called concatenation. The strcat() function joins two strings

together.

**Syntax:**

**Example:**

**Example1:** write a c program to concatenate two strings.

#include<stdio.h>

#include<string.h>

#include<conio.h> void main()

{

char st1[100], st2[100]; clrscr();

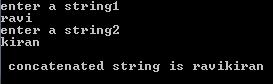
printf(“enter a string1\n”); gets(st1);

printf(“enter a string2\n”); gets(st2);

printf(“\n concatenated string is %s”,strcat(st1,st2)); getch();

}

**Output:**



**4. strcmp() FUNCTION:**

In C you cannot directly compare the value of two strings in a condition Like this:

If(st1==st2)

Instead, use the strcmp() function found in most compilers. The syntax of the function is illustrated below:

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

String1 and string2 may be string variables or string constants. Some compilers return a negative if the string1 is alphabetically “less than” the second, and a positive number if the first string is greater than the second.

**Example1:** strcmp(“MYSORE”,”MYSORE”) will return 0 because two strings are

equal





**Example2:** strcmp(“their”,”there”)will return -9 which is the numeric differencebetween ASCII “i” and ASCII “r”.

**Example3:** strcmp(“there”,”their”)will return 9 which is the numeric differencebetween ASCII “r” and ASCII “i”.

**Example4:** strcmp(“THE”,”the”)will return -32 which is the numeric differencebetween ASCII “T” and ASCII “t”.

**Example5:** write a c program to compare two strings.

#include<stdio.h>

#include<string.h>

#include<conio.h> void main()

{

char st1[100], st2[100]; int n;

clrscr();

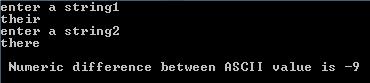
printf(“enter a string1\n”); gets(st1);

printf(“enter a string2\n”); gets(st2); n=strcmp(st1,st2);

printf(“\n Numeric difference between ASCII value is %d”,n); getch();

}

**Output:**



1. **strcmpi() FUNCTION**

This function is same as strcmp(), which compares two strings, but is not case

sensitive.

**Example1:** write a c program to compare two strings using strcmpi().

#include<stdio.h>

#include<string.h>

#include<conio.h> void main()

{

char st1[100], st2[100]; int n;

clrscr();

printf(“enter a string1\n”);





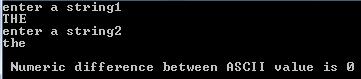
gets(st1);

printf(“enter a string2\n”); gets(st2); n=strcmpi(st1,st2);

printf(“\n Numeric difference between ASCII value is %d”,n); getch();

}

**Output:**



**6. strlwr() FUNCTION:**

This function converts all uppercase characters in a string to lowercase **Syntax:** strlwr(string);

**Example1:** write a c program to convert uppercase character to lowercase character.

#include<stdio.h>

#include<string.h>

#include<conio.h> void main()

{

char name[100]; clrscr();

strcpy(name,”COMPUTER”);

printf(“\n Converted string is %s”,strlwr(name)); getch();

}

**Output:**



**7. strupr() FUNCTION:**

This function converts all lowercase characters in a string to uppercase

|  |  |  |
| --- | --- | --- |
| characters. |  |  |
| **Syntax:** | strupr(string); |  |
| **Example1:** write a c program to convert lowercase character to uppercase character. | | |
| #include<stdio.h> |  |  |
| #include<string.h> |  |  |
| #include<conio.h> |  |  |
| void main() |  |  |
| { |  |  |
|  | |  |



char name[100]; clrscr();

strcpy(name,”computer”);

printf(“\n Converted string is %s”,strupr(name)); getch();

}

**Output:**



**8. strrev() FUNCTION:**

This function reverses the characters in a string.

**Syntax:** strrev(string);

**Example1:** write a c program to check whether a given word is palindrome or not.

#include<stdio.h>

#include<string.h>

#include<conio.h> void main()

{

char st1[100], st2[100]; clrscr();

printf(“Enter a string”); scanf(“%s”,st1); strcpy(st2,st1); strrev(st2); if(strcmp(str1,str2)==0)

printf(“%s is a palindrome”,st1);

else

printf(“%s is not a palindrome”,st1); getch();

}

**Output:**



**ARRAY OF STRINGS**

An array of strings is a two-dimensional array. The size of first index (rows) determines the number of strings and the size of second index (columns) determines maximum length of each string.

**Example:** consider the following array declaration



char name[8][20];

Declares an array of 8 strings, each of which can hold maximum of 20 characters. An array of strings appears in memory as shown in figure.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | 11 | | 12 13 | | 14 | | 15 16 | | | 17 | | 18 | | 19 | |  |
| name[0] |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D | H | A | R | W | D | \0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| . |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| G | U | L | B | A | R | G | A | \0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| . |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| . | S | H | I | M | O | G | A | \0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| . |  |  |  |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C | H | I | K | A | M | A | G | A | L | | U | | R | \0 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| . |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| M | Y | S | O | R | E | \0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| . |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| . | B | A | N | G | A | L | O | R | E | \0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| . | K | A | R | A | W | A | R | \0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| . |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| H | A | S | S | A | N | \0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Name[7] |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**Example**: A C program to arrange N names in alphabetical order.#include<stdio.h>

#include<string.h>

#include<conio.h> void main()

{

char name[10][20]; char tempname[20]; int n,i,j;

clrscr();

printf(“Enter number of names\n”); scanf(“%d”,&n);

printf(“Enter list of names\n”); for(i=1;i<n;i++)

gets(name[i]);

for(i=1;i<n;i++) for(j=0;j<n-i;j++)

if(strcmp(name[j],name[j+1])>0)

{

strcpy(tempname,name[j]);

strcpy(name[j],name[j+1]);

strcpy(name[j+1],tempname);

}

printf(“\n Sorted names”); for(i=0;i<n;i++) printf(“\n%s”,name[i]); getch();

}





**Output:**





**FUNCTIONS**

**MODULE:** A self contained collection of steps that solves a problem or sub-problem.

Large programs can be made manageable by dividing it into smaller sub-programs or modules. These sub-programs are called as **functions**.

OR

A function is a named unit of a group of program statements designed to perform a specific task and returns a single value.

C supports two types of functions. They are

1. Built-in or library functions.
2. User defined functions.
3. **Built-in or library functions**:

A function defined by the C compiler.

**Example**: sqrt(), strcpy(), pow(), cos() and so on.

1. **User defined functions:**

A function defined by the user.

**Example:** sum(), mul(), search(),read() and so on.

**NEED OF FUNCTIONS:**

1. **Repetition:** If there is a portion of the program, which is to be repeated in a numberof places in the program, a function may be used to avoid rewriting the same sequence of code at two or more, locations in the program. This is especially useful if the code involved is long or complicated.
2. **Universal use:** Some tasks might be needed in more than one program or a largegroup of programmers may need the same task. By writing a function and making it available to others the duplication of effort can be avoided.
3. **Modularity:** A program for complex problems can be developed in a modular fashionby dividing the problem into modules, or functions, each function performing a definite task. Since the functions are independent of each other, each one of them can be developed and tested without worrying about the details of the other modules.
4. **Teamwork**: If a program is divided into sub-programs, a number of members canform a team and each person in the team can develop a separate sub-program, rather than having a single person work on the complete program.

**USE OF FUNCTIONS:**

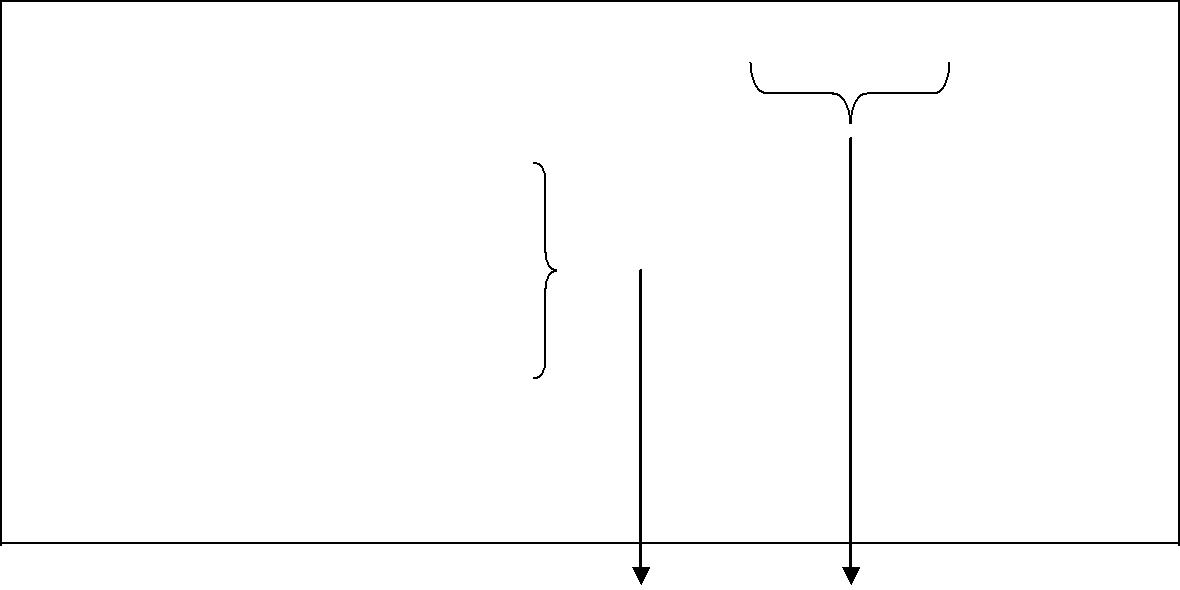
* Programs with functions are compact thus decreasing the number of lines of code to a very large extent.
* Debugging the code is much simpler because errors can be localized and corrected easily.





* Functions increase the readability of the program and helps in proper and good documentation.
* Many other programs may use a function, thus helping us to create our own libraries.

**FUNCTION DEFINITION OR SYNTAX**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Return\_type\_specifier function\_name (type arg1, type arg2............** | | | **)** |  |
| **{** |  |  |  |  |
| **Local variable declarations;** | | |  |  |
| **Statement 1;** | | |  |  |
| **Statement 2;** | | |  |  |
| **.......................** |  |  |  |  |
|  |  |  |  |
| **.......................** |  |  |  |  |
| **Statement n;** | | |  |  |
| **Return (Expression);** | | |  |  |
| **}** |  |  |  |  |

Body of the function Argument list with declaration

1. **Return\_type\_specifier**

The return\_type\_specifier identifies the type of value, which will be sent back after the function has performed its task. The return\_type\_specifier can include the normal data types such as **int**, **float**, **char** the data type **void** is used if the function does not return a value.

1. **Function\_name**

The function\_name thus helps us to uniquely identify and call a function.

1. **Argument list with declaration**

The argument list identifies the set of values, which are to be passed to the function from either the main program or a sub-program.

1. **Body of the function**

The body of the functions first includes the identification of those components, which will be used only within the function and not outside it. Thus those components are declared as local variables.

Next a function contains a set of executable statements, which will perform the required task for the user. The last executable statement of the function is the word





“**return**” with an expression, which contains the value that has to be sent back to the

main program.

**FUNCTION CALL:**

A statement that transfers control to a function. In c, this statement is the name of the function, followed by a list of arguments.

A function can be called by simply using the function name in a statement. A function call has the following **syntax**.

Variable=function\_name (arg1, arg2 ...);

Values are passed on to a sub-program by copying values from the actual paramenters (function call) to the formal parameters (function header)

**Formal parameter**: A variable declared in a function heading is also called formal argumentor formal parameter.

OR

The definition of a function or the function body also includes a set of parameters or arguments these arguments are called as formal arguments or dummy arguments.

**Actual Argument:** A variable or expression listed in a call to a function is also called actualargument or actual parameter.

OR

A function call includes the function name with some parameters or arguments, these arguments are known as actual arguments.

**Function prototypes:**

A function prototype is a declaration of the function that tells the program about the type of the value returned by the function and the number and type of arguments.

**Syntax:** return\_type\_specifier function\_name(); OR

Return\_type\_specifier function\_name(type1,type2,.....);



returns value



**Example 1:** C program to find the sum of the digits of a number until the resulting value is a

one digit number.

#include<stdio.h>

#include<conio.h>

int sum\_of\_digits(int);  function prototype

void main()

{

int num,sum,n;

clrscr();

printf(“Enter a number\n”);

scanf(“%d”,&num);

n=num;

while(n>=9)  calling function



{

sum=sum\_of\_digits(n);  n is an actual argument

n=sum;

}

printf(“the number is=%d\n”,num);

printf(“sum of digits until one digit=%d\n”,sum); getch();

}

int sum\_of\_digits(int n) n is a formal argument

{

int d,s=0; local variable

while(n>0)

{

d=n%10;

n=n/10;

s=s+d;

}

return(s);

}





**Output:**



**LOCAL AND GLOBAL VARIABLES:**

**Local variable:** A variable declared within a block and not accessible outside of the block

**Example:**

Function(int m, int n)

{

int a,b;  a and b are local variables

--------

--------

}

**Global variable:** A variable declared outside of all the functions in a program.

**Example:**

|  |  |
| --- | --- |
| int a,b; | a and b are global variables |
| main() |  |
| { |  |
| int p,q; | p and q are local variables |
| -------- |  |
| -------- |  |
| } |  |
|  |  |

**Example 2:** C program to find the largest of two numbers using the global variabledeclaration.

|  |  |  |  |
| --- | --- | --- | --- |
| #include<stdio.h> | |  | |
| #include<conio.h> | |  | |
| int num1,num2,lar;  int largest(); | | Global declaration | |
| void main() | |  | |
| { | |  | |
|  | |  | |
| clrscr(); | |  | |
| printf(“enter two numbers”); | |  | |
| scanf(“%d%d”,&num1,&num2); | |  | |
|  |  | | |  |
|  | | |  |  |
|  |  | |  |  |
| printf(“Largest of %d and %d is %d”,num1,num2,largest()); | | | |  |
| getch(); |  | | Calling function |  |
|  | |  |
| } |  | |  |  |
| int largest() | Called function or definition of the function | | |  |
| { |  | |  |  |



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

return(lar);

}

**Output:**



**CALL BY VALUE:**

A parameter that receives a copy of the value of corresponding argument.

**Example 1:** C program to illustrate the function using call by value mechanism.#include<stdio.h>

#include<conio.h> void main()

{

void function(int,int); int a=20,b=30; clrscr();

printf(“a=%d b=%d before function call \n”,a,b); function(a,b);

printf(“a=%d b=%d after function call\n”,a,b); getch();

}

void function(int x,int y)  call by value function

{

x=x+x;

y=y+y;

}





**Output:**



The corresponding formal parameter represents a local variable in the called function. The current value of the corresponding actual parameter becomes the initial value of the formal parameter. The value of formal parameter may then change in the body of the subprogram by assignment or input statements. This will not change the value of the actual parameter.

**TYPES OF FUNCTIONS OR CATEGORIES OF FUNCTION:**

1. Functions with no arguments and no return values.
2. Functions with arguments and no return values.
3. Functions with arguments and with return values.
4. Recursive functions.
5. **Functions with no arguments and no return values:**

In this type of function the first function calls the second function. However no arguments are passed to the second function. The second function performs its calculations but no value is sent back to the first function.

|  |  |  |  |
| --- | --- | --- | --- |
| **Calling Function** | **Action** | **Called Function** |  |
|  |  |  |  |
| Function1() |  | Function2() |  |
| { | No arguments | **{** |  |
| ---------------- |  | **----------------** |  |
| ----------------- |  | **----------------** |  |
| Function2(); |  | **----------------** |  |
| } | No return values | **}** |  |
|  |  |
|  |  |  |  |



**Example:** C program to illustrate a function with no arguments and no return values.

#include<stdio.h>

#include<conio.h> void main()

{

void Our\_address(); clrscr(); Our\_address(); getch();



void Our\_address()

{

printf(“\n JSS”); printf(“\n Mysore”);

}

**Output:**

JSS

Mysore

1. **Functions with arguments and no return values:**

In this type of function the first function calls the second function by passing some arguments or values to the second function. The second function performs its calculations but no value is sent back to the first function.

|  |  |  |  |
| --- | --- | --- | --- |
| **Calling Function** | **Action** | **Called Function** |  |
|  |  |  |  |
| Function1() |  | Function2(a) |  |
| { | Arguments | **{** |  |
| ---------------- |  | **----------------** |  |
| ----------------- |  | **----------------** |  |
| Function2(x); |  | **----------------** |  |
| } | No return values | **}** |  |
|  |  |
|  |  |  |  |



**Example:** C program to find the type of character entered using functions.#include<stdio.h>

#include<conio.h> void main()

{

void char\_type(char); char ch;

clrscr();

printf(“enter a character\n”); scanf(“%c”,&ch); char\_type(ch);

getch();

}





void char\_type(char c)

{

if((c>=‟a‟)&&(c<=‟z‟))

printf(“%c is a lower case alphabet\n”,c); else if((c>=‟A‟)&&(c<=‟Z‟))

printf(“%c is an upper case alphabet\n”,c); else if((c>=‟0‟)&&(c<=‟9‟))

printf(“%c is a digit\n”,c);

else

printf(“%c is a special character\n”,c);

}

**Output:**



1. **Functions with arguments and with return values:**

In this type of function the first function calls the second function by passing some arguments or values to the second function. The second function performs its calculations and returns the answer or a value back to the first function.

|  |  |  |  |
| --- | --- | --- | --- |
| **Calling Function** | **Action** | **Called Function** |  |
|  |  |  |  |
| Function1() |  | Function2(a) |  |
| { | Arguments | **{** |  |
| ---------------- |  | **----------------** |  |
| ----------------- |  | **----------------** |  |
| Function2(x); |  | **----------------** |  |
| } | Return values | return(e); |  |
|  |  |
|  |  | **}** |  |
|  |  |  |  |



**Example:** C program to find all prime numbers in the range N and M using functions.

#include<stdio.h>

#include<conio.h>

void main()

{

int a,b,g,l;





int gcd(int,int); clrscr();

printf(“Enter the numbers a and b\n”); scanf(“%d%d”,&a,&b);

g=gcd(a,b);

l=(a\*b)/g;

printf(“GCD of %d and %d=%d\n”,a,b,g); printf(“LCM of %d and %d=%d\n”,a,b,l); getch();

}

int gcd(int x,int y)

{

while(x!=y)

if(x>y)

x=x-y;

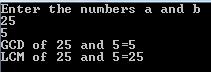
else

y=y-x;

return(x);

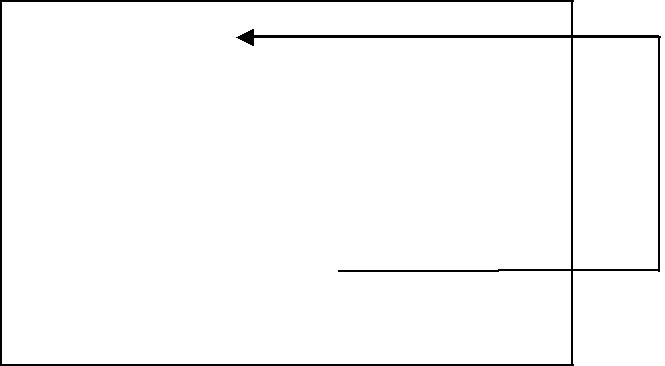
}

**Output:**



1. **Recursive functions:**

Functions that call itself directly or indirectly again and again is called recursive function.



Function1(a)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Function1(x);

}





**Example:** C program to find the factorial of the given number using recursive function.

#include<stdio.h>

#include<conio.h> void main()

{

int n;

int fact(int); clrscr();

printf(“enter any non-negative integer\n”); scanf(“%d”,&n);

if(n<0)

printf(“Invalid Input”);

else

printf(“factorial of %d is=%d\n”,n,fact(n)); getch();

}

int fact(int n)

{

if(n==0)

return(1);

else

return(n\*fact(n-1));

}

Output:



**FUNCTIONS AND ARRAYS:**

We can pass an entire array from one function to another. To pass an entire array to a function we will have to just pass the name of the array as the actual argument. When we pass a normal variable other than an array, c makes a copy of the data and places it in a memory location associated with the receiving variable.

**//Transpose of a matrix using functions**

#include<stdio.h>

#include<conio.h>

void display(int a[][20],int,int);

void input(int,int);

int a[20][20],b[20][20];

void main()

{

int m,n;

clrscr();

printf("Enter the order of Matrix\n");

scanf("%d%d",&m,&n);

printf("Enter the elements\n");

/\*OUTPUT

Enter the order of Matrix

2 3

Enter the elements

4 5 6

7 8 9

Entered matrix is

4 5 6

7 8 9

After Transpose

4 7

5 8

6 9 \*/

input(m,n);

printf("Entered matrix is\n");

display(a,m,n);

printf("After Transpose\n");

display(b,n,m);

}

void input(int m,int n)

{

int i,j;

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

{

for(j=0;j<n;j++)

{

scanf("%d",&a[i][j]);

b[j][i]=a[i][j];

}

}

}

void display(int x[][20],int p,int q)

{

int i,j;

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

{

for(j=0;j<q;j++)

{

printf("%d\t",x[i][j]);

}

printf("\n");

}

}

**STRUCTURES AND UNIONS**

**STRUCTURE:**

C provides a way to collect different types of data together into one entity, called a structure.

OR

It is a group of logically related data items of same type or different types.

**DIFFERENCE BETWEEN ARRAY AND STRUCTURE:**

|  |  |
| --- | --- |
| **Array** | **Structure** |
| Array is a collection of homogeneous data | Structure is a collection of heterogeneous |
|  | data |
| Array elements are referred by subscript | Structure elements are referred by unique |
|  | name |
| Array elements are accessed by its position | Structure elements are accessed by its objects |
| or subscript | as „.‟ Operator |
| Array is a derived data type | Structure is user defined data type |
| **Syntax:** | struct structure\_name |
|  | { |
| Data\_type array\_name[size]; | Data\_type member 1; |
|  | Data\_type member 2; |
|  | ---------------------------- |
|  | ---------------------------- |
|  | Data\_type member n; |
|  | }struct\_var\_name; |
| **Example:** | struct item |
| int rno[5]; | { |
|  | int rno; |
|  | char name[50]; |
|  | }itm; |

**DEFINITION AND DECLARATION OF A STRUCTURE OR SYNTAX:**

struct structure-name

{

Data\_type member\_name\_1;

Data\_type member\_name\_2;

Data\_type member\_name\_3;

...................................................

Data\_type member\_name\_n;

};





**DECLARATION OF A STRUCTURE:**

**Syntax:** Struct structure\_name variable;

**Instance 1:**

**struct** emp\_info

{

**char** emp\_id[10]; **char** nm[100]; **float** sal;

}emp;

**Instance 2:**

**struct** emp\_info

{

**char** emp\_id[10]; **char** nm[100]; **float** sal;

};

**struct** emp\_info emp;

**ACCESSING THE ELEMENTS OF A STRUCTURE:**

Structure members can be accessed using member operator '**.**' . It is also called as '**dot** **operator**' or '**period operator**'.

**Syntax:** structure\_variable\_name**.**member\_name

**Example:** C program to show the use of structures.

#include<stdio.h>

#include<conio.h> void main()

{

struct item

{

char name[20]; float price;

};

struct item i1,i2,i3,i4,i5; clrscr();

printf(“Enter the name and prices of four items\n”);

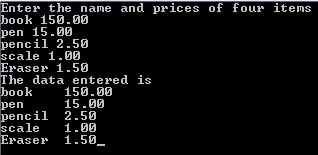




scanf(“%s%f”,i1.name,&i1.price); scanf(“%s%f”,i2.name,&i2.price); scanf(“%s%f”,i3.name,&i3.price); scanf(“%s%f”,i4.name,&i4.price); scanf(“%s%f”,i5.name,&i5.price); printf(“The data entered is \n”); printf(“%s\t%.2f\n”,i1.name,i1.price); printf(“%s\t%.2f\n”,i2.name,i2.price); printf(“%s\t%.2f\n”,i3.name,i3.price); printf(“%s\t%.2f\n”,i4.name,i4.price); printf(“%s\t%.2f”,i5.name,i5.price); getch();

}

**Output:**



**INITIALIZING STRUCTURE VARIABLE IN C:**

1. Memory is not allocated for un-initialized variable.
2. Let us discuss very familiar example of structure student, we can initialize structure variable in different ways

**Way 1: Declare and Initialize.**

struct student

{

char name[20]; int roll;

Float marks; }std1={“prakash”,67,78.5};





**Way 2: Declaring and Initializing multiple variables.**

struct student

{

char name[20]; int roll;

float marks;

}

std1={“prakash”,67,78.5}; std2={“ramesh”,75,86.4};

**STRUCTURES WITHIN STRUCTURES (NESTED STRUCTURES):**



Structures can be used as structures within structures. It is also called as 'nesting of structures'.

**Syntax:**

**struct** structure\_nm

{

<data-type> element 1; <data-type> element 2;

* - - - - - - - - - -
* - - - - - - - - - -

<data-type> element n;

**struct** structure\_nm

{

<data-type> element 1; <data-type> element 2;

* - - - - - - - - - -
* - - - - - - - - - -

<data-type> element n; }inner\_struct\_var;

}outer\_struct\_var;

**Example :**

struct stud\_Res

{

int rno;

char nm[50]; char std[10];

struct stud\_subj

{





char subjnm[30]; int marks;

}subj;

}result;

In above example, the structure stud\_Res consists of stud\_subj which itself is a

structure with two members. Structure stud\_Res is called as 'outer structure' while stud\_subj

is called as 'inner structure.' The members which are inside the inner structure can be

accessed as follow :

result.subj.subjnm

result.subj.marks

**Example 1:** Program to demonstrate nested structures.

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

**struct** stud\_Res

{

**int** rno; **char** std[10];

**struct** stud\_Marks

{

**char** subj\_nm[30]; **int** subj\_mark;

}marks;

}result;

**void** main()

{

clrscr();

printf("\n\t Enter Roll Number : "); scanf("%d",&result.rno); printf("\n\t Enter Standard : "); scanf("%s",result.std);

printf("\n\t Enter Subject Code : "); scanf("%s",result.marks.subj\_nm); printf("\n\t Enter Marks : "); scanf("%d",&result.marks.subj\_mark); printf("\n\n\t Roll Number : %d",result.rno); printf("\n\n\t Standard : %s",result.std);

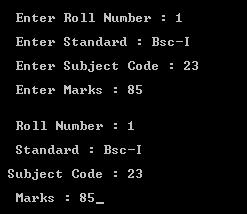
printf("\n\n\tSubject Code : %s",result.marks.subj\_nm); printf("\n\n\t Marks : %d",result.marks.subj\_mark); getch();

}





**Output :**



**Example 2:** C Program to illustrate the use of nested structures.#include<stdio.h>

#include<conio.h> void main()

{

struct date

{

int year; int month; int day;

};

struct student

{

int reg\_no; char name[20]; struct date dob;

}std1;

clrscr();

printf(“Enter student Reg\_no and name\n”); scanf(“%d%s”,&std1.reg\_no,std1.name); printf(“Enter date of birth of the student\n”);

scanf(“%d%d%d”,&std1.dob.day,&std1.dob.month,&std1.dob.year); printf(“Student Information\n\n”);

printf(“Reg-no : %d\n”,std1.reg\_no); printf(“Name: %s\n”,std1.name);

printf(“%d/%d/%d”,std1.dob.day,std1.dob.month,std1.dob.year);

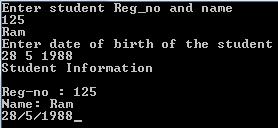
getch();

}





**Output:**



**STRUCTURE WITH ARRAY:**



We can create structures with array for ease of operations in case of getting multiple same fields.

**Example:** Program to demonstrate Structure with Array.

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

**struct** emp\_info

{

**int** emp\_id; **char** nm[50];

}emp[2];

**void** main()

{

**int** i;clrscr();

**for**(i=0;i<2;i++)

{

printf("\n\n\t Enter Employee ID : "); scanf("%d",&emp[i].emp\_id); printf("\n\n\t Employee Name : "); scanf("%s",emp[i].nm);

}

**for**(i=0;i<2;i++)

{

printf("\n\t Employee ID : %d",emp[i].emp\_id); printf("\n\t Employee Name : %s",emp[i].nm);

}

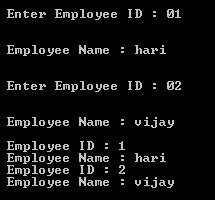
getch();

}





**Output :**



**UNION:**

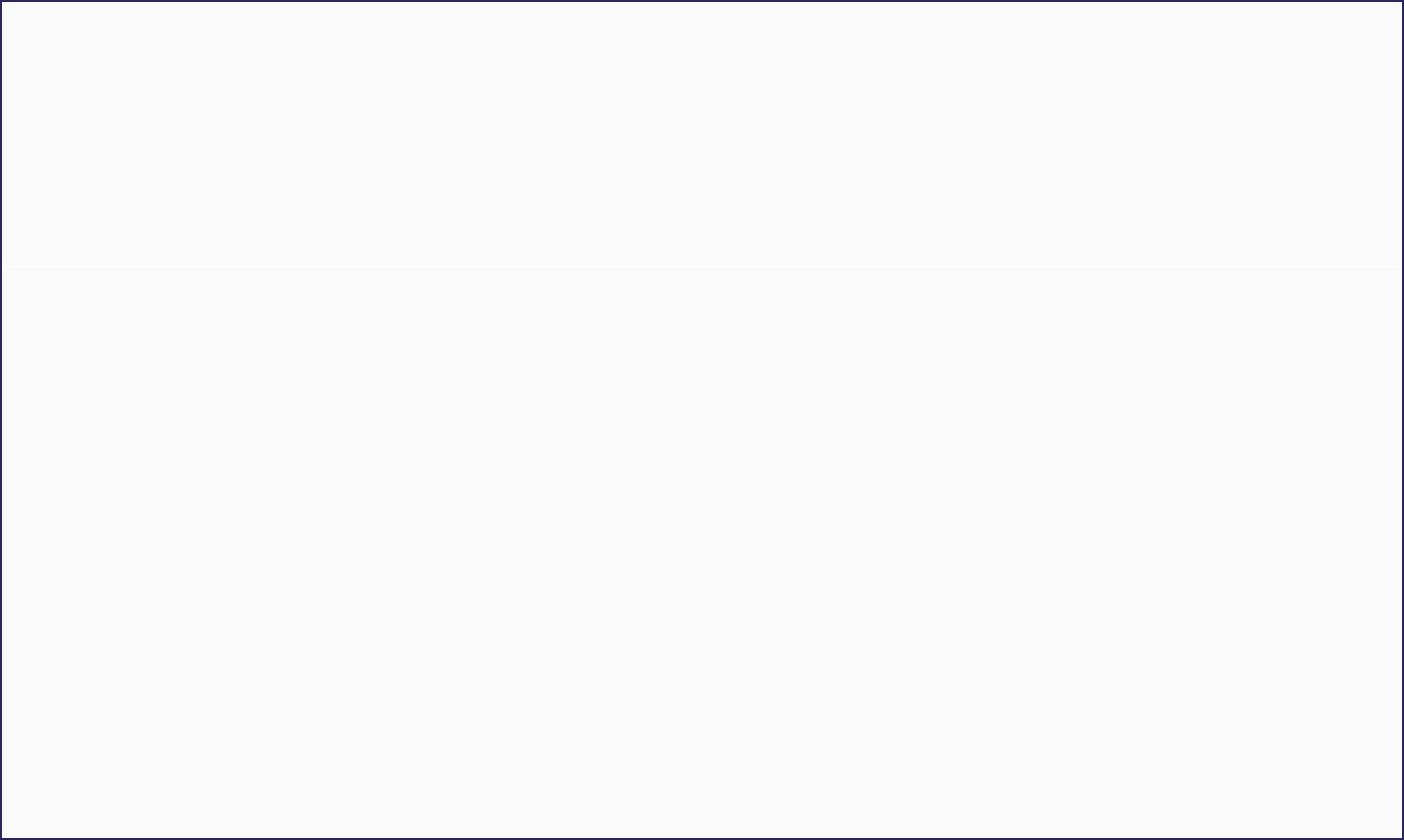


Union is user defined data type used to stored data under unique variable name at single memory location.

Union is similar to that of structure. Syntax of union is similar to structure. But the major **difference between structure and union is 'storage.'** In structures, each member has its own storage location, whereas all the members of union use the same location. Union contains many members of different types; it can handle only one member at a time.

To declare union data type, **'union'** keyword is used.

Union holds value for one data type which requires larger storage among their members.



**Syntax:**

**union** union\_name

{

<data-type> element 1; <data-type> element 2; <data-type> element 3;

}union\_variable;

**Example:**

**union** techno

{

int comp\_id; char nm; float sal;

}tch;





In above example, it declares tch variable of type union. The union contains three members as data type of int, char, float. We can use only one of them at a time.

**\* MEMORY ALLOCATION :**

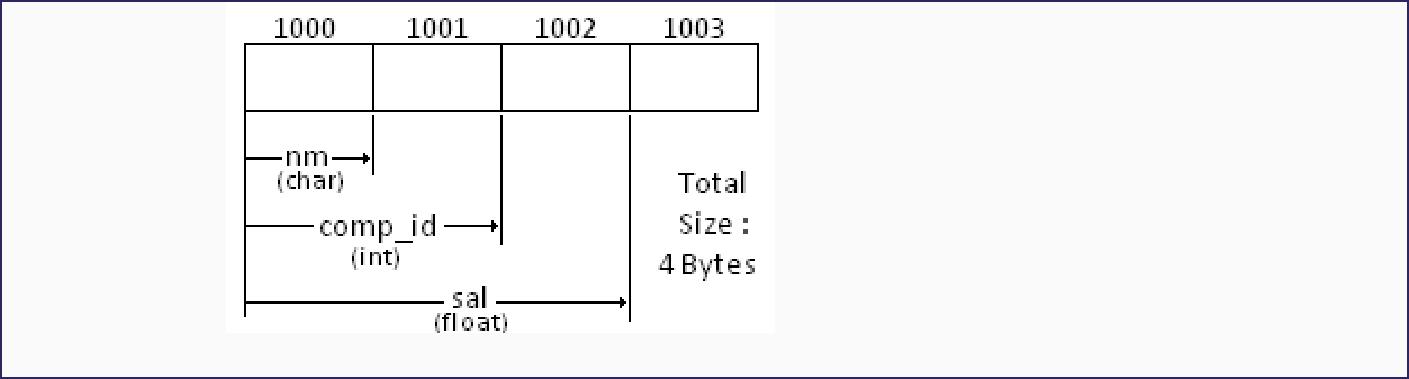


Fig : Memory allocation for union

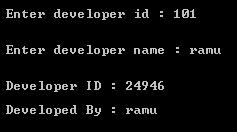
To access union members, we can use the following **syntax**.



tch.comp\_id tch.nm tch.sal

**Example:** Program to demonstrate union.



#include <stdio.h> #include <conio.h> **union** techno

{

**int** id;

**char** nm[50];

}tch;

**void** main()

{

clrscr();

printf("\n\t Enter developer id : "); scanf("%d", &tch.id);

printf("\n\n\t Enter developer name : "); scanf("%s", tch.nm);

printf("\n\n Developer ID : %d", tch.id);//Garbage printf("\n\n Developed By : %s", tch.nm); getch();

}

