Teacher's Sign / Remarks

classmate
Date 7/7/2017
Page

Chapter-1

Introduction to 'C' language

The History of the 'C' starts with a language called [BCPL] Beginners Combined programming language by Martin Richards in 1970. Later was modified to "B" language & in 1972 it Dennies Richie modified the version of 'B' language & Brought up 'C' language. 'C' i high level programming language it is the langu which a rich set of Building functions & opera that can be used to write any Complex progra The 'C' Compiler Combines the capabilities of a assembly language with the features of high leve language & therefore it is well suited for writing both system software & application soft

Features of "C" language

1. It provides flexibility for writing Programmes Compilers, operating systems, application software e
2. It can also be used for writing scientific, Engin & Business application.
3. 'C' is a portable language that is the program written on 1 Computer can be executed on anot with little or no modifications. That is we can r the same 'C' programs with different operating systems.
4. There are only 32 Keywords
5. Programmes written in 'C' are found to execute faster then Compare to other languages.
6. 'C' is a Procedure oriented language that is it is a Collection of functions modules or block
7. 'C' program runs with various operating system such as UNIX, LINUX, & MS-DOS.

A 'C' program contains 1 or more sections & they are as follows.

1. Documentation Section
2. Link Section
3. Definition Section
4. Global declaration Section
5. Main function Section
6. Sub program Section.

1. Documentation Section: It contains a set of Command lines giving the name of the program or title. The Compiles ignores the Commands. The Command line starts with /* & ends with */.
2. Link Section: The link section provides instructions for the compiler which contains pre defined function that can be used in the program that is header file inclusion.
   ex: # include <Stdio.h>
3. Definition Section: The Definition section defines all symbolic constants by using # define directive.
4. Global declaration section: It contain variables that are declare outside all the function i.e visible in all the functions of a program.
5. Main function Section: Every 'C' Program must have 1 main function. It is divided into 2 parts ① Declaration part ② executable part.

1. The Declaration part declares all the variables that are used in executable part.
The executable part contains executable statements.
The 2 parts must appear b/w the opening & closing braces or main functions.

main( )

6. Sub Program Section :- It Contains all the user defined functions that are called in main function.

Note :- All Section except link & main function section may be absent when they are not required.

The following example illstrate the above

```
/* To find the area of circle. */ } → (1)
# include <StDio.h> } — (2)
# include <Conio.h> }
# define Pi = 3.141 } — (3)

Void main ( )
{
  float A,R ;
  clr Screen ( ) ; } declaration part

Print f ("enter the Radius value ; ") ;
  Scan f ("%f" &R) ;
  A = (Pi * (R*R)) ; } executable part
Print f "The Area of circle is : %f", A) ;
  getch ( ) ;
}
```

o/p enter the radius value : 10
Area of circle is : 314

PROGRAMING STYLE

C is free from language i.e the 'C' Compiler does not care where on the line we begin typing. Although several programing styles are possible Conside

```
{
    printf ("welcome to \n");
    printf ("C programming");
}
```

The same program can be written in different styles as given below.

1. # include < >
```
Void main ( )
{
printf ("welcome to \n"); printf
        ("C programming")
}
```

2. # include <Stdio.h>
```
void main ( ) { printf ("welcome to \n"); printf
                ("C programming"); }
```

Execution of 'C' program :- Execution of 'C' Program Compress following steps.

1. Creating a program
2. Compiling the program
3. linking the program with functions of C-library
4. executing the program.

Character Set :-

To learn any language the first step is to know the characters supported by that language. The characters are used to form variables, numbers, keywords, expressions & Statements

The characters in 'C' are grouped as follows.

1) letters 3) Special characters

1. Letters :- Upper case A-Z, lower case a-z

2. Digits :- 0-9

3. Special character :- , ; : [ ] % # @

4. White Spaces :- Blank space, New line, Horizontal tab etc.

'C' Tokens

An individual word & punctuation marks are called Tokens. Similarly in 'C' smallest units are known as 'C' Tokens.

'C' has 6 types of Tokens.

1. Key words
2. Identifiers
3. Constants
4. Strings
5. Operators
6. Special Symbols.

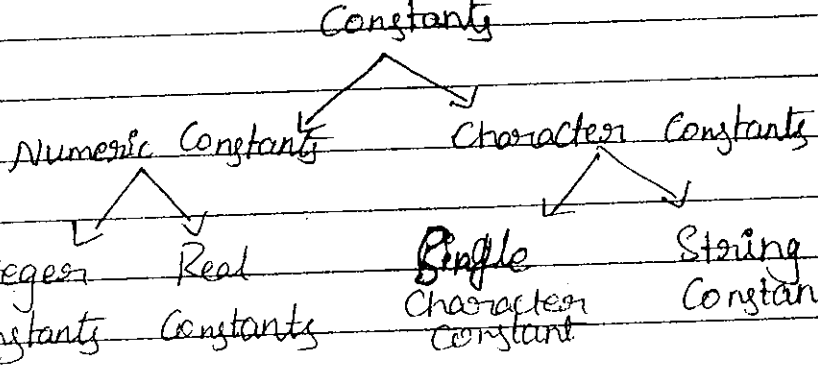
1. Keywords :- Keywords are predefined words having special meaning to the C-Compiler. 'C' supports 32 keywords & all are a lower case. It is also called as Reserved words. These keywords must not be used as a file name or variable name or function name. The list of keywords is as follows

1	Auto	Double	Int	Stauct
2.	Break	Else	Long	Switch
3	Case	Enum	Register	Typedf
4	Char	Extern	Return	Union
5	Const	Float	Short	unsigned
6	Continue	For	Signed	void
7	default	Goto	Size of	Volatile
8.	Do	IF	Static	while

2. Identifiers :- A variable is denoted by a name that is made up of letters & digits both upper case & lower case letters are permitted these variable names are called identifiers. It is used to name the functions, arrize, Structures, & any General declaration int marks ; float last - balance.

The Underscore is used as a link b/w 2 words in long identifiers if the name is a single letter then it is a variable.

3. Constants :- is a fixed value that does not change during the execution of the program. Constants are catogorised as follows



Integer Constant :- is a Sequence of digits without a decimal point that is a whole number.
There are 3 types of Integer Constants.
1. Decimal :- Eg 1, 2, 3, 4, 5
2. Octal :- 5, 6, 7, 3, 4
3. Hexa decimal :- 0, 9, A, B, D

Real Constants :- The quantities that are represented by numbers containing fractional part are called Real constants or floating point constants. There are 2 types in Real Constants

number followed by fractional part ex: 143.72

2. Exponential notation :- If we express the real Constants using the scientific notation is called exponential notation that is 123.22 = 123.22E2

where E is mantissa is a real number expressed as ~~Decimal~~ integer.

1. Single character Constants :- If the character is enclosed within a pair of single quotes (' '). Or any of the escape sequence leading with (slash 0) '\0' Or character Constants example :- 'A', '0', '10x1A'

2. String Constants :- is a sequence of characters enclose in double quotes. The characters may be letters, numbers, special characters & Blanks space. Example :- "ECPEPT", "1985", " ", "welcome".

Variable :- A variable is a name used to store the value it is Constant during execution of the Program. example :- int a;

data type variable

int num

data type identifier

Identifier,

Variable names consists of letters, digits & the underscore (_) & the character subject to the following rules.

1. The first character must be a letter or underscore
2. Rest of the characters that follow the first character may be either letters, digits or underscore.
3. The variable name should not be a keyword
4. The length of the name should not exceed 8 characters.

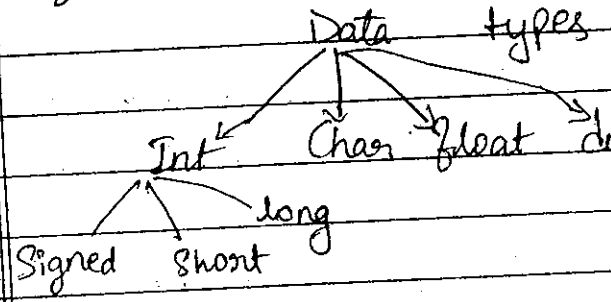
5 it is case sensitive that is upper case & lower case letters are different.

6 Special Symbols cannot be used as the first letter for variable or identifier.

Valid
a
num
_total
marks_madhu
marks_90

Invalid
##
int
Score
Count digit (Spaces not allowed)
2 num (number not allowed)

Data types: The Data types is used to specify the type of value that can be stored in a variable. 'C' has 4 basic data types. The Data types are briefly classified as shown below



Char - 1 byte
Int - 2 bytes
Float - 4 bytes
double - 8 bytes

Character data type: A Single character can be declared as a character data type i.e. char each character requires 1 byte of internal storage area i.e memory. A character is enclosed b/w a pair of Single quotes (' ')
The general Declaration of character data type
data_type Variable_name, example:-
character 'c'; Character name; char acctype='b';

Integer data type: In 'C' an integer comprises of sequence of one or more digits that is a whole number usually 2 bytes of storage area is required to store the integer value. The range of values can be stored is -32768 to 32767 that is -2^{15} to $2^{15}-1$ The general Declaration is given by data type variable name

Example: int num;
int marks = 80;

In order to provide some control over the range of numbers & Storage space 'C' has 3 types of integers namely int, short int, long int in both signed & unsigned numbers.

Short int: It represent fairly small integer values & requires half the amount of storage area of regular int. i.e 1 byte

Long int: we use long int & unsigned integers to increase the range of values. it requires double the amount of int storage area.

The following table gives the brief clarification of int data types

	Data type	Size
1.	int :	2 bytes
2.	Short int	1 byte
3.	long int	4 bytes

Floating data type: A floating data type is used to indicate precision point values that is Decimal points. The float type requires 4 bytes of storage area & provides 6 digits of precision. The general declaration is given by data type variable name

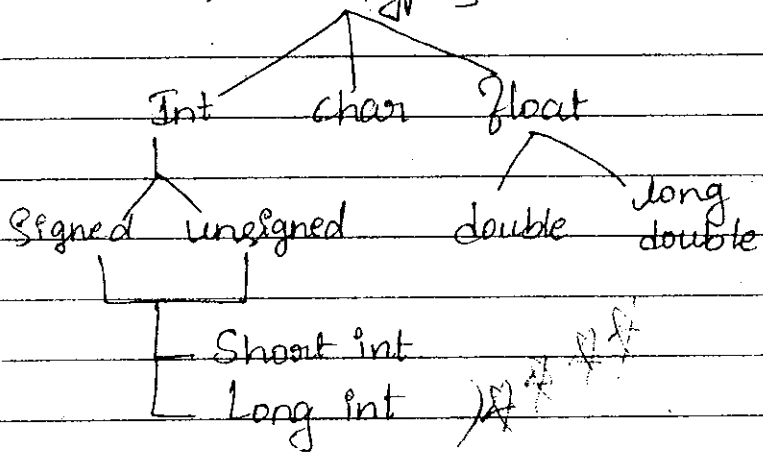
Double data type :- when the accuracy provided by the float number is not sufficient then double & long double can be used which requires a storage area of 8 bytes & 10 bytes respectively. Ex :- Double Balance ;
Long double avg = 112.11b ;

The following table gives a list of floating data types.

Data type	Size
Float	4 bytes
bits	32 bits
Double	8 bytes
long. Double	10 bytes

Char 1 byte
int 2 byte

Data types in 'C'



operators in 'C'

The operators are used in programs to manipulate Data & variables. They are a part of Arithamatic or logical expression. The 'C' operator are classified as follows.

1. Arithamatic operators
2. Relational operators
3. Logical operators

6. Conditional operator or Ternary operator
7. Bitwise operators
8. Special operator

1. Arithmetic operators: The Arithmetic operators are the symbols used in Arithmetic expression to combine the values to evaluate. The Arithmetic operators can be classified as
Unary — which operates on 1 operant
Binary — which operates on 2 operants
The list of Arithmetic operators with the meaning, precedence & associativity are as shown below.

operator	Description	precedence	associativity
−	unary −	1	R → L
*	multiplication	2	L → R
/	division	2	L → R
%	Reminder	2	L → R
+	addition	3	L → R
−	Subtraction	3	L → R

Example: a−, a+b, a−b, a/b, a%c

Here a & b are the variables & are known as operants.

Precedence
The order in which operators are evaluated or executed is called precedence of operators. As indicated above Level 1 indicates highest precedence & level 3 the lowest. The operators with highest preceder are executed first

The Operators with the same precedence are evaluated from left to Right (L → R) This is known as associativity property of an operator.

Relational operator: Any 2 quantities can be Compared with relational operator for example: we can Compare marks of 2 Students etc. The Result of Comparision is always True or false. In 'C' a non-zero value is treated as True & zero is treated as false. 'C' supports Six(6) relational operators & are given below

operator	discription	precedence	Associativity
<	less than	1	L → R
<=	less than/equal to	1	L → R
>	Greater than	1	L → R
>=	Greater than = to	1	L → R
==	Equal to	2	L → R
!=	Not equal to	2	L → R

A Simple Relational expression contains only one Relational operators & is of the form,

ae_1 relational operator ae_2

where ae_1 & ae_2 are arithmetic expression. example: A > B, a <= b, a+b == C+D. Note: Relational operators are used in Decision making Statements such as if, for, while.

Logical operator: 'C' has 3 logical operators

! Logical NOT

&& logical AND

|| logical OR

The logical operator && (AND) || (OR) used when want to test more than one condition & make decision. An expression involving the relational operator & logical operator then the expression is called logical expression For example:- (a>b) && (B<c) In the above expression it yeilds the value either true or false depending on the condition. The above expression is true if (a>b) & (b<c) is true. Otherwise the result is false The precedence of logical operator is as follows

! Logical NOT 1
&& Logical AND 2
|| Logical OR 3

Increment & Decrement operator
'C' has a unary operator called ++ incriment operator.
-- Decriment operator.

++ Operator (Incriment)
The ++ operator is a unary operator used to increment the value of an integer by one. The incriment operator is of 2 types
1. Prefix (precedes the variable name)
Ex:- ++a, ++y
2. Postfix (follows the variable name)
Ex:- a++, y++

-- operator (Decriment)
The -- operator is a unary operator used to decriment the value of an integer by one. The decriment operator is of 2 types
1. prefix Ex:- --a, --y
2. post fix Ex:- a--, y--

To Differenciate b/w prefix & postfix operators consider the following program.

Post increment :-

```
void main ( )
{
int i, j;
i = 2;
j = i++;
printf ("%d", j, i);
}
```

o/p 2 3

In the above statement J is equal to i++ the value of I is first assigned to J & then incremented by 1 hence the values 2 & 3 are displayed for j & i

Pre fix :-

```
Void main ( )
{
int i, j ;
i = 2;
j = ++i;
printf ("%d", j, i);
}
```

o/p 3 3

In the above statement j = ++i the value of I is first incremented by 1 & then assigned to J hence the values 3 & 3 are displayed for j & i.

Bit wise operators

'C' supports Bitwise operators for manipulating the data at bit level. These operators are

or right. Bitwise operators cannot be applied for floating values.

The list of Bitwise operators is as shown below.

Operator	meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR
<<	Left shift
>>	Right shift
~ (Tilday / Taldy)	1's compliment

Example:- y = a & b ;
c = a >> b ; 6
d = ~b ; b

Size of operator

The size of operator is used to return the number of bytes the operant occupies. The operant may be a variable constant or a data type. The general syntax is as given below.

Size of (variable, constant, data type);

Example Size of (a);
Size of (int);
Size of (float);

```
Void main ( )
{
Printf ("The size of int data type = %d
        bytes \n", Size of (int));
printf ("The size of float data type = %d
        bytes \n", Size of (float));
}
```

O/P The size of int data type = 2 bytes

TERINARY Operator :->

Terinary operator is also called as conditional operator which is denoted by (?). This operator requires the operands hence the name terinary. The general form of terinary operator is given by expression 1 ? expression 2 ; expression 3 ; expression 1 is evaluated first, if it is true then expression 2 is evaluated & becomes the value of the expression.

If expression 1 is false expression 3 is evaluated & it's value becomes the value of the expression. Note that only one of the expression either expression 2 or 3 is evaluated for example:-

big = (a>b) ? a ; b ; (exp1 ? exp2 ; exp3 ;) If (a>b) Big=a; else Big=b;

In the above statement if condition (a>b) is true then a is assigned to Big otherwise B is assigned to Big If (a>b) big = a; big = b;)

Operator, presedence & associativity

Each operator in 'C' has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated. There are different levels of operators & precedence belongs to one of the levels. The operator at higher level of precedence are evaluated first. The operators of same level are executed either from left to Right or Right to left. This is known as the associativity property of an operator. The following table provides complete list of operators & there precedences

executed

Operator	Description	Associativity	precedence (or) level
( )	function call	Left-Right	1
[ ]	array element Reference	Left-Right	1
+	Unary plus	Right-left	2
−	unary plus	Right-left	2
++	Increment	Right-left	2
−−	Decrement	Right-left	2
!	logical NOT	Right-left	2
~	1's Compliment	Right-left	2
*	indirection operator	Right left	2
&	address operator	Right-left	2
Size of	Size of an data type	Right left	2
(type)	type casting	Right left	2
*	multiplication	Left-Right	3
/	Division	Left-Right	3
%	Modulas	Left-Right	3
+	Addition	Left-Right	4
−	Subtraction	left-Right	4
<<	Bitwise left shift	Left-Right	5
>>	Bitwise Right shift	Left-Right	5
<	less than	Left-Right	6
<=	less than equal to	Left-Right	6
>	Greater than	Left-Right	6
>=	Greater than equal to	Left-Right	6
==	equal to	Left-Right	7
!=	Not equal to	Left-Right	7
&	Bitwise AND	Left-Right	8
^	Bitwise X-OR	Left-Right	9
|	Bit wise OR	left-Right	10
&&	logical AND	left-Right	11
||	logical OR	left-Right	12
?:	Conditional Operator (Ternary)	Left-Right	13
=, *=, /=, +=, -=, %=, ^=, |=, <<=,	Assignment operator	Right-left	14

1	Comma operator	Left-Right	15

Expressions in 'C'

Arithmatic expression | logical expression | Relational expression

Expressions in 'C' are catogarised depending on the type of operators used. Every expression is formed out of operands & operators. Depending on the type of operators used Expressions in 'C' are divided into 3 types as shown above. They are

1. Arithamatic Expression
2. Logical Expression
3. Relational Expression

1. Arithamatic Expression

They are formed out of Arithmatic operators & operands. The general syntax is as given.

operand 1 <operator> operand 2 ;

Eg :- a+b ;
b+c
(b+a) + (c/d);

2. Logical & Relational Expression

Logical expression is formed by the combination of logical operators & relational expression inturn relational expression are those which involve relational operators. The logical expression is also called as conditional Expression. Ex :- (a>b) && (b<c);

Here a is greater than b (a>b) is one

The execution of whole expression depends on individual execution of condition followed by final logical operator.

Ex:- (a>b) && (b<c);

(a && b) || (b && c)

The example shown above includes only logical operators

Type Conversion.

Converting a value of one type to another type is called Type conversion. In C there are 2 types of type conversion.

1. Implicit type Conversion.
2. Explicit type Conversion.) ****

Implicit type Conversion: that takes place automatically during the execution of an expression or in the assignment statement. It is also called as internal conversion. If operants are of different types the lower type is automatically converted to higher type & hence the result is higher type. The conversion is strictly followed by type conversion rules.

For example
```
int a = 3;
float b = 4.5;
int sum;
sum = a+b;
```

The following are the Conversion rules.

Short int & char values are converted into int while float is converted into double.

If either of the operand is float or double the other is converted into float or double & the result is float or double.

either of the operand is long the other is converted into long & the result is also long. If either of the ~~unsigned~~ is operand is unsigned the other is converted into unsigned & the result is also unsigned.

ex:-
```
int a=3;
float b=4.5;
int Sum;
Sum= a+b;
```

as shown in the above example `a` is of type int, b is of type float & Sum is of type int. In the expression Sum = a+b as one of the type is float then according to the rule. The other operand is converted to float & result is also in float type.

Explicit Expression:

If we convert the type explicitly then that type of conversion is called Explicit type Conversion. The general syntax is as given below. (type_name) expression;

where type name is any 'C' data type. The expression may be a Constant or variable or an expression. Type name is called cast operator. The following example area = 1/2 x B x H

B = Base
h: hig

In the above expression base & hight one of type integer & Subexpression of should be evaluated first the result will Zero due to both are integers. This can b explicitly Converting one of the operants of / to float as shown below.

solved

area = (float) 1/2 x Base x hight;

By using the automatic type Conversion the division is done in floating point mode.

Preprocessor Directive

The statement starting with '#' symbol are called preprocessor directives. They are placed before the main function. The preprocessor directive are processed before the source program are compiled by the compiler. Note that the preprocessor directive should not end with semicolon; example :- #include <stdio.h>. The above statement is a preprocessor directive. The file "stdio.h" is a header file consisting of 'C' i/p / o/p operations. The above statement includes an external file "stdio.h" in the program making function such as printf & scanf available to use.

Header files

Files that are placed at the top before main function of 'C' program are called header files. The header files have .h extension & are used to provide information of various library functions.

The header files are entered into the program through "#include" Statement. The "#include" Statement is a preprocessor directive & is written at the beginning of the program.

The following are the examples of header file Declaration.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <malloc.h>
```

Some commonly used header files with their

meaning is given in the table below

	Header	Meaning
1.	Stdio.h	input output functions
2.	Math.h	Mathamatical function
3.	String.h	String manipulation function
4.	Malloc.h	Memory allocation / Read allocation function
5.	Stdlib.h	Some Standard library function
6.	Ctype.h	character manipulation function.
7.	DOS.h	Functions linking DOS Rountiey
8.	Graphics.h	Graphical function

Basic input output Statement OR Standard input output Statement

In 'c' an input statement is a function that allows the program to read the data from the keyboard. An output statement displays (indicating / printing) the processed data on the monitor.

In 'C' there is no built in statements to perform basic input output operations. to perform these operations 'C' provides a library function called Standard i/p o/p library function. For example: Stdio.h

Basically there are 2 types of i/p / o/p Statement.

1. unformatted input output Statement
2. formatted input output Statement.

The unformatted i/p / o/p Statement do not specify the type of the data the way

written out.

Formatted i/p /o/p Statement specify the type of the data. The below table gives the type of i/p/o/p Statements

Type	i/p Statement	o/p Statement
unformatted	getchar()	Putchar()
	gets()	Puts()
Formatted	Scanf()	Printf()

Unformatted i/p Statement

These statements are used for reading the character type data from the keyboard. Here the type of data is not specified. Unformatted i/p Statements are of 2 types

1. Get Char()
2. Get String()

Get char():

This function reads a single character from the keyboard. The general syntax is given by variable-name = GetChar(); where variable-name is a valid 'C' name that is declared of character type which accepts a single character. When this statement appears the Computer waits until a letter (key) is pressed & then assigns that character to variable name.

For example Char letter,

letter = getchar();

The following simple program illustrates get char(); function

```
#include<stdio.h>
void main()
{
  char ans;
printf("would you like to display name'\n);
  ans = getchar();
if (ans=='Y' || ans=='y');
  printf("\n Amith \n"),
  else
printf("Cannot display name");
  }
```

O/p

would you like to display name

y

Amith

A

Cannot display name

This function getString() reads everything entered from the keyboard until the enter keys pressed the general syntax is given by gets (variable name); where variable name is a sequence of characters & is of type character Eg: char name[20]; gets(name);

where name is a character array of type char having 20 elements

```
#include <stdio.h>
main()
{
  char name[20];
printf("Enter your name"\n);
  gets(name);
```

gets(variable_name);

Eg char name[20];
gets(name);

O/P
Enter your name
ABCD

Unformatted output Statement
These statements are used to Display the character on the screen. The Put char() [function] & put String() are the 2 types of unformatted O/P statement. These 2 are included in header file Stdio.h

1. Put Char()
This function prints or displays a single character on the screen. The General syntax is given by putchar(variable_name);
where variable name is of type character it will display the character on the screen.
Ex: Putchar(ans);
- The following example illustrate put char(

```
#include <stdio.h>
Void main()
{
  Char ans;
  Printf("Enter the char to display\n")
    ans = getchar();
    Put char(ans);
}
```

O/P
Enter the char to display
H

Put String ( )

This function displays or prints a string of characters on the screen. The general syntax of puts() is given by

puts (String — name);

Eg:- puts (name);

where String — name is a sequence of character. The following example illustrate puts ( )

```
#include <stdio.h>
Void main ( )
{
  Char name [20];
  Printf ("Enter the name to display \n");
  name = gets (name);
  puts (name);
}
```

O/P

Enter the name to display

Formatted Input Statement :-

It refers to an i/p for taking data of different types. Formatted i/p refers to an i/p data that has been arranged in a particular format there is one type of formatted i/p ~~function~~ Statement Scanf( ); the general form of Scanf( ) is

Scanf ("format String", arg1, arg2, — argn);

where the format String contains the message with format specifier %d -> int

%c -> char

The format specifier specifies the type of data i.e to be assign to the variable associated. It should be enclose within a pair of double coats.

The arguments arg1, arg2, ... argn, specifies the addresses of location where the data is to be stored.

The Scanf function scans & accepts the values from the Standard i/p device & stores them in the argument list.

The format specifiers used with Scanf( ) are given below.

Format Specifier	meaning
%.c	Reading a character
%.d	Reading a decimal integer
%.f	Reading a float
%.g	Reading a float or double
%.I	Reading decimal, octal or Hexadeci.
%.O	Reading octal
%.X	Reading Hexa decimal
%.S	Reading a String
%.p	Reading a pointers

Example : Scanf ("%.d %.f %.c", &a, &b, &c)

In the above example the scanf function accept integer, float & character & stores them into the memory locations reserved for variables a, b & c.

The following program illistrate Scanf function in finding sum of two variables.

```
#include <stdio.h>
void main()
{
    int a, b, Sum;
printf("enter the value of a & b\n");
scanf("%d %d", &a, &b);
    Sum = a+b;
printf("\n The Sum is %d", Sum);
}
```

o/p

enter the value of a & b

10 20

The sum is 30

printf( ) Formatted output

The printf( ) o/p's a message & data in the required format & returns the number of bytes it displayed. The outputs are produced in a easy understandable form the general form of printf( ) is,

printf("format String", arg1, arg2 ... argn)

The format string must be enclosed with in a pair of double coats. It specifies the type of data that can be displayed from the variables with the required format the format string contain the message & format specifiers. The arguments arg1, arg2, ... argn.

printed. The arguments should match in number, order & type with format specifier.

Ex: printf ("a = %d, b = %f", a, b);
printf ("welcome to EC" \n);

The following program illistrates printf() to find Sum & average

```
# include <stdio.h>
void main()
{
  int a, b, Sum;
    float avg;
printf ("enter the value of a & b" \n);
 Scanf ("%d %d", &a, &b);
   Sum = a+b;
   average = Sum / 2;
printf ("\n The sum is %d", Sum);
 printf ("\n The avg is %f", avg);
  }
```

O/P

enter the value of a & b

10 20

The sum is 30

The avg is 15.00

Write a Program to find the cube of number.

```
# include <stdio.h>
# include <conio.h>
void main ()
{
  int a, cube;
printf ("enter the value of a" \n);
  scanf ("%d", &a);
    cube = a*a*a;
printf ("\n The cube is %d", cube);
printf ("\n");
}
```

O/P
enter the value of a
2
The cube ~~is axaxa~~ of a is 8

write a program to find the area of triangle

```
# include <stdio.h>
# include <conio.h>
void main ()
{
  float area, Base, height;
  printf ("enter the value of Base & hight");
  scanf ("%f %f", &Base, &hight);
  Area = (base * height /2);
printf ("\n The area of the triangle
      is %f", area);
}
```

Output

enter the value of base x height

1 1

2 2

The area of the triangle is 21.00

Write a program to evaluate the expression $y = A^2 + B^2 + C^2$

```
#include <stdio.h>
#include <conio.h>
void main()
{
    float y, A, B, C;
printf("enter the value of A, B, C");
scanf("%f %f %f", &A, &B, &C);
y = ((A*A) + (B*B) + (C*C));
printf("\n The y is %f", y);
}
```

O/P

enter the value of A, B, C

3

4

5

The value of y is 50.

Write a program to evaluate the expression : $y = (A^2/B^2)(C^2/D^2)$

```
#include <stdio.h>
#include <conio.h>
void main()
{
    float y, A, B, C, D;
printf("enter the value of A, B, C, D");
scanf("%f %f %f %f", &A, &B,
        &C, &D);
y = ((A*A)/(B*B)) * ((C*C)/(D*D));
printf("\n The y is %f", y);
}
```

o/p

enter the value of A, B, C, D

5

4

6

3

The value of y is 6.25

Write a program to evaluate the expression: $\left(\frac{A^2 B^2}{C^2 D}\right) + \left(\frac{E^2 DC}{A^2 C D^2}\right)$

```
#include <stdio.h>
#include <conio.h>
void main()
{
    float y, A, B, C, D, E;
printf("enter the value of A, B, C, D, E");
scanf("%f %f %f %f %f", &A, &B,
      &C, &D, &E);
y = (((A*A)*(B*B))/((C*C)*D)) +
    (((((E*C)*D*C))/(((A*A)*C*(D*D))))
printf("\n the y is %f, y);
}
```

o/p

enter the value of A, B, C, D, E

3

4

3

2

5

The value of y is 5.388888889

write a Program to Display name Branch & Sem

```
include <Stdio.h>
include <conio.h>
  void main()
  {
    char name [20], Branch [10];
     Int Sem;
printf ("enter the Std name");
  Scanf ("%s",&name);
printf ("enter the Branch");
  Scanf ("%s",&Branch);
~~printf ("%d",&Sem);~~

printf ("enter the Sem");
  Scanf ("%d", &Sem);
printf ("%s \t %s \t %d", name,
                Branch, Sem);
  }
```

O/P

enter the Student name

Nethra

enter the Branch

E C

enter Sem

2

ANIL EC 2

Write a program to Display Account name, ~~name~~, acc no, Balance

```
include <Stdio.h>
include <Conio.h>
Void main()
{
    Char Acc name[20];
    Int acc no;
    Float Balance;
printf ("enter the Acc_name");
  Scanf ("%.S", & Acc name);
printf ("enter the Acc_no");
  Scanf("%.d", & Acc no");
printf ("enter the Balance");
  Scanf ("%f", & Balance");

Printf ("%.S \t %.d \t %f", Acc name,
                Acc no, Balance);

}
```

o/p

enter the Acc_name
Nethra
enter the Acc_no
08035892
enter the Balance
28,000.95

Nethra 08035892 28.000.95

```
#include <stdio.h>
#include <conio.h>
void main()
{
  char name[20], Branch[10];
  Int Sem, marks in maths;

printf ("enter the Std name");
 Scanf ("%.s", & Std name);
printf ("enter the Branch");
 Scanf ("%.s", & Branch");
printf ("enter the Sem");
 Scanf ("%.d", & Sem");
printf ("enter marks in maths");
 Scanf ("%.d", & marks in maths);

printf ("%.s \t %.s \t %.d \t
        %.d", name, Branch, Sem
        marks in maths);
}
```

o/p

enter the Std name
ANIL
enter the Branch
EC
enter the Sem
2
enter the marks in maths
80

Name :- ANIL
Branch :- EC
Sem :- 2
maths :- 80

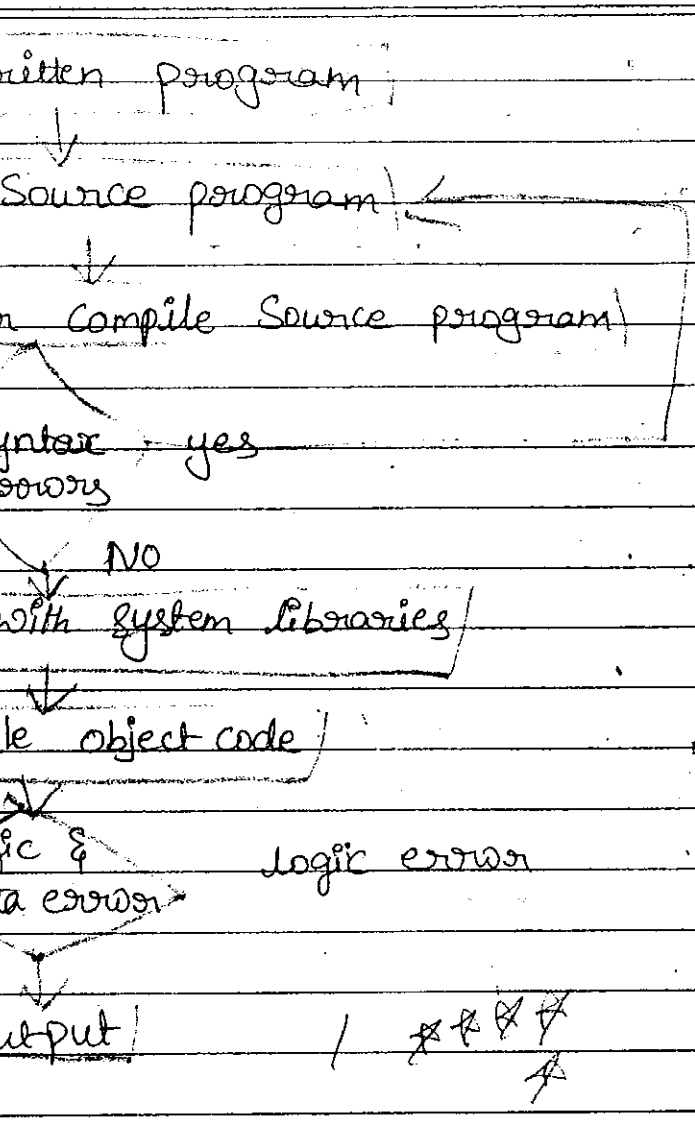
main() function

The main() is a part of every C program. 'C' permits different forms of main statements that are given below.

main ()
main (void)
void main()
void main(void)
int main()
int main(void)

In the above declaration the NT pair of parandhesis ( ) indicates that the function has no arguments & does not return any value. That is a nol value is retiirn. This may also be indicated by using the keyword "void" inside & outside the parandhesis as shown above. "void" means the function that does not return any information to operating system we can also specify the keyword int before the woord main. int means the functio returns an integer value to the operating syeten when int is specified the last statement in the program must be "return zero".

Executing a 'C' Program



Process & compiling & Running a C- program

The executing of program written in C consists of following steps

1. Creating the program
2. compiling the program
3. linking the program with functions that are needed by C library.
4. Executing the program

The figure shown above illistrate the process of creating, compiling & executing a C- program. These steps are same for different operating system An operating

environment to use the Computer & controls the entire operation of computer system. It is the interface b/w the hardware & the user

Write a program To swap the values of 2 variables without using the 3rd variable

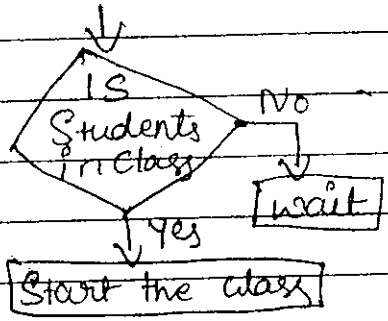
```
#include <stdio.h>
#include<conio.h>
void main()
{
  int a, b;
  printf ("enter two values");
  scanf ("%d %d", &a, &b);
printf ("Before swaping value of a & b is", a,b);
      a = a+b;
      b = a-b;
      a = a-b;
printf ("value of a = %d and
        value of b = %d after
        swaping", a, b);
  getch();
}
```

o/p
enter any 2 values
1 2
before swaping value of a & b is
value of a=2 & value of b=1

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

Conditional transfer & Branching or Conditions & loops.

In procedure oriented language the processing depends on flow of control during program Execution different action need to be performed on the basis of decision taken or decision making statements. The control statements or decision statements are mainly used for breaking the sequencial flow & Jumps to another part of the program. This is called Branching the below figure illistrates it.



In the above example if the students are present in the class then start the class otherwise wait for few minutes. These type of statements are called breaking Statements. These are several branching Statements that are classified depending on condition is true or false. The Branching statements are classified as shown below.

Branching Statements

Conditional	unconditional
If	goto
If -Else	Break
Nested If	continue
Switen	Exit

IF Statement :- The IF statement is a powerful dicision making statement. It is 2 way branching statement. Different types of IF statements are

1. IF Statement
2. IF Else Statement
3. Nested IF else Statement
4. Switen Statement

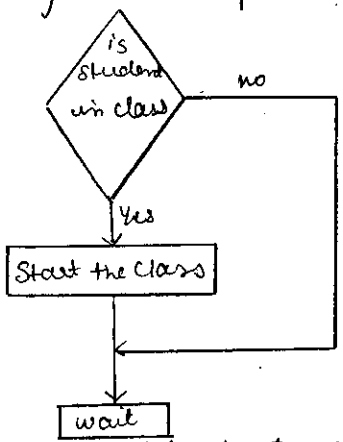
CHAPTER - 2.

CONDITIONAL TRANSFER AND BRANCHING

In procedure oriented language processing depends on the flow of control during program execution.

During programming different actions need to be performed on the basis of decision taken or decision making statement. The control statement and structure in 'C' are mainly used for breaking the sequential transfer of control and taking it to the other part of the program.

The following is simple example:-



The above example illustrates if the student present in class start the class other wise wait for few minutes. These type of statements are called as branching statement. There are several branching statements can be classified into condition satisfied or not. The brief classification of branching statements are shown below.

BRANCHING STATEMENT

- CONDITIONAL
  - If
  - If else.
  - nested if
  - switch
- UNCONDITIONAL
  - goto
  - Break
  - continue.
  - Exit.

If Statement [OR] construct:-

The "if" construct is required to perform an event [OR] activity to be performed if and only if condition is true. The general syntax of if construct is shown below.

```
if (test expression)
{
statement 1;
Statement 2;
    :
statement n;
}
```

eg:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=10, b=c=0;
if (a=10)
{
b=21;
c=31;
}
printf("The value of b=%d & c=%d",b,c);
getch();
}.
```

If else construct:-

If else construct is required to select and perform one among two optional based on condition i.e if condition is true event one is performed otherwise event two performed. The general syntax of "if else" construct is as shown below.

```
if (test expression)
{
statement1;
  :
statement n;
}
else
{
statement 1;
  :
Statement n;
}
```

Eg:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
float num, rem;
Printf ("Enter the number \n");
scanf ("%f", &num);
rem = num % 2;
if (rem == 0)
{
Printf (."The number is even\n");
else
Printf ("The number is odd\n");
}
getch ();
}
```

output:-

Enter the number
12
number is even
Enter the number
11
number is odd.

Write a program to find wether the number is small or large.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int a, b;
Clrscr ();
Printf ("Enter any two number \n");
Scanf ("%d %d", &a, &b);
If (a > b)
Printf ("The larger number is %d \n", a);
else
Printf ("The smaller number is %d", b);
If (b > a)
printf ("The smaller number is %d", a);
else
printf ("The smaller number is %d", b);
getch ();
}
```

Output:-

Enter any two numbers
12 16
The larger no is 16
The smaller number is 12

Manisha

Write a program to find the largest of 2 numbers.

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int a, b;
Printf("Enter two numbers\n");
Scanf("%d %d", &a, &b);
If (a>b)
Printf("The larger number is %d", a);
else
Printf("The larger number is %d", b);
getch();
}
```

Output :- Enter two numbers.
16 19
19 is larger number.

Nested if else :-

'C' language allows nesting of "if" statements Nesting means using one if statement in an other The dipth of nesting is limited only by conuience and the requirment of programer. One thing to remember in nesting of if statements is that the else part of the statement is associated with the nearest If statement.

The general syntax of "nested if else" statement is as given bellow.

```
If (expression1)
  statement 1;
else
{
 if (expression 2)
  statement 2;
else if (expression 3)
  statement 3;
}
```

Eg:-

```
#include<stdio.h>
#include<conio.h>
Void main()
{
clrscr();
int i;
printf("Enter either 1 or 2\n");
scanf("%d", &i);
if (i==1)
Printf("The number is one\n");
else
{
if (i==2)
printf("The number is two\n");
else
Printf("Unknown number\n");
getch();
}
```

Output :- Enter either 1 or 2.
1
The number is one.
4
unknown number.

Switch Statement :-

If we want to make a choice between multiple events rather than one or two at this time switch statement is used. It is a control statement which allows us to make a decision from multiple choices is called switch statement or switch construct or switch case default. Since these three key words together to make up the control statement.

The general syntax of switch statement is as shown below.

```
Switch(expression)
{
case constant 1:
  do this;
case constant 2;
  do this;
case constant 3:
   do this;
  .
  .
default;
 do this
}
```

The integer expression following the key word switch in any expression will yeild an integer value. The key word case is followed by an integer or character constant. Each constant in each case must be different from all others. The "do this" line is any valid C statement

Eg:-
```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr()
int i;
printf("Enter the value \n");
scanf("%d",&i);
switch(i)
{
case 1:
Printf("I am in case 1 \n");
case 2:
Printf("I am in cas 2 \n");
case 3;
printf("I am in case 3 \n");
default;
Printf("I am not in any case \n");
}
getch();
}
```

output :-

```
Enter the value
     2
I am in case 2
I am in case3
I am not in any case
```

4/21/2011

Unconditional statement :-

1) Break statement :- The break statement in 'C' is used to break the sequence of execution of 'C' statements. When break statement is used as the last statement in the set of statements of each case in the switch statement it executes the set of statements and come out of the loop.

NOTE:- There is no need of break statement in the default switch statement. The general syntax is as given below.

```
Switch (expression)
{
case 1
 Statement 1;
Statement 2;
Break;
Case 2
Statement1;
Statement2;
Break
   :
Case n
Statement 1;
Statement 2;
}
```

Eg:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
Printf("Enter the value \n");
Scanf("%d", &i);
Switch (i)
{
Case 1:
Printf("I am in case 1");
Break
Case 2:
Printf("I am in case 2");
Break
Case 3:
Printf("I am in case 3");
Break
Default:
Printf("I am not in any case");
}
getch();
}
```

17/2/2011

"go to" Statement:- The "go to" statement is considered useful for unconditional transfer of control. The programmer used the "go to" statement to unconditionaly branch to some statements other than the sequentially next statement.

The syntax of the "go to" statement is given by

"go label;"

The "go to" statement takes the control to the statement which is preceded by label.

i.e., label statement;

Eg:-

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

```
int marks;
printf("\n Enter the marks: ");
Scanf("%d", &marks);
if (marks >=75)
go to dist
else
{printf("\n perfoemance is good \n");
exit();
}
dist:
printf("\n perfoemance is extrodinary \n");
getch();
}
```

Output :-

Enter the ~~number~~ marks :

69

Perfoemance is good.

Enter the marks :

79

perfoemance is extrodinary.

Exit Statement :- If the condition is satisfied the "go to" statement transfers control to the label "dist" causing printf() statement to be executed. The label can be on seperate line as shown are on the same line.

Exit Statement :- In 'C' "exit" is a control statement which is used to return control to the operating system. The general syntax of "exit statement" is given as,

```
exit();
```

Eg :-

```
#include<stdio.h>
#include<conio.h>
void main()
{
int marks;
printf("\n Enter the marks \n");
scanf("%d", &marks);
if (marks <35);
go to fail;
else
{
printf("\n the student is pass \n");
exit();
}
fail:
Printf("\n the student is fail \n");
getch();
}
```

Output :- Enter the marks -

32

The student is fail

Enter the marks

36

The student is pass.

Write a program to swap the values of two numbers

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a, b;
Printf("Enter any two numbers\n");
Scanf("%d %d", &a, &b);
Printf("Value of a is %d & value of b is %d before
        swapping\n", a, b);
a = a+b;
b = a-b;
a = a-b;
Printf("value of a is %d & value of b is %d after
        swapping\n", a, b);
getch();
}
```

output:-

Enter any two numbers -
a = 1 b = 2 before swapping

after swapping
a = 2 b = 1

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a, b, temp;
printf("Enter the values\n");
scanf("%d %d", &a, &b);
temp = a;
a = b;
b = temp;
printf("the swaped value of a = %d & b = %d", a, b);
getch();
}
```

output.

Enter the values.
a = 1
b = 2

after swaping.
b = 1
a = 2.

Write a program to display the numbers from 0 to 9 otherwise unknown number using switch statemen

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i;
```

```
printf("Enter the value \n");
scanf("%d", &i);
switch(i)
{
case 1 = 0
printf("The number is 0 \n");
break;
case 2 = 1
printf("the number is 1 \n");
break;
case 3 = 2
printf("the number is 2 \n");
break;
case 4 = 3
printf("the number is 3 \n");
break;
case 5 = 4
printf("the number is 4 \n");
break;
case 6 = 5
printf("the no is 5 \n");
break;
case 7 = 6
printf("the no is 6 \n");
break;
case 8 = 7
printf("the no is 7 \n");
break;
case 9 = 8
printf("the no is 8 \n");
break;
case 10 = 9
printf("the no is 9 \n");
break;
default;
printf("the no is invalid \n");
getch();
}
```

output:-

25/12/2011

LOOPING IN PROGRAMS

Needs for looping

Most of the programs written in computer language consists of 3 types of control flow namely sequential, conditional and loop execution. In sequential execution it is always the next statement in a sequence of statement that is executed. Conditional execution is performed on the basis of certain decisions or conditions. Many times there is a need for executing a sequence of statement repetedly or until the condition is satisfyed. There exists the need for looping structures or constructs. Depending up

Depending upon the control statement in the loop the control structure may be classified into

i) Entry controlled loop

ii) Exit controlled loop.

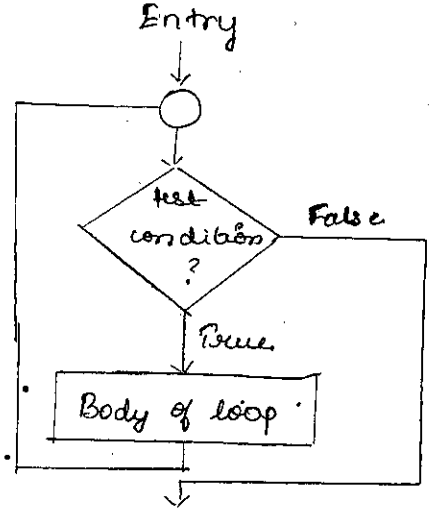


fig (a)

Entry controlled loop

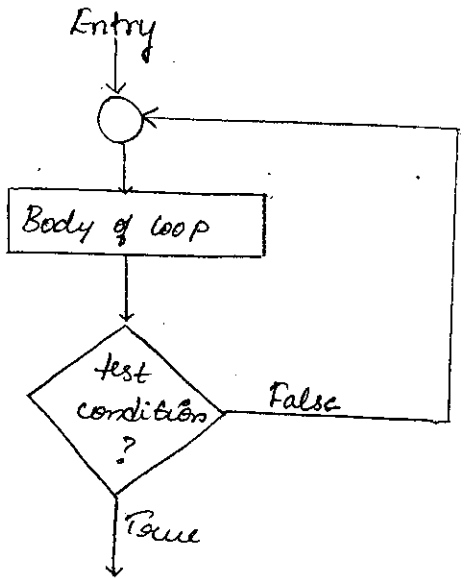


fig (b)
exit controlled loop

The figures above shows the entry controlled loop and exit controlled loop. In entry controlled loop the control condition are tested before the loop (i.e start of the loop). If condition is not satisfied then the body of the loop will not be executed.

In case of exit controlled loop the condition is tested at the end of the loop hence the body of the loop is executed unconditionally for the 1st time. A looping process in generall includes following 4 steps.

Setting Initialization of condition variable

* Execution of the statements in the loop
* Test for condition in the loop
* Incrementing or updating the condition variable

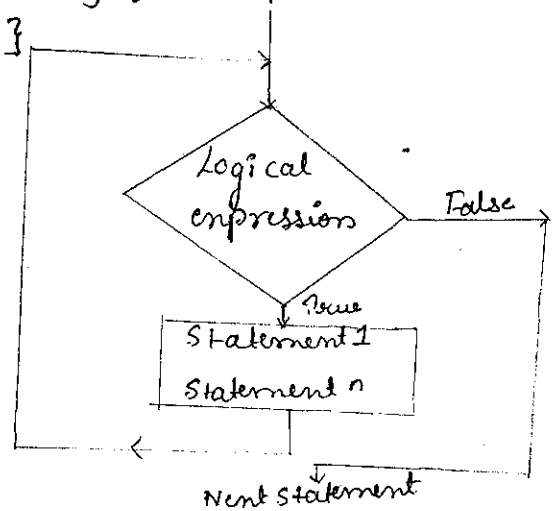
The 'C' language provides 3 loop constructs they are.

1) The "while" statement
2) The "do while" statement
3) The "for" statement

The "while" Statement :-

The simplest form of looping structure in 'C' is the "while" statement. The generall format is as shown bellow.

```
while (test condition)
{
  Body of the loop
}
```



To read 'n' integers and find the sum and average.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n, sum=0, count num;
float average;
printf("Enter how many no to add \n");
scanf("%d", &n);
```

```
count = 1; while ( count <= n)
{
printf ( "Enter num %d", count );
scanf ( "%d", num );
Sum = Sum + num;
count ++;
}
average = Sum / n;
printf ( "The sum = %d, average = %f", Sum, average);
getch ( );
}.
```

The above expression shows the general syntax of while statement or while loop. The statements S_1 to S_n are enclosed within the braces ({}). The statements are executed repetedly as long as the expression evaluates to true. when the expression is false the loop terminates and the control of the program reaches the statement following the loop i.e., next statement. The flow chart for the while loop is as shown in above figure.

output :-

Enter how many number to add 3

Enter the num 1 = 10

Enter the num 2 = 20

Enter the num 3 = 30

The sum = 60 average

= 20.000

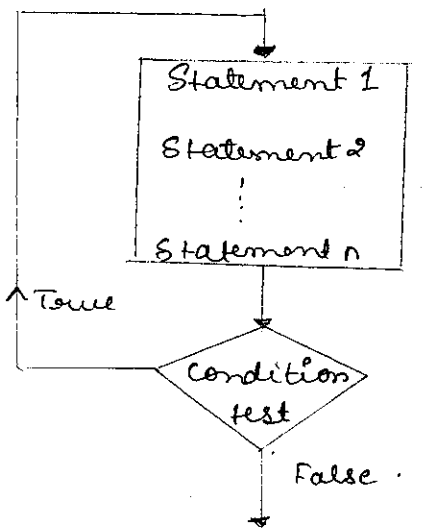
8/3/2011

do while loop :- [Exit control loop]

On some occasision it might be necessary to execute the body of the loop before the test is performed. such situation can be written with the help of do statement or double statement. The general syntax is given bellow.

```
do
{
statement 1;
Statement 2;
:
Statement n;
}
while (condition);
```

loop



The body of the loop is executed once irrespective of the condition. At the end of the loop the test condition is executed. If the condition is true, the program executes the loop once again until the condition is false. Since the condition is checked (executed) at the end of the loop it is called as exit controlled loop

The above example will illustrate

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int I=1, Sum=0;
do
{
Sum = Sum + I;
I = I + 2
}
while (Sum < 10 && I < 5);
printf ("%d %d\n", I, Sum);
getch();
}.
```

Output :-

I = 1 Sum = 0.

1st Sum=1 I = 3

2nd Sum = 4 I = 5

output is 4 5

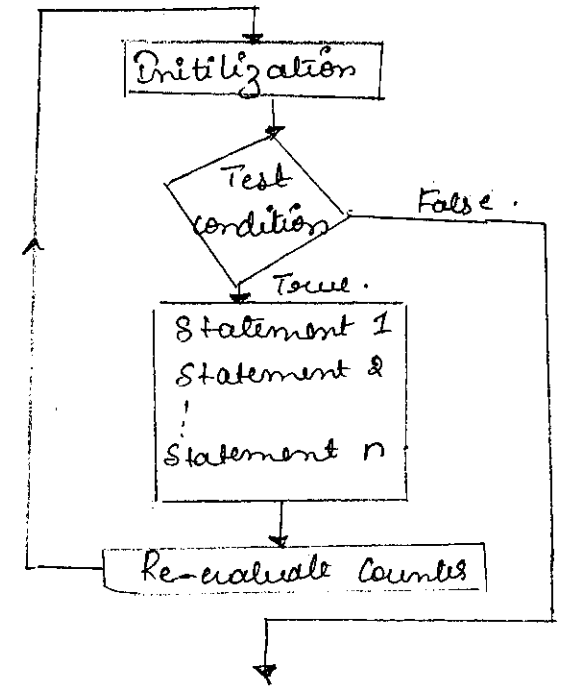
For the above example the value of I and sum are calculated as long as both of the three condition are true

For loop

The "for" loop is an entry controlled loop. The general syntax for "for" loop is;

```
for (initilization; testcondition; Increment)
{
Statement 1;
Statement 2;
:
Statement n;
}
```

loop



The "for" loop contains 3 parts

i) initilization

ii) test condition

iii) increement.

The initilization of loop control variable is done first i.e., I = 1 where I is loop control variable

The value of control variable is tested using test condition, if condition is true the body of the loop is executed otherwise the loop is terminated

~~The 3rd part is~~

The 3rd part is incrementing the control variable and to assign the new value to the control variable for further process.

The flow chart for "for" is as shown above.

Eg:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
clescr();
int i=0;
for(i=0; i<10; i++)
 {
printf("\n %d", i);
getch();
}
```

Output

0
1
2
3
4
5
6
7
8
9

Write a program to find sum and average of n numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clescr();
int i, n, sum=0, num;
float avg;
printf("Enter n numbers \n");
scanf("%d", &n);
printf("Enter the values");
for(i=1; i<=n; i++)
{
 scanf("%d", &num);
  sum = sum + num;
}
avg = sum / n;
printf(" sum = %d, avg = %f", sum, avg);
getch();
}
```

Output

Enter n numbers
5
Enter the values
01
02
03
04
05
Sum = 15
avg = 7.5

Write a program to find the number of students who scored greater than 60 and less than 75 in a subject of n students

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i, n, marks, count=0;
printf("Enter n numbers");
scanf("%d",&n); printf("Enter the marks");
for(i=0; i<n; i++)
{
scanf("%d", &marks);
if(marks>60 && mark<70)
count=count+1;
}
printf("The number of students scored >60 && <75 = %d", count);
getch();
}
```

Output:-

Enter n number of students

5

Enter the marks

62
71
38
45
55

The no. of students who scored >60 && <75 = 2.

Continue Statement

In some programming situations we want to take the control to the begining of the loop by passing the statements inside the loop which have not get yet been executed. When the key word "continue" is encountered inside any loop control automatically passes to the begining of the loop. The syntax is

```
continue;
```

The program illustrates the use of continue statement to display all numbers from one to 'n' which are not divisible by 5.

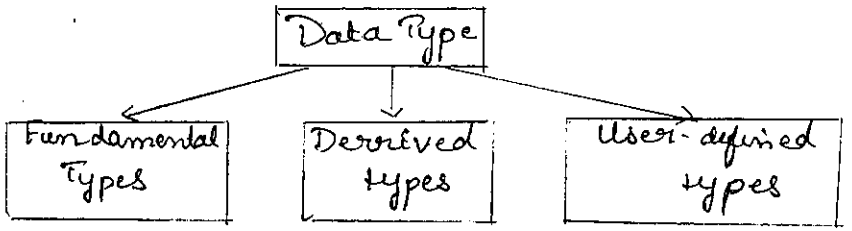
```
#include<stdio.h>
#include<conio.h>
void main()
{
int i=0, num;
printf("Enter the num");
scanf("%d", &num);
while(i++<=num)
{
if(i%5==0)
continue;
else
}
printf("%d", i);
getch();
}
```

Output:-

Enter the num 10

1 2 3 4 6 7 8 9

ARRAYS:-



An array is a structured data type consisting of group of components of same type.

Eg:- To find the largest of few numbers we can ~~need~~ use nested if statement in which each if statement would compair two numbers. How ever suppose there are 1000 numbers a set of if statements would be difficult to analyse.

To ~~declair~~ delare 1000 number it is given as

int num1, num2 - - - - - num 1000;

Consened to above problem if we group these numbers under one name say num and ~~acc~~ access the individual components of the group. This group of elements is called an array.

Definition:-

Array is defined as grouping of similar data-types i.e., group of int's, group of float's etc.

[OR]

An array is named a collection of data items all of which have the same data type.

Eg:- int num[100];
char name[80];
float avg[10];
double bal[20];

Declaration of array's:-

As ever variable used in 'C' program must be declaired array's also must be declaired before they are used. We declare an array by giving the data type of its elements, its name and the number of elelements. The generall Syntax is given by

data type array name [size];

where data type is any allowed data type and array name is a valid 'C' identifier.

Eg:- int num[20];
double bal[50];
int table[100]; ← Index (how many elements an array contains)
↓ data type, ↓ Array, (or) Subscript

In above example table is an array consisting of 100 elements of type integer (int)

Types of Array's :-

Array's can be classified into 3 types

i) 1 dimensional array :-

If the array is having only 1 subscript or index is called one dimensional array

ii) 2 dimensional array :-

If the array has 2 subscripts or index is called 2 dimensional array.

iii) Multi dimensional array :-

If the array is having more than 2 subscripts or index then its called as multi dimensional array

Initialization of One dimensional array :-

(or)

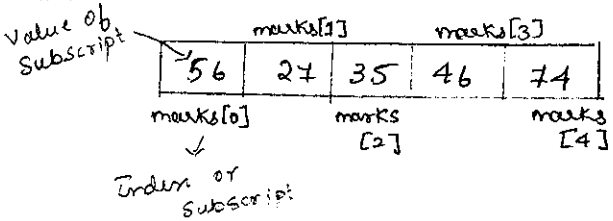
Array initialization

Once you declare an array you can enter information in to it. You can assign the array initial values when you declare it.

Eg :- int marks[5] = {56, 27, 35, 46, 74};

This creates 5 elements with these values.

marks[0] = 56
marks[1] = 27
marks[2] = 35
marks[3] = 46
marks[4] = 74



The above diagram shows the memory representation of one dimensional array for above example. The values of arrays or elements store in contigous [continues] memory locations

The subscript or index indicates the position of array elements.

NOTE : (1) If you declare less values to that of initilized subscript then it will asign the garbage value for remaining subscript variables

(2) If you declare more values to that of initilized subscript then it will (compiler) take only upto initilized subscript

PROCESSING WITH ARRAYS :-

We can use an array element in any instruction that is input or output comands or expressions. We always refer to the individual element by its subscript number. The following examples shows the processing of arrays.

To enter values into an array

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

```
int marks[10];
int i, n;
printf("Enter the number of values to be entered");
scanf("%d", &n);
printf("Enter %d
for(i=0; i<n; i++)
{ printf("Enter the %d value", i);
scanf("%d", marks[i]);
printf("The value %d elements is marks[%d]", n, i
                                    , marks[i]);
}
getch();
}
```

output:-

Enter the number of values to be entered 03
Enter the 0 element value 12
The value of 0th element is marks[0] = 12

Enter the 1 element value 15
The value of 1st element is marks[1] = 15

Enter the 2 element value 32
The value of 2nd element is marks[2] = 32

15/3/2011

Write a program to find the largest element in an array and the position of its occurance.

```
#include<stdio.h>
#include<conio.h>
void main
{
int a[10];
int largest, position, i, num;
printf("Enter number of elements in array\n");
scanf("%d", num);
for(i=0; i<num; i++)
scanf("%d", &a[i]);
largest = a[0];
position = 0;
}
for(i=1; i<num; i++)
{
if(largest<a[i])
{
largest = a[i];
position = i;
}
}
printf("Largest number is %d\n", largest);
printf("The largest number position is %d\n", position+1);
getch();
}
```

Output :-

Enter the number of elements in array
10.

52
71
93
21
63
95
57
90
31
10

The largest number is 95

The largest number position is 6.

Write a program to read in student marks and find the class average using arrays.

```
#include <stdio.h>
#include <conio.h>
void main( )
{
int a[20];
int marks, sum=0, num, i;
float avg
printf("Enter the number of students in class");
scanf("%d", &num);
for(i=0; i<num; i++)
{printf("Enter the marks\n");
scanf("%d", &a[marks]);
marks=i.
sum = sum + a[i];
}
average = sum/num;
printf("Class average is %f", average);
getch();
}
```

Output :-

Enter the number of students in class.
5.

24
25
92
71
62

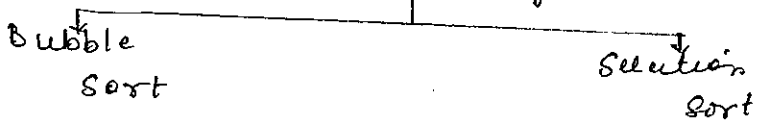
class average = 54.8

a[0] = 24
a[1] = 25
a[2] = 92
a[3] = 71
a[4] = 62

Sorting of Arrays :-

Sorting is the technic of arranging elements in accending or decending order.

There are two types of sorting of arrays



BUBBLE SORT :-

In bubble sort each element is compaired with the adjacent element (nent element). If the 1st element is largest than the second then the position of the elements are inter changed other wise it is not changed. The nent element is compaired with adjacent element and same process is repeated for remaining elements in the array.

Write a program to arrange 'n' numbers in ascending order using bubble sort method

```
#include <stdio.h>
#include <conio.h>
void main()
{
int array[100];
int n, i, J, temp;
Printf("Enter the array size");
Scanf("%d", &n);
printf("Enter the array elements");
for(i=0; i<n; i++)                    Reading array elements
{
scanf("%d", &array[i]);
}
for(i=1; i<n; i++)                    main loop.
{
for(J=0; J<n-i; J++)                  Inner loop.
{
if(array[J] > array[J+1])
{
temp = array[J];
array[J] = array[J+1];
array[J+1] = temp;
}
}
}
printf("List of bubble sorted array in ascending order");
for(i=0; i<n; i++)                    printing array element.
Printf("\t%d", array);
getch();
}
```

Output :

```
Enter the array size 5
Enter the array elements
85 92 12 11 05
List of Bubble sorted array in ascending order is
05 11 12 85 92
```

85 92 12 11 05

① 85 12 11 05 92

② 12 11 05 85 92

③ 11 12 05 85 92

④ 05 11 12 85 92

~~05 11 12 85 92~~

Write a program to arrange 'n' number in decending order using bubble sort method.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int array[100];
int n, i, J, temp;
printf("Enter the array size");
scanf("%d", &n);
printf(" Enter the array element ");
for(i=0; i<n; i++)
{
scanf("%d", &array[i]);
}
for(i=1; i<n; i++)
{
for(J=0; J<n-i; J++)
{
if(array[J]<array[J+1])
{
temp = array[J];
array[J] = array[J+1];
array[J+1] = temp;
}
}
}
printf("\n List of bubble sorted array in decending order is \n");
for(i=0; i<n; i++)
printf("\t %d", array[i]);
getch();
}
```

output :-

```
Enter the array size
5
Enter the array element
85 92 12 11 05
List of bubble sorted array in decending order is
92 85 12 11 05
```

Selection Sort:-

The selection sort is based on the extension of minimum and maximum technique i.e the means of nested for loops. A pass through the array is made to locate the minimum value. Once this is found it is placed in the 1st position of the array i.e position (0). Another pass through the remaining elements is made to find the next smallest elements and is placed in position [1] one and so on

Write a program to arrange 'n' numbers in ascending order using selection sort method.

```
#include<stdio.h>
#include<conio.h>
void main():
{
int array[100];
int n, i, j, temp;
printf("Enter the array size");
scanf("%d", &n);
printf("Enter the array element");
for(i=0; i<n; i++)
{
scanf("%d", &array[i]);
}
for(i=0; i<n; i++)
{
for(j=i+1; j<n; j++)
{
if(array[i]>array[j])
{
temp = array[i];
array[i]= array[j];
array[j]= temp;
}
}
}
printf("List of selection stored in ascending order\n");
for(i=0; i<n; i++).
printf("\t%d", array[i]);
getch();
}
```

output:-

```
Enter the array size 5
Enter the array element
85 92 12 11 05
```

85 92 12 11 05

05 92 85 12 11

05 11 92 85 12

05 11 12 92 85

05 11 12 85 92

List of selection sorted array is ascending order is

05 11 12 85 92

Write a program to arrange 'n' numbers in descending order using selection sort method.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int array[100];
int n, i, j, temp;
printf("Enter the array size");
scanf("%d", &n);
printf("Enter the array element");
for(i=0; i<n; i++)
{
scanf("%d", &array[i]);
}
for(i=0; i<n; i++)
{
for(j=i+1; j<n; j++)
{
if(array[i]<array[j])
{
temp = array[i];
array[i] = array[j];
array[j] = temp;
}
}
printf("\n list of selecton sorted array in descending order is");
for(i=0; i<n; i++)
printf("%d", array[i]);
getch();
}
```

output :

```
Enter the array size
        5
Enter the array element
        85
        92
        12
        11
        05
List of selection sorted array in ascending order is
        05
        11
        12
        85
        92
```

17/5/22

Linear Search- To search the number in an array

It is used to find the location where element or number is available or not. In linear search we access each element of the array and see whether it is the desired number or not

The search will be unsuccessfull if all the elements in the array is searched and the number is not found.

Write a program to search the given number and its position in an array 'n' numbers

```
#include<stdio.h>
#include<conio.h>
void main()
{
int array[100];
int n, i, num, pos=-1, or pos=0;
printf("Enter the array size");
scanf("%d", &n);
printf("\n Enter the array elements");
for(i=0; i<n; i++)
{
scanf("%d", &array[i]);
}
printf("Enter the number to search");
scanf("%d", &num);
for(i=0; i<n; i++)
{
if(num==array[i])
{
pos=i;
break;
}
}
if(pos>=0) or (pos==0)
printf("The num %d is found in position %d", num, pos+1);
else
printf("The num not found");
getch();
}
```

Output

Enter the array size
7
Enter the array elements
2 3 5 9 11 15
19
Enter the number to search
19
The number is found and its position is
7

Enter the array size
5
Enter the array element
5 6 9 11
12
Enter the number to search
13
The number is not found.

Two Dimensional Arrays:-

The two dimensional array have two subscripts The 1st subscript reffers to the row and the 2nd subscript reffers to column. The generall syntax is ~~data~~

data type array name [K_1] [K_2];

where [K_1] and [K_2] are integer constants (row, column)

[K_1] reffers to row and [K_2] reffers to column.

Eg: int matrix [3] [3];
(Row, column)

The above example shows a two dimensional array matrix having three rows and three columns That can be given as

matrix [0][0]	matrix [0][1]	matrix [0][2]
matrix [1][0]	matrix [1][1]	matrix [1][2]
matrix [2][0]	matrix [2][1]	matrix [2][2]

Due to this representation a two dimensional array is also called a table or rectangular array

18/3/2011

Write a program to read and display a square matrix or example for 2 dimensional array

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int mat[10][10];
int row, col, n, Order;
printf("Enter the order of matrix");
scanf("%d", &order);
printf("Enter matrix elements");
for(row=0; row<order; row++)
{
for(col=0; col<order; col++)
scanf("%d", &mat[row][col]);
}
printf("square matrix is \n");
for(row=0; row<order; row++)
{
for(col=0; col<order; col++)
printf("%d\t", mat[row][col]);
printf("\n");
}
getch();
}
```

Reading values into matrix

Printing the matrix

Reading of loop:

mat [r] [c]
mat [0] [0]
[0] [1]
[0] [2]
[1] [0]
[1] [1]
[1] [2]
[2] [0]
[2] [1]
[2] [2]

Output

Enter the order of matrix... 3
Enter the matrix elements... 1 2 3 4 5 6 7 8 9
Square matrix is

1	2	3
4	5	6
7	8	9

Write a program to read the matrix order and to display the matrix and to display transpose of the matrix.

```
#include< stdio.h >
#include< conio.h>
void main( )
{
clrscr( );
int a[10][10], b[10][10];
int row, cols, order;
printf("Enter the order of matrix");
scanf("%d", &order);
printf("Enter matrix elements");
for(row=0; row<order; row++){
{
for(col=0; col<order; col++)
scanf("%d", &a[row][col]);
b[col][row] = a[row][col];
}
printf("Square matrix is \n");
for(row=0; row<order; row++)
{
for(col=0; col<order; col++)
{
for col= printf("%d\t", a[row][col]);
printf("\n");
}
printf("Transposed matrix of A is \n");
for(col=0; col<order; col++)
{
for(row=0; row<order; row++)
printf("%d\t", b[col][row]);
printf("\n");
}
getch( );
}
```

Output :-

Enter the order of matrix...3

Enter the matrix elements

1 2 3 4 5 6 7 8 9

matrix A is

1	2	3
4	5	6
7	8	9

Transposed matrix of A is

1	4	7
2	5	8
3	6	9

Differentiate between while and do while

While	Do while

1) General Syntax
while (test condition)
{
Body of the loop
}
	General Syntax.
	do
	{
	Statement 1;
	Statement 2;
	:
	Statement n;
	}
	while (condition);

2) The loop is executed if and only if condition is true.	The loop is executed once irrespective of the conditions.
3) It is entry controlled loop	It is exit controlled loop
4) Condition is tested at the top (beginning)	Condition is tested at the last (end)
5) Frequently used	Rarely used for particular application

Eg:-
while (a > b)
{
temp = a;
}
	Eg:-
	do
	{
	temp = a;
	}
	while (a > b);

Differentiate between break and continue Statement

Break Statement	Continue Statement

1) General Syntax.
switch (expression)
{
case 1 (break;)
Statement 1;
Statement 2;
break
:
case n
Statement 1
Statement 2;)
}
	1) General syntax.
	continue;

2) The break statement is used to terminate the loop { to come out of the loop }	2) The continue statement is used to take the control to the beginning of the loop
3) The statements within the loop are executed	3) The statement within the loop are bipassed
4) It is used with switch statement	4) It is not used with switch statement.

Eg:- while (i <= n)
{
if (num % i == 0)
{
printf ("number is even");
break;
}
	Eg:- while (i++ <= num)
	{
	if (i % 5 == 0)
	continue;
	else
	printf ("%d", i);
	}.

WAP to input 2 nxn matrix A and B and display the sum of A and B.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10][10], b[10][10], c[10][10], i, j, n, m;
printf("Enter the order of the matrix ");
scanf("%d, %d", &n, &m);
printf("\n enter matrix A elements \n");
for(i=0; i<n; i++)
{
for(j=0; j<n; j++)
scanf("%d", &a[i][j]);
}
printf("Enter matrix B elements:\n");
for(i=0; i<n; i++)
{
for(j=0; j<n; j++)
scanf("%d", &b[i][j]);
}
for(i=0; i<n; i++)
{
for(j=0; j<n; j++)
c[i][j]=a[i][j]+b[i][j];
}
printf("\n Sum of matrix A & B is");
for(i=0; j<n; i++)
{
for(j=0; j<m; j++)
printf("%d\t", c[i][j]);
printf("\n");
}
getch();
}
```

Reading for matrix A elements

Reading for matrix B elements

Calculating Sum of A+B

Displaying C matrix

output :-

```
Enter the order of matrix
2 2
Enter the matrix of A elements
1 2 3 4
Enter the matrix of B elements
1 2 3 4
Sum of matrix A and B is
2 4
6 8
```

28/3/2011

WAP to input 2 nxn matrix A and B and display the product of A & B.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a[10][10], b[10][10], c[10][10], i, j, n, k;
float norm;
printf("Enter the order of the matrix");
scanf("%d", &n);
printf("Input A matrix elements\n");
for(i=0; i<n; i++)
for(j=0; j<n; j++)
scanf("%d", &a[i][j]);
printf("Input matrix B elements\n");
for(i=0; i<n; i++)
for(j=0; j<n; j++)
scanf("%d", &b[i][j]);
/* Find the product of two matrix */
for(i=0; i<n; i++)
for(j=0; j<n; j++)
{
c[i][j]=0
for(k=0; k<n; k++)
c[i][j]+=(a[i][k]*b[k][j]) + c[i][j]
}
printf("\n Product matrix");
for(i=0; i<n; i++)
{
for(j=0; j<n; j++)
printf("%d\t", c[i][j]);
printf("\n");
}
getch();
}
```

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

i=0
j=0
k=0
= 0 + [a[0][0] * b[0][0]]
= 0 + 1 × 1 = 1

i=0
j=0
k=1
c[0][0] = 1 + [a[0][1] * b[1][0]]
= 1 + (2 * 3) = 1 × 6
= 7

output:-

Enter the order of the matrix

2
2

Input A matrix elements

1 2 3 4

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

Input B matrix elements

1 2 3 4

$\begin{bmatrix} 1+6 & 2+8 \\ 3+12 & 4+16 \end{bmatrix}$

Product matrix

7 10

15 22

CHAPTER – 4 :-

~~Find the product of two matrix */~~

30/3/2011

STRUCTURES AND UNIONS

We all know that array is a group of similar data types. A structure is a meaningful collection of data items of different type under a ~~~~ unique name. In simple structure can be defined as "A group of dissimilar data types".

A ~~structure~~ contains integer elements, floatingpoint, character elements, array of characters, pointers and other structures. ~~The general syntax~~.

Declaration of structure

The general syntax of structure is as given below.

```
Struct tag {
        member1;
        member2;
          :
        membern;
};
```

As shown above ~~Struct~~ STRUCT is the key word, tag is the ~~name~~ name that appeared in the structure declaration. member 1, member 2 to member n are individual member declaration. The members can be int's, float's, array's and structures.

Eg:-

```
Struct account {
        int acct.no;
        char acct_name[80];
        char acct-type;
        float balance;
};
```

In above example Struct is the key word, account is the structure name having four members

Declaring Structure variables:-

Once the structure composition has been defined individual Structure variables can be defined as follows

Storage-class Struct tag var1, var2, ---Varn, (variable)

where

Storage_class is optional.

Struct is the required key word, tag is the name that appear in the structure declaration and var1, var2 --- varn are Structure variables of type tag

Eg:- Struct account old customer, new customer

As shown above old customer and new customer are structure variables of type account

1/4/11 - The structure variables can be defined as,

```
Struct account
    {
     int acct_no;
     char acct_name[80];
     char acct_type;
     float balance;
     } old customer, new customer;
```

where old customer and new customer are two structure variables that as, storage space for these individual members i.e. for above example of 4 members.

Structure within a structure (OR) Nested Structure

```
struct date
    {
     int day;
     int month;
     int year;
    }
account
    {
     int acct_no;
     char acct_name[80];
     char acct_type;
     float balance;
     struct that last payment;
    } old customer, new customer;
```

A structure within another structure is known as nested structure. The embedded structure known as nested structure is declared as a member of the main structure. The embedded structure is declared outside in the above example account is the main structure. The date is the embedded structure. The example is shown above.

Processing a structure:

The member of a structure is usually processed individually as separate entities. Therefore we must be able to axis the individual structure member.
A structure member can be accessed by writing
variable.member;
where,
variable refers to name of structure type variable
and
member refers to a name of a member within the structure

The "." operator is referred to as:
→ period operator which separates the variable name from the member name. It is a member of highest precedence through group. The following example illustrate the above.

```
struct date
    { int day;
     int month;
     int year;
    }
```

```
Struct account
  {
   int acct_no;
   char acct_name[80];
   char acct type;
   float balance;
   Structure dat last payment;
   } customer;
```

In the above example customer is a structure variable of type account where, account is a structure. To access the customer account number and balance we can write,

```
customer.acctno;
customer.balance;
```

If a structure is one of the member of the nested structure, then the members of the embedded structure can be accessed by writing

```
customer_last payment.date;
customer_lastpayment.year;
```

The following example illustrates the processing of array.

WAP to illustrate the accessing of a structure variable.

```
# include <stdio.h>
Struct date
  {
   int day;
   int month;
   int year;
  }
  Day;
  Main()
{
Printf("Please enter payment date");
Scanf("%d%d%d", & pay.date, & pay.month, & pay.year);
Printf("day=%d\n Month=%d\n Year=%d\n", Pay.date,
      pay.month, pay.year");
```

O/P

Please enter year payment date

01 07 2010

day= 01
month=07
year=2010

Assignment of structure (OR) Assigning values of 1 structure variable to another

It is possible to assign the values of 1 structure variable to another structure variable.

The following example illustrate the passing values from one structure to another structure

```
# include <stdio.h>
  Struct marks
  {
   char name[80];
   int submarks;
  };
 main()
  {
  struct marks std1, std 2;
```

```
Printf(" enter name,marks of student 1 \n");
Scanf ("%s%d ",& Std 1.name,& Std 1.submarks);
Printf (" enter the name of Std 2 \n");
Scanf("%s," & Std2.name);
Printf ("Marks of %s is %d", Std 1.name, Std 1 submarks);
Printf ("Marks of %s is %d", Std 2.name, Std 2. submarks);
  }
```

O/P

```
Enter name,marks of std1
     Manju 48
Enter the name of Std 2
     Sathi

Marks of Manju is 48
Marks of Sathi is 48
```

Structure and Arrays:

Array of structure:

Array is a group of similar data elements and structure is a dissimilar data element. As we can declare an array, an array of structures can also be defined.

For example:

The RS student data based application stores and manipulate the information about many students, hence, it is necessary to use array of structures for storing information about many students. The following example illustrates the array of structure concept.

```
# include < stdio . h>
   struct student
      {
        char name[30];
        int reg no;
        int sem;
        char branch[10];
       };
   main()
    { int j, i;
      struct student std dat[2];
    Printf ("enter name, Reg no, Sem and branch of 2 students \n");
    for(i=0;i<2;i++)
   Printf("%s %d %d %s", Stddat[i].name,
                          Std dat[i].reg no,
```

```
                    Stddet[i].Sem,
                    Stddet[i].Branch);
    }
    Printf("Student details as follows");
    for(j=1; j<=2; j++)
    {
    Printf("%s%d%d%s", Stddet[j].name,
                       Stddet[j].reg no,
                       Stddet[j].Sem,
                       Stddet[j].Branch);
    }
}
```

o/p

```
Enter name, Reg no, Sem and Branch of 2 students
Aruu    1178    3    Electronics
Ganesh  1179    3    Electronics
Student details as follows
Aruu    1178    3    Electronics
Ganesh  1179    3    Electronics
```

12/1/2011

Size of Structure :-

The unary operator sizeof() can be used to find the number of bytes of memory occupied by a variables arrays, structures and unions, hence the size of structure i.e., bites of memory occupied can be known by using "size of( )"

The below program illustrates the above concept

```
# include <stdio.h>
struct payment {
        int date;
        int month;
        int year;
       };
    void main()
    {
    struct payment last;
    Printf("size of structure payment is %d", size
          of(struct payment));
    }
```

Output :-

size of the structure payement is 6.

UNIONS :- Like structures unions is a collection of dissimilar data elements. In unions the member share the same storage area within the computers memory. The unions are used to conserve memory. The main difference between structure and union is the union members accept the value for only one member at a time. The general syntax is as given bellow.

```
union tag_name {
        datatype 1   member 1;
        datatype 2   member 2;
                :
        datatype n   member n;
} var 1, var 2 --- var n;
```

where union is the required keyword, tag is the name of union defination, data type 1 --- to data type n are the basic data types. member 1, member 2 so on to member n are the members of the union and var 1 var 2 --- to var n are the union variables.

```
Eg:- Union value {
        float f;
        int i;
     } univar;
```

Value variable whose value is the union. It is having two variable members. It is having two members float and int occupying a size of 4 bit union may be member of structure and vis a vis a. An individual union members can be accessed in the same manner as individual.

```
#include<Stdio.h>
void main()
{
union value {
        float f;
        int i;
} num;
num.i = 4444 ; 1* statement * 1
printf("i = %d  f = %f", num.i, num.f);
num.f = 4444.123; 1* statement 2 */
printf("i = %d  f = %d", num.i, num.f);
}
```

output

i = 4444 f = 5.234578 9

i = 1278912 2 f = 4444.123

As shown above after statement 1 data stored in num is an integer and the float member is the garbage value funch value. After statement to the data stored in num is a float and the integer value is meaning-less.

Difference between union and structure

Structure	Union
* The key word struct is used to define a structure	The key word union is used to define a union.
* Once a structure is declared compiler will alocate the memory to all of its members	In union it will alocate the memory to only one member that requires max memory size.
* minimum memory utilization when all the members are not accessed	maximum memory utilization

* Can pass value to all the members	can pass value to only one member at a time.
* Every member has its unique storage area	Memory is shared by all the members
struct value { float f; int i; } num; For above example the size of structure is 6 bits	union value { float f; int i; } num; For above example the size of union is 4 bits

13/4/2011

Bit fields:-

We can group variables requiring a few bits as a single word i.e., a single integer instead of defining each variable as an integer or character. this is known as bit fields.

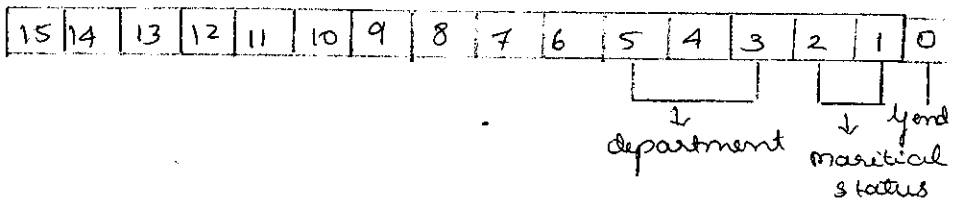
One bit can give values true or false.

A 2 bit can range from 0 to 3 while a value of 3 bits can range from 0 to 7. Several such data items can be combined into an individual word of memory. Such data items are called bit names and are defined as member of structure. The general syntax is as shown bellow

```
struct tag {
    member 1 : size;
    member 2 : size;
    .
    .
    member n : size;
} var 1, var 2 ... var n;
```

Each member name is followed by a semicolon and an unsigned integer indicating field size. The variable declaration and accessing each member is same as structure

Eg:-
```
struct emp {
    unsigned gender : 1;
    unsigned mar_status : 2;
    unsigned dept : 3;
} C;
```



The number after the colon (:) specifies the size of each bit field. It is defined as a structure which is subdivided into 3 bit fields. The members have width of one, 2 and 3 bits. Hence they occupy 6 bits within a word of memory

Here member 1 :- indicates the gender either as male or female.

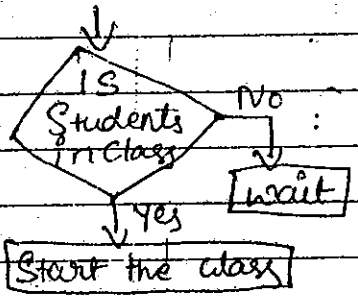
member 2 :- indicates the marital status unmarried, married, divorced, widow.

member 3 :- Indicates 8 different departments

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

Conditional transfer & Branching or Conditions & loops

In procedure oriented language the processing depends on flow of control during program Execution different action need to be performed on the basis of decision taken or decision making statements. The Control statements or decision statements are mainly used for breaking the sequencial flow & Jumps to another part of the program. This is called Branching the below figure illistrates it



In the above example if the students are present in the class then start the class otherwise wait for few minutes. These type of statements are called breaking Statements. These are several branching Statements that are classified depending on condition is true or false. The Branching statements are classified as shown below.

Branching Statements

Conditional	unconditional
If	goto
If-Else	Break
Nested If	continue
Switch	Exit

IF Statement :- The IF statement is a powerful dicision making statement. It is 2 way branching statement. Different types of IF statements are

1. IF Statement
2. IF Else Statement
3. Nested IF else Statement
4. Switch Statement

1. IF Statement :- It may be necessary to carry some operation. If the condition is true, in such cases the simple 'IF' control statement is used. The general syntax of simple 'if' statement is given by. if (condition)

```
{
    Statement 1;
    Statement 2;
    Statement n;
}
Statement X;
```

(The braced block is the If block)

The condition is a relational expression. When the condition is true. The statements within 'IF' block are executed. otherwise the control is transferred to next statement. i.e 'Statement 'x'.

The following program illustrate the IF control statement.

```
#include <stdio.h>
void main()
{
  int marks;
  printf("Input marks \n");
  scanf("%d", &marks);
  if (marks >= 35)
  {
    printf("you are passed");
  }
  printf("end of program");
}
```

O/P

```
Input marks
38
you are passed
end of program
```

2. If else Statement:

The 'if else' statement is required if there are 2 options & to select one at a time. It is a two way branching statement. The general form of 'If else' statement is,

```
if (condition)
{
    Statement 1;
    Statement 2;
        .
        .
    Statement n;
}
```

If block

```
else
{
    Statement 1;
    Statement 2;
        :
    Statement n;
}
Statement X;
```

else block

If the result of condition is true then, the statement within if block are executed. And the control is transferred to Statement x'. Otherwise the statement in the else block are Executed. Note that the statement x' is executed, after the execution of either blocks

```
#include <stdio.h>
Void main()
{
  int marks;
  printf ("input marks \n");
  scanf ("%d", &marks);
  if (marks >= 35)
   printf ("you are passed \n");
   else
   printf ("you are failed \n");
  }
```

o/p

Input marks

24

you are failed

```
else
printf ("c is greater, the value is %d", c);
}
if (b>c)
{
printf ("b is greater, the value is %d", b);
else
printf ("c is greater, the value is %d", c);
}
}
```

o/p

enter three no

1 2 3

c is : greater. value is 3

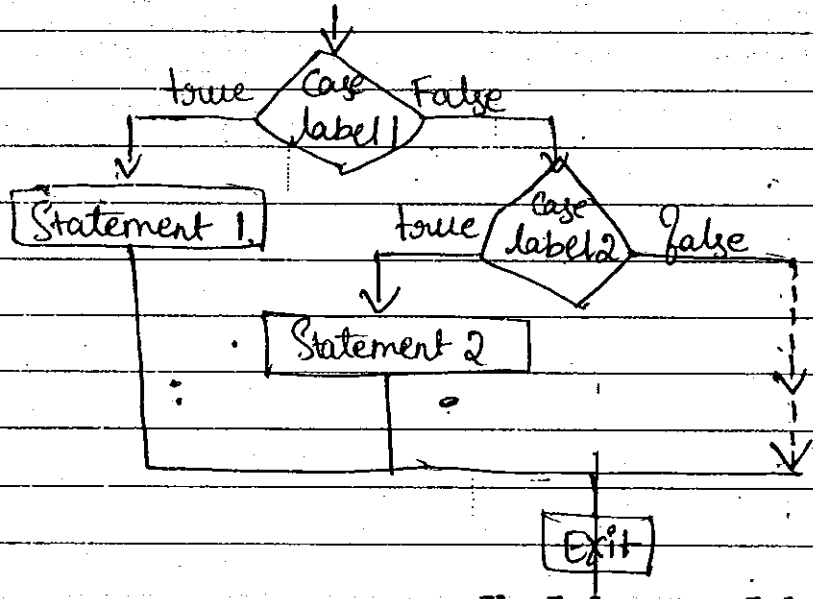
Switch Statement (multiple Selections)

The switch statement allows the programer to select 1 statement for execution out of set of statements. During the execution of switch statement only one of the possible statement will be executed & the remaining statement are skipped, the general format of switch statement is given below.

```
Switch (Expression)
{
case case-label-1 : Statement_list_1;
break;
```

```
case case_label_2 : Statement_list_2;
                    break;
case case_label_3 : Statement_list_3;
                    break;
.
.
.
default : Statement_list_n;
          break;
}
```

When the switch statement is executed the control expression is executed first & the value is compared with the case label value in the given order. If the label matches with the value of expression then statements following that label are executed If non of the value matches then default statement is executed. The default is optional in switch statement



Eg:
```
Switch (j)
{
    Case 1 : k = k+1 ;
             break ;
    Case 2 : k = k+2 ;
             break ;
    Case 3 : k = k+3 ;
             break ;
    default : k = 1 ;
              break ;
}
```

with a program that accepts the day number of the week & displays corresponding week day.

```
#include <stdio.h>
main()
{
    int day no;
printf ("enter day number of the week");
 scanf ("%d", &day no);
Switch (day no)
{
 Case 1 : printf ("Sunday") ; break ;
 Case 2 : printf ("Monday") ; break ;
 Case 3 : printf ("Tuesday") ; break ;
 Case 4 : printf ("Wednesday") ; break ;
 Case 5 : printf ("Thursday") ; break ;
 Case 6 : printf ("Friday") ; break ;
 Case 7 : printf ("Saturday") ; break ;
 default : printf ("invalid day") ; break ;
}
```

o/p
enter day no of the week
5
Thursday

write a program to i/p in number
If the Right most digit of n is zero then
o/p '0' or display '0' If it is 1 then
display 1 so on to 9

```
# include <Stdio.h>
# include <conio.h>
void main ()
{
  int num, r;
  Printf ("enter a number");
  Scanf ("%d", & num);
    r = num % 10;
Switch (r)
{
Case 0 : Printf ("Zero"); break;
Case 1 : printf ("one"); break;
Case 2 : print f ("two"); break;
Case 3 : printf ("three"); break;
Case 4 : printf ("Four"); break;
Case 5 : printf ("Five"); break;
Case 6 : printf ("Six"); break;
Case 7 : printf ("Seven"); break;
Case 8 : printf ("Eight"); break;
Case 9 : printf ("Nine"); break;
}
}
```

o/p ①

enter a number
1246
Six

o/p ②

enter a number
1460
Zero.

UNCONDITIONAL

Goto Statement: The goto statement is useful for unconditional transfer of control it is used to transfer the program control from one statement to other statement unconditionally.

The General syntax of goto syntax is given by

```
goto label;
```

Eg:-
```
loop : printf ("Bangalore");
goto loop;
```

In the above example loop is the label & go to Jumps to loop statement unconditionally The goto statement is rarely used in C-programing The program may contain several goto statement. The label name must be same

The following program illustrates it.

Program to find the sum of N natural number using go to statement

```
#include <stdio.h>
#include <conio.h>
Void main()
{
    int n, sum=0, i=0;
```

```
printf (" enter a number");
Scanf (" %d", &n);
- loop : i++;
Sum = Sum + i ;
  if (i<n) goto loop;
printf (" \n sum of %d natural no = %d",
          n, sum);
    }
```

o/p
enter a number
3
Sum of 3 natural no is 6

i=1
Sum=1
i=2
Sum=3
i=3
Sum=6

Break Statement:
The Break Statement is used to terminate loops or exit the loop. It can be used with in while, do while, or for statement. when break is encountered inside any loop the control automatically comeout of the loop. It provides a convinient way to terminate the loop - the general syntax is given by break;
program to test whether the number is prime or not to illustrate break Statement

```
# include <stdio.h>
# include <conio.h>
  void main ()
{
  int num, i;
    printf (" enter a number");
    Scanf (" %d", &num);
}
```

```
For (i=2; i<n; i++)
    if (num % i == 0)
{
    printf ("\n %d is not prime", num);
    break;
}
    if (i>=n)
printf ("%d is a prime no", num);
    }
```

The Continue Statement

In some programming situations we want to take the control to the beginning of the loop by passing the statements inside the loop which have not yet been executed when the keyword continue is encountered inside any loop control automatically passes to the beging of the loop the syntax is Continue program to display all numbers from one to n which are not divisible by 5

```
# include <stdio.h>
# include <conio.h>
  void main ()
  {
     int i=0, num;
  printf ("enter a number");
  scanf ("%d", &num);
   while (i++ <= num)
      {
        if (i % 5 == 0)
    continue;
    else
    printf ("%d \t", i);
```

o/p
Enter the number
10
1 2 3 4 5 6 7 8 9

The Exit function ( ) :

The purpose of exit ( ) function is to terminate the execution of the program. Break statement terminates the execution of loop but exit statement terminates the program itself. Headerfile stdio.h should be included to use this function. The general syntax is exit ( ); program to display all numbers from 1 to n which are not divisable by 5.

```
#include <stdio.h>
#include <stdlib.h>
Void main ()
  {
   int i=0, num;
  printf ("enter a number");
  Scanf ("%d", &num);
   while (i++<=num)
        {  if (i%5==0)
             exit ();
        else
        printf ("%d /t", i);
        }
  }
```

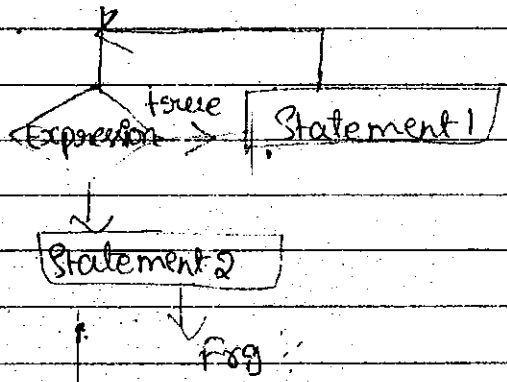
o/p
enter the number
10
1 2 3 4
C:\TC\Bin

looping Statements : In certain situations it is necessary to execute a set of statements more than once repeatedly In such cases the looping statements are used a looping statements executes the statements repeatedly for specified number of times as long as the condition is true
There are 3 types of looping Statements

1. The while loop
2. The Dowhile loop
3. The For loop

1. The while loop

while loop



Flow chart of while

The while Statements permits a block of Statements to be executed repeatedly as long as the Condition is true. The while is an entry Controlled statement because the Condition is tested at the begining before executing the body of the loop. The flow chart is as shown above The general syntax of while statement is given by

Syntax
while (expression)

```
{
    Statement 1;
    Statement 2;
    .  .  .
    Statement n;
}
    Statement X;
```

— Body of while

The Condition is executed & if the condition is true then the body of the loop is executed again the condition is once again tested & if true the body of the loop is executed once again. The process is repeated until the condition is false once the condition becomes false the program execution continues with the Statement after the body of the loop i.e Statement X.

Program to find the sum & average of n numbers.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int n, sum=0, count=1, num;
    float avg;
printf ("enter How many no's to add");
Scanf ("%d", &n);
    While ( count <=n);
{
  printf ("Enter the %d num", count);
  Scanf ("%d", &num);
    Sum = Sum + num;
    count ++;
}
```

```
avg = sum/n;
printf("The sum of %d numbers is %d", n, sum);
printf("The average of %d number is %f", n, avg);
}
```

O/P

enter how many numbers to add

3

enter the 1 num 10

enter the 2 num 20

enter the 3 num 30

The sum of 3 nos is 00

The average of 3 number is 20.00

Count = 1
Sum = 10
Count = 2
Sum = 30
Count = 3
Sum = 60
Count = 4
avg = 60/3
= 20.00

write a program to display all the numbers from 1 to 10

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int i = 1
  while (i <= 10)
  {
    printf("%d", i);
    i++;
  }
}
```

O/P

1 2 3 4 5 6 7 8 9 10.

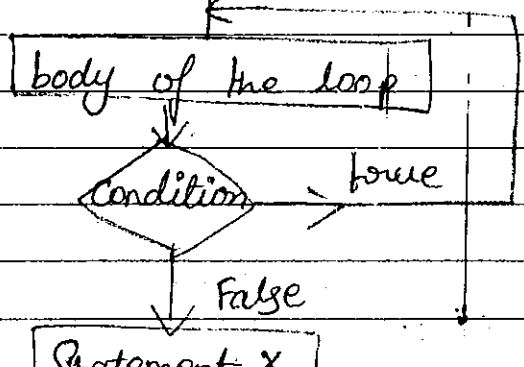
The Do while loop

The Do while loop always executes the body of the loop at least once. It is a exit controlled loop the condition is tested at the bottom of the loop. The general syntax of do while statement

The do-while loop

```
do
{
    Statement 1;
    Statement 2;      } body of the
    Statement n;        dowhile loop
}
while (condition);
    Statement x;
```

The execution of the do while loop is as follows. The body of the loop is executed first then the condition is tested if the condition is true the body of the loop is executed once again. This process continues as long as the condition is true, when the condition is false the loop is terminated & control is transferred to next statement i.e statement X. The flow chart of do while loop is as shown below



classmate
Date
Page

program to illustrate Do while loop

```
# include <stdio.h>
# include <conio.h>
Void main()
{
  Int I=1, Sum=0;
  do
  {
    Sum = Sum + I;
    I = I+2;
  }
  while (Sum < 10 && I < 5);
printf ("Sum = %d \n I = %d", Sum, I);
}
```

O/P

Sum = 4, I = 5

I = 1
Sum = 0
Sum = 1
I = 3
Sum = 4
I = 5

For loop

A "For loop" is an entry control loop It will execute a loop body a specified number of times until a particular condition is satisfied the general form of "For loop" is given by

For loop

```
for (initialization ; test condition ; Increment)
    {
      Statement 1;
      Statement 2;
      ---------
      Statement n;
    }
```

as shown above the form loop containg 3 parts. each part should be separated by semicolon(;) the 3 parts are

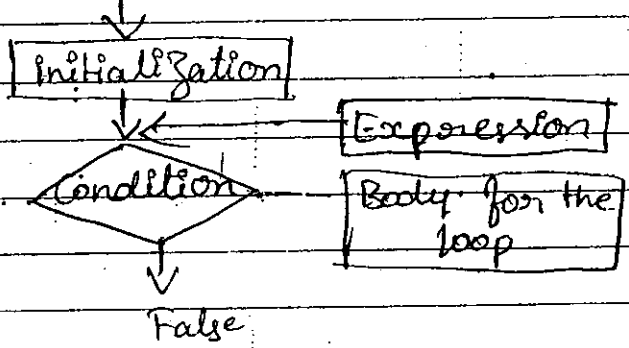
1. Initialization
2. Test condition
3. Increment

The first is the initialization expression. It executes once at the beginning & allows for the initialization of the variables which acts as a counter that controls the loop

The second part is the condition is a relational expression. The condition is executed at the beginning of each iteration of the loop

If it is true the loop continues if false the loop terminates

The third part expression is executed at the end of each iteration It increments or decrement the loop control variable which helps in termination of the loop. The flow chart representation of for statement is as shown.



Program to illustrate for loop

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i=0;
for(i=0; i<=10; i++)
{
printf("\n %d", i);
}
}
```

O/P

1
2
3
4
5
6
7
8
9
10

Comparision b/w while & do while

	while	do while
1.	It is a entry control loop	It is a exit control loop
2	The loop is executed if & only if the condition is true	The loop is executed irrespective of the condition either true or false
3.	The condition is tested at the starting of the loop	The condition is tested at the ending of the loop.
4.	Frequently used	Rarely used for particular condition.
5.	The general syntax is	The general syntax

Date Page 2/9/2011

```
while (condition)
{
Statement 1;
Statement 2;
.
.
Statement n;
}
Statement X;
```

Eg: while a>b
{
temp=a;
}

```
dowhile (condition)
do
{
Statement 1;
Statement 2;
- - - - -
Statement n;
}
while(condition);
Statement x;
```

Eg do
{ temp=a;
} while (a>b)

Differenciate b/w break & continue Statement

	Break	Continue
1.	The break is used to terminate the loop	The continue is used to take the control to the beginning of the loop
2.	It is used in Switch condition.	It is not used in Switch Statement.
3.	The Statements within the loop are executed	The Statements with in the loop are bypassed
4.	The general Syntax: break;	The general Syntax Continue;
5.	Eg while (i==n) { printf("%d",n);	while (i==num)
5.	{ Switch n	{ if (i%5==0)
	{ case 1: printf("%d",n); break;	Continue;
	case 2: printf("%d",n); break;	else printf("%d",i);
	}	}

classmate
Date 2/9/21
Page

Arrays

Data types

- Fundamental types
  - char
  - int
  - float
  - double
- Derrive types
  - arrays
  - functions
  - pointers
- User-defined types
  - Structures
  - Unions
  - Enumeration

An 'array' is a derrived data type consisting of group of similar elements. All data items share a common name & individual data item is accessed by using an index or subscript in brackets after the array name.

Definition

(Array is defined as a group of similar data types i.e a group of characters, group of int's a group of floats & a group of doubles.)

Ex:- char name [2];
int num [10];
float bal [20];
double bal [30];)

In the first example array of 2 characters is defined with the name. Similarly num is an array which consists a group of 10 elements of integer data

Declaration of Arrays

The arrays must be declared before they are used. The general syntax of array declaration is given by

data type array name [array size];

where data type is any allowed data type & array name is an identifier & array's size is the size of the array that is number of element enclosed in square brackets. The square bracket indicates it is an array. Ex:- int num[5]; (data type, array name, array size)

0	1	2	3	4
11	12	13	14	15
1	2	3	4	5

In the above example num is an array consisting of 5 elements of type integer.

Types of arrays

Arrays can be broadly classified into 3 types

1. One Dimensional
2. Two Dimensional
3. Multi Dimensional

One Dimensional array :- If the array is having only one subscript or 1 Index or 1 array size or 1 [ ]. They are called one Dimensional array.

Ex:- int, num, [5]

Two Dimensional array :- If the array is having two subscript or 2 Index or 2 array size or 2 [ ].

They are called two Dimensional array.
Ex:- int num [ ] [3].

MultiDimensional array.
If the array is having more than 2 Index or subscript or array size or [ ], then it is called multidimensional array.
Eg:- int num [ ] [ ] [ ] [ ]

Array's Initialization OR Initialization of one ~~array's~~ dimensional array's

Once array has been declared & necessary memory locations are reserved for array the next step is to enter information into the array created. This process is known as initialization. This is done as follows.

Array_name [subscript] = value;

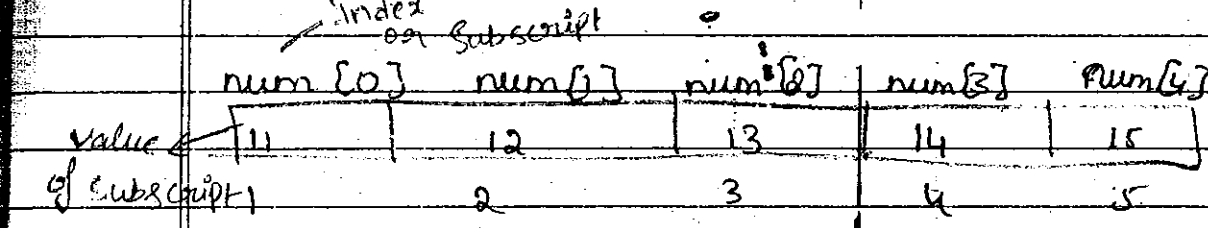
Eg:- int num [3];
num [0] = 11;
num [1] = 12;
num [2] = 13;

we can also assign the values to array's initially when we declare

Eg:- int num (5) = { 11, 12, 13, 14, 15 };

In the above example num is an array of 5 elements & will assign values 11, 12, 13, 14, 15 to num [0], num [1], num [2], num [3], num [4]. i.e

```
num[0] = 11;
num[1] = 12;
num[2] = 13;
num[3] = 14;
num[4] = 15;
```



The values of num is stored in memory location as follows

The above diagram shows the memory representation of one dimensional array.
The values of array elements Store in continues memory locations.
The subscript or index indicates the position of array element.

One dimensional

(Program to illistrate array processing of array's that takes the array size & display the array.

```
# include <stdio.h>
# include <conio.h>
    void main();
{
    int num[3], i=0, n;
    printf("enter array size");
    scanf("%d", &n);
    /* Reading array elements */
    for (i=0; i<n; i++)
    {
```

```
printf ("enter %d element", i);
scanf ("%d", &num[i]);
}
/* To display array elements */
for (i=0; i<n; i++)
{
printf ("The array & array elements
in num[%d] = %d", i, num[i]);
}
}
```

O/P

enter array size: 3
enter 0 element: 11
enter 1 element: 12
enter 2 element: 13) * In

The array & array elements are
num[0] = 11
num[1] = 12
num[2] = 13

write a program to find the largest number in an array.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int array[50], largest, num, i;
printf("enter the array size\n");
scanf("%d", &num);
/* Reading array elements */
for(i=0; i<num; i++)
{
scanf("%d", &array[i]);
largest = a[0];
}
/* To find largest no */
for(i=1; i<num, i++);
{
if (largest < array[i]);
{
largest = array[i];
}
}
printf("The largest number is %d",
largest);
}
```

o/p Enter the array size

array[0] = 11 · largest = 40

[1] = 12 — i = 1, num = 5 — 11 < 12

[2] = 15 — i = 2, num = 5 — 12 < 15

[3] = 25 — i = 3, num = 5 — 15 < 25

[4] = 40 — i = 4, num = 5 — 25 < 40

i = 5, num = 5

Program to find the smallest in an array of .

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int array[50], smallest, num, i;
printf ("enter the array size \n");
  scanf ("%d", &num);
  /* Reading array elements */
  for (i=0, i<num; i++)
  {
    scanf ("%d", &array[i]);
    Smallest = array[0];
  }
  /* To find smallest no */
  for (i=1; i<num; i++)
  {
    if (Smallest > array[i]);
    {
      smallest = array[i];
    }
  }
  printf ("The smallest number is %d", smallest);
  }
```

Smallest = 10

O/P enter the array size

array[0] = 40 — 40 > 12

[1] = 12 — 12 > 15

[2] = 15 — 12 > 10

[3] = 10 — 10 > 11

[4] = 11

Date
Page

Sorting of Arrays

Bubble Sort — Selection Sort

Sorting is a technique of arranging elements either in ascending or descending order. There are 2 types sorting techniques for array elements. They are
1) Bubble sort 2) Selection sort

Bubble sort :- In bubble sort each element is compared with the adjacent element i.e the next element. If the first element is larger than the second element then the position of the elements are interchanged otherwise it is not changed. Then the next element is compared with adjacent element & same process is repeated for all the elements in the array.

During the first pass the largest element occupies the last position. During the next pass the same process is repeated leaving the largest element. The same process is repeated until no more elements are left for comparision & the resulting array is sorted array.

Write a program to arrange n numbers in Asending order using bubble sort method.

```
#include <stdio.h>
#include <conio.h>
Void main ()
{
  int arr[100], n, i, temp;
Printf ("enter the array size");
  Scanf ("%d", &n);
/* Reading array elements */
Printf ("enter the array elements");
 for (i=0; i<n; i++)
  {
    scanf ("%d", &arr[i]);
  }
/* Sorting the array */
for (i=1 ; i<n ; i++)
  {
 for(J= 0 ; J<n-i ; J++)
  {
   if (arr [J] > arr [J+1])
     {
       temp = arr [J];
   arr [J] = arr [J+1];
   arr [J+1] = temp;
}
}
}
  /* Printing sorted array */
Printf ("The array sorted in
ascending order using bubble sort method is");
```

```
for (i=0; i<n; i++)
    {
    printf ("\t%d", array");
    }
    {
    Getch ();
    }
```

o/p

enter the array size

~~enter~~

enter the array elements

	85	12	92	11	62	arr(0)=85
①	12	85	11	62	92	arr(1)=12
②	12	11	62	85	92	arr(2)=92
③	11	12	62	85	92	arr(3)=11
④	11	12	62	85	92	arr(4)=62

i = 1 J = 0
i<5 J<4 ④

i = 2 J = 0
i<5 J<3 ③

i = 3 J = 0
i<5 J<2 ②

i = 4 J = 0
i<5 J<1 ①

is");

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int arr[100], n, i, temp;
    printf("enter the array size");
    scanf("%d", &n);
/* Reading array elements */
    printf("enter the array elements");
    for (i=0; i<n; i++)
    {
        scanf("%d", &arr[i]);
    }
/* Sorting the array */
    for (i=1; i<n; i++)
    {
        for (J=0; J<n-i; J++)
        {
            if (arr[J] > arr[J+1])
            {
                temp = arr[J];
                arr[J] = arr[J+1];
                arr[J+1] = temp;
            }
        }
    }
}
```

/* Printing sorted array */
printf ("The array sorted in ayending Order using bubble sort method is");

```
for (i=0; i<n; i++)
    {
    printf ("\t %d", array[i]);
    }
getch();
}
```

O/P

enter the array size
enter the array elements

85 12 92 11 62

① 85 92 12 62 11

② 92 85 62 12 11

③

④

Selection Sort

Selection sort is based on the extension of minimum & maximum technique i.e the means of nested for loop. A pass through the array is made to locate the minimum value. Once it is found it is placed in the first position of the array i.e position (0). Another pass through the remaining elements is made to find the next smallest element & it is placed in position one & so on.

Once the next number is compared with the last one, all the elements are sorted in Asending Order.

```
# include<stdio.h>
# include<conio.h>
void main()
{
int arr[100], n, i, J, temp;
printf("enter the array size");
scanf("%d", &n);
printf("enter the array elements");
   /* Reading array elements */
   for (i=0; i<n; i++)
   {
   scanf("%d", &arr[i]);
   }

/* To sort the array in Asending order */
   for(i=0; i<n; i++)
   {
   for(J=i+1; J<n; J++)
   {
```

```
if (arr[i] > arr[j])
{
    temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
}

/* To print the sorted array */
printf("Array sort in Asending order");
for (i=0; i<n; i++)
{
    printf("\t%d", arr[i]);
}
getch();
}
```

o/p

enter the array size [~~100~~ 5]

enter the array elements.

85 12 91 72 11

n = 5 arr[0], arr[1] => 85 > 12

i = 0 t = 85 arr[0] = 12

j = 1 arr[1] = 85

12 85 91 72 11

n = 5 arr[0] > arr[2]

i = 0 12 > 91

j = 2 12 85 91 72 11

n = 5 arr[0] > arr[3]

i = 0 12 > 72

j = 3 12 85 91 72 11

n=5 arr[0] > arr[4]

i=0 12 > 11

J=4 11 85 91 72 12

85 12 91 72 11

12 85 91 72 11

12 85 91 72 11

12 85 91 72 11

11 85 91 72 12

11 85 91 72 12

11 72 91 85 12

11 12 91 85 72

11 12 91 85 72

11 12 85 91 72

11 12 72 91 85

11 12 72 91 85

11 12 72 85 91

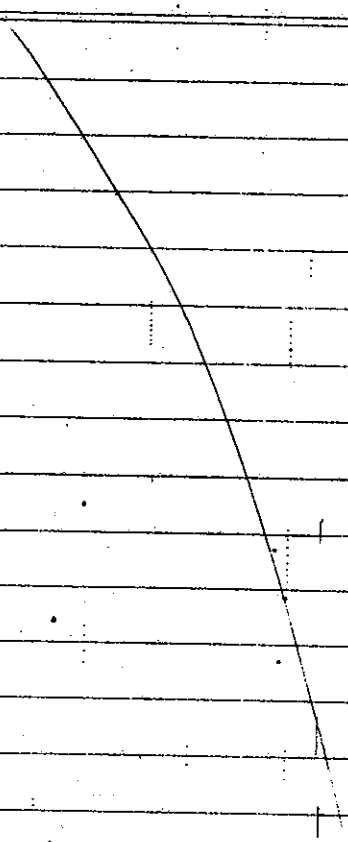
Exam

Searching an element in the array
There are 2 types to search the element in a array they are
1. linear search
2. Binary Search

linear Search
In linear search we access each element of the array in wether it is the desired or required element or not. If found the search is successful. Search will be unsuccessful when all the element are accessed & desired element is not found.

The following program illustrate i.e. Write a 'C' program to search the given number in an array of N numbers using linear search method.

```
#include <stdio.h>
#include <conio.h>
void main
{
int array [100], n, i, num;
res;
printf ("enter the array size");
scanf ("%d", &n);
printf ("enter the array elements");
```



```
/* Reading array elements */
for (i=0; i<n; i++)
{
    Scanf ("%d", &arr[i]);
}
printf ("enter the num to search in the array")
Scanf ("%d", &num);
/* To find the num present or not */
for (i=0; i<n; i++)
{
    if (num == arr[i])
    {
        res = arr[i];
        break;
    }
}
if (res == num)
    printf ("The num %d found", num);
else
    Printf ("The num %d not found", num
}
```

Enter the array size
5
Enter the array elements
2 8 9 11 78
Enter the num to search
11
The num 11 found

n=5
arr(0) = 2
arr(1) = 8
arr(2) = 9
arr(3) = 11
arr(4) = 78

i=0	i=0	i=0	i=0
n=5	n=5	n=5	n=5
num=11	num=11	num=11	num=11
arr(0)=02	arr(1)=8	arr(2)=9	arr(3)=11
11=2	11=8	11=9	11=11

o/p

Enter the array size

5

Enter the array elements

02 8 9 11 78

Enter the num to search

72

The num 72 is not found

i=0	i=0	i=0	i=0
n=5	n=5	n=5	n=5
num=72	num=72	num=72	num=72
arr(0)=02	arr(1)=8	arr(2)=9	arr(3)=11
72=2	72=8	72=9	72=11

i=0

n=5

num=72

arr(4)=78

72=78

Program to search the element in the array of n numbers & display the position if found using linear search method.

```
void main()
{
  int arr[100], n, i, num, res = 0, position = 0;
  printf ("enter the array size");
  Scanf ("%d", &n);
  printf ("enter the array elements");
  /* Reading array elements */
  for (i=0; i<n; i++)
  {
    Scanf ("%d", &arr[i]);
  }
  printf ("enter the num to search in the array
  Scanf ("%d", &num);
  /* To find the num present or not */
  for (i=0; i<n; i++)
  {
    if (num == arr[i])
    {
      res = arr[i]; pos = i+1;
      break;
    }
  }
  if (res == num)
    printf ("The num %d found in %d
        position", num, pos);
  else
    printf ("The num %d not found", num
}
```

o/p

Enter the array size

5

Enter the array elements

2 8 9 11 78

Enter the num to search
8
The num 11 found
at 4 position

n=5
arr(0)=2
(1)=8
(2)=9
(3)=11
(4)=78
num=11

i=0	i=0	i=2	i=3+1=4
n=5	n=5	n=5	n=5
num=11	num=11	num=11	num=11
arr(0)=2	arr(1)=8	arr(2)=9	arr(3)=11
11=2	8=8	11=9	

Two Dimensional Array

Two dimensional array is a array having 2 subscripts or 2 square brackets
The first subscript refers to the row & the second subscript refers to the column

The general syntax of 2 Dimensional array is given by

data type array_name[row size][column size]

where row size & column size are the integers specify the number of rows & number of columns respectively.

ex:- int a[3][3];

In the above example A is a two dimensional array that contains 3 rows & 3 columns. The two square bracket indicates it is a two dimensional array

The two dimensional array is stored in memory as shown below

a[0][0]
a[0][1]
a[1][2]
a[1][0]
a[1][1]
a[1][2]
a[2][0]
a[2][1]
a[2][2]

The array elements can be accessed by using 2 subscript. The first selects the row & the second selects the column within that row.

example: a[1][2] → refers to element present in 1st row & 2nd column.

Declaration of 2 dimensional array. The 2 dimensional array can be declared & initialised with the initial values for example:-

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

a is a 2 dimensional array with 3 rows & 3 columns. The initial values are enclosed in braces. The values are initialised to the array elements given by,

a[0][0]=1 a[0][1]=2 a[0][2]=3
a[1][0]=4 a[1][1]=5 a[1][2]=6
a[2][0]=7 a[2][1]=8 a[2][2]=9

In the above illustration each row is 1 dimensional array hence 2 dimensional array is treated as an array of 1 dimensional arrays. Therefore the above declaration can be written as

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

Program to illustrate 2 dimensional array to read & display a matrix square.

```
#include<stdio.h>
#include<conio.h>
Void main()
{ int a[10][10];
  int n, i, j;
printf ("enter order of matrix");
Scanf ("%d", &n);
printf ("enter matrix elements");
    for (i=0; i<n; i++) /* Row no */
{
   for (j=0; j<n; j++)
{
  Scanf ("%d", &a[i][j]);
}
}
/* printing matrix elements */
   printf ("matrix A is \n");
     for (i=0; i<n; i++)
       {
          for( j=0; j<n; j++)
{
  printf ("%d\t", d[i][j]);
}
  printf ("\n");
}
}
```

enter order of matrix
2
24 enter matrix elements
6 10 2 4 6 10
matrix A
2 4
6 10

n = 2 order
i = 2 row
j = 2 column
a[0][0] = 2 a[0][1] = 4
a[1][0] = 6 a[1][1] = 10

Date
Page

a program to read the order of the matrix
Page
display the matrix & transpose of that matrix

```
void main()
{
  int a[10][10], b[10][10];
  int n, m, i, j;
  printf("enter order of matrix");
  scanf("%d%d", &n, &m);
  printf("enter matrix elements");
  /* Reading element */
  for(i=0; i<n; i++)  /* Row no */
  {
    for(j=0; j<m; j++)
    {
      scanf("%d", &a[i][j]);
      b[i][j] = a[i][j];
    }
  }
  printf("/* Transpose of A is */");
  for(i=0; i<m; i++)
  {
    for(j=0; j<n; j++)
    {
      printf("%d\t", b[i][j]);
    }
    printf("\n");
  }
}
```

matrix A,
2 4 6
10 9 8
Transpose of matrix A is
2 10
4 9
6 8

Read to any 2 N matrix A & B, display the sum of A & B

```
void main ()
{
  int a[10][10], b[10][10], c[10][10];
  int n, i, j;
  printf ("enter order of matrix A & B");
  scanf ("%d", &n);
  printf ("enter matrix A elements");
      /* Reading elements */
  for (i=0; i<n; i++)  /* Row no */
  {
    for (j=0; j<n; j++)
    {
      scanf ("%d", &a[i][j]);
    }
      /* printing matrix B elements */
    printf ("enter matrix B is \n");
      for (i=0; i<n; i++)
    {
      for (j=0; j<n; j++)
      {
        scanf ("%d", b[i][j]);
      }
```

```
void main ()
{
int a[10][10], b[10][10], c[10][10];
int n i,j;
printf("enter order of matrix A & B");
scanf("%d", &n);
printf("enter matrix A elements");
  /* Reading elements */
for (i=0; i<n; i++)  /* Row no */
{
 for (j=0; j<n; j++)
  {
  scanf ("%d", &a[i][j]);
  }
 printf (/* Sum of A&B");
  for (i=0; i<n; i++)
    {
     for (j=0; j<n; j++)
{
  c[i][j] = a[i][j] + b[i][j]
  printf ("%d \t", c[i][j]);
}
  printf ("\n");
    }
    }
```

Structures & Unions

We know that array is a group of similar data type. In some situations it is necessary to store a group of data of different data types. This can be implemented by Structures.

Structure is a user defined data type which is a collection of different data types i.e integers, floats, characters. In simple Structure is defined as a group of dissimilar data elements.

```
Struct Accounts {
    char name [80];
    char acct_type [5];
    float balance;
} cust1, cust2, ------, custn;
```

Structure members

Structure variable

Structure Declaration.

The Structure must be declared before it is processed. So general syntax of Structure declaration is given by.

```
Struct Structure_name {
        member 1;
        member 2;
        member n;
    };
```

The Structure is declared with the keyword "Struct" followed by the Structure name, a valid identifier in C.

The different members with different data types are declared by enclosing them with in braces

Structure declaration must end with semicolon

```
Struct Account {
    int account no;
    char name [80];
    char acct type [5];
    float balance;
} cust 1, cust 2 ---- cust n;
```

Structure members

In the above example account is the Structure name & it is having four members account no, name, acc type, balance

Structure variable declaration:

Once the Structure has been defined the Structure variables can be declared of that type

The general Syntax of Structure variable declaration is given by,

```
Struct Structure — name variable 1, variable 2,
    ---- variable n ;
```

Example: Struct account cust 1, cust 2 ---- cust n;

In the above example Struct is the required keyword, account is the name of the Structure. Customes 1, cust 2, so on to cust n are the Structure variables of type account where account is a Structure

It is possible to combine Structure declaration & Structure variable declaration as shown below

```
Struct account {
    int acct-no; - - - - - - - - - - - -
    char name [80];
    char acct-type [5];
    float balance;
```

};
cust 1, cust 2 ;

processing a Structure or accessing Structure members :

Once the Structure composition has been written the members of Structure are process or accessed has separate entities.

To access the individual Structure member, the '.' Dot operator is used. It is also called as period operator.

The general syntax is given by

Structure - Variable - name . member ;

It is illustrated using following example

```
Struct Account {
    int acct - no ;
    Char name [80] ;
    Char acct - type [5] ;
    float balance ;
};
Customer ;
```

In the above example customer is a Structure variable of type account where account is a structure. to access the Structure members we use the dot operator as shown below

```
customer . acct - no ;
customer . balance ;
customer . name ;  - - - - - -
```

Program to illustrate processing of Structure :-

```
Struct Student {
void main()
{
  int roll no;
Char name [20];
  int mark;
};
Void main ()
{
  Struct Student S1;
  S1. roll no = 1234;
  S1. name = "Rakesh";
  S1. marks = 78;

/* Printing the Student details */
Printf ("Roll no = %d \n", S1. roll no);
Printf ("Name = %S \n", S1. name);
Printf ("Marks = %d \n", S1. marks);
}
```

O/P

Roll no = 1234
Name = Rakesh
Marks = 78

```
#include <stdio.h>
#include <conio.h>
Struct Student {
int roll no ;
char name [20];
int marks ;
};
void main ()
{
    Struct Student S1;
Printf ("enter the student details");
Printf ("enter the Student roll no");
Scanf ("%d", & S1.roll no);
Printf ("enter the Student name");
Scanf ("%s", & S1.name);
Printf ("enter marks");
Scanf ("%d", S1 marks);
```

O/P

```
enter the Student details
enter Student roll no
1234
enter the Student name
Arun
enter marks
17
The Student details are
Roll no = 1234
Name = Arun
Marks = 17
```

Structure within a Structure OR Nested Structure

A Structure that contains the other Structure has a member is called a Nested Structure that is one of the member is a Structure

Example:-

```
Struct account {
    int acc no ;
    Char acct name [80];
    Char acct type [5];
    float Balance ;
};

Struct date {
    int day ;
    int month ;
    int year ;
};
Struct account {
    int acct no ;
    Char acct name [80];
    float bal ;
    Struct date Doj ;        Date of joining
} cust ;

Struct Sub marks {
    int m2 ;
    int Sc ;
    int SCD ;
    int Cprog ;
};
```

```
Struct Std name [20];
int roll no;
int Sem;
Struct Subman Stud,
{
 Sl;
```

```
Sl.Std na = Nethra
Sl. roll no = 23
Sl. Sem = 2
Sl. Studm.M2 = 92;
Sl-Studm.SC = 95;
Sl. Studm.SCD = 98;
Sl. Studm. C-P = 96;
```

write a program to display Student details & Subject marks using nested Structure.

```
#include <stdio.h>
#include <conio.h>
Struct Student {
  int roll no;
  char name [20];
  int marks [ SCD, M-2, SC, C-P];
  int Sem;
};
  void main ( )
  {
    Struct Student Sl;
printf (" enter the Student details");
printf (" enter the Student roll no");
scanf (" %d", & Sl. roll no);
printf (" enter the Student name");
scanf (" %d", & Student.name);
printf (" enter
```

```
Struct subm {
    int m2, SC, SCD, C-Pro;
};
Struct Stud {
    char name[20];
    int RN;
    int Sem;
    Struct Subm Ind;
};
    void main()
    {
    Struct Student S1;
printf("enter Student details");
printf("enter name \n");
scanf("%s", &S1.name);
printf("enter Roll no \n");
scanf("%d", &S1.RN);
printf("enter Sem \n");
scanf("%d", &S1.Sem);
printf("enter marks \n");
scanf("%d %d %d %d", &S1.Ind.m2,
                     &S1.Ind.SC,
                     &S1.Ind.SCD,
                     &S1.Ind.C-P.);
printf("The Student details are \n");
printf("\nName = %s", St.name);
printf("\n Roll no = %d", S1.R.No);
printf("\n Sem = %d", S1.Sem);
printf("\n m1 = %d \n SC = %d
\n SCD = %d \n C-P = %d",
S1.Ind.m2,
S1.Ind.SC,
S1.Ind.SCD,
S1.Ind.C-P);
}
```

o/p

enter the Student details
enter name Santhosh
enter roll no 123
enter sem 2
enter marks 38 50 60 50

The Student details are
Name = Santhosh
Roll no = 123
Sem = 2
m1 = 38
SC = 50
SCD = 60
C-P = 50

```
Struct Employee {
  char name [80] ;
  Char Department [80] ;
  int ID no ;
  Int Contact no ;
} ;
  void main ( )
  {
      Struct Employee S1 ;
printf ("enter Employee details") ;
printf (" enter name \n") ;
  Scanf (" %.s ", &S1.name) ;
printf (" enter Department \n") ;
  Scanf (" %.s", &S1.Department) ;
printf (" enter ID no \n") ;
  Scanf (" %d", & S1. ID no) ;
printf (" enter Contact no \n") ;
  Scanf (" %d", & S1. Contact no) ;
```

```
printf ("The Employee details are \n");
printf ("Name = %s", S1.name);
printf ("Department = %s", S1.department);
printf ("ID no = %d", S1.ID no);
printf ("Contact no = %d", S1.Contact no);
}
```

O/P

enter the Employee details
enter name : Santhosh
enter Department : R&D
enter ID no : 1345
enter contact no : 9035987364

The Employee details are
Name : Santhosh
Department : R&D
ID no : 1345
Contact no : 9035987364

Array of Structures

Array is a group of similar data types As we can declare an array an array of Structures can also be declared.

An array that will have each element as a structure is called array of Structure

Ex
```
Struct Employee {
    char name [80];
    char dept [20];
    Int Id no;
    int ph no;
} emp [100];
```

In the above example m of [100] is a array of Structure - variable of type empdte where emp det is the Structure which contains four members now emp of [100] is 100 Structures each structure having four members

Declaration & Initialization of Array of Structures:-

The initialization & declaration of array of Structures can be done in the same way as a common Structure. where each members are seperated by, the below example illustrate it

```
Struct Emp det {
char name [80];
char Dept [20];
int id no;
};
```

} initialization

```
Struct Emp det Emp [2] =
{ { "Hariish", "Production", 1120 }
  { "mahesh", "R&D", 1180 } };
```

} declaration

Program to illustrate array of Structure.

To read & display n student details using array of Structures.

Date ______
Page ______

```
Struct Student {
    Char name [80];
    Int RN;
    int Sem;
};

void main ()
{
    int i, n;
    Struct Student S1 [100];
    Printf ("enter no of Students");
    Scanf ("%d", &n);
    for (i=0; i<n; i++)
    {
        Printf ("enter Student [%d] details",
                i+1];
        Printf (" enter name \n");
        Scanf ("%s", & S1[i].name),
        Printf (" enter Roll no \n");
        Scanf (" %d", & S1 [i]. RN);
        Printf (" enter Sem " \n);
        Scanf (" %d", & S1[i]. Sem);
    }

    for (i=0; i<n; i++)
    {
        Printf (" Student [%d] details are, i+1);
        Printf (" Name= %S", S1[i]. name");
        Printf (" Roll no= %d", S1 [i]. RN");
        Printf ("Sem = %d", S1 [i].Sem ");
    }
}
```

o/p

enter no of Students
2
Enter Students 1 details
enter name Akshay
Enter roll no 1123
enter Sem 2

enter ~~no~~ Student 2 details
enter name Nethra
enter roll 1156
enter Sem 2

Students 1 determined

Name = Akshay
Roll no = 1123
Sem = 2

Students 2 details are
name = Nethra
Roll no = 1156
Sem = 2

Write a C program to read & display cust acc details using array of Struct.

```
#include <stdio.h>
#include <conio.h>
: Struct date {
    int day;
    int Month;
    int year;
    };
Struct acct {
```

```
Char name [80];
Int acct no;
float bal;        balance
Struct date lp;   - last payment
};

void main()
{
 Struct acc cust [10];
 Int n, i;
 printf ("enter no of customers");
 Scanf ("%d", &n);
for (i=0; i<n; i++)
  {
  printf ("enter %d cust details", i+1);

Scanf ("%s %d %f | %d %d %d",
          & cust[i].name,
          & cust[i].acct no,
          & cust[i].bal
        & Cust[i].lp.day,
        & cust[i].lp.month,
        & cust[i].lp.year);
  }
for (i=0; i<n; i++)
{
printf ("cust %d details are", i+1);
printf ("Name = %s \n acct no = %d \n
        bal = %f \n
        last payment = %d-%d-%d",
            cust[i].name,
            cust[i].acct no,
            cust[i].bal,
```

acc details
Struct

```
cust[i].dp.day,
cust[i].dp.month,
cust[i].dp.year);
}
}
```

O/P

~~enter~~ ~~no of~~ customer 1 details are
Name = Avinash
acct no = 1234
bal = 111.2
last payment = 12-09-2011

customer 2 details are
Name = Santhosh
acct no = 1235
bal = 156.2
last payment = 15-08-2011.

Unions

The union is a data type having the similar Concept of structures. The Unions can be defined as

" union is a collection of dissimilar data type in which the members share the same storage area with in the memory.

The unions are created & processed As same as structures. but the keyword is union. The major difference b/w

structure & union is storage space.

In structures each member has it's own storage space - but in case of union all the members share only one common storage space i.e largest member of union.

The other main difference is in unions only one member is processed ate a time but in structures all the members can be processed at a time.

Example of union.

```
union value {
    int i;
    float f;
} variable;
```

difference b/w structure & union.

	Structure	union
1.	The keyword STRUCT is used to Define a Structure	The keyword UNION is used to define a union
2.	Once the structure is declared the Compiler will reserves memory to all of it's members.	In unions the Compiler will Reserve memory only to the largest member of union
3.	In structures all the members can be accessed at a time.	In unions only one member can be accessed at a time.
4.	maximum memory utilization	minimum memory utilization
5.	Generally used	Used for a specific task
6.	Ex: struct value { int i; float f; } var;	In which memory is at most concern. Ex: union value { int i; float f; } v;

7) For above example The compiler Reserves 6 bytes of memory

For above example the compiler reserves 4 bytes of memory.

Declaration of Unions

The general syntax of union declaration is given by

```
union union_name {
    data type member 1 ;
    data type member 2 ;
    :
    data type member n ;
} var1, var2, var3, ..... varn ;
```

where union is the required keyword. union name is the any valid identifier, data type member 1 is the member of the union of basic data type. variable 1, variable 2, ----- variable n ; are the union variables

Example:

```
union value {
    int i ;
    float f ;
} t, sum ;
```

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

union value {
    int i ;
    float f ;
```

xample
reserves
memory.

ation

'arn;

- union
type
nion of
= 2,
n

```
};
Void main ()
{
  union value sum;
  clrscr ();
  printf ("enter the value for i\n");
  Scanf ("%d", & Sum.i);
  printf ("enter the value for f\n");
  Scanf ("%f", & Sum.f);
  printf ("the value of i = %d\n", Sum.i
  printf ("the value of f = %d\n", Sum.f
  getch ();
}
```

o/p
enter the value for i
& 2
enter the value for f
the value of i = 0
the value of f = 23.0000

Program to illustrate the size of union & Structure.

```
# include <stdio.h>
# include <conio.h>
  union value {
  int i;
  float f;
  };
  Struct num {
  int k[10];
```

```
float i[10];
};
void main()
{
union value sum[10];
struct num val[20];
clrscr();
printf("the size of union = %d \n", size of
(sum));
printf("the size of structure = %d \n",
size of (val));
getch();
}
```

o/p

the size of union = 40
the size of structure = 1200

```
int arr[100], n, i;
clrscr();
printf("enter the array size \n");
scanf("%d", &n);
printf("enter the array elements \n");
for(i=0; i<n; i++)
{
printf("enter the %d array
element \n", i+1);
scanf("%d", &arr[i]);
}
printf("The entered array
elements are \n");
```

```
for (i=0; i<n; i++)
{
printf ("arr[%d] = %d  %d
position \n", i, arr[i], i+1);
}
getch();
}
```

O/P

enter the array size
4
enter the array elements
enter the 1 array element
11
enter the 2 array element
22
enter the 3 array element
33
enter the 4 array element
44
enter the 5 array element
55

The entered array elements are

arr[0] = 11	1 position
arr[1] = 22	2 position
arr[2] = 33	3 position
arr[3] = 44	4 position
arr[4] = 55	5 position

```
#include <conio.h>
void main()
{
  int arr[100], n, i, larg, pos;
  clrscr();
  printf("enter the no of array elements\n");
  scanf("%d", &n);
  for(i=0; i<n; i++)
  {
    scanf("%d\n", &arr[i]);
  }
  larg = arr[0];
  pos = i;
  for(i=1; i<n; i++)
  {
    if(larg < arr[i])
    {
      larg = arr[i];
      pos = i+1;
    }
  }
  printf("the largest no in entered
    array elements is %d\n", larg);
  printf("the largest is %d in %d
  position\n", larg, pos);
  getch();
}
```

o/p

enter the no of array elements

2

11

the largest no in entered array elements is 12 the largest element is 12 in 2 position.

: Selection Sort

```
#include<stdio.h>
#include<conio.h>
void main
{
int arr[100], n, i, j, temp;
clrscr();
printf("enter the no of array elements \n");
scanf("%d", &n);
for(i=0; i<n; i++)
{ scanf("%d", &arr[i]);
} for(i=0; i<n; i++);
{
for(j=i+1; j<n; j++)
{
if(arr[i] > arr[j])
{
temp = arr[j];
arr[j] = arr[i];
arr[i] = temp;
}
}
}
printf("the list of bubble sorted array
is \n");
for(i=0; i<n; i++)
{
printf("the list of bubble sorted
array is \n");
```

```
for (i=0; i<n; i++)
{
printf (" %d \t", arr[i]);
}
getch();
}
```

O/P

enter the no of array elements
5

34, 4, 5, 56, 67

the list of bubble sorted array is
4 5 34 56 67.

write a program to search the array number

```
#include <stdio.h>
#include <conio.h>
void main()
{
int arr[100], n, i, num, pos = -1;
clrscr();
printf ("enter the no of array elements \n");
scanf ("%d", &n);
for (i=0; i<n; i++);
{
scanf ("%d", &arr[i]);
}
printf ("enter the no to search in the array");
```

```
scanf("%d", &num);
for (i=0; i<n; i++)
  {
    if (num == arr[i])
    {
    pos = i;
    exit(0);
    }
  }
  if(pos >=0)
printf("the number %d is found in
        %d position of the array", num, pos+1);
else
printf("
```