Page No.:
Date :

Unit-1

Programming Language:

A Language is system or means of communications. Language can be classified into two groups in general sense.

1. Natural Language
2. Computer Language.

1. Natural Language:

It is a means of communication used to communicate in between human beings. The some examples of the natural languages are Nepali, English, German, French etc. Certain symbols & letters form the natural languages. These symbols & letters can be understood by human beings so means of communication is established some points.

The features of natural language are:

- It has large vocabulary
- Human being use knowledge for the communication
- Energy consumption is different in different stage

2. Computer Language:

It is a means of communication between computer and operator. This is also called programming language. There are many programming language available & each one has its own symbolic characters set and syntax but each language has an instruction set that permits those operations to be performed.

Types of computer Language

1. Low Level language
2. High level language

Page No.:
Date :

Low Level Language:

This is special kind of Language used in the computer in terms of coded format called digits (0 & 1). Low-level language is not written in our understandable format. The machine codes are used to make means of communication between computer & user. e.g. 1001100111011.

2. High Level Language:

This is some developed form of low-level language because low level language is coded format but high level language is our understandable format therefore. The written is systematic basic building of language syntax. This language can't perform final result without help of middle language called as interpreter or compiler. Some examples of high level languages are C, Basic, TurboC, Foxpro, Fortran etc.

Classifications of low level language

1. Machine Language
2. Assembly Language

Page No.:
Date :

Machine language:

Machine language is the fundamental language of computer. It is similar to low level language. Although computers can be programmed to understand many different computer languages this is the only one language understand by the computer without using a translation.

A machine language would have two parts. The first is command or operation. Every computer has an operating code or upcode for of its functions. Second part of the instruction is the operand that tells the computer where to find or store data or other instructions that are to be manipulated.

Opcode	Operand
(Operation Code)	(Address/location)

Page No.:
Date :

Compiler	Interpreter	Assembles
• It scans the entire program first and translate it into machine code.	It translates the program line by line.	It also converts the entire program into machine code.
• It converts the entire program to machine code; When all the syntax errors are removed, execution takes place.	Each time the program is executed, every line is checked for syntax errors and then converted to equivalent machine code.	Same as compiler
• Slow for debugging errors.	Faster for debugging errors.	Fastest for debugging the errors.
• Over all execution time is less.	Over all execution time is more.	Over all execution time is more less.
• It creates only one objecte program after execution of program	It does not create object program	
• Separate for program	Separate for program	Separate for program.

Page No.:
Date :

sserables

also converts
entire program
machine code.

ame as
mpiler

est for
gging the
s.

all execution
is more less.

ate for
am.

Page No.:
Date :

Loader:

It is responsible for loading, linking & relocation. The basic functions of the loader are as follows.

> To assign a load time storage area to a program - storage allocation.

> Making a program to execute properly from its load time storage area - program relocation.

> Keeping the program into assigned memory area - loading

> linking of programs with one another - linking.

Types of Loaders:

Absolute loader: It is a primitive type of loader that executes only the loading function. It loads the object program from translation time address & simply transfers control to it. It doesn't perform linking & program relocation.

Bootstrap loader: When the computer is turned on, absolute is executed. This loader is responsible for loading the operating system & transferring control to it. The bootstrap loader is present in the ROM area of main memory.

Relocatable loader: This loader is responsible for relocation and loading.

Linker (software, part of translator):

The linker established the linking - Relationship between software and hardware. Linker is a program that does the merging of individual program modules so that they could communicate with each other.

All the high level languages are using in-built functions like calculating square root sine, cos etc. The programmer doesn't code these functions but the standard execution. this higher level rd library of the high level language provides these functions.

Types of loader:

a) Linking Loader:

The linking loader performs all linking & relocation operations & loads & linked program directly into the main memory for execution. The functions can be pointed as:

1) Shifting the instruction to the brain of the computer.
2) Reloading - Managing the instructions.
3) Storing into main memory.

b) Linking Editor:

After storing the information into main memory. It is checked for correction. A linkage editor produces a linked version of the program called the Load Module or an Executable Image. This Load module is written onto file or library for execution at a large storage.

c) Dynamic Linker:

The total duration of the program execution is the dynamic period. This scheme postpones the linking function until execution time. Any subroutine is loaded & linked dynamic linking. Dynamics loading or loads on call.

(*) Syntax (Grammar)	Semantics (logic)
1. It is a common format.	It is a Pseudo format.
2. It is the rule of the semantics	It is the examples of syntax.
3. Computer can understand	Computer can not understand
4. It is the error of spell mistake & grammar mistakes	It is the mistake of doing error inputing.
5. Single programmer can easily detect error.	Need of group of programmers
6. It is simple to remove	It is difficult to remove.
7. It take less time to detect.	It take long time to detect,
8. Compiler can show error program can't run.	Compiler can't show errors program can Run.

Page No.:
Date :

Algorithm :

A set of well defined steps for performing a task or solving a problem.

A good algorithm has the following characteristics:

- Precision - the steps are precisely stated (defined.)
- Uniquenessss - result of each step are uniquely defined and only depend on the input and the result of the preceding steps.

Finiteness - the algorithm stops after a finite number of instructions are executed.

Input - the algorithm receives input.
Output - the algorithm produces output
Generality - the algorithm applies to a set of inputs.

Flowchart :-

A flowchart is a pictorial representation of the steps involved in the processing of the data. It indicates the logic sequence in which these steps are to be performed.

Flowchart Symbols:

- ○ → Start / End
- ▭ → Input/Output
- ▭ → Processing
- ◇ → Decision
- ▭ → Hardcopy output display
- ⬡ → Softcopy display

Advantages:

* Better communication
* Effective analysis
* Proper program documentation
* Effective coding
* Easy to study program.

Pseudo Code :

> Fake code
> Just like real program but could not run.
> Written in any language.
> Gives clear idea about semantics (logic)
> Imitation of actual code.
> Pseudo means imitation or false
> Code means instruction
> Pseudo code = false instruction.

Page No.:
Date :

Features of Good program.

1. The program should be user-friendly.
2. The program should be portable and must be able to run on different platforms.
3. The program should produce accurate results quickly and should use the least amount of memory.
4. The program should use meaningful names for constants and variables.
5. The program should produce meaningful output for any set of input data.
6. Every program should have a users guide.

* Simplicity
* Efficiency
* Compactness
* Security
* Clearity
* Integrity
* Modularity
* Abstraction

KISS: Keep It Short & Sweet

Page No.:
Date :

Unit 2 : Introduction to C

C is a general purpose programming of programming language by 1960 ; almost which used for specific purpose only : FORTRANC (formula Translation) developed by IBM in 1960 & used for engineering and scientists.

- COBOL (Common Business-oriented long) developed in 1956 & was used only for commercial area.

- Computer scientists started to think for all possible areas.

- ALGOL (All Gorithm language)
→ CPL (Combine Programming Language) 1963 Cambridge University developed by Mathematical laboratory
→
→ BCPL (Basic Combined) 1967, Martin Richard at CU.
→ B → 1970 → Ken thompson at AT & Tbell lab
→ C → 1972 → Dennis Ritche at AT & Tbell lab

Middle level language (Directly impact with hardware) Structured Programming language because it is based on. • Sequence • Selection • Iteration

Application of C

- System Programming
- Network Software Programming
- Graphics Programming
- Embedded System

Page No.:
Date :

Basic structure of C programme

Comments/Documentation
Preprocessor directives/Header file
Global Variable
main() function
{
local variable
statements
}
function 1 ()
{
statements
}
function 2 ()
{
statement
}

#include <stdio.h> Command Preprocessor directives
<conio.h> header file

Character set:

The Character set that can be used to form words, numbers and expression depend upon the computer on which the program is run.

The characters has following categories.

1. Letters or Alphabet
   A, B, C, - - - - Z Uppercase
   a, b, c. - - - z lowercase

2. Digits

All decimal digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Special Characters

, comma	. Period	; semicolon
: Colon	? question mark	' apostrophe
" quotation mark	! Exclamation mark	| vertical bar
/ slash	\ backslash	~ tilde
_ Underscore	$ dollar sign	% percent sign
& ampersand	^ caret	* asterisk
- minus sign	+ plus sign	< Opening angle bracket
> closing angle bracket (greater than sign)		
(left parenthesis	) right parenthesis	
[left bracket	] Right bracket	
{ left brace	} Right brace	# number sign

Tokens

In a passage of text, individual words and punctuation marks are called token. The smallest individual units are known as C tokens.

C Tokens

Keywords	Constant	String	Operators
float	-15.5	"Ans"	+ -
while	100	"Year"	* /

Identifiers	Special symbols
main	[ ]
amount	{ }

Page No.:
Date :

Keywords

All keywords have fixed meaning and these meaning cannot be changed. Keywords serve as basic building blocks for program statement.

C keywords are listed below.

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

Rules of Identifiers

1. Name should consist of alphabet, digit and underscore.
2. First character should be alphabet or underscore.
3. name should not be keyboards
4. It is case sensitive
5. No commas and blanks are allowed
6. The max length of the variable should be 8 character.

Datatype + Identifier = Variable

Page No.:
Date :

Datatype:

There are various types of data for eg 10 and 10.6 are data of different types. The data 10 is integer number and 10.6 is a fractional number. Datatypes are classified into two categories

1. Primary datatype
2. Secondary datatype

Primary datatype:-

(a) Integer (b) float (c) Double-precision floating
(d) character type (e) void type

Secondary datatype:

(a) Array (b) Pointer (c) Structure (d) Union

Datatype	Category	Range	Conversion character	bytes
Integer (int)	Unsigned int	0 to 65,536	%-u	2
	Signed int	-32768 to 32768	%d	2
	Long Unsigned int	0 to 4294967295	%-Lu	4
	Long signed int	-2,147,483,648 to 2147483,648	%-Ld	4
float	float	3.14×10^{-38} to 3.4×10^{38}	%-f	4
	double	1.7×10^{-308} to 1.7×10^{308}	%-Lf	8
	long double	3.4×10^{-4932} to 3.4×10^{4932}		10
char	Signed char	-128 to 127	%-c	1
	Unsigned char	0 to 255	%-c	1

Escape sequence	Meaning	ASCII Value	Result
\a	beep	007	alert
\b	backspace	008	Move cursor to previous position of line.
\r	Carring return	013	Move to the begining of line.
\n	new line	010	new line
\0	null	000	null
\t	horizontal tab	089	horizontal tab
\v	Vertical tab	011	Vertical tab
\f	form feed	012	Move to initial position of next page back slash sign
\\	back slash	092	Back slash sign
\"	Qrotation mark	034	Print quotation
\'	Apostrophy	039	Print
\?	Question mark	063	Question mark

Delimiters: Used for static meaning

Delimiter	Name	Meaning
:	Colon	Used for label
;	Semicolon	End of statement
( )	Paren thesis	Used in expression
[ ]	Square bracket	Used for array
{ }	Curly Bracket	Used for block of statement
#	hash	Preprocessor directive
,	Comma	Variable delimeters

Page No.:
Date :

Comments:

Single line comments //---

Multiple line comments /* ---*/

Conversion specification (format specifier)

1. %c single character
2. %d decimal integer
3. %f floating point number
4. %e floating point number
5. %g floating point number
6. %lf long range of floating point (double datatype)
7. %h short integer
8. %o Octal integer
9. %x Decimal integer
10. %i Decimal, Octal or Hexadecimal
11. %s string without space
12. %u Unsigned integer
13. %ld long integer (signed)
14. %lu long integer (unsigned
15. %lf long double
16. %[^n] or %[...] string with space

1. Single character input & Output

(i) getch() → Read a single character from standard input device

2. Putchar() → It gives output a character at a time to the standard output device.

Page No.:
Date :

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char ch;
printf("Enter a character\n");
ch = getchar();
printf("The entered character is:\n");
putchar(ch);
}
```

Mixed data input and output

① Printf() → To give output

syntax:

```
Printf(control string, argument);
```

② Scanf() → To read data

syntax:

```
scanf(control string, argument);
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,large;
printf("Enter 2 number");
scanf("%d %d", &a, &b);
large=(a>b)? a:b);
printf("large number is %d", large);
getch();
}
```

Page No.:
Date :

gets(); and puts();

```
# include <stdio.h>
# include <conio.h>
void main()
{
char name [50], address [50];
printf ("Enter name");
gets (name);
print f("Enter address");
gets (address);
printf("\n name and address are \n");
puts (name);
puts (address);
getch();
}
```

getchar()	getche()	getch()
ch=getchar()	ch=getche()	ch=getch()
① need to press enter to assign value to ch	no need to press enter to assign value to ch	Same as getche()
② Display character before assignment	Display character before assignment	character will not be displayed

Page No.:
Date :

Statement

1. Expression statement
2. Compound statement
3. Selection statement (if, if--else, switch)
4. Iteration (Looping) statements (for, while, do-while)
5. Jump statement (goto, continue, break, return)
6. Label statement (case, default, label)

Operators :

1. Unary operator
   a. Unary minus (-)

```
{
int a,b;
a=5
b=-a;
}
```

   b. Increment (++)

Prefix

```
a=2
b=++a;
```

```
a=2
b=3
```

Post fix

```
b=a++;
a=2
b=a++;
```

```
a=2
b=2
```

   c. decrement (--)
   d. size of () operator
   e. cast operation

Prefix

```
int a,b
a=5;
b=--a;
b=4
a=4
```

postfix

```
int a,b
a=5
b=a--;
b=5
a=4
```

size of () operator

```
int i,j
j= size of (i)
j=2
```

Output

```
printf("%d",size of(i));
```

Page No.:
Date :

Cast operator

```
int a=10;
float b;
b = (float) a;
```

Binary operator

Arithmetic	Relational	Equality	logical	Assignment
+, -, *, /, %	<, >, <=, >=	==, !=	&& (AND)	=
			|| (OR)	+=, -=,
			||	*=, /=
			! (Not)	%=

Ternary operator (conditional)

Syntax ? :

exp1? exp2: exp3 a=5, b=6

find large num between a&b

```
c = (a>b? a:b);
```

WAP to read three numbers and find large numbers using ternari operator.

```
#include <stdio.h>
#include <conio.h>
void main()
{ int a,b,c,large;
printf("Enter the three numbers");
scanf("%d%d%d", &a,&b,&c);
large= (a>b && a>c?? a:(c>a && b>c)?b:c);
printf("%d", large);
getch();
}
```

Page No.:
Date :

Bitwise operator

(a) Ones complement operation

- Used for low level programming
- Used to support the manipulation data at bit level

```
int a=10;
printf("%d", ~a);
```

Output 5

a = 10 = 1010

Ones complement of a = 0101

(b) Bitwise logical operation

Bitwise AND(&)

```
int a=10, b=11;
printf("%d", a&b);
```

Output is 10

a = 1010 (10)
b = 1011 (11)
1010 → 10

Bitwise OR(|)

```
int a=10, b=11;
printf("%d", a|b);
```

Output 11

1010
1011
1011

Bitwise XOR(^)

```
int a=10, b=11;
printf("%d", a^b);
```

Output 1

Page No.:
Date :

Unit 3: Control structure. (Decision making statement)

· With the help of decision making statement, we can logicaly test the condition and according to that the instruction to be executed.

① If statement
* If else statement
* Nested if else statement
* Switch statement.
* else if ladder statement
* Condition operator (?:)

② Looping statement
* While loop
* Do-While loop
* for loop
* Nested loop

③ Jumping statement
* goto statement
* break statement
* Continue statement

If statement:

It is the decision making statement where the expression inside the if block will be executed if and only if the condition is true.

Syntax:
```
If (Condition)
{
    // statement or expression
}
```

Page No.:
Date :

WAP to read two integer number and find largest number between them do this using if statement.

```
#include <stdio.h>
#include<conio.h>
void main()
{
int a,b.;
printf("Enter any two numbers");
scanf("%d %d", &a, &b);
if (a<b)
{
printf("%d is greater", b);
}
if (b<a)
{
print ("%d is greater", a);
}
getch();
}
```

If else statement:

It is the decision making statement where the expression inside the if block will be executed if the condition is true else the statement of else block will be executed.

Syntax:

```
If (condition)
{ //Expression or statement
}
else
{ //Expression or statement
}
```

Page No.:
Date :

WAP to read two integer number and find smallest number using if else statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a, b;
printf ("Enter any two integer");
scanf ("%d %d", &a, &b);
if ( a<b)
{
printf ("%d is smallest number", a);
}
else
{
printf ("%d is smallest number", b);
}
getch();
}
```

Nested if else statement (if-else ladder statement)

In this case we can put the if-else condition inside other if else statement. inner if else statement will be executed if and only if ~~output~~ outer condition is satisfied.

Page No.:
Date :

Syntax

```
if (condition)
{
    if(condition)
    {
    }
    else
    {
    }
}
else
{
    if(condition)
    {
    }
    else
    {
    }
}
```

(8) WAP read three numbers and print largest number using Nested if else statement.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b,c;
printf("Enter any three number \n");
scanf("%d %d %d", &a, &b, &c);
if(a<b)
{
```

```
        if(c<b)
        {
        printf("%d is largest", b);
        }
        else
        {
        printf("%d is largest", c);
        }
    }
    else
    {
        if(a<c)
        {
        printf("%d is largest", c);
        }
        else
        {
        printf("%d is largest", a);
        }
    }
    getch();
}
```

Switch statement:

This is a multi directional statement. The statement block is executed which match with the given case. If the ~~b~~ value does not match with any if the cases the control is transfered to default level if it is exist.

Page No.:
Date :

Syntax:

```
switch(exp)
  {
   case value 1:
        //statement;
        break;
   case value 2:
        //statement;
        break;
        :
        :
   Default:
        //Statement;
```

* WAP read a single value & check wheather it is digit or not using switch statement.

```
#include<stdio.h>
#include<conio.h>
void main()
 {
  int ch;
 printf("Enter value");
  scanf("%d", &ch);
  switch(ch)
   case 1:
     printf("1 has entered");
   break;
   case 2:
   printf("2 has entered");
   break;
```

Page No.:
Date :

```
case 3:
 printf("3 has entered");
 break;
case 4:
printf("4 has entered");
break;
case 5:
printf("5 has entered");
 break;
case 6:
printf("6 has entered");
 break;
case 7:
printf("7 has entered");
break;
case 8:
printf("8 has entered");
break;
case 9:
 printf("9 has entered");
 break;
case 10:
printf("10 has entered");
 default:
printf("digit has not entered");
 getch();
}
```

Page No.:
Date :

```
#include<stdio.h>
#include<conio.h>
void main()
{
int ch;
printf("Enter value");
scanf("%d", &ch);
switch (ch);
{
case 1:
case 2:
case 3:
case 4:
case 5:
case 6:
case 7:
case 8:
case 9:
case 0:
printf ("number has entered");
break;
default:
printf("number has not entered");
}
```

Page No.:
Date :

Looping statement:

Looping (iteration) statement allow a set of instruction to be executed repeatedly until a certain condition is reached. condition may be predefine or open ended as in while and do while loop.

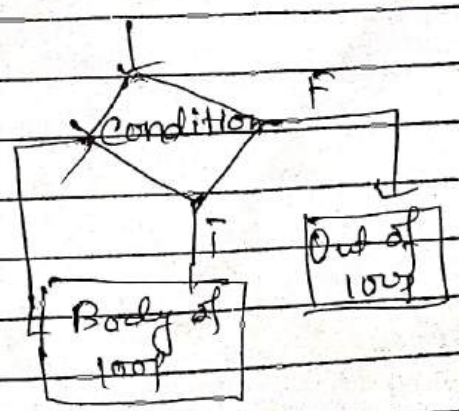
While loop:

This is on which executed set of statements repeatedly till condition is true it is also called dynamic loop. because it asked the condition from the user. Based on the user choise to makes ~~condit~~ continue or discontinue. it is also called antilevel loop because to entered into the loop condition has should be satisfied

- It is called entry level loop
- It executes set of statements repeadly till condition is true
- called dynamic loop because it ask the condition from the user based on user choise it makes continue or discontinue.

Syntax:

```
While (condition)
{
statement;
!
!
}
```



flowchart

Page No.:
Date :

WAP to print your name as much as when you want.

```
#include<stdio.h>
#include<conio.h>
void main();
{
char name[30];
char decision;
decision='y';
name="Ram";
While (decision=='y')
{
printf("%s", name);
getch();
printf("do you want to print name
again\n");
scanf("%c", &decision);
}
getch();
}
```

Page No.:
Date :

WAP to print your name numbers of times that you want name must taken from user.

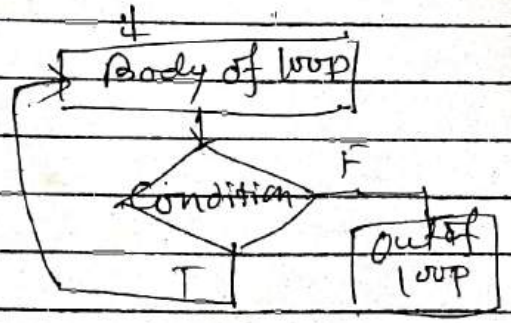
```
#include<stdio.h>
#include<conio.h>
void main()
{
char name[50];
int nooftime, start, start=1;
printf("Enter how many time do you want to
print your name\n");
scanf("%d", &nooftime);
while (start ≤ nooftime);
{
printf("name is %s\n", name);
start ++;
}
getch();
}
```

Do while loop:-

- exit level loop
- Similar to while loop but the testing condition is at the bottom of the loop.
- In this case condition is initially false the statement is inside do while will be executed at least once.

Syntax:

```
do
{
statement;
} while (condition);
```



Page No.:
Date :

WAP print your name number of times that you want using do while loop name must be taken from user.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char name[50];
int noftime, start, start=1;
printf("Enter the number. how many
 times do you want to print\n");
scanf("%d", &noftime);
printf("\n Enter your name\n");
scanf("%s", name);
do
{
printf("%s \n", name);
 start++;
}
while (start <noftime);
getch();
}
```

Page No.:
Date :

For loop

- This is used to execute set of statement repeatedly till the condition is satisfied.

- It has three expression and semicolons are used for seperating these expression.

Syntax:

```
for (expression1; expression2; expression3)
  {
    statements;
    ---
    ---
  }
```

Expression1 is initialization expression
Expression 2 is test (condition) expression
Expression 3 is update expression

WAP to print number from 1 to 10 using for loop.

```
#include <stdio.h>
#include <conio.h>
void main()
{
 int a;
 for (a=1; a<=10; a++)
  {
   printf("%d \n", a);
  }
 getch();
}
```

Page No.:
Date :

Nested for loop

Syntax:

```
for (exp1; exp2; exp3)
{
    for (exp1; exp2; exp3)
    {
    }
}
```

Jumping statement:

- continue
- break
- goto

Break statement:

- Used inside loop & switch statement.
- Used to terminate the loop or exit from switch.

Syntax: break;

* WAP to print number from 1 to 5 from the integer array having number from 1 to 10.

```
#include <stdio.h>
#include<conio.h>
void main()
{
int a[10]= {1,2,3,4,5,6,7,8,9,10};
int i;
for (i=0; i<=10; i++)
  {
  if (a[i]>5)
    { break;
    } printf ("%d \n", a[i]);
  }
getch();
}
```

Page No.:
Date :

Continue:

It is used when we want to goto the next iteration of the loop after skipping some statement of the loop.

Syntax:

```
continue;
```

When continue statement is encountered all the remaining statement after continue statement in the current iteration are not executed and the loop continues with the next iteration.

WAP to print number from 1 to 10 except 5 from the int array having no. from 1 to 10.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a[10]= {1,2,3,4,5,6,7,8,9,10}
int i;
for (i=0; i<10; i++)
  {
   if (a[i] ==5)
     {
      continue;
     }
   printf("%d \n", a[i]);
  }
getch();
}
```

Page No.:
Date :

WAP to print numbers from 1 to 5 except 5 & 6 from the int array having no. from 1 to 10.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a[10]={1,2,3,4,5,6,7,8,9,10}
int i;
for (i=0; i<10; i++)
 {
if a[i]==5 || a[i]==6)
  {
  continue;
  }
printf("%d\n", a[i]);
 }
getch();
}
```

goto statement:-

Transfer the flow to another part of the program.

Syntax: goto label name;
- - -
label name:
- - -
- - -

except
ing no.

}

part of

WAP read a number and find number is even or odd using goto statement.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a;
printf(" Enter a number \n");
scanf("%d", &a);
if (a%2==0)
{
go to even;
}
else
{
go to odd;
}

even:printf("Number is even");
goto xyz;

Odd:
printf("Number is odd");
xyz:
getch();
}
```

Page No.:
Date :

Unit4 : Function

Library function:-

- Supplied with C compiler
- User to perform predefine task.
- Source code of library function is not given to user.
- library function are pre compiled and only object code is available to user.

function defination:-

Predefined, precompiled and present in the library.

function declaration:-

In header file (file with .h extension)

function call :

→ by the programmer

Function	data type return	Description	library file
abs(i)	int	Return absolute value	<stdio.h>
pow(d1,d2)	double	Return $d_1^{d_2}$	<math.h>
sqrt(d)	double	Return Square root of d	<math.h>
toascii(a)	int	Return ASCII value of A	<ctype.h>
tolower(c)	char	Convert to lower case	<stdlib.h>
toupper(c)	char	Convert to uppercase	<stdlib.h>
isalpha(c)	int	Return nonzero value if argument is alphabet	<ctype.h>

Page No.:
Date :

Function	datatype return	Description	library file
function	return type	Return non-zero value if argument is alphabet	<ctypy.h>
isdigit()	int	Return non-zero value if argument is digit	<type.h>
isupper()	int	Return non-zero value if argument is in uppercase	<type.h>
islower	int	Return non-zero value if argument is in lowercase	<type.h>
rand	int	Return random positive integer	<stdlib.h>
strcmp(S_1,S_2)	int	return 0 if $S_1 = S_2$ return -1 if $S_1 < S_2$ return positive 1 if $S_1 > S_2$	<string.h>
strcpy(S_1,S_2)	char	Copy string S_2 to S_1 value if S_1 will be overide	<string.h>
strlen(S_1)	int	return number of character in string	<string.h>
strcat(S_1,S_2)	char	concat string S_2 to S_1	<string.h>
strrev(S_1)	char	Reverse string S_1	<string.h>
strupr(S_1)	char	Convert S_1 to uppercase	<string.h>
strlwr(S_1)	char	Convert S_1 to lower case	<string.h>

Page No.:
Date :

A function is a self contain block of statement that perform some specific and well defined task.

A C program make contain one or more function. If a program have only one function then that must be main function.

Types:-
Library function
User define function

User define function:

User can create their own function for performing any specific task of a program. this types of functions are called user defined function.

Steps required for function:-
1. Declerating (Prototyping)
2. Calling
3. Definition (Body of function)

statement
defined

re function:
then that

or perform
, this
fined

Page No.:
Date :

WAP read two number and find sum of them using function with argument and no return type.

```
#include<stdio.h>
#include<conio.h>
void main()
{
void sum(int,int); //Decleration
int A,B;
printf("Enter two number \n");
scanf("%d %d", &A, &B);
sum(A,B); //Calling
getch();
}

void sum(int A, int B)
{
int sum;
sum = A+B;
printf("Sum is %d", sum);
}
```

Defination

Types of user define function

1. function with no argument and not return type (void type)
2. function with argument and return type (void type)
3. function with argument and no return type
4. function with no argument and return type.

Page No.:
Date :

* function with argument & no return type

```
#include<stdio.h>
#include<conio.h>
void main()
{
void biggest (int, int);
int a,b;
printf("Enter any two number:\n\n");
scanf("%d %d", &a,&b);
biggest (a,b);
getch();
}

void biggest (int a ,int b)
{
if(a<b)
{
printf("%d is biggest number", b);
}
else
printf("%d is biggest number", a);
}
```

Page No.:
Date :

Example of function without argument without return type.

```
#include <stdio.h>
#include <conio.h>
void main()
{
void sum();
sum();
getch();
}
void sum()
{
int a, b, sum;
printf("Enter two values");
scanf("%d%d", &a, &b);
sum = a+b;
printf("sum is: %d", sum);
}
```

Example of function with argument with return type.

```
#include <stdio.h>
#include <conio.h>
void main()
{ int sum (int, int);
int a, b, result;
printf ("Enter two values\n");
scanf ("%d %d", &a, &b);
result = sum (a,b);
printf ("Sum of values%d \n", result);
getch();
}
int sum (int x, int y)
{ int value;
value = x+y;
return value; }
```

Page No.:
Date :

Example of function without argument and return type.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int sum();
int result;
result = sum();
printf("Sum of values %d \n", result);
getch();
}

int sum()
{
int a,b, result;
printf("Enter values \n");
scanf("%d %d", &a, &b);
result = a+b;
return result;
}
```

Page No.:
Date :

Parameter passing mechanism of function

Call by value	Call by reference
1. In call by value only the values of arguments are sent to the function.	In call by reference address of arguments are sent to the function
2. Why change made to the argument in function (formal argument) do not change actual argument.	Why change made to the argument in function (formal argument) will be reflected on a actual arguments.
3. It doesn't use pointer.	It uses pointer.

WAP read two number and swap them using call by value and call by reference using different function

```
#include<stdio.h>
# include<conio.h>
        void main()
  {
     int a,b; // actual argument
     void swap1 (int, int);
     void swap2( int *, int *);
     printf ("Enter two number \n");
     scanf("%d %d", &a, &b);
  print f("Values of a is %d & values of b is
        %d \n", a, b);
```

Page No.:
Date :

```
swap1 (a,b); // Passing value
printf (" Values of a is %d \t Values of b is
        %d \n", a,b); // passing address

swap2 (&a,&b); // passing address
printf("Values of a is %d \t Values of b is %d",
        a,b);
  getch();
}

void swap1 (int x, int y)
  {
   int c;
   c=x;
   x=y;
   y=c;
printf(" Value after swap through call by value
        is %d \t %d", x,y);
  }

void swap2( int *x, int *y)
  {
   int c;
   c= *x
   *x = y;
   *y = c;
printf("\n Value after swap through call
        by reference is %d \t %d", *x, *y);

  }
```

Page No.:
Date :

WAP to print fibonancy series of given number using function.

```
# include<stdio.h>
# include<conio.h>
void main()
{
 int  n;
 void fibo(int n);
 clrscr();
 printf("Enter a number:");
 scanf("%d",&n);
 printf("fibonancy Series \n0\n1\n");
 fibo (n);
 getch();
}

void fibo(int n)
{
 int  a=0, b=1, c, i;
 for(i=1; i<=n-2; i++)
   {
    c=a+b
    printf("%d \n", c);
    a=b;
    b=c;
   }
}
```

Page No.:
Date :

Recursion function:

Recursion is a process by which a function call itself repeatedly until some specified condition has been satisfied.

This is used for repeative computation in which each action is stated in terms of a previous result.

The condition for recursion are
1. function must be recursive in nature
2. There must be stoping condition

WAP to print an using recursion

```
#include<stdio.h>
#include<conio.h>
void main()
{
long int raise(int, int);
int a,n; long int result;
printf("Enter any two number");
scanf("%d %d", &a, &n);
result = raise(a,b);
printf("Values of an is %ld", result);
getch();
}
long int raise(int a, int n)
{ if (n==0)
    { return 1;
    }
  else { return (a* raise(a,n-1))
    }
}
```

Page No.:
Date :

WAP to print factorial of given number using recursion

```
#include <stdio.h>
#include <conio.h>
void main()
{
long ~~int~~ facto (int);
int n;
long result;
printf ("Enter a number");
scanf ("%d", &n);
result = facto (n);
printf ("factorial is %d", result);
getch();
}

long facto (int n)

{
if (n==0)
{
return 1;
}
else
{
return n*facto(n-1);
}
}
```

Page No.:
Date :

WAP read a number and find sum of individual digit using recursion.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int sum(int a);
int n,result;
printf("Enter the number:");
scanf("%d", &n);
result = sum(n);
printf("Sum of digits is %d", result);
getch();
}

int sum(int n)
{
if (n != 0)
{
return 0;
}
else
{
return (n % 10 + sum(num/10));
}
}
```

Page No.:
Date :

WAP to print reverse of number using recursion

```
#include <stdio.h>
# include<conio.h>
long reverse (long);
int main()
{
long n, r;
printf("Enter a number");
scanf("%d", &n);
r= reverse (n);
printf("Reverse of entered number is %d", r);
}

long reverse (long n)
{
static long r=0;
if (n==0)
return 0;
r= r*10;
r=r+n%10;
reverse (n/10);
return r;
}
```

Unit 5: Array, Pointer and String

- An array is a collection of homogeneous value that are represented by common name.

- Specific element in an array is accessed by an index number.

- It consist of continuous memory location

- lowest address correspond to ~~last~~ first element

- highest address correspond to last element

- Array may have from one to several dimension

- Common array are one and two dimension

- Index always start with 0.

One dimensional

int a[5]

index	0	1	2
value	10	12	13

Two dimensional (int a[5][3]

i	0	1	2
0	[0][0]	[0][1]	[0][2]
1	[1][0]	–	–
2	[2][0]	–	–
3	[3][0]	–	–
4	[4][0]	–	–

Page No.:
Date :

int a[5]

datatype → int

Array name → a

Size → 5

Eg 1: char name[20];
~~nam~~ char P[] = {"Santosh"}

Eg 2: char name[10];
name = {"Santosh"}

0	1	2	3	4	5	6	7	8	9
s	a	n	t	o	s	h	\0		

Eg 3:
char name[2][10] = {{"Khushi"}, {"Ram"}, {~~Sita~~}}

	0	1	2	3	4	5	6	7	8	9
0	K	h	u	s	h	i	\0			
1	R	a	m	\0						

Page No.:
Date :

WAP to read n numbers of values in array and find the sum of them.

```
#include<stdio.h>
#includ<conio.h>
void main()
{
int n, A[100], i, sum=0;
printf(" Enter how many number you want to entered:");
scanf("%d", &n);
printf("\n Enter the value in array:");
for( i=0; i<n; i++)
{
scanf("%d", &A[i]);
}
{for (i=0; i<n; i++)
sum=sum+ A[i];
}
printf ("sum = %d", sum);
}
```

Page No.:
Date :

WAP to read n numbers of values in array and find only even numbers from array.

```
#include <stdio.h>
# include<conio.h>
void main()
{
int n, A[100], i;
printf("Enter how many number :");
scanf("%d", &n);
print f("\n Enter the value in array");
for(i=0; i<n; i++)
{
scanf("%d", &A[i]);
}
printf("The even numbers in given array are");
{
for (i=0; i<n; i++)
{
if (A[i] %2 == 0)
printf("%d\t", A[i]);
}
}
getch();
}
```

Page No.:
Date :

1 WAP to read n numbers of values in array and find only odd numbers from array.

```
#include <stdio.h>
# include <conio.h>
void main()
    int n, A[100], i;
printf("Enter how many number:");
scanf("%d", &n);
printf("Enter the values in array");
for(i=0; i<n; i++)
  {
  scanf("%d", &A[i]);
  }
printf("The odd numbers in given array
        are:");
  {
  for (i=0; i<n; i++)
    {
    if(A[i]%2 !=0)
    printf("%d\t", A[i]);
    }
  }
  getch();
}
```

Page No.:
Date :

WAP to find the smallest no from array.

```
# include <stdio.h>
# include <conio.h>
void main()
{
int j, i, n, a[50], small;
printf(" Enter size of array less then 50");
Scanf("%d", &n);
printf("Enter value in array \n");
for(i=0; i<n; i++)
  {
  scanf("%d", &a[i]);
  }
  small = a[0];
  for(j=1; j<n; j++)
    {
      if(a[j] < small);
        {
        small = a[j];
        }
    }
  printf("Smallest number is %d", small);
  getch();
}
```

Page No.:
Date :

WAP to read n numbers from user into array and find largest number among them.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[100], n, large;
printf("Enter how many number\n");
scanf("%d", &n);
printf("Enter value in array\n");
for(i=0; i<n; i++)
    {
    scanf("%d", &a[i]);
    }
large= a[0];
for (j=1; j<n; j++)
    {
    if(large< a[j])
        {
        large= a[j];
        }
    }
printf("large num is %d", large);
getch();
}
```

Page No.:
Date :

WAP to search weather given no. is in array or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[100], n, extranum, i, j;
printf("How many numbers :\n");
scanf("%d", &n);
printf("Enter value in array \n");
for(i=0; i<n; i++)
{
scanf("%d", &a[i]);
}
printf("Read extranum\n");
scanf("%d", &extranum);
for(j=0; j<n; j++)
{
if (extranum == a[j])
{
printf("number found");
exit;
}
getch
if(j==n)
{
printf("num not found");
}
getch();
}
```

Page No.:
Date :

WAP to sort number in array using selection sort (ascending)

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,i,temp,j, A[100];
printf(" Enter how many number upto 100);
scanf("%d", &n);
for (i=0; i<n; i++)
{
scanf("%d", &A[i]);
}
for (i=0; i<n-1; i++)
{
for (j=j+1; j<n; j++)
{
if (A[i]> A[j])
{
temp = A[i];
A[i] = A[j];
A[j] = temp;
}
}
}
printf(" Values in sorted order\n");
for (i=0; i<n; i++)
{
printf("%d\n", A[i]);
}
getch();
}
```

Page No.:
Date :

Descending order

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,i,temp,j,A[100];
printf(" Enter how many number upto 100");
scanf("%d",&n);
printf(" Enter value in array");
for(i=0; i<n; i++)
    {
    scanf("%d", &A[i]);
    }
for(i=0; i<n-1; i++)
    {
    for(j=i+1; j<n; j++)
    {
    if(A[i] > A[j])
    {
    temp = A[i];
    A[i] = A[j];
    A[j] = temp;
    }
    }
    }
printf("Values in descending order");
for (i=n-1; i>-1; i--)
    {
    printf("%d \n", A[i]);
    }
getch();
}
```

WAP to read mxn matrix and display transpose of that.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int A[100][100], m,n,i,j;
printf("Enter size of m & n of mxn matrix:\n");
scanf("%d %d", &m, &n);
printf("Enter value in array");
for (i=0; i<m; i++)
  { for(j=0; j<n; j++)
    { scanf("%d", &A[i][j]);
    }
  }
printf("Given matrix is:\n");
for (i=0; i<m; i++)
 { for (j=0; j<n; j++)
   { printf("%d\t", A[i][j]);
   } printf ("\n");
 }
printf("\n Transpose of given matrix is\n");
for (i=0; i<n; i++)
 {
  for(j=0; j<m; j++)
   { printf("%d\t", A[j][i]);
   }
  printf("\n");
 }
 getch();
}
```

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$= \begin{bmatrix} a\times e + b\times g & b\times f + b\times h \\ c\times e + d\times g & c\times f + d\times h \end{bmatrix}$$

Page No.:
Date :

WAP to read two matrix of size m×n and product of them and put in third matrix of same size.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int A[100][100],m,n,i,j,B[100][100],C[100][100]K;
printf("Enter the size of matrix");
scanf("%d %d",&m,&n);
printf("Enter the elements of first matrix:\n");
for (i=0; i<m; i++)
{
for(j=0; i<n; j++)
{
scanf("%d", &A[i][j]);
}
}
printf("Enter elements of second matrix:\n");
for (i=0; i<m; i++)
{
for(j=0; i<n; j++)
{
scanf("%d", &B[i][j]);
}
}
printf("The product is:\n");
```

Page No.:
Date :

```
printf("The product of two matrix is :\n");
    for(i=0; i<m; i++)
        for (j=0; j<n; j++)
        {
        c[i][j]=0;
        for(k=0; k<m; k++)
        c[i][j]=c[i][j]+m[i][k]*n[k][j];
        }
        for(i=0; i<m; i++)
        printf("\n");
        {
        for (j=0; j<n; j++)
        printf("%d", c[i][j]);
        }
        getch();
    }
```

Page No.:
Date :

```
#include <stdio.h>
#include<conio.h>
void main()
{
int m,n,p,q,i,j,k,a=0;
int A[100][100], B[100][100], C[100][100];
printf("Enter size of matrix A\n");
scanf("%d %d", &m, &n);
printf("Enter size of matrix B\n");
scanf("%d %d", &p, &q);
~~print~~ printf("Enter value in A matrix\n");
for (i=0; i<m; i++)
  {
    for (j=0; j<n; j++)
      {
        scanf("%d", &A[i][j]);
      }
  printf("Enter value in B matrix\n");
  for(i=0; i<p; i++)
    {
      for(j=0; j<q; j++)
        {
          scanf("%d", &P[i][j]);
        }
    }

printf("Product of matrix is\n");
  if(n==p)
    {
      i=0
      while (i<m)
```

Page No.:
Date :

```
{
for(j=0; j<q; j++)
  {
  c[i][j] =0;
  for (k=0; k=p; k++)
  {
  c[i][j] = c[i][j] +(A[i][k] * B[k][j]);
  }
  }
  i=i+1;
  }
}
else
printf (" Matrix product is not possible\n");
a=1;
}
if (a==0)
{
printf (" Product of A*B matrix is\n");
  for (i=0; i<m; i++)
  {
  for(j=0; j<q; j++)
  {
  print f(" %d \t", c[i][j]);
  }
  print f("\n");
  }
}
getch();
}
```

Page No.:
Date :

WAP read n number of value in 2D array and print value in that array. Implement these to show the concept of passing array to the function.

```
#include <stdio.h>
#include <conio.h>
void main()
int A[100][100], n, i;
void disp(int [], int);
printf ("Enter how many number");
scanf ("%d", &n);
printf ("Enter value in array\n");
for (i=0; i<n; i++)
    {
    scanf ("%d", &A[i]);
    }
disp (A, n);
getch();
}
void disp (int x[], int n)
int j;
{
printf("\n Values in array are\n");
for (j=0; j<n; j++)
    {
    printf ("%d \n", x[j]);
    }
}
```

Page No.:
Date :

WAP read values in 2D array and print all the values in array. Implement this using the passing the array to the function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int A[50][60], raw, col, i, j;
void disp(int [][60], int, int);
printf("Enter size of 2D array");
scanf("%d%d", &row, &col);
print f("Enter values in array");
for(i=0; i<row; i++)
  {
   for (j=0; j<col; j++)
    {
     scanf("%d", &A[i][j]);
    }
  }
  disp(A, raw, col);
  getch();
}
  void disp(int x[][60], int row, int col)
  { int i, j;
  print f("Values in array are \n");
  for(i=0; i<row; i++)
    { for (j=0; j<col; j++)
      { print f("%d", x[i][j]);
      }
    }
}
```

Page No.:
Date :

String

Array of character terminator by '\0' is called string.

Data type used for string is char * for single character %c is used.

~~%c is used~~ for single string (whole word) %s is used.

string constant:-
A string constant is a sequence of characters enclosed in double quate.
It is sometime called a literal
Double quate is not a part of string

Example
char name[] = "Kamal";
↓ string variable ↘ string constant

char name[20];
↳ Capable of string is character and one \0
\0 = null

Page No.:
Date :

Initializing string

(1) Character by character
char name[] = {'S', 'a', 'n', '\0'};

(1) strlen() → strlen(s)
gives length of string

(2) strlwr() → strlwr(s)
Return string in lowercase

(3) strcmp(Case sensitive) → strcmp(s1,s2)
Check weather two string are same or not

(4) strrev() → strrev(s)
→ Give reverse of string

(5) strupr() → strupr(s)
Return string in uppercase

(6) strcpy() → strcpy(s1, s2)
Copy s2 to s1

(7) strcat() → strcat(s1, s2)
→ Oppen s2 to end of s1

(8) strcpi() → strcpi(s1, s2)
Compare two string ignoring case.

CamScanner

Page No.:
Date :

WAP to print length of string without using Library function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 char str[100];
 int length (char []);
 printf("Enter string \n");
 scanf("%s", str);
printf("length of string is %d", length(str));
 getch();
 }
int length (char str[])
 {
  int i=0;
  while (str[i] != '\0')
   {
    i=i+1;
   }
   return i;
 }
```

Page No.:
Date :

Pointer:

- A variable that store address of another variable is called pointer
- Simply a pointer is a variable that hold memory address. This address is the location of another object of memory.

Advantages

> Pointer provide a way to use address of element directly.

> Speedup the execution of programs because accessing address is faster than accessing value.

> It enable to pass information back on fourth function and refarence point.

> Enable to return more than one value from a function.

> Help to pass array and string more efficient to function.

> Complex data structure (queue, stack linked list, trees) can be easily implemented.

> Memory can be efficiently utilized

> Hardware interface is possible with the help of pointer.

> Accessing dynamically allocated memory dynamic memory creation and deletion.

Page No.:
Date :

Drawback

> Pointer variable can not be declare as global because if it is global then automatically initialize to zero this ~~mean~~ means indicating 0 of memory.

> static pointer variable does not work because data will be replaced.

> Memory lick (once memory is allocated to ovon we can not reallocated before realizing explicitly.

> Dangling pointer.

Pointer Variable:

Syntax

datatype * variable name

Example

int *p;

↳ pointer variable

Pointer Operator:

> * indirection operator (Return value of address)

> & address operator (Return address

Page No.:
Date :

```
# include<stdio.h>
#include <conio.h>
 void main()
  {
   int a=5;
    int *p;
    p=&a;                    // Output
   printf("%d", a);          // 5
   printf("%d", *p);         // 5
   printf("%x", &a);         // address of a
   printf("%x", p);          // address of a
   printf("%x", &p);         // address of p
    getch();
   }
```

WAP to find biggest number among three no. using pointer.

```
 #include <stdio.h>
 #include<conio.h>
  void main()
  { int a,b,c,large;
   int *p, *q, *r;
   print f("Center three number");
   scanf("%d%d%d", &a, &b, &c);
      p=&a;
      q=&b;
      r=&c;
  large=((*p>*q)?(*p>*r?*p:*r):
          (*q>*r? *q:*r));
  printf("larg number is %d", large);
   getch();
  }
```

Page No.:
Date :

Pointer Arithmetic

Example 1:

Addition of a number to a pointer

```
int *p, a=5, b;
p=&a; → Assign address of a
p=p+1; → increase address of p by 2 bytes
(*p)++; → increase the value of a by 1
p++; → increase address pointer by p
       by 2 bytes
```

Example 2:

```
float *p, a=5;
p=&a; → Assign address of a
p=p+1; → increase address of p by 4 bytes
p=p-1; → decrease address of p by 4 bytes
(*p)++; → increase the value of a by 1
p++ → increase address of p by 4 bytes.
```

Example 3:

```
int *p, a=5
p=&a; Assign address of a
p=p-1; decrease address of
       p by 2 bytes
(*p)--; decrease the
       value of a by 1
p--; decrease
       address of
       p by 2 bytes.
```

```
float *p, a=5;
p=&a; Assign address of a
p=p-1 → decrease address of
       p by 4 bytes
(*p)--; decrease the
       value of a by 1
p=; decrease address
       of p by 4 bytes
```

Page No.:
Date :

Subtracting the pointer from another pointer.

Ex:

```
int *pa, a=5;
int *pb, b=10;
pa = &a;
pb = &b;
c = pb-pa;
```

~~Effective~~
Efficient use of memory by knowing the leakage memory.

It gives number of bytes between the address pointer by them

Assign value of one pointer to another pointer

```
int *pa, *pb, a=5
pa = &a
pb = pa;
```

pb contain value of a

Initialize pointer with null value

```
int *p;
p = Null;
```

- Arithmetic operator that are not applicable to pointer.
- Address of two pointer variable
- Multiplication of two pointer variable
- Devision of two pointer variable

Page No.:
Date :

Memory allocation

Static Memory allocation	Dynamic memory allocation
Eg. int a[100]; int y float sum;	Memory is allocated during running of program
* Memory is allocated at the time of compilation	Allocated by programmer (end user) not by compiler
* Size is fixed.	Memory saving
* during execution we can't change the size of allocated memory area	fleribility is allocation memory
	Malloc(), Calloc(), Realloc use for this
	free() used to ~~delear~~. deallocate memory.

Syntax:

```
int *p
p=(int *) malloc (n* sizeof (int));
```

Type cast

Example:

```
int *p
int n=10;
p =(int *) malloc(n* sizeof (int));
```

20 bytes memory allocation where we could store 10 integer value.

Page No.:
Date :

WAP read n values and print biggest number
use dynamic memory allocation (DMA) concept

```
#include <stdio.h>
#include <conio.h>
# include <malloc.h>
void main()
{
int *a, n;
int big(int *, int);
~~printf~~ printf("How many number = \n");
scanf("%d", &n);
a = (int *) malloc (n*sizeof(int));
printf("\n read values \n");
for(i=0; i<n; i++)
  {
  scanf("%d", (a+i));
  }
print f("Biggest num is %d", big(a,n));
free(a);
getch();
}
int big(int *p, int n);
  {
  int i;
  big = *(a+0);
  for(i=1; i<n; i++)
    {
    if(big < *(a+i));
    {
    big = *(a+i);
    }
    }
    return big;
  }
```

Page No.:
Date :

WAP to calculate the average of n values using pointer to an array an function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 float result;
 float avg (int *, int);
 int i, n, A[100], *P;
printf ("Enter how many numbers \n");
scanf ("%d", &n);
printf (" Enter values : \n");
for (i=0; i<n; i++)
  {
  scanf ("%d", &a[i]);
  } P=A;
  result = avg (P,n);
  printf (" Average of %f", result);
  getch();
  }
float avg (int *P, int n)
{ float sum, i;
  float res;
  for (i=0; i<n; i++)
   {
   sum= sum + *(P+i);
   }
  res = (float) sum/n;
  result  res;
 }
```

Page No.:
Date :

WAP read n values in array and find largest number using pointer to an array concept.

```
#include<stdio.h>
#include <conio.h>
void main()
{
long array[100], *largest, size, c, location=1;
printf("Enter the numbers of elements in array\n");
scanf ("%ld", &size);
printf(" Enter %d integers\n", size);
for(c=0; c<size; c++)
{ scanf ("%ld", &array[c]);
}
largest = array;
*largest = *array;
for(c=1; c<size; c++)
{
if(*(array+c)> *largest)
{
*largest= *(array+c);
location= c+1;
}
}
printf("largest element is present at
%d location and its value is %ld",
location, *largest);
getch();
}
```

Page No.:
Date :

WAP read n values in array and print them in ascending order using concept of pointer to an array.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i, *p, asc, temp, num[100];
printf("How many numbers you want to enter:\n");
scanf("%d", &asc);
printf("Enter values in array\n");
for(i=0; i<asc; i++)
{scanf("%d", &num[i]);
}
p=num;
for(i=0; i<asc-1; i++)
{
temp=*(p+i);
if(temp > *(p+(i+1)))
{
num[i] = *(p+(i+1));
num[i+1] = temp;
}
printf("\n Values in ascending order\n");
for (i=0; i<asc; i++)
{
printf("%d\t", *(p+i));
}
getch();
}
```

Array of pointer

a	b	c
5	10	15
2012	2030	2060

1D array

p[0]	p1	p[2]
2012	2030	2060

```
int *p[5];
int a,b,c;
p[0] = &a;
p[1] = &b;
p[2] = &c;
```

Output

*p[0] = 5	p[0] = 2012	&p[0] = 5000
*p[1] = 10	p[1] = 2030	&p[1] = 5002
*p[2] = 15	p[2] = 2060	&p[2] = 5004

2D array

```
#include<stdio.h>
#include<conio.h>
void main()
{ int i,j,a[3][3]={{1,2,3},{4,5,6},{7,8,9}};
int *p[3];
for (i=0; i<3 i++)
{
p[i]= a[i]; //Assign address to pointer
}
for(i=0; i<3 ; i++)
{ for (j=0; j<3 ; j++)
{ *(*(p+i)+j);
}
a[i][j]= *(a[i]+j)
=(*(a(a+i)+j))
*a[i][j]=(a[i]+j)
=(*(a+i)+j)

}
```

Page No.:
Date :

Unit 6: Structure and Union

Structure:

Syntax:

```
struct structure_tag-name
  {
   variable decleration
   - - -
   - - -
  };

Struct structure_tag-home
  {
   variable decleration
   - - -
   - - -
   - - -
  } structure variable;
```

Example:

```
struct student
{
char name[50];
int roll no;
float marks;
};
struct student stu1, stu2, stu3;
```

```
struct student
{
Char name[20];
int roll no;
float marks;
} stu1, stu2, stu3;
```

```
struct student [User define datatype
  { char name[20];
    int roll no;
    float marks;
  } stu [100]; [Array of structure
```

Page No.:
Date :

WAP read name, age and salary of fifty employee and display those information implement this using concept of structure (concept of user define datatype)

```
#include <stdio.h>
#include <conio.h>
struct employee
{ char name[30];
  int age;
  float salary;
} emp[50];
void main()
{
  printf("Enter employee information\n");
  for (i=0; i<50; i++)
  {
scanf("%s %d %f", emp[i].name, &emp[i].age,
        &emp[i].salary);
  }
printf("\n Employee information:\n");
  for (i=0; i<50; i++)
  {
  ~~printf ("%s%t %d%t %f%t~~
printf("%s\t%d\t%f\n", emp[i].name, &emp[i].age,
        emp[i].salary);
  }
  getch();
}
```

Page No.:
Date :

structure

- Structure is on which is used to store related fields of different data types.
- It is capable of storing hetrogeneous data
- Data of different datatype can be grouped together under a single name using structure.

In structure

> Each member must be unique.
> We can't initialize member within structure
> It is user defined datatype

Defining structure:

```
struct structure name
  {
    datatype member1;
    datatype member2;
      :
    datatype member n;
  } structure variable
```

Declare structure Variable

(1) With structure defination

```
struct student
  { char name[20];
    int rollno;
    float marks;
  } stu1, stu2, stu3;
```

Page No.:
Date :

② Using structure tag

```
struct student
{
char name[20];
int rollno;
float marks;
};
struct student stu1, stu2; struct student stu3;
```

Initialize structure variable

```
struct student
{ char name[20];
  int roll;
  float marks;
} stu1 = {"Kamal", 1, 66.5};
```

Accessing member of structure

```
struct student
{ char name[20];
  int rollno;
  float marks;
};
struct student stu1, stu2;
```

to access name

```
stu1.name;
stu2.name;
```

to access roll no.

```
stu1.rollno;
stu2.rollno;
```

Array of structure

```
struct student
{
Char name[20];
int roll no;
float marks;
} stu [100];
struct student stu2[50];
int A[100]
int B[50]
```

Page No.:
Date :

WAP read name, rollno. and marks of 100 students and print those information using structure.

```
#include <stdio.h>
#include <conio.h>
struct student
{
char name[20];
int rollno;
float marks;
}
void main()
{
int n=100, i;
struct student stu[100];
printf("Read student info \n");
for (i=0; i<n; i++)
{
scanf("%s %d %f", stu[i].name, &stu[i].rollno,
&stu[i].marks);
}
printf("\n Student info are. \n");
for(i=0; i<100; i++)
{
printf("%s\t %d\t %f\t \n",
stu[i].name, stu[i].rollno, stu[i].marks);
getch();
}
```

Page No.:
Date :

Structure and pointer

```
#include<stdio.h>
#include<conio.h>
struct student
{
char name[20];
int rollno;
float marks;
} stu, *p;
void main()
{
p=&stu;
printf("Enter name of students\n");
scanf("%s", p->name);
printf("Enter marks\n");
scanf("%f", p->marks);
printf("Name & marks are\n");
scanf("%s\t%f", p->name, p->marks);
getch();
}
```

Page No.:
Date :

Pass structure to function

```
#include <stdio.h>
#include <conio.h>
 struct student
  {
  char name[20];
   int rollno;
   float marks;
   };
  void main()
  {
  void display (char [], int, float);
  struct student stu={"Ram", 12, 87.5};
 display (stu.name, stu.rollno, stu.marks);
  getch();
   }
void display (char name[20], int rollno, float marks)
  {
 printf("Name \t \n", name);
 printf(" Rollno \t \n", rollno);
 printf(" Marks \t \n", marks);
 }
```

Page No.:
Date :

Pass structure variable to function

```
#include<stdio.h>
#include<conio.h>
 struct student
 {
 char name[20];
 int rollno;
 int marks;
 };
 void disp(struct stu);
  void main()
 {
 struct student stu;
 printf("Read student detail \n");
 scanf("%s %d %d", stu.name, &stu.rollno, &stu.marks);
  disp(stu);
  getch();
 }
 void display(struct student stu)
   {
   printf("Name \t %s \n", stu.name);
   printf("roll no \t %d \n", stu.marks);
    }
```

Page No.:
Date :

Pass Array of structure to function

```
#include <stdio.h>
#include <conio.h>
struct student
{
char name[20];
int rollno;
int marks;
};
void disp(struct stu[], int);
void main()
{
int i, n;
struct student stu[100];
printf("Enter how many student \n");
scanf ("%d", &n);
printf("Enter student info \n");
for(i=0; i<n; i++)
{
scanf("%s %d %d", stu[i].name, &stu[i].rollno,
&stu[i].marks");
}
disp (stu, n);
getch();
}
void disp(struct student stu[], int n);
{ int i;
printf(" student details are \n");
for (i=0; i<n; i++)
{ printf("Name\t %s\n", stu[i].name);
printf("rollno\t %d\n", stu[i].rollno);
printf("Marks\t %d\n", stu[i].marks);
}
}
```

Page No.:
Date :

rction

n");

");

e, &stu[i].rollno

nt n);

n);

rame);

);

rks);

Page No.:
Date :

Union

Union is a derived datatype like structure and it can also contain members of different datatype.

The syntax used for defination of union, declaration of union variables and for accessing members is similar to that used is structure but the keyword union is used instead of struct.

The main difference between union and structure is int the way memory is allocated for the members.

In the structure each member has it's own memory location, where as member of union share the same memory location. When a variable of type union is decleared compiler allocates sufficient memory to hold the largest memory in the union.

Since all members share the same memory location we can use only one member at a time.

Thus, union is used for save memory.

The concept of union useful when it is not necessary to use all member of union at a time.

Page No.:
Date :

Advantages of Union

> Saving of memory by sharing memory.

> Area where we need to access only one member is sufficient at a time.

> Union can be used to declare an array that can hold values of different datatype

Limitation of Union

> We can access one member of union at a time.

> We can not initalized the union members at the time of declaration because it can be overlaped or they share the same memory address.

Different between structure and Union

Structure	Union
Each member has their own memory location	Member share the memory location
struct keyword is used for structure	Union keyword is used for union
More memory	Less memory

Page No.:
Date :

Numbers of member can be initialized and use simultaneously.	Only one member can be initialize and use at a time
Size of structure is sum of size of each member	Size of union is the highest size of member of union.
We can initalized member at the time of dederation	We cannot initialized member at the time of decleration.
Syntax struct student { char name[50]; int age; float marks; } stu;	Syntax Union ~~structu~~ student { char name[50]; int age; float marks; } stu;

```
struct student
{int rollno;
 char name[50];
 float marks;
 };

struct hostel
{ int hostel id;
  char hosname;
  float charge;
  int roomno;
};
```

```
Union stuhos
{
struct student stu;
struct hostel hos;
} sh;
```

Page No.:
Date :

Unit 7: Input Output and File Handling

Steps for file operations
- Open a file
- Read, write or open data to file
- Close the file

Opening file
<stdio.h> contain opening file

```
FILE *fp
fp = fopen ("filename", "mode of file");
```

Example

```
FILE *fp;
fp = fopen ("test.txt", "r");
FILE *fp1, *fp2, *fp3;
fp1 = fopen ("test.txt", "w");
fp2 = fopen ("test.txt", "a");
```

Mode of opening file

1. "r"
2. "w"
3. "a"
4. "r+" Read and write
5. "w+" Read and write
6. "a+" Read & append

Page No.:
Date :

Open and exiting file for reading only.

W: Open a new file for writing only if a file with the specified file name currently exit, it will be destroyed and a new file created in its place

a: Open & exiting file for appending that is for adding new data at the end of file. A new file will be created if the file specified file name does not exit.

r+ : (Useful mode) Open & exiting file for both reading & writing.

w+ : Open a new file for both reading and writing. If a file with specified file name currently exit, it will be destroyed and new file created in its place.

a+ : Open & exiting file for both reading and appending. A new file will be created if the file with the specified file name doesnt exit

Page No.:
Date :

Error in opening a file

```
FILE *fp
fp = fopen ("Kamal.txt", "r");
~~if = fopen~~ if (fp == NULL)
  {
    printf ("Error in opening file");
    exit;
  }
  else
  {
    ---
  }
```

file close

```
fclose (fp);
```

Syntax;

```
fclose (filepointer);
```

Function for I/O

① fopen() : Open new file as open exiting file.

② fclose() : Close the open file

③ get ()/ fget() : read a character from file

④ putc()/fputc() : Write a character on file

Page No.:
Date :

(5) fgets() : read string from file.

(6) fputs() : Write string on file

(7) getw() : Read a single integer from file

(8) putw() : Write a integer to file

(9) fscanf() : Read data from file

(10) fprintf() : Write data to file

(11) fread() : Read entire block from file

(12) fwrite : Write entire block from given file

(13) fseek() : Used to set the cursor to specified place

(14) ftell() : Gives the current position of cursor on file

(15) rewind : Place the cursor on the beginning the file.

Character oriented I/O operation fgetc/getc()
fputc/putc()

Page No.:
Date :

fputc/putc()

WAP to demonstrate fputc() & fgetc()

```
#include<stdio.h>
#include<conio.h>
#include <IO.h>
void main()
{
  FILE *fp;
  char ch;
  fp=fopen("abc.txt", "w");
  if(fp==NULL)
  {
   printf("Unable to open file\n");
   exit(1);
  }
  else
  {while(ch=getchar())!=EOF)
    { fputc(toupper(ch), fp);
    }
    fclose(fp)
  }
 printf("\n Successfully inserted\n");
  fp= fopen("abc.txt", "r");
 while((ch= fgetc(fp))!=EOF)
  {
   printf("%c", ch);
  }
 printf("\n Successfully read from file");
 fclose(fp);
 getch();
}
```

Page No.:
Date :

WAP to demonstrate getw() and putw()

```
#include<stdio.h>
#include<conio.h>
#include<io.h>
void main()
{
int i, n, num, sum;
float avg;
FILE *fp;
fp= fopen("abc.txt", "w");
if (fp== NULL)
{
printf(" file cant be create\n");
exit(1);
} else
{ printf(" Enter how many num\n");
scanf("%d", &n);
for(i=0; i<n; i++)
{ scanf("%d", &num);
putw(num, fp);
}
fclose(fp);
fp=fopen("abc.txt", "r");
for(i=0; i<n; i++)
{ num= getw(fp);
sum = sum+num;
printf("%d", num);
} fclose(fp);
printf(" Sum is %d", sum);
avg = (float) sum/n;
printf(" Average is %f", avg);
getch();
}
```

Page No.:
Date :

WAP by which first you enter number data on data.txt. Then read values from it and determine each number weather even or odd. If even write that to even.txt otherwise on odd.txt. Then after read even and odd from their own file and display on video display screen.

```
#include<stdio.h>
#include<conio.h>
#include<IO.h>
void main()
{
int num,n,i;
FILE *fp1, *fp2, *fp3;
printf ("Contents of file data.txt");
fp=fopen("data.txt", "w");
if (fp1==NULL)
{ printf (" file doesnot exit");
exit(1);
}
else
{
printf ("How many numbers\n");
scanf("%d", &n);
for (i=0; i<n; i++)
{
scanf ("%d", &num);
put w(num, fp1);
}
fclose (fp1);
fp1= fopen ("data.txt", "r");
fp2 = fopen( "odd.txt", "w");
fp3 =fopen ("even.txt", "w");
```

Page No.:
Date :

```
While (num=getw(fp1)) != EOF)
  {
   if(num %2 ==0)
   putw(num, fp3)
   else
   putw(num, fp2);
   }
  fclose(fp2);
  fclose(fp3);
 fp2= fopen("odd.txt", "r");
 fp3 = fopen("even.txt", "r");
 printf("Odd number \n");
 While (num= getw(fp2) != EOF)
   printf("%d", num);
 printf("Even number \n");
 While (num= getw(fp3)) != EOF);
 printf("%d", num);
  fclose(fp2);
  fclose(fp3);
   getch();
 }
```

Page No.:
Date :

fgets() and fputs()

fputs() This function writes the null terminated string pointed to by string to a file.

The null character that marks the end of string is not return to profile unsuccess it returns the last character and error it returns end of file.

Syntax: int fputs(const char *str, File *fp)

```
#include <stdio.h>
#include <conio.h>
#include <io.h>
void main()
{
FILE *fp;
char s[80];
fp= fopen ("abc.txt", "w");
printf("Enter a reco line\n");
While (gets(s) != NULL)
   {
   fputs (s, fp);
   fputs ("\n", fp);
   }
   fclose (fp);
}
```

fgets(): This function is used to read characters from a file and these characters are stored in the string. It reads atmost n-1 characters from the file where n is the second arguments

fp is a file pointer which points to the file from which characters are read.

This functions returns the string unsuccess and on error or end of file it returns null.

This function reads characters from the file untill either a new line or an end of file is encounter.

Syntax:

```
char *fgets(Char *str, int, FILE *fp);
```

```
#include<stdio.h>
#include<conio.h>
#include<IO.h>
void main()
{
FILE *fp;
char s[80];
fp=fopen("abc.txt", "r");
while (fgets (s, 79, fp)!= NULL)
  {
  printf("%s", s);
  printf("\n");
  }
  fclose (fp);
}
```

Page No.:
Date :

WAP to copy the containt of abc.txt to xyz.txt file.

```
#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <string.h>
void main()
{
FILE *fp1, *fp2;
char s[20], ch[20];
int i;
fp= fopen ("abc.txt", "w");
if (fp1 == NULL)
  { printf (" File does not exist");
   exit (1);
   else
   printf ("Enter string \n");
   gets(s);
  for (i=0; i<strlen(s); i++)
   { fput c(s [i], fp1);
   }
   fclose (fp1);
   }
fp1=fopen ("abc.txt", "r");
fp2 =fopen ("xyz.txt", "w");
if (fp1 = NULL || fp2 = NULL)
 { printf (" The file doesnot exit");
  exit(1);
 } While ((str = fget ((fp1)) != EOF)
  { fput (str, fp2);
  } fclose (fp1);
   fclose (fp2);
  printf ("Successfully copied");
 getch();
}
```

Page No.:
Date :

fprintf()

This function is used to write formated data into the file.

It returns the no. of characters inserted into the file success and EOF and error.

fprint f(file *fp, control string, arguments)

fscanf()

This function is used to read data from file instead a VDU.

fscanf (FILE *fp, control string, arguments)

Page No.:
Date :

WAP read rollno, name and marks of students, write into file and read from file and then print on screen.

```
#include <stdio.h>
#include <conio.h>
#include <io.h>
struct student
{
 int roll;
 char name[50];
 int marks;
}
void main()
{
 char tname[50];
 int tmarks, roll_no;
 struct student stu[100];
 FILE *fp;
 int r, n;
 fp = fopen ("test.txt", "a+");
 printf ("Enter how many students\n");
 scanf ("%d", &n);
 for(i=0; i<n; i++)
 {
 printf("%s %d %d", stu[i].name,
  &stu[i].marks, &stu[i].rollno);
fprint f(fp, "%d %s %d", stu[i].rollno,
 stu[i].name, stu[i].marks);
 }
```

```
printf("Data saved in file successfully");
 rewind(fp);
printf("\n Student records are\n");
printf("roll\t name \t marks\n");
while(fscanf(fp, "%d%s %d", &troll, tname, &tmarks)!=EOF)
 {
  printf("%d\t %s\t %d", troll, tname, tmarks);
  printf("\n");
 }
 fclose(fp);
 getch();
}
```

Block read & write to file

It is useful to store blocks of data into file rather than individual elements. Each block has some fixed size. It may be a structure or an array. It is easy to read the entire block from the file or write the entire block to the file. There are two functions for this purpose.

1. fread
2. fwrite

Page No.:
Date :

fwrite()

- This is used to write to entire block to a given file.

- fwrite(*ptr, sizeof data, numbers of data, FILE*fp)

- *ptr points to the block of memory that contains the information to be return to the file.

- Size of data denote the length of each items in bytes. No. of data denoted the no. of items to be return to the file.
- fp is the file pointer

fread()

- This function is used to read an entire block to a given file.

- Syntax: fread (*ptr, size of date, number of date, FILE*fp)

- *ptr points to the block of memory which receives the data from the file.

- Size of data denote length of each items in bytes. No. of data denote no. of item to be read from the file. fp is a file pointer.

- Mode for block read/write

① wb ② rb ③ ab ④ wb+ ⑤ rb+ ⑥ ab+