Unit 4

PROGRAMMING IN C

Review of C Programming

- C is a general-purpose programming language used to write programs
- High level language and easy to use
- Base programming language for other
- Provides core concept of loop, array, string, function, structure, file handling etc.

Review: Features of C

- Simple to learn and understand
- Simple in size only having 32 keywords
- Portable or machine independent
- Provides lot of inbuilt functions
- Faster execution
- Extensible
- Reusable

Review: Structure of C

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

void main(){
  int x=5;
  printf("value of x = %d", x);
  getch();
  }
```

Review: Preprocessor and Header files

- Header files are those files that are defined to be included at the beginning of program that contains definition of data types and declaration of variables used by function
- Functions and variables are defined in-built in header files
- Ex. stdio.h, conio.h, math.h, process.h etc

Review: C character set

- Alphabets: A-Z, a-z
- Numbers: 0-9
- Special characters:

;	:	{	}	,	"	'	|
\	>	<	/	~	_	[	]
!	$	?	*	#	^	@	&

Review: Use of comments

- Single Line Comment:
  - To comment a single line, we simply place double slashes (//) at beginning
  - Single line comment does not have ending slashes
  - Eg. //This is single line comment

- Block/Multi Line Comment:
  - To comment multiple line or block of lines we use /* as opening and */ as closing
  - We can comment multiple lines or blocks of line
  - It has both opening and closing portion
  - Eg.

```
/* This is our
Multiple line comment */
```

Review: Identifiers, Keywords & Tokens

- Basic element recognized by the compiler is known as tokens
- They are text that does not break down into component elements
- Keywords are words that are used in source code i.e. predefined words
- Keywords like int, float, goto, if, else, etc are also called tokens
- Identifiers are the names of variable or functions that identifies some value or procedure

Review: Basic Data Types in C

- There are basically two types of data

1. Primary data types

Char	: for character types of data
Int	: for whole number integer types of data
Float	: for real number with decimal point of data
Double	: for large or lengthy numbers
void	: for NULL or empty type of data

2. Secondary data types

Array	: collection of similar objects or data elements
Pointer	: holds memory location address point
Structure	: user defined collection of various data types
Union	: similar to structure but saves memory usage
Enum	: similar to array but with prefixed values

Review: Constant and Variables

1. Constant
   - A constant is a fixed value which cannot be changed during the program execution
   - Constant can be defined using preprocessor directives
   - e.g. #define PI 3.14;
2. Variable
   - The value of variable can be changed during program execution.
   - Variable can hold values at a time
   - e.g. int x = 2;

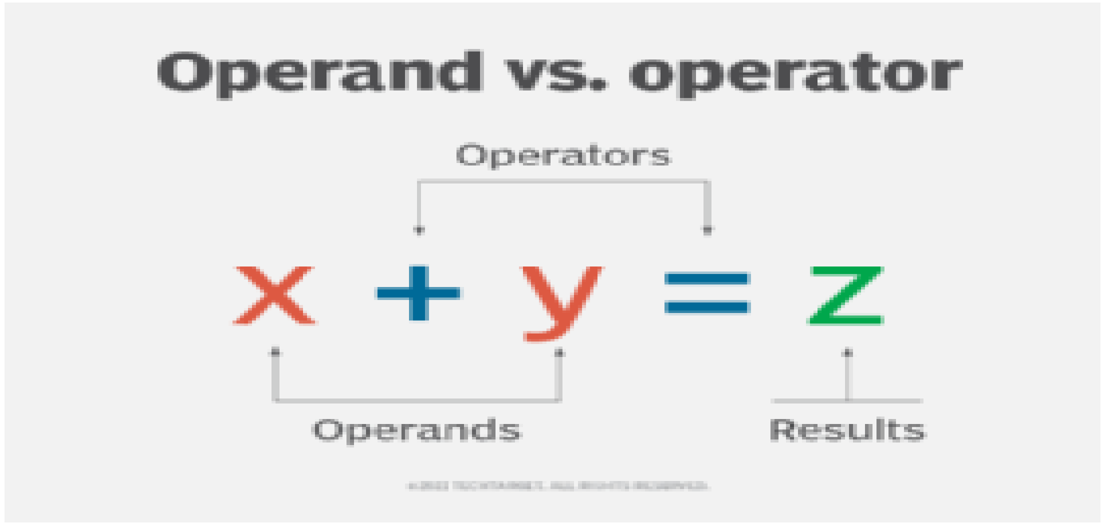
Review: Type of specifier

- The input and output data are formatted by specific pattern
- These patterns are generated by using specific tokens in C programs
- Tokens that are used to format data are called specifiers
- Some of the specifiers are :

1. Escape sequence
   - \n   \t   \v   \a
2. Format specifier
   - %d   %c   %f   %l

Review: Operators and Expressions

- An operator is a sign or symbol, which performs an operation or evaluation on one or more operands
- Operands are values or variables declared within program
- For an expression: a + b
  - here a and b are operands and + is operator



Review: Operators

Arithmetic operators :	+	-	*	/	%	
Relational operators :	<	>	<=	>=	==	!=
Logical operators :	&&		||	!		
Assignment operators :	=	+=	-=	*=	/=	%=
Conditional/Ternary operators:	? :					
Bitwise operators :	&	|	^			
Comma operators :	,					
Increment/Decrement operators:	++		--			

Review: Input/output (I/O) functions

- Input/output functions are categorized into following types
  1. Formatted I/O functions

Function	Description
printf()	It is an output function that prints a character or string or numeric values on the screen. It offers %d, %c, %s, %u conversion characters for various data types
scanf()	It is an input function that reads the input from keyboard and different data can be entered like int, float, char, string etc into C program
fprintf()	It is an output function that is used to write strings into files
fscanf()	It is an input function that is used to read strings from files

Review: Input/output (I/O) functions

- Input/output functions are categorized into following types
  1. Unformatted I/O functions

Function	Description
getch()	It is an input function which reads only one character at a time without echoing (displaying) on screen
getche()	It is an input function which reads only one character at a time with echoing (displaying) on screen
putch()	It is output function that outputs a single character on the screen
getchar()	It is an input function to read character from keyboard
putchar()	It is an output function to print a character on the screen
gets()	It is an input function that reads a single or multiple words(string) from keyboard
puts()	It is an output function that prints a single or multiple words(string) on screen

Review: Control Statements

- They define flow of control in a program and enable us to specify order of instruction execution.
- Control structures are very core part of programming
- There are 3 basic control structures

1. Sequential Control Structure
2. Decision Control Structure
3. Looping Control Structure

Review: Control Statements

1. Sequential Control Structure

- They are default control structure.
- In sequential control each commands or statements are executed one after another in sequential fashion
- Statements are executed from the very first line from main function

Review: Control Statements

1. Decision Control Structure

- They are used to divert the flow of control/execution on the basis of condition

    1. if statement
    2. if … else statement
    3. if … else if … else statement
    4. Nested if … else statement
    5. switch statement

Review: Control Statements

1. Looping Control Structure

◦ They are used to execute statement or block of statement repeatedly for until certain condition is satisfied.

1. for loop
2. while loop
3. do-while loop

Functions

- A function is a block of code that performs a specific task when called.
- There must be at least one function i.e. main(). Every program in C starts from main() function
- It helps to break down the large and complex program into small and manageable codes

Functions

- Syntax:
  - return_type function_name (parameter1, para2..)

```
{
    function block
}
```

advantages of using function

- avoids repetitions of code
- increases program readability
- divide the large complex codes into small simplex ones
- easy debugging, modification and updating
- easy code organization and reuse
- saves time and effort of programmer

Library vs user defined function

- 1. Library functions/Built-in function/Predefined function
  - also called built-in function or pre-defined function
  - already defined, compiled and stored in headers files
  - easier to write and use
  - not necessary to declare and define such function
  - we can just call wherever and whenever required
  - e.g. printf(), scanf(), strlen(), strcpy() etc.

advantages of Library function

- easy to use and are 100% accurate
- each library function performs specific tasks
- reduction in program size
- saves time of development

Self Study

- List out the built-in functions defined in following header files
  - stdio.h
  - math.h
  - string.h
  - stdlib.h
  - time.h
  - ctype.h

User Defined Function

- Function written and used by programmers is called user defined function
- function must have following characteristics
  - function prototype (declaration)
  - function body (definition)
  - function call
  - return statement (optional)

User defined function

- syntax

```
return_type function_name (arg1, arg2, ...)
    {
            function_body
    }
```

characteristics of user defined function

- function name must be unique
- can perform task without interfering other i.e. independent
- function can receive value from the calling program
- can pass and return values to and from function
- not executed until called
- can be called multiple times once defined

advantages of user defined function

- code can be reused multiple times
- large complex code can be divided into small simple sub programs
- easy to debug and maintain
- makes easy to understand logic
- avoids re-writing of codes

User defined Vs library

Basis	User defined function	Library function
function creation	created by user as their own requirements	can not be created by user as their own
storage	not stored in library or header file	stored in header or library files
renaming	name can be changed	name can not be changed
function declaration	user must declare and define these function	declaration and definition is not required
function definition	not pre defined	pre-defined in header files
function call	part of program, compiled at runtime	part of header files compiled at runtime
example	sum(), calculate(2,3) etc	printf(), scanf(), getch(), sqrt() etc

Types of user defined functions

1. With no return type and with no arguments
2. With no return type but with arguments
3. With return type but with no arguments
4. With return type and with arguments

Question:

Write a C Program that accepts radius and displays area or circle using different types of functions.

self study

- Types of user defined function on the basis of return type and arguments

1. No Return Type & No Arguments

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

void calcArea();

void main(){
	calcArea();
	getch();
}
```

```
void calcArea(){
	float a, r;
	printf("Enter radius: ");
	scanf("%f", &r);
	a = 3.14*r*r;
	printf("Area of circle is: %f", a);
}
```

2. No Return Type But Arguments

```
#include<stdio.h>
#include<conio.h>
void calcArea(float r);
void main(){
float r;
printf(" Enter radius: ");
scanf("%f", &r);
calcArea(r);
getch();
}
```

```
void calcArea(float r){
float a;
a = 3.14*r*r;
printf("Area of circle is: %f", a);
}
```

3. With Return Type But No Arguments

```
#include<stdio.h>
#include<conio.h>
float calcArea();
void main(){
float a;
a = calcArea();
printf("Area of circle is: %f", a);
getch();
}
```

```
float calcArea(){
float r, a;
printf(" Enter radius: ");
scanf("%f", &r);
a = 3.14*r*r;
return a;
}
```

2. With Return Type and Arguments

```
#include<stdio.h>
#include<conio.h>
float calcArea(float r);
void main(){
float a, r;
printf(" Enter radius: ");
scanf("%f", &r);
a = calcArea(r);
printf("Area of circle is: %f", a);
getch();
}
```

```
float calcArea(float r){
float a;
a = 3.14*r*r;
return a;
}
```

Accessing Function By Passing Values

- Arguments or parameters are used to receive required values from the function call
- Number of arguments must be matched with the called and calling function
- Actual parameters: that appear in a function call
- Formal parameters: that appear in function declaration

Parameter Passing Mechanism

- There are two parameter passing mechanism
  1. Call by Value (pass by value)
  2. Call by Reference (pass by reference)

1. Call By Value (Pass By Value)

- Actual value of arguments are sent to the function
- Change made to formal arguments does not change the actual arguments.
- Values are actually copied to new formal variables

1. Call By Value (Pass By Value)

```
#include<stdio.h>
int num(int a);
void main(){
		int x=3, y;
		y = num(x); //pass by value
		printf("Value of x = %d and y=%d" ,x,y);
		getch();
}

int num(int i){
i++;
return i;
}
```

2. Call By Reference (Pass By Reference)

- Address of arguments are sent to the function
- Any change made to the formal arguments will be reflected on actual arguments.
- Values are not copied to new formal variables instead all arguments points to same values.
- Used to return more than one variables from function

2. Call By Reference (Pass By Reference)

```
#include<stdio.h>
int num(int *a);
void main(){
int x=3, y;
y = num(&x); //pass by reference
printf("Value of x = %d and y=%d" ,x,y);
getch();
}

int num(int *i){
*i++;
return *i;
}
```

pass by value vs pass by reference

Basis	Pass by Value	Pass by Reference
Return value	Function cannot return more than one value at a time.	Function can return more than one value at a time
Concept of pointer	No concept of pointer used	Concept of pointer used
Change in value	Changes made in formal arguments do not change the actual argument	Changes made in formal arguments will be reflected in the actual arguments
Value send	Copy of data is sent to the function	Memory address of data is sent to the function
Example	Area (l , b);	Area (&l, &b)

Storage class

- each data types has one more attribute known as storage class which specifies lifetime and visibility of the variable
- visibility or scope is the accessibility of variable
- proper use of storage class makes program fast and efficient
- syntax: storage_class data_type variable_name

Types of storage class

- 1. auto : default variable
- 2. external : global variable
- 3. static : local variable
- 4. register : variable stored inside a register

1. Auto

- all variables declared inside a function without any storage class are called automatic variables.
- 'auto' keyword can be used to declare such variables but not compulsory
- also called local or internal variables
- initially takes garbage value by default
- initialized when program starts and destroyed after completion
- can't access local variables outside function
- e.g. auto int i;

2. External

- also called global variables
- declared above main function or outside function
- initial value of external variable is zero
- 'extern' keyword is specified in declaration
- memory is allocated at the time of definition
- e.g. extern int i;

3. Static

- static is local variable which is capable of returning a value even when control is transferred to the function call
- static variables have initial value zero and initialized only once on its lifetime.
- e.g. static int count = 1;

4. Register

- register class can be applied only to local variables
- scope, lifetime and initial value of register variable is same as auto variables
- stored in CPU registers
- can be accessed much faster than others
- 'register' keyword is used
- e.g. register int a;

Concept Of Recursion

- Recursion is the process by which a function calls itself repeatedly until some specified condition has been satisfied.
- it makes the code simple and easy to understand
- code execution may be slow and program execution may enter into infinite loop if proper condition is not specified
- there exists non recursive function for every recursive function

Concept Of Recursion

- To be recursive the conditions should be satisfied
  - function must be recursive in nature
  - there must be stopping condition

```
eg. void abc(){
.....
abc();
}
```

- //Program to find the factorial

```
#include<stdio.h>
#include<conio.h>
int fact(int);
void main(){
int n;
printf("Enter a number");
scanf("%d",&n);
printf("Factorial of number is %d",fact(n));
getch();
}
int fact(int n){
if(n==0){
return 1;
}
else{
return n*fact(n-1);
}
```

Structures

- Structure is a user defined data types that stores different data types of logically related data whereas array is collection of similar data types.
- Each member must be unique
- No storage class can be used or attached with member
- Can not initialize member variables
- No memory is allocated for structure before declaring variable
- Capable of storing heterogeneous data
- Used to manage database
- Program becomes systematic and efficient

Defining Structure

```
struct structureName
{
        datatype member1;
        datatype member2;
        ……
        datatype memberN;
} variable1, variable2;
// or
struct structureName variable1, variable2;
```

Structure Example

```
struct student
{
        char name[20];          //20 bytes
        int roll;               //2 bytes
        float marks;            //4 bytes
};                              //total size = 20+2+4 =26 bytes

struct student s1, s2, s3;
```

accessing Structure member

- for accessing structure member a dot(.) operator is used which is also known as the period or membership operator
- syntax:

```
    structureVariable.member
```

e.g.

```
    s1.name;
    s1.roll;
    s1.marks;
```

Initialization of Structure Variables

```
struct student
{
        char name[20];          //20 bytes
        int roll;               //2 bytes
        float marks;            //4 bytes
};                              //total size = 20+2+4 =26 bytes
struct student s1={"Ram", 4, 55.6};
                                   OR

s1.name = "Ram";
s1.roll = 4;
s1.marks = 55.6;
```

Classwork

- Write a program that accepts book title, isbn number, author, price of a book and show the details using structure.
- Write a program that accepts name, roll, marks, contact number of student, store them in structure and show them.

Array of Structure

- The array of structure is used to store large amount of similar records
- e.g. store the record of 100 employees
- Array of structure can be declared just like other
- e.g. struct student st[100];

Classwork

- Write a program that accepts book title, isbn number, author, price of 10 books and show the details using structure.
- Write a program that accepts name, roll, marks, contact number of 100 students, store them in structure and show them.

Union

- Union is user-defined data type like structure where member variables share a common memory space
- Variables save the shared memory that is not used.
- Size of Union is the largest data member which can hold the enough memory space
- In structure, each member has its own memory location but in Union each member shares the same memory location

Advantages of Union

- Union occupies less memory space compared to structure saving lots of memory
- Union are very helpful for low level programming
- Same memory can be used differently for different members of the union
- The last variable can be directly accessed
- Union can also be used for declaring array that can hold value of different data types

Syntax of Union

```
union unionName{
        Datatype member1;
        Datatype member2;
        Datatype member3;
        .....
        Datatype memberN;
}
```

Union Example

```
union student{
        int roll;               // 2-bytes
        char name[30];          // 30-bytes
        float marks;            // 4-bytes
}
```

size of union = 30 bytes

size of structure = 36 bytes

accessing Union member

- for accessing Union member a dot(.) operator is used which is also known as the period or membership operator
- We can access only the recently used variables
- syntax:

```
        unionVariable.member
```

e.g.

```
        s1.name;
        s1.roll;
        s1.marks;
```

Pointers

- Pointer is a variable that stores the references to another variable instead of storing the actual value
- Pointer variable contains the memory address of another variable, object or function.
- Pointer is declared to be specific type depending on what it points to
- Pointer of various data types can be created such as integer, character, string, structure or union

Pointer Variable

- Pointer variables are declared using the asterisk symbol * with the data type name and name of pointer to be declared.

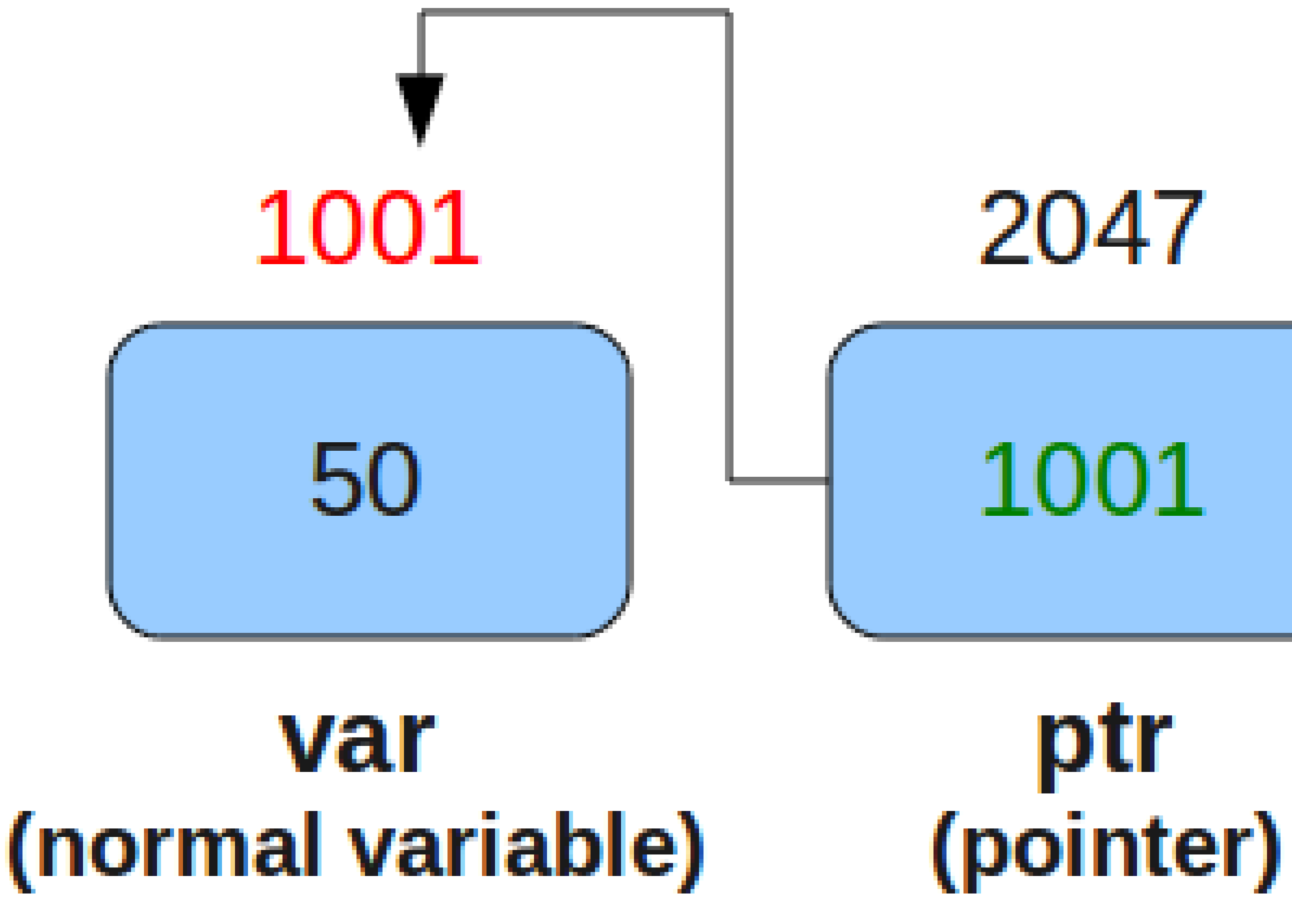
Syntax:

```
data_type * pointer_variable;
```

Example:

```
int *p;
char *ptr;
```

Pointer Variable



Benefits of Pointer

- speed up the execution of program
- complex data structures like queues, link, lists, stack, trees can be easily implemented
- memory can be efficiently utilized
- hardware interpretation is possible
- can interact with operating system
- dynamic memory creation and deletion is possible
- enables to return more than one value from a function
- pointers support dynamic memory allocation (DMA)
- can access memory faster because accessing memory location is faster than value

Drawbacks of Pointers

- Pointer variable can not be declared as global variable because all global variables are initialized zero (0), which indicates the address 0 of memory
- dynamically allocated memory can not be automatically released. It will exist till the end of program. It is programmer's responsibility to release that memory using free() function which is also called memory leak problem.
- If memory is deleted by dynamic memory allocation (DMA) but the pointer to the location where it does not exist any object. It is called dangling pointer.

Null Pointer

- Null pointer is a special pointer which "points to nothing"
- Null pointer points to NULL.
- C Language uses the symbol NULL for this purpose

Address (&) and Indirection (*) Operator

1. Address of (&) Operator

- The address of operator (&) will return its operand's address. This can be done as follows
  - int a=5;
  - int *p;
  - p = &a;

2. Indirection (*) Operator

- It returns the memory address of the operands i.e. indirection operator can give the actual value of a variable
  - int a=5;
  - int *p;
  - p = &a;

Pointer Expression and Assignment

We can perform following operations on pointers

- increment (++)
- decrement (--)
- addition of integer to a pointer (+)
- subtraction of integer from a pointer (-)
- subtraction of pointer from another pointer
- comparison of two pointers

We can not perform following operations on pointers

- addition of two pointers
- multiplication of two pointers
- division of two pointers

1. Incrementing a pointer

- any pointer variable when incremented (++) points to the next memory location of its type.
- pointer gets incremented according to the data type of the value it stores

```
e.g. int *p, a=5;
     p=&a;
     p++; //increases address pointed by p by 2 bytes
     (*p)++;    //increases the value of a by 1;
```

2. Decrementing a pointer

- any pointer variable when decremented (--) points to the previous memory location of its type.
- pointer gets decremented according to the data type of the value it stores

```
e.g. int *p, a=5;
     p=&a;
     p--;   //decreases address pointed by p by 2 bytes
     (*p)--;      //decreases the value of a by 1;
```

3. Addition of Integer to a pointer

- Addition of integer number to a pointer is allowed.
- Addition of any integer number to a pointer increases the address in pointer by that amount

```
e.g. int *p, a = 5;
     p=&a;
     p = p+2;
//increases the address pointed by pointer p by 4 bytes
```

4. Subtraction of Integer to a pointer

- Subtraction of integer number to a pointer is allowed.
- Subtraction of any integer number to a pointer decreases the address in pointer by that amount

```
e.g. int *p, a = 5;
     p=&a;
     p = p-2;
//decreases the address pointed by pointer p by 4 bytes
```

5. Subtraction of one pointer from another

- A pointer variable can be subtracted from another pointer variable only if they point to the same data type
- If two pointers are of different data type then type mismatch occurs.

```
int *pa, *pb;
int a=5, b=15, c, d;
pa=&a;
pb=&b;
c = *pa - *pb;          //subtracts values pointed by pa & pb
d = pa - pb;            //subtracts addresses of pa & pb pointers
```

6. Comparison of two pointers

- Comparison of two pointer variables is possible only if the two pointer variables are of the same type.
- They can check equality and inequality
- Result is TRUE if both the pointers point to the same location or address and result if FALSE if they point to different location in the memory

```
int *pa, *pb;
int a=5;
pa=&a;
pb=&a;
if(pa==pb){
printf("Both pointer points to same location");
}else{
printf("Two pointers points to different location");
}
```

Classwork

1. Write a program to enter any two numbers and find their sum using pointers.
2. Write a program to display the array elements using pointer

File Handling

- A data file is any file containing information but not code only meant to be read or write
- C program can be used to read or write files permanently
- Data is lost when data is stored using variables after program termination
- Variables hold data on Primary memory only which is volatile in nature
- Data must be stored in permanent secondary memory if it is needed later
- It is difficult to handle large amount of data using variables only

File Handling

- File handling in C enables us to create, update, read and delete files stored on the local file system through our C program
- Operations that can be performed on a file are:
  - Creation of the new file
  - Opening an existing file
  - Reading from the file
  - Writing to the file
  - Deleting the file

Needs for file handling in C program

- Helps in permanent storage of data or information generated after running the program
- Large amount of data can be stored for later use
- Easy to transfer contents of a file from one system to other
- Saves lot of time as we don't need to enter data from keyboard
- Files are easily portable to carry from one computer to another

Types of files

- There are two types of files

1. Text files
   - Text files contains text information like alphabets, digits, special symbols, punctuation marks which are easily readable
   - The ASCII codes are stored in these files
   - Text file has the **.txt** extension
   - Text files are used for files containing plain text that can be opened and viewed with simple text editors like notepad
2. Binary files
   - Binary files can be interpreted and understood by the computer and is not always printable on screen
   - Binary files have **.dat** extension
   - Binary files are often used for all kinds of objects to store data that is not just plain text

End of File (EOF)

- EOF is a sign or symbol that the end of file is reached and there are no more data to be read
- EOF is condition where no more data can be read from data source
- Character reading functions such as getc(), getch() or getchar() will return a value of EOF to indicate that an end-of-file condition has occurred. The value returned is mostly -1

Sequential Vs Random Access File

S.N.	Sequential Access	Random Access
1.	The computer system reads or writes information to the file sequentially, starting from the beginning of the file and going on step by step	The computer system can read or write information anywhere in the data file
2.	File is accessed slowly	File is accessed quickly
3.	Sequential Files must search through each and every file	Random files need not to search every files
4.	Insertion and updating of data is difficult	Insertion and updating of data is easy
5.	fread(), fscanf() function is used	fseek(), ftell(), rewind() functions are used

File Manipulation Function

S.N.	Functions	Descriptions
1.	fopen()	This function opens new or existing file
2.	fprintf()	This function write data into the file
3.	fscanf()	This function reads the data from the file
4.	fputw()	This function writes an integer value to file
5.	fgetw()	This function reads an integer value from a file
6.	fputc()	This function writes a character into a file
7.	fgetc()	This function reads a character from a file

File Handling Functions

fopen()

- It is used to open a file to perform operations such as reading, writing etc.
- In a C program, we declare a file pointer and use it
- The function creates new file if the mentioned file name does not exist.

Syntax:

```
    FILE *fp;
    fp = fopen("filename", "file_openingmode");
```

Example:

```
    fp = fopen("book.txt", "r");
```

File Handling Functions

fprintf()

- This function writes any types of data(integer, float, string, char etc) into a file pointed by fp pointer.
- This function is same as printf() function but instead of writing data on monitor, it writes the content to the file
- It has one extra parameter which points to a file

Example:

```
FILE *fp;
fp = fopen("book.txt", "w");
fprintf(fp, "computer science");
```

File Handling Functions

fscanf()

- This function is same as scanf() function but it has one extra parameter that points to a file
- Instead of reading the data from a standard keyboard, it reads the data from memory

Example:

```
FILE *fp;
fp = fopen("book.txt", "r");
fscanf(fp, "%d", &a);
```

File Handling Functions

putw()

- It is used to write an integer into a file

Syntax:

```
    putw(i, fp);where i is integer value & fp is file pointer
```

Example:

```
    FILE *fp;
    int i=4;
    fp = fopen("book.txt", "w");
    putw(i, fp);
```

File Handling Functions

getw()

- It reads an integer value from a file pointed by file pointer

Syntax:

```
    getw(fp);where fp is file pointer
```

Example:

```
    FILE *fp;
    int i;
    fp = fopen("book.txt", "r");
    i = getw(fp);
```

File Handling Functions

fgetc()

- It returns a single character from the file
- It gets a character from the stream
- It returns EOF at the end of file

Syntax:

```
fgetc(fp);where fp is file pointer
```

Example:

```
FILE *fp;
fp = fopen("book.txt", "r");
c = fgetc(fp);
```

File Handling Functions

fputc()

- It is used to write a single character into file
- It outputs a character to a stream

Syntax:

```
fputc(c, fp);where c is character & fp is file pointer
```

Example:

```
FILE *fp;
fp = fopen("book.txt", "w");
fputc('a',fp);
```

File Handling Functions

fclose()

- It closes the file that is being pointed by file pointer fp

Syntax:

```
    fclose(filename or file pointer)
```

Example:

```
    FILE *fp;
    fp = fopen("book.txt", "w");
    fputc('a',fp);
    fclose(fp);
```

Creating a file

- To work with file, we must first create it
- To create a file

Syntax:

FILE *fp;

FILE is defined in the stdio.h header file

fp is a pointer variable

Opening a file

- A file must be open before any READ/WRITE operations can be performed on that file
- The process of establishing a connection between the program and file is called opening a file
- To open a file fopen() function is used

Syntax:

```
pointer_variable = fopen("filename.extension","file_mode");
```

Example:

```
fp = fopen("data.txt","w");
```

File Opening Modes

Mode	Meaning	Description
r	Reading	this mode opens a file for reading only. The file to be opened must exist. If file is opened successfully fopen() loads into memory otherwise returns NULL
w	Writing	This mode opens an empty file for writing only. If the file already exists the previous file will be erased and new file is created otherwise it will simply create new file
a	Append	This mode opens a file for appending (i.e. adding the new information at the last). It file does not exist, new file will be created.
w+	Write & Read	This mode opens a file for both writing and reading. If file is already exist, previous data will be erased.
r+	Read & Write	This mode opens a file for both reading and writing. If the file to be opened exist, previous data will not be erased. This mode is also called update mode
a+	Append & Read	this mode opens a file ofr both reading and appending. A new file is created if the file does not exist. We cannot modify existing data in this mode.

Closing a file

- The file that was open must be closed whe no more operations are to be performed on it
- After closing file, the connection between file and a program is lost
- It is good to close a file when there is no more operations to be performed on a file

Syntax:

fclose(filename or file pointer);

Example:

fclose(fp);

Example

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

void main(){
FILE *fp;
fp = fopen("greeting.txt","w");
fprintf(fp, "Good Morning all");
fclose();
getch();
}
```

Thank you

END OF UNIT 4