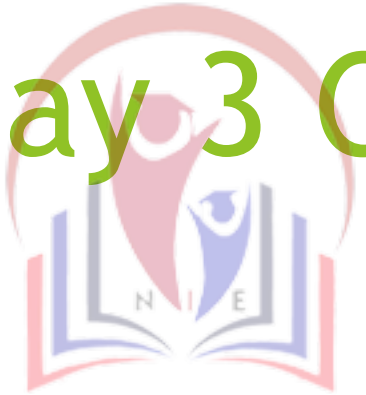
Day 3 C-Programming



Topics to be Covered

- Pointer arithmatic
- Pointer and array
- Passing pointer to function
- Structure Vs Union
- Array of Strucure
- Passing structure to function
- Structure and Pointer
- I/O operation on file
- Sequential and random access of file

Introduction

- A Pointer in C language is a variable which holds the address of another variable of same data type.
- Pointers are used to access memory and manipulate the address.

Declaration of Pointer Variable

Syntax:

Data_Type *Variable_Name;

Example:

int *ptr; => pointer to integer variable.

float *fptr; => pointer to floating variable

char *cptr; => pointer to character variable

Initialization of Pointer Variable

- **Pointer Initialization** is the process of assigning address of a variable to a **pointer** variable.
- Pointer variable can only contain address of a variable of the same data type.

Example1:

```
void main()
{
        int a = 10;
        int *ptr;          //pointer declaration
        ptr = &a;          //pointer initialization
}
```

Pointer to Pointer

```
int main()
 {
    int  a = 65;
    int  *p1;
    int  **p2
    p1 = &a;
    p2 = &p1;
    printf("Address of a = %u\n", &a);                  //  Address of a = 5000
    printf("Address of p1 = %u\n", &p1);               // Address of p1 =  6000
    printf("Address of p2 = %u\n\n", &p2);             // Address of p2 =  7000
    printf("Value of   a = %d", a);                    // Value of a = 65
    printf("Value of *p1 = %d", *p1);                   // Value of  *p1 =  65
    printf("Value of *p2 = %d", *p2);                   // Value of  *p2 =  5000
    printf("Value of   **p2 = %d\n", **p2);             // value of  **p2 = 65
    return 0;
}
```

Array of pointers

- The multiple pointers of same type can be represented by an array. Each elements of such array represents pointer and they can point to different locations.
- Array of pointers can be declared as:

Syntax:

```
data_type * pointer_variables[size];
```


Eg.

```
int ptr[5];
int a,b,c,d,e;
```

Here ,

ptr[0] = &a, ptr[1] = &b, ptr[2] = &c, ptr[3] = &d and ptr[4] = &e,

Ways of Passing Arguments to functions

- C provides two different mechanisms to pass arguments
  - Pass By Value
  - Pass By Reference

- **Pass By Value:** The process of passing actual value of variables is know as passing by value.In this mechanism, the values of actual arguments are copied to the formal arguments of the function definition. So the value of arguments in the calling function are not even changed even they are changed.

```
#include<stdio.h>
#include<conio.h>
void swap(int x,int y);
int main()
{
        int a,b;
        a=10;
        b=20;
        printf("value before swapping a=%d and b=%d",a,b);
        swap(a,b);
        printf("\nvalue after swapping a=%d and b=%d",a,b);
        getch();
        return 0;
}
void swap(int x,int y)
{
 int temp;
 temp=x;
 x=y;
 y=temp;
}
```

Pass by Reference

- Reference means address
- The process of calling a function giving address of variables as arguments is called passing by reference. Pointer variables are used for this purpose.
- Here, address of variables(&) are passed and pointer variables(*) to receive the address.

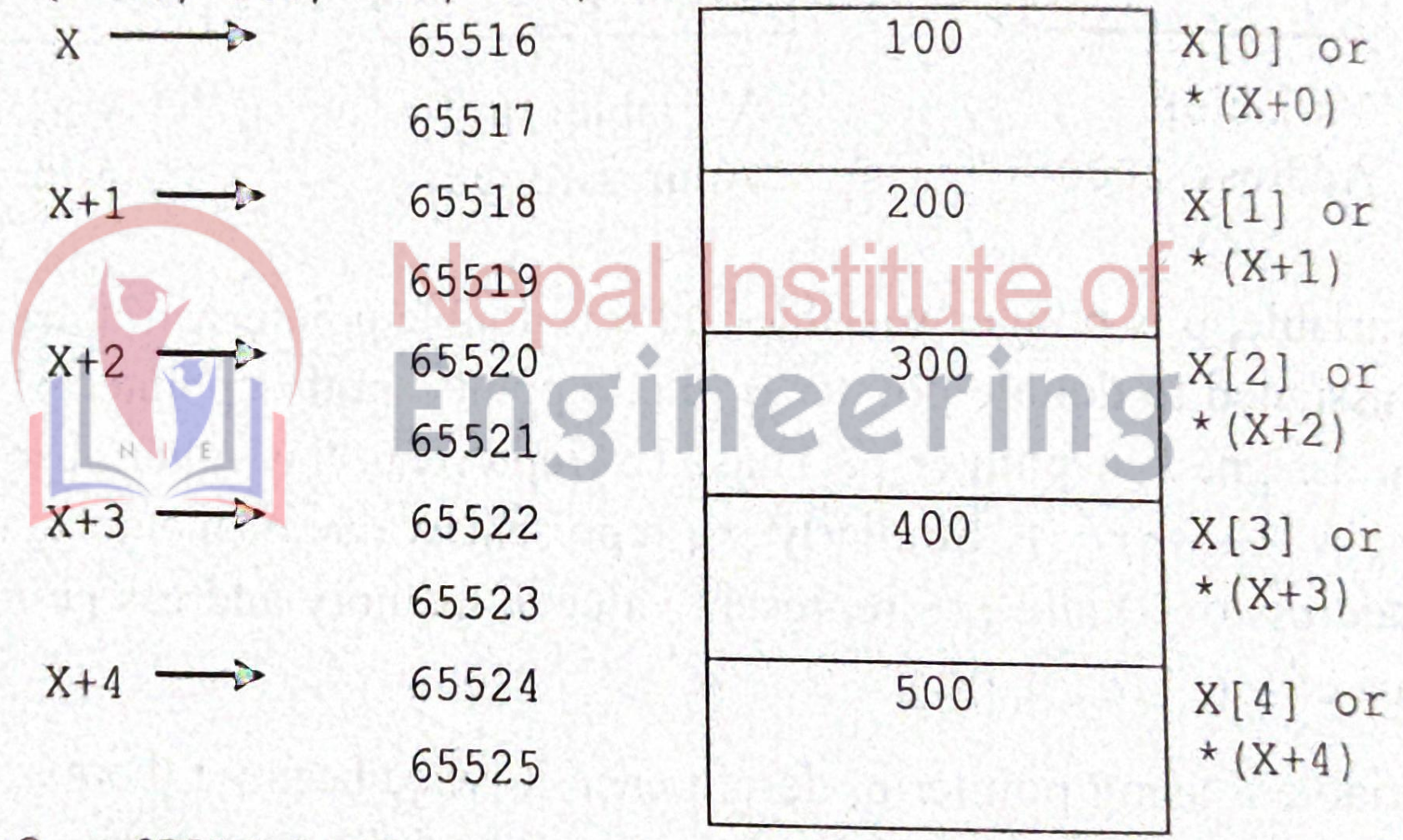
```
#include<stdio.h>
#include<conio.h>
void swap(int *x,int *y);

int main()
{
 int a,b;
 a=10;
 b=20;
 printf("The value before swapping are a=%d and b=%d",a,b);
 swap(&a,&b);
 printf("\nThe values after swapping are a=%d and b=%d",a,b);
 getch();
 return 0;
}

void swap(int *x,int *y)
{
 int temp;
 temp=*x;
 *x=*y;
 *y=temp;

}
```

1 D Array and Pointer

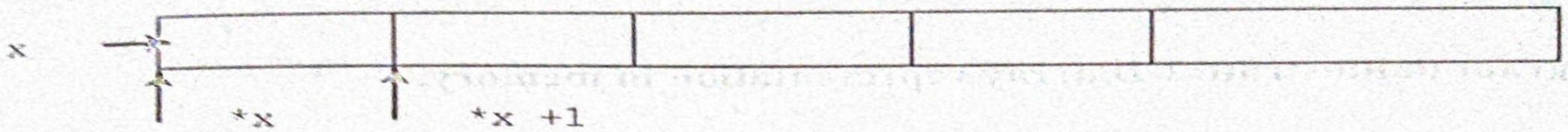
- An array name itself is a pointer which represents base address of array. Thus if x is 1D ARRAY, then address of the first array elements can be expressed as either &x[0] or simply as x.
- The address of second elements can be written as either &x[1] or simply as x + 1, and so on.
- So the address of ith elements can be written as either &x[i] or simply as (x +i).
- And the value of first elements of array can be accessed by x[0] or simply *(x + 0)
- Likewise the value of second elements of array can be accessed by x[1] or simply *(x + 1)
- So the value of ith elements of array can be accessed by either x[i] or simply as *(x +i).

1 D Array and Pointer

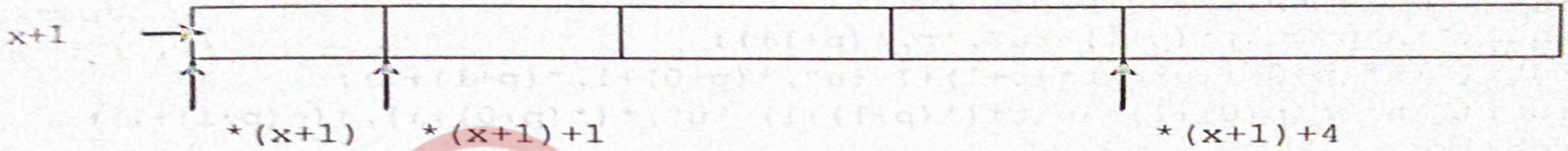
```
int X[] = { 100, 200, 300, 400, 500};
```



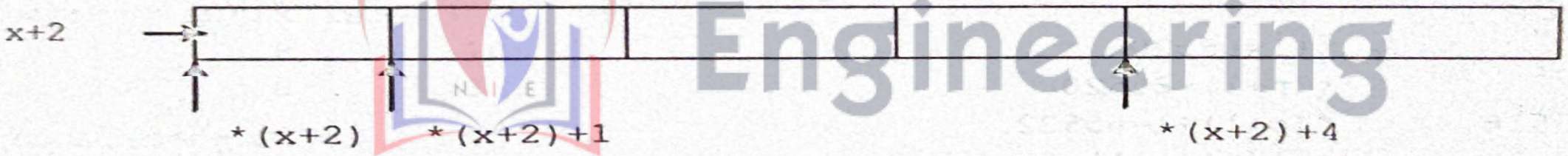
X+3 = 65516+3*2(number of bytes per integer) = 65522



1st 1-D array or first row



2nd 1-D array or second row



3rd 1-D array or third row

It can be summarized as:

- `x` → pointer to 1st row
- `x+i` → pointer to i^{th} row
- `*(x+i)` → pointer to first element in the i^{th} row
- `*(x+i)+j` → pointer to j^{th} element in the i^{th} row
- `*(*(x+i)+j)` → value stored in the cell `i, j`

Thus, in 2-D array,

`&x[0][0]` is same as `*x` or `*(x+0)+0`

`&x[0][1]` is same as `*x+1` or `*(x+0)+1`

`&x[2][0]` is same as `*(x+2)` or `*(x+2)+0`

`&x[2][4]` is same as `*(x+2)+4`

and

`x[0][0]` is same as `**x` or `*(*(x+0)+0)`

`x[0][1]` is same as `*(*x+1)` or `*(*(x+0)+1)`

`x[2][0]` is same as `*(*(x+2))` or `*(*(x+2)+0)`

`&x[2][4]` is same as `*(*(x+2)+4)`

In general, `&x[i][j]` is same as `*(x+i)+j` and `x[i][j]` is same as `*(*(x+i)+j)`.

2 D Array and Pointer

Eg. WAP to read a matrix of n x m order and display it using pointer notation

```
void main()
{
    int num[100][100],i,j,m,n;
    printf("How many Row");
    scanf("%d",&m);
    printf("How many Column");
    scanf("%d",&n);
    printf("Enter %d x %d Elements",m,n);
    for(i=0; i<m; i++)
    {
        for(j=0; j < n; j++)
        {
            scanf("%d", (*(num + i) + j ));
        }
    }
    printf("Elements of array are:\n");
    for(i=0; i<m; i++)
    {
        for(j=0; j < n; j++)
        {
            printf("%d ", *(*(num + i) + j ));
        }
        printf("\n");
    }
    getch();
}
```

Difference between Structure and Union

union	structure
union keyword is used to define the union type.	structure keyword is used to define the union type.
Memory is allocated as per largest member.	Memory is allocated for each member.
All field share the same memory allocation.	Each member have the own independent memory.

We can access only one field at a time.
 	
union Data {
int a; // can't use both a and b at once
char b;
} Data;
 	
union Data x;
x.a = 3; // OK
x.b = 'c'; // NO! this affects the value of x.a
	We can any member any time.
	
	struct Data {
	int a; // can use both a and b simultaneously
	char b;
	} Data;
	
	struct Data y;
	y.a = 3; // OK
	y.b = 'c'; // OK

Altering the value of the member affect the value of the other member.	Altering the value of the member does not affect the value of the other member.
Flexible array is not supported by the union.	Flexible array is supported by the structure.
With the help of union we can check the endianess of the system.	We cannot check the endianess. https://aticleworld.com/

Structure and Function

C allows passing of structure as arguments to function. There are 2 methods by which values of structure can be transferred from one function to another.

1. pass by value

2. pass by reference

- **Pass by value**

It involves passing a copy of the entire structure to the called function. Any changes to the structure members within the function is not reflected in the original structure (in the calling function). Therefore, it becomes necessary for the function to return entire structure back to the calling function.

Passing Value To Function

```
#include <stdio.h>
struct student {
   char name[50];
   int age;
};
void display(struct student s);
int main() {
   struct student s1;
   printf("Enter name: ");
   scanf("%[^\n]%*c", s1.name);
   printf("Enter age: ");
   scanf("%d", &s1.age);
   display(s1); // passing struct as an argument
   return 0;
}
void display(struct student s) {
   printf("\nDisplaying information\n");
   printf("Name: %s", s.name);
   printf("\nAge: %d", s.age);
}
```

Passing by reference

- It uses the concept of pointer to pass structure as an argument.
- The address location of the structure is passed to the called function.
- The function can access indirectly the entire structure and work on it. This method is more efficient than the first one. Structure pointer operator is used to access the member.

Example

```
#include<stdio.h>
#include<conio.h>
struct student
{
char name[50];
int roll;
} ;
void display(struct student *);
int main()
{
struct student s;
printf("Enter name of student:");
scanf("%s",s.name);
printf("Enter roll number:");
scanf("%d",&s.roll);
display(&s);
return 0;
}
void display(struct student *st)
{
printf("Name=%s\nRoll=%d",st->name,st->roll);
}
```

Structure and Pointer

If a pointer variable is assigned the address of a structure, then a member of the structure can be accessed by:

pointer_to_structure -> structure_member_name

```
#include <stdio.h>
struct person
{
  int age;
  float weight;
};
int main()
{
   struct person *personPtr, person1;
   personPtr = &person1;
   printf("Enter age: ");
   scanf("%d", &personPtr->age);
   printf("Enter weight: ");
   scanf("%f", &personPtr->weight);
   printf("Displaying:\n");
   printf("Age: %d\n", personPtr->age);
   printf("weight: %f", personPtr->weight);
   return 0;
}
```

Introduction Of File

The input output function:

- Like printf( ), scanf( ), getchar( ), putchar( ), gets( ), Puts( ) are known as console oriented I/O functions which always use keyboard for input device.

- While using these library functions, the entire data is lost when either the program is terminated, or the computer is turned off.

File Operations

In every file I/O, one has to do three basic operations. They are as follows:

i. Opening a file.

ii. Performing read, write, and append operation etc on an opened file.

iii. Closing a file.

Opening & Closing a data file

- Before a program can write to a file or read from a file, the program must open it.
- Opening a file establishes a link between the program and the operating system.
- This provides the operating system, the name of the file and the mode in which the file is to be opened.
- While working with high level data file, we need buffer area where information is stored temporarily in the course of transferring data between computer memory and data file.

Opening & Closing a data file

- The buffer area is established by:

```
FILE *ptr_variable;
```

- And file is opened by using following syntax

```
ptr_variable = fopen( file_name, file_opening_mode);
```

- where fopen( ) function takes two strings as arguments, the first one is the name of the file to be opened and the second one is file mode that decides which operations (read, write, append, etc.) are to be performed on the file.
- On success, fopen() returns a pointer of type FILE and on error it returns NULL.
- For example:

```
FILE *fp1, *fp2;
fp1 = fopen("myfile.txt", "w");
fp2 = fopen("yourfile.dat", "r");
```

Opening & Closing a data file

- The file that was opened using fopen() function must be closed when no more operations are to be performed on it.
- After closing the file, connection between file and program is broken. Its syntax is:

  fclose(ptr_variable);
- On closing the file, all the buffers associated with it are flushed and can be available for other files.
- For example:

  fclose(fp1);

  fclose(fp2);

File Opening Modes

- The file opening mode specifies the way in which a file should be opened (i.e. read, write, append, etc.).
- In other word, it specifies the purpose of opening a file.
- They are:

1. Write mode “w”
2. Read mode “r”
3. Append mode “a”
4. Write + Read Mode “w+”
5. Read + write Mode “r+”
6. Append + Read Mode “a+”

File Opening Modes

1. “w” (write):

- If the file doesn’t exist then this mode creates a new file for writing, and if the file already exists, then the previous data is erased, and the new data entered is written to the file.

2. “a” (append):

- If the file doesn’t exist then this mode creates a new file, and if the file already exists then the new data entered is appended at the new data entered is appended at the end of existing data.
- In this mode, the data existing in the file is not erased as in “w” mode.

File Opening Modes

3. “r” (read):

- This mode is used for opening an existing file for reading purpose only.
- The file to be opened must exist and the previous data of file is not erased.

4. “w+” (write + read):

- This mode is same as “w” mode but in this mode, we can also read and modify the data.
- If the file doesn’t exist, then a new file is created and if the file exists then previous data is erased.

File Opening Modes

5. “r+” (read + write):

- This mode is same as “r” mode but in this mode, we can also write and modify existing data.
- The file to be opened must exist and the previous data of file is not erased.
- Since we can add new data and modify existing data, so this mode is also called update mode.

6. “a+” (append + read):

- This mode is same as the “a” mode but in this mode, we can also read the data stored in the file.
- If the file doesn’t exist, a new file is created and if the file already exists then new data is appended at the end of existing data in this mode.

Binary Files

Opening modes:

- “rb”: Binary file opened in read mode
- “wb”: Binary file opened in write mode
- “ab”: Binary file opened in append mode
- “rb+” or (“r+b”): Binary file opened in update mode
- “wb+” or (“w+b”): Binary file opened in write + read mode
- “ab+” or “a+b”: Binary file opened in appended + read mode

Random Access to File

- We can access the data stored in the file in two ways, sequentially or random. So, far we have used only sequentially access in our programs.
- For example, if we want to access the forty-fourth record then first forty-three records should be read sequentially to reach the forty-fourth record.
- In random access, data can be accessed and processed randomly i.e. in this case the forty-fourth record can be accessed directly. There is no need to read each record.
- Sequentially, if we want to access a particular record. Random access takes less time than the sequential access.
- C supports these functions for random access file processing:
  - fseek( )
  - ftell( )
  - rewind( )

Random Access to File

fseek( ):

- This function is used for setting the file position pointer at the specified byte.
- Syntax:

```
fseek(fptr, offset, mode);
```

  - where fptr is a file pointer,
  - Offset is an integer which specifies the number of bytes by which the pointer is moved,
  - Mode specifies from which position the offset is measured.
  - The value of mode may be 0, 1 or 2 which is represented below.

Constant	Value	Position
SEEK_SET	0	Beginning of file
SEEK_CURRENT	1	Current position
SEEK_END	2	End of File

Random Access to File

fseek( ):

- Examples:

fseek(fptr, 0, SEEK_SET) => moves the file pointer at the beginning of the file

fseek(fptr, 0, SEEK_END) => moves the file pointer at the end of the file

fseek(fptr, 10, SEEK_SET) => moves the file pointer at 10 bytes right from the beginning of file

fseek(fptr, -2, SEEK_CUR) => moves the file pointer at 2 bytes left from the current position

Random Access to File

rewind():

- This function is used to move the file pointer to the beginning of the file and then re-open it.
- Syntax:

```
rewind(fptr);
```

- using `rewind(fptr)` is equivalent to `fseek(fptr, 0, SEEK_SET)`

ftell( ):

- This function returns the current position of the file position pointer.
- The value is counted from the beginning of the file.
- Syntax:

```
ftell(fptr);
```

Thank You