Chapter: 8
Pointers

Department of Electronics and Computer Engineering

Er. Pradip Khanal

Er. Bikash Acharya

Er. Bigyan Karki

ADVANCED COLLEGE OF ENGINEERING & MANAGEMENT

Affiliated to Tribhuvan University (T.U.)

ESTD. 2000

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.

Advantages of Pointer

- ✓ Pointers are more efficient in handling Arrays and Structures.
- ✓ Pointers allow references to function and thereby helps in passing of function as arguments to other functions.
- ✓ It reduces length of the program and its execution time as well.
- ✓ It allows C language to support Dynamic Memory management.

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
```

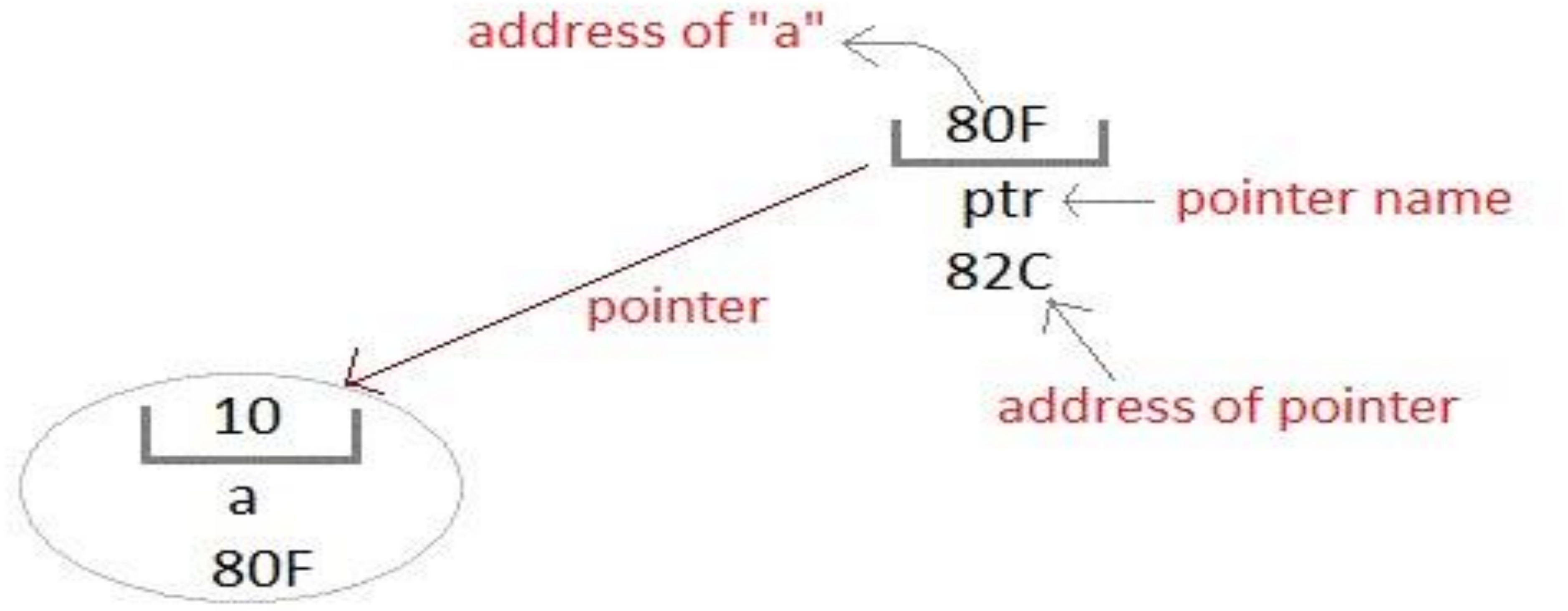
Initialization of Pointer Variable

- ✓ **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.

<u>Example1:</u>

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

Initialization of Pointer Variable



Initialization of Pointer Variable

Example2:

```
void main()
{
        float a = 10.06;
        int *ptr;      //pointer declaration
        ptr = &a;      //Error. Type Mismatch
}
```

Deference operator (*)

- ✓ The operator ‘*’ used infront of variable is called dereference operator.
- ✓ This operator is used in two distinct ways
  1. Declaration
  2. Dereference operator

Declaration

Eg. int x, *xptr; $\longrightarrow$ * used as declaration of pointer variable.

xptr = &x; // Here * is used to declare pointer variable.

Deference operator (*)

Dereference

```
Eg      int main()
        {
            int  x=6;
            int *xptr;        → * used as declaration of pointer variable.
            xptr = &x;
            printf(“Value of x= %d”, x);        //  Value of x = 6
             printf(“Value of x= %d”, *xptr);      // Value of x = 6
        }                              ↓
                                   * used as dereference operator.
```

Bad Pointer

- ✓ When the pointer is first declared it does not have a valid address. This pointer is uninitialized or bad and it is called bad pointer.
- ✓ The dereference operation on bad pointer is serious run time error. Thus, pointer must be assigned with a valid address before dereferencing it.

Eg.

```
int main()
    {
          int  x=6;
          int *xptr;
          printf(“Value of x= %d”, x);        //  Value of x = 6
          printf(“Value of x= %d”, *xptr); // Error!!  Bad Pointer
      }
```

void pointer

- ✓ A void pointer is special type of a pointer.
- ✓ It can point to variables of any data types.

Eg.

```
int main()
    {
        int a = 10;
        float b = 12.5;
        void *vptr;        ⟶        void pointer
        vptr = &a;     // points to address of int type variable
        printf("a = %d", ((int *)vptr));
        vptr = &b;     // points to address of floating type of variable.
        printf("a = %d", ((float *)vptr));
        return 0;
      }
```

NULL Pointer

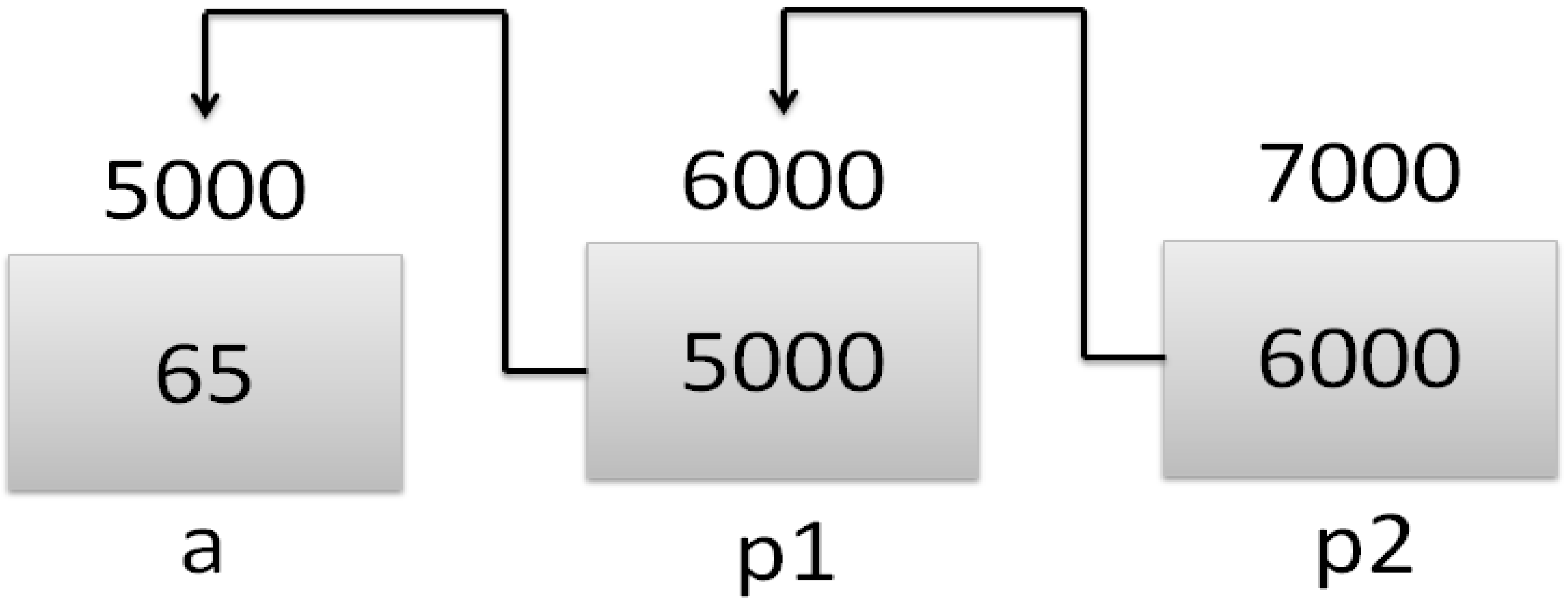
- ✓ A NULL pointer is special pointer value that points nowhere or nothing in memory address.
- ✓ We can define NULL pointer using predefined constant NULL.

Eg.

```
int *ptr = NULL;
```

Pointer to Pointer (double pointer)

- ✓ Pointers are used to store the address of other variables of similar data type. But if you want to store the address of a pointer variable, then you again need a pointer to store it. Thus, when one pointer variable stores the address of another pointer variable, it is known as Pointer to Pointer variable or Double Pointe.



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.

Example: WAP to swap two numbers using pass by value.

```
void  swap(int,int);
void main()
{
int a,b;
printf("Enter values of a and b");
scanf("%d %d", &a,&b);
swap(a,b);
printf("In Calling Function");
printf("a= %d, b=%d", a , b);
}
```

```
void swap(int a, int b){
int temp;
temp =a;
a=b;
b=temp;
printf("In called function");
printf("a= %d, b=%d", a, b);
}
```

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.

Example: WAP to swap two numbers using pass by reference.

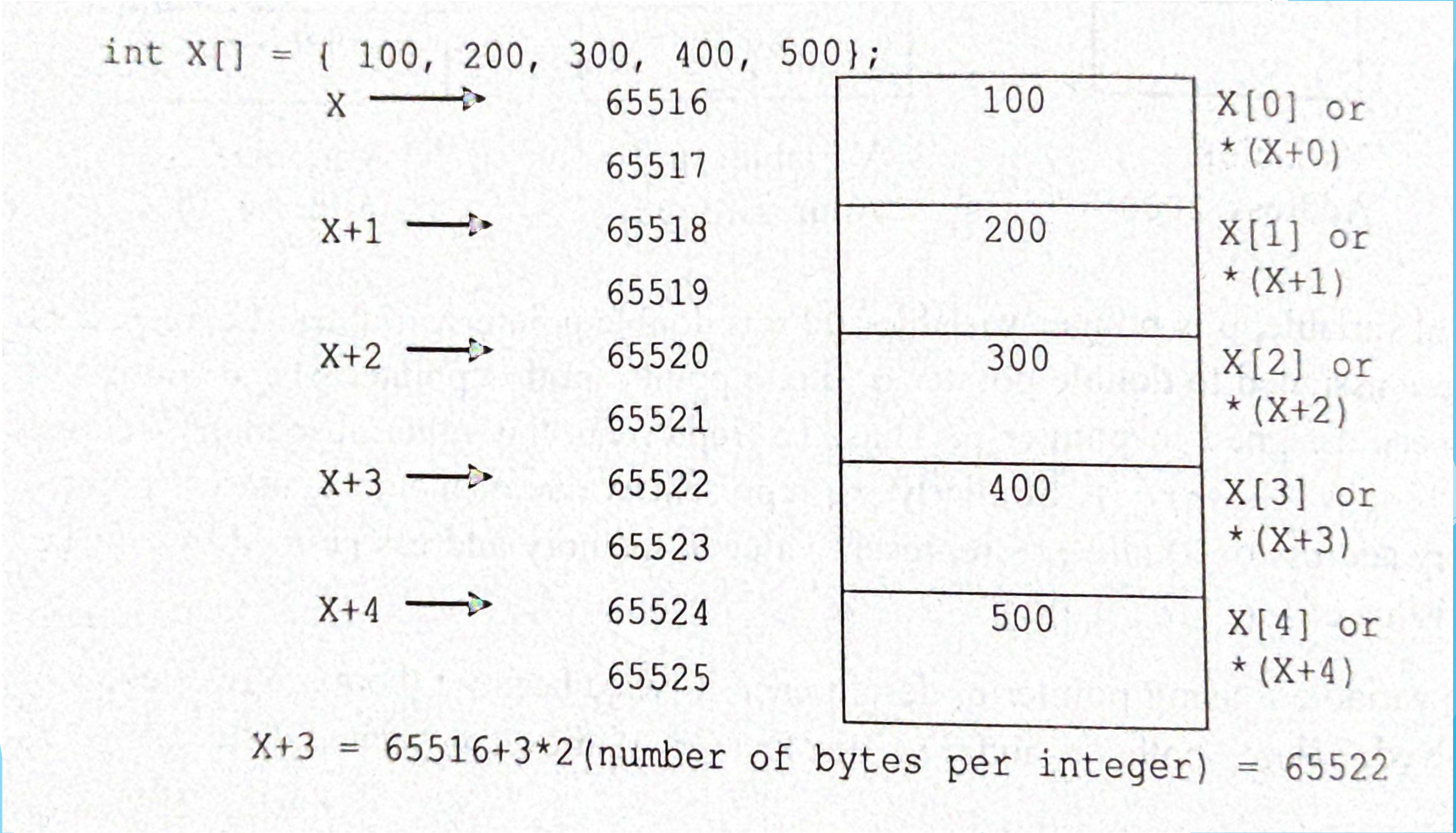
```
void swap(int*,int*);
void main()
{
int a,b;
printf(“Enter values of a and b");
scanf("%d %d", &a,&b);
swap(&a, &b);
printf(“In Calling Function”);
printf(“a= %d, b=%d”, a , b);
}
```

```
void swap(int *a, int *b){
int temp;
temp =*a;
*a=*b;
*b=temp;
printf(“In called function”);
printf(“a= %d, b=%d”, a, b);
}
```

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

1 D Array and Pointer

Eg. WAP to ask n number of elements in array and display using pointer notation.

```
void main()
{
    int num[100],i,n;
    printf("How many elements?\n");
    scanf("%d",&n);
    printf("Enter %d Elements",n);

    for(i=0; i<n; i++)
    {
        scanf("%d", (num + i));
    }
```

```
    printf("Elements of array are:\n");
    for(i=0; i<n; i++)
    {
        printf("%d\t",*(num + i));
    }

    getch();
}
```

2 D Array and Pointer

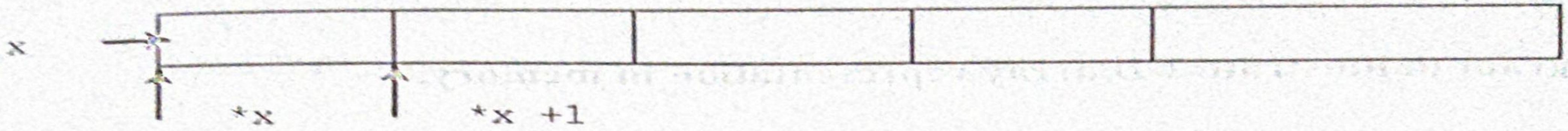
- ✓ The 2D array can be represented with an equivalent pointer notation like in 1D array.
- ✓ The 2D array is actually a collection of 1D array, each indicating a row.

Syntax for declaration of 2D array

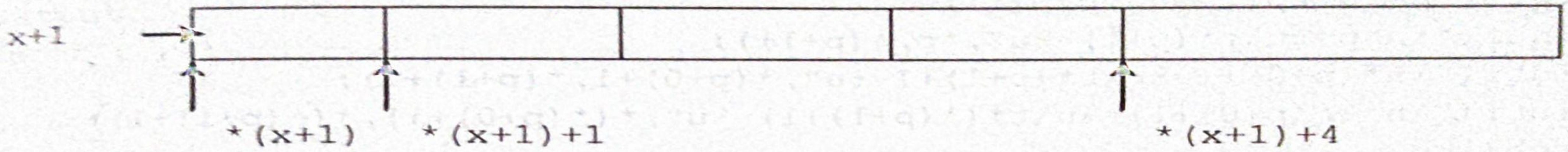
```
data_type (* pointer variable) [size];
```

Example

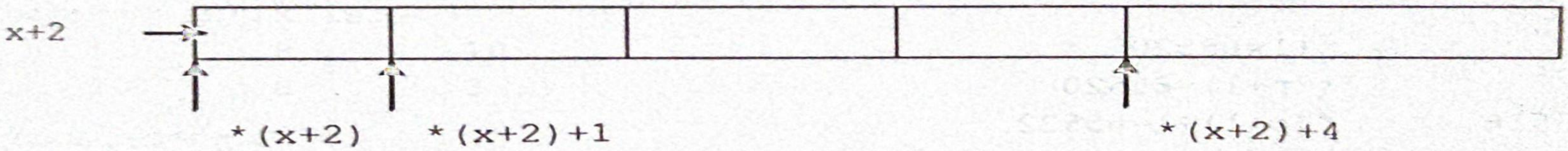
```
int (*Mat) [5];
          OR
int Mat[5][5];
```



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 ith row
`*(x+i)`	→ pointer to first element in the ith row
`*(x+i)+j`	→ pointer to jth element in the ith 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();
      }
```

THE END