GE3 COMPUTER SCIENCE

C AND C ++ LECTURE SERIES *FOR*

B.SC 3RD SEMESTER *BY*

SUBHADIP MUKHERJEE

DEPARTMENT OF COMPUTER SCIENCE

KHARAGPUR COLLEGE

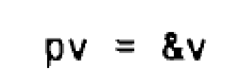
LECTURE 11

POINTERS

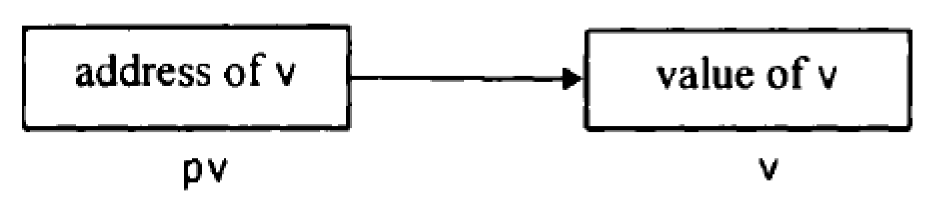
A ***pointer*** is a variable that represents the ***location*** (rather than the ***value)*** of a data item, such as a variable or an array element.

Pointers are also closely associated with arrays and therefore provide an alternate way to access individual array elements.

- let us assign the address of v to another variable, pv.
- pv is referred to as a ***pointer variable.***



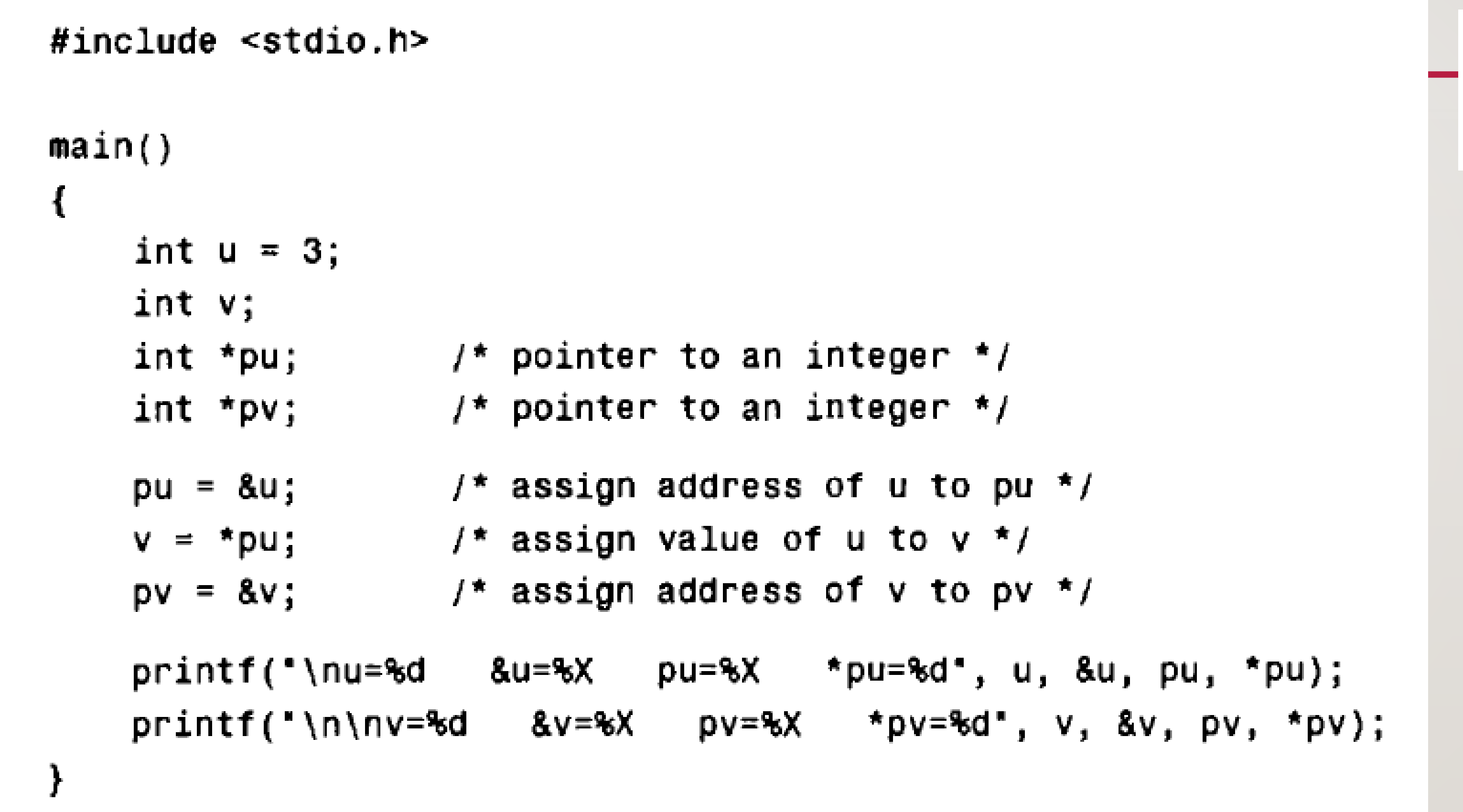
```
pv = &v
```



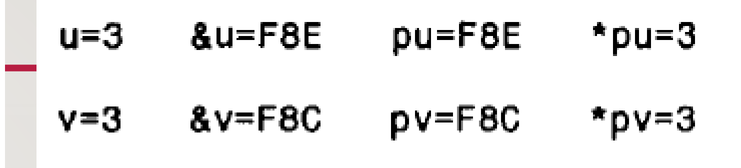
Therefore, ***pv** and **v** both represent the same data item

POINTERS

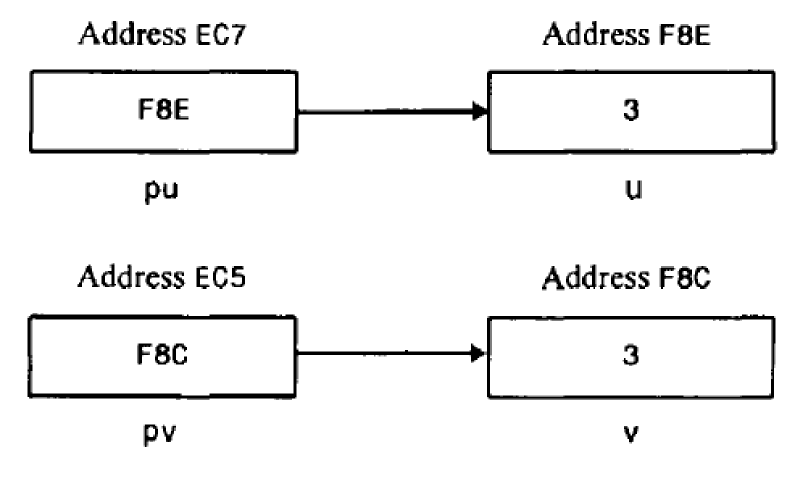
Example 1



```
#include <stdio.h>

main()
{
    int u = 3;
    int v;
    int *pu;        /* pointer to an integer */
    int *pv;        /* pointer to an integer */

    pu = &u;        /* assign address of u to pu */
    v = *pu;        /* assign value of u to v */
    pv = &v;        /* assign address of v to pv */

    printf("\nu=%d   &u=%X   pu=%X   *pu=%d", u, &u, pu, *pu);
    printf("\n\nv=%d   &v=%X   pv=%X   *pv=%d", v, &v, pv, *pv);
}
```

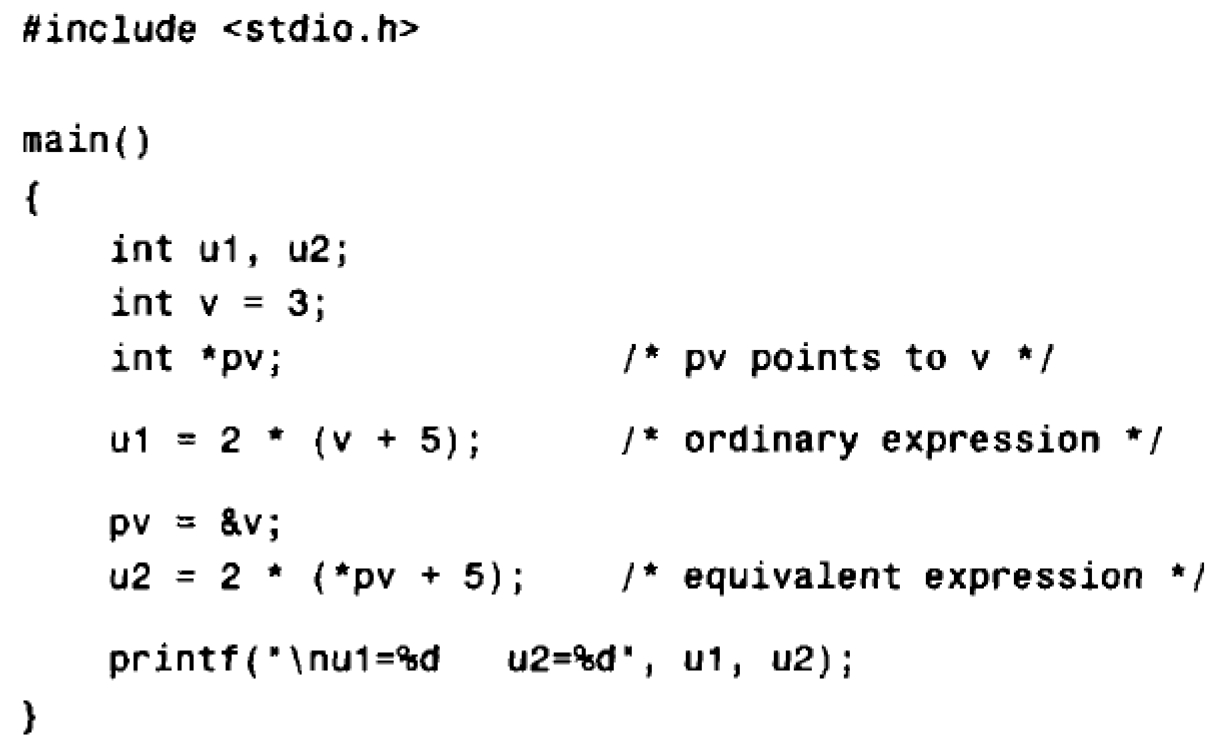


```
u=3    &u=F8E    pu=F8E    *pu=3

v=3    &v=F8C    pv=F8C    *pv=3
```

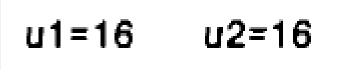


POINTERS

Example 2



```c
#include <stdio.h>

main()
{
    int u1, u2;
    int v = 3;
    int *pv;               /* pv points to v */

    u1 = 2 * (v + 5);      /* ordinary expression */

    pv = &v;
    u2 = 2 * (*pv + 5);    /* equivalent expression */

    printf("\nu1=%d   u2=%d", u1, u2);
}
```



```
u1=16   u2=16
```

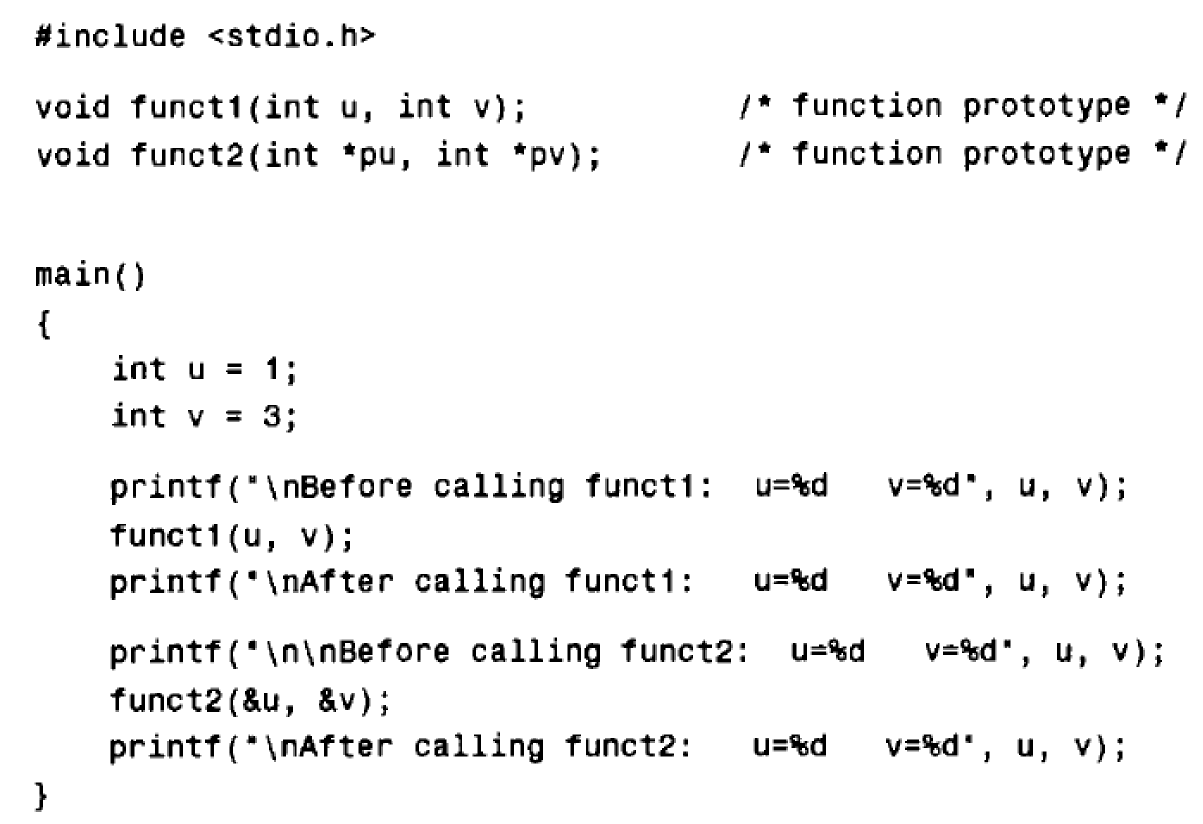
POINTERS

Passing pointer to an argument

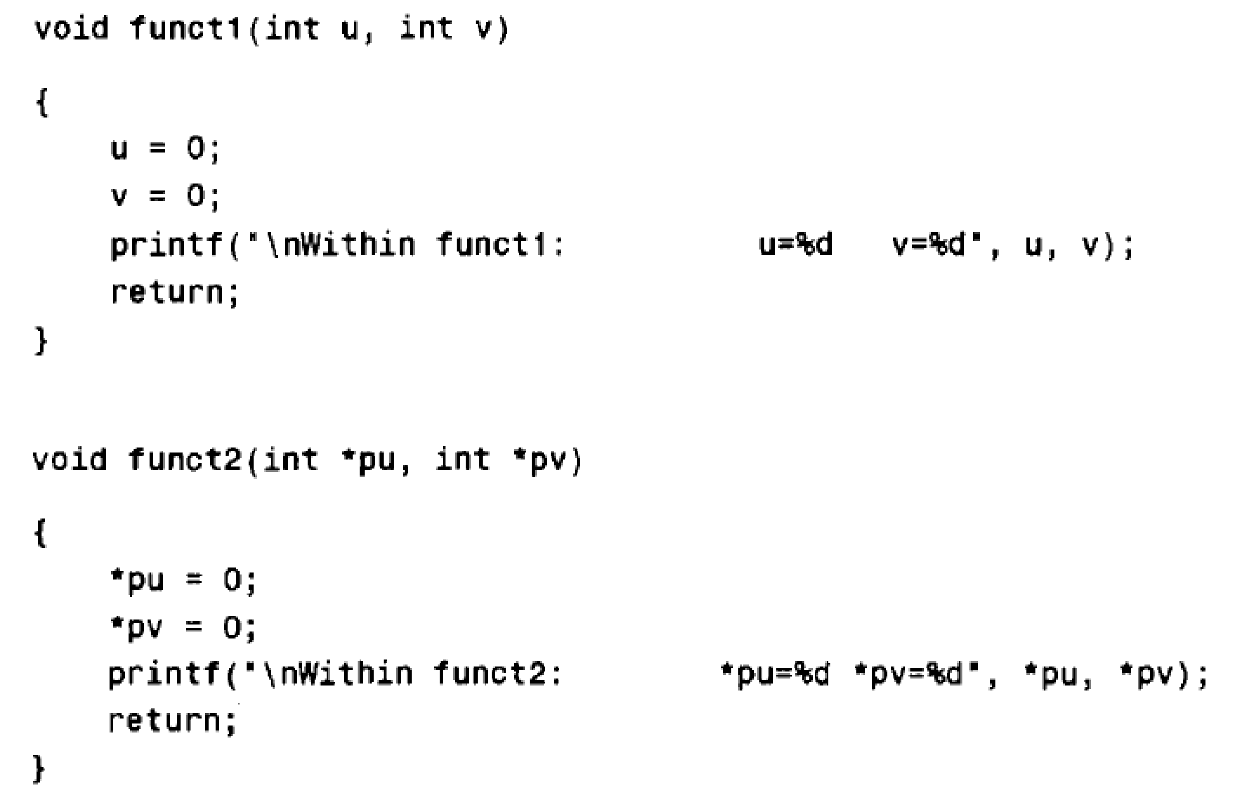
- Pointers are often passed to a function as arguments. This allows data items within the calling portion of the program to be accessed by the function, altered within the function, and then returned to the calling portion of the program in altered form. We refer to this use of pointers as passing arguments by ***reference*** (or by ***address*** or by ***location),***in contrast to passing arguments by ***value.***

POINTERS

Passing pointer to an argument (Continued)



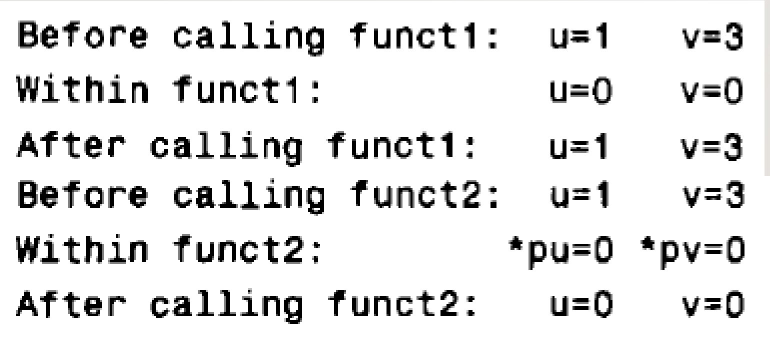
```
#include <stdio.h>

void funct1(int u, int v);             /* function prototype */
void funct2(int *pu, int *pv);         /* function prototype */


main()
{
    int u = 1;
    int v = 3;

    printf("\nBefore calling funct1:  u=%d   v=%d", u, v);
    funct1(u, v);
    printf("\nAfter calling funct1:   u=%d   v=%d", u, v);

    printf("\n\nBefore calling funct2:  u=%d   v=%d", u, v);
    funct2(&u, &v);
    printf("\nAfter calling funct2:   u=%d   v=%d", u, v);
}
```



```
void funct1(int u, int v)

{
    u = 0;
    v = 0;
    printf("\nWithin funct1:          u=%d   v=%d", u, v);
    return;
}


void funct2(int *pu, int *pv)

{
    *pu = 0;
    *pv = 0;
    printf("\nWithin funct2:        *pu=%d *pv=%d", *pu, *pv);
    return;
}
```

POINTERS

Passing pointer to an argument

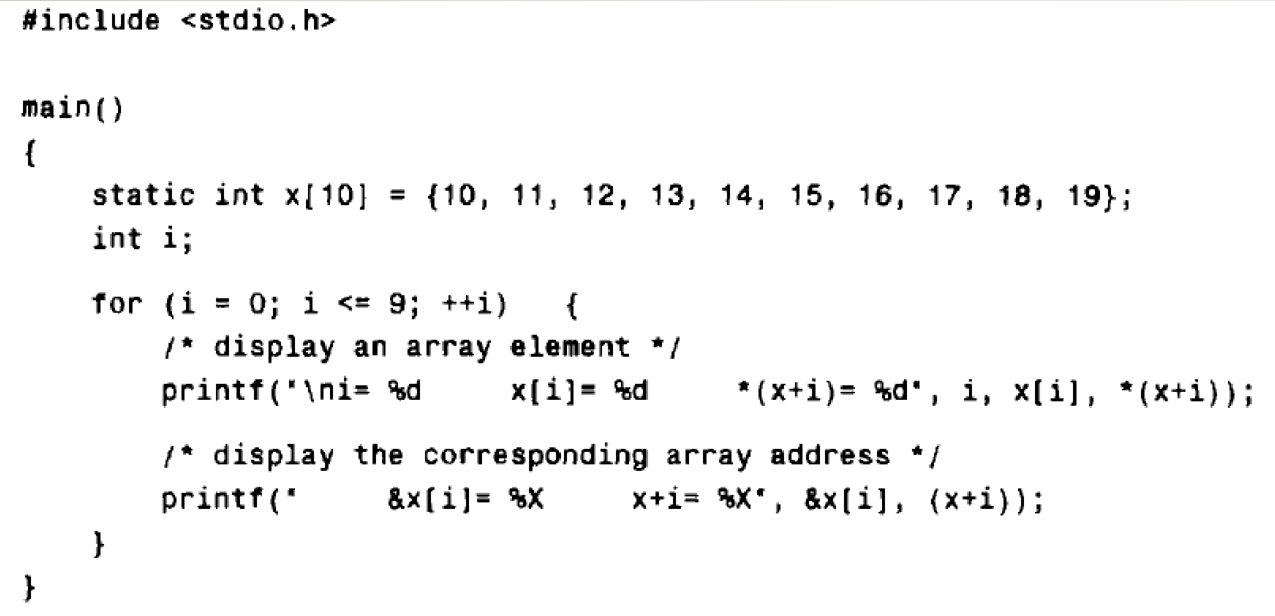


```
Before calling funct1:   u=1    v=3
Within funct1:           u=0    v=0
After calling funct1:    u=1    v=3
Before calling funct2:   u=1    v=3
Within funct2:         *pu=0  *pv=0
After calling funct2:    u=0    v=0
```

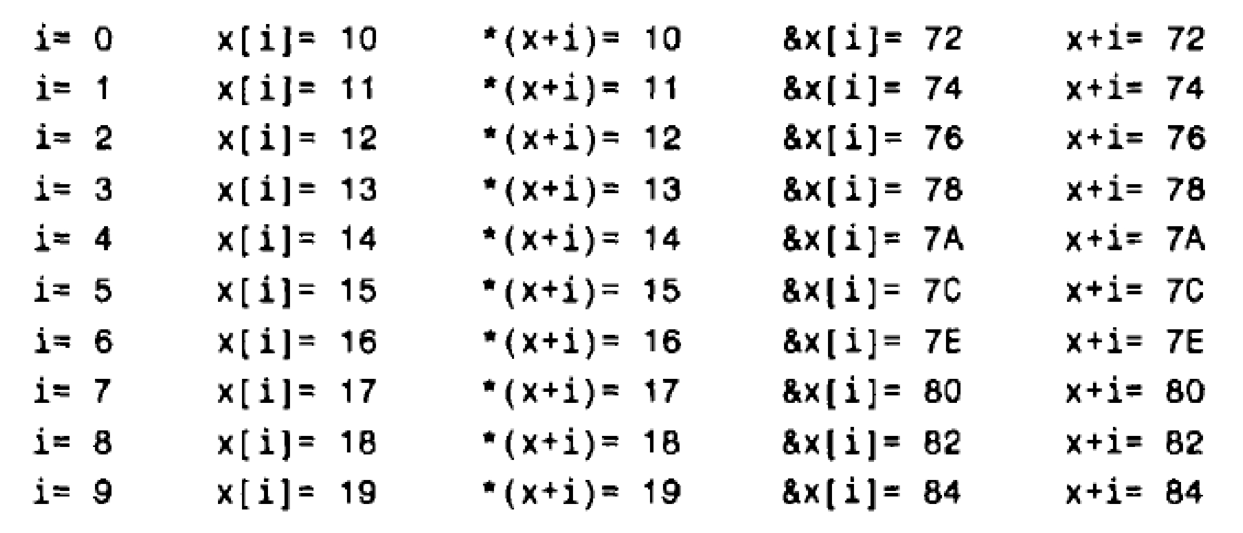
POINTERS

POINTERS AND ONE-DIMENSIONAL ARRAYS

- Recall that an array name is really a pointer to the first element in the array. Therefore, if **x** is a one dimensional array, then the address of the first array element can be expressed **as** either **&x [01]** or simply **as x.**



```
#include <stdio.h>

main()
{
    static int x[10] = {10, 11, 12, 13, 14, 15, 16, 17, 18, 19};
    int i;

    for (i = 0; i <= 9; ++i)    {
        /* display an array element */
        printf("\ni= %d     x[i]= %d     *(x+i)= %d", i, x[i], *(x+i));

        /* display the corresponding array address */
        printf("     &x[i]= %X     x+i= %X", &x[i], (x+i));
    }
}
```

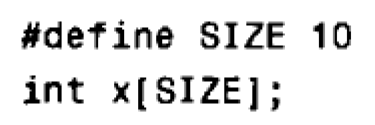


```
i= 0     x[i]= 10     *(x+i)= 10     &x[i]= 72     x+i= 72
i= 1     x[i]= 11     *(x+i)= 11     &x[i]= 74     x+i= 74
i= 2     x[i]= 12     *(x+i)= 12     &x[i]= 76     x+i= 76
i= 3     x[i]= 13     *(x+i)= 13     &x[i]= 78     x+i= 78
i= 4     x[i]= 14     *(x+i)= 14     &x[i]= 7A     x+i= 7A
i= 5     x[i]= 15     *(x+i)= 15     &x[i]= 7C     x+i= 7C
i= 6     x[i]= 16     *(x+i)= 16     &x[i]= 7E     x+i= 7E
i= 7     x[i]= 17     *(x+i)= 17     &x[i]= 80     x+i= 80
i= 8     x[i]= 18     *(x+i)= 18     &x[i]= 82     x+i= 82
i= 9     x[i]= 19     *(x+i)= 19     &x[i]= 84     x+i= 84
```

POINTERS

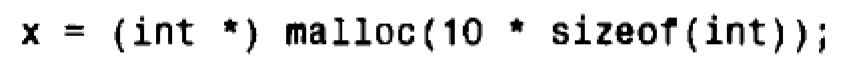
DYNAMIC MEMORY ALLOCATION

- The use of a pointer variable to represent an array requires some type of initial memory assignment before the array elements are processed. This is known as ***dynamic memory allocation***
- Suppose **`X`** is a one-dimensional, 10-element array of integers. It is possible to define **`X as`** a pointer variable rather than an array. Thus, we can write **int *x;** rather than **int x [lO] ;**
- **Or**



```
#define SIZE 10
int x[SIZE];
```

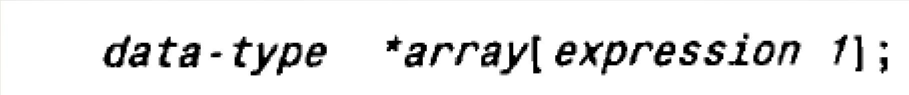
- To assign sufficient memory for **`x,`** we can make use of the library function **`malloc,`** as follows.



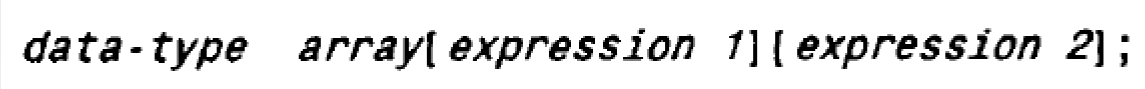
```
x = (int *) malloc(10 * sizeof(int));
```

POINTERS

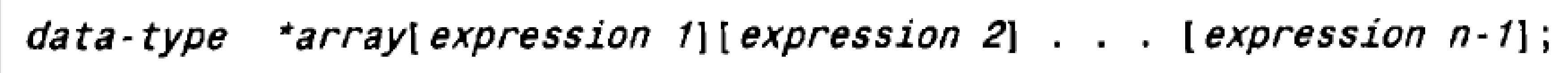
ARRAYS OF POINTERS



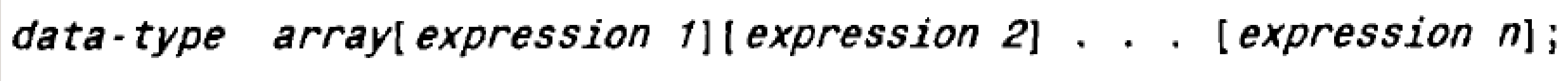
```
data-type  *array[expression 1];
```



```
data-type  array[expression 1][expression 2];
```



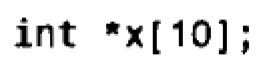
```
data-type  *array[expression 1][expression 2] . . . [expression n-1];
```



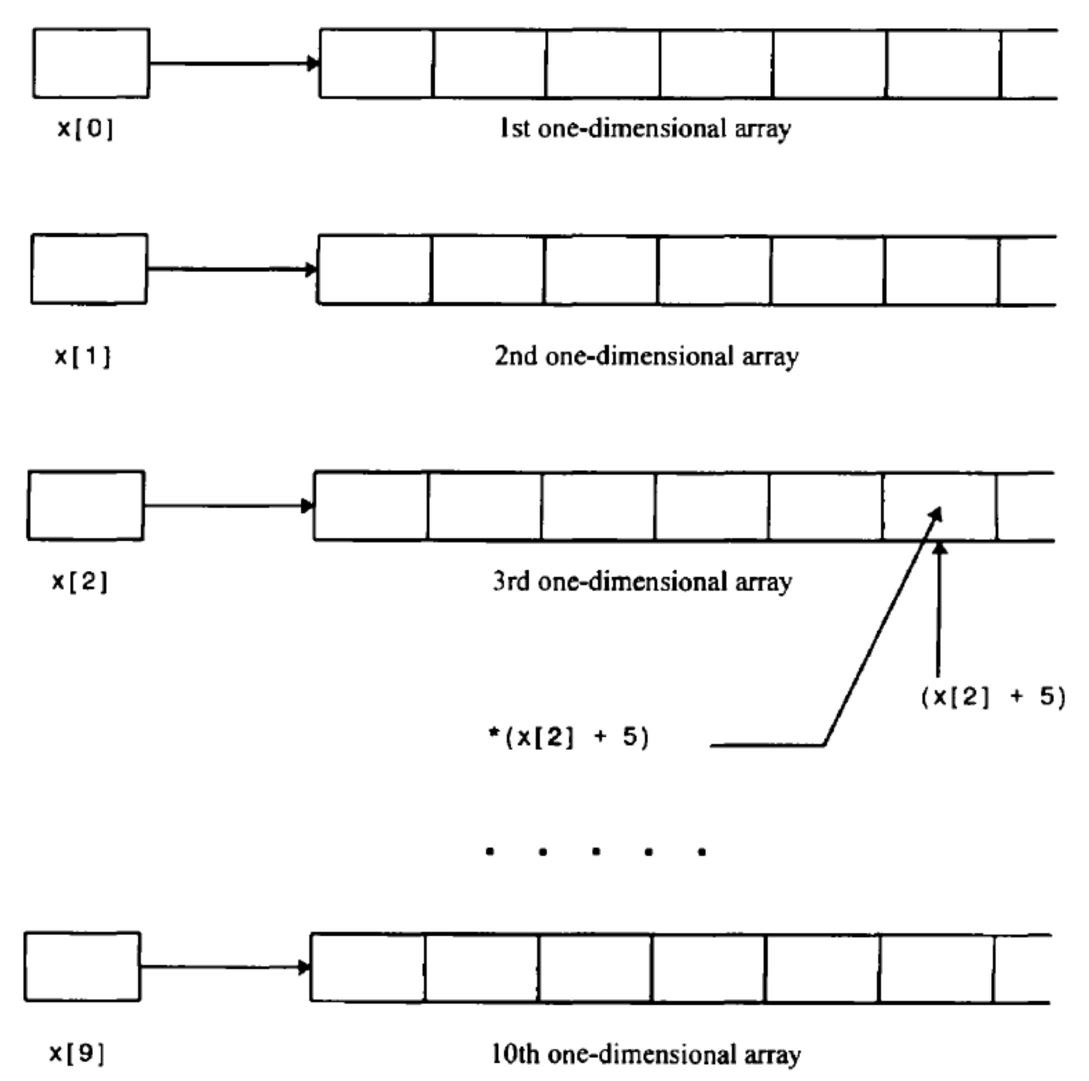
```
data-type  array[expression 1][expression 2] . . . [expression n];
```

POINTERS

ARRAYS OF POINTERS



```
int *x[10];
```


x[0]
1st one-dimensional array
x[1]
2nd one-dimensional array
x[2]
3rd one-dimensional array
(x[2] + 5)
*(x[2] + 5)
. . . . .
x[9]
10th one-dimensional array

COMPILE AND RUN A C CODE

Thank You

End of Lecture 11

Subhadip Mukherjee

Department of Computer Science

Kharagpur College

Kharagpur, India