MA 511: Computer Programming

Lecture 9: Pointers (Contd..)

http://www.iitg.ernet.in/psm/indexing_ma511/y10/index.html

Partha Sarathi Mandal

psm@iitg.ernet.ac.in

Dept. of Mathematics, IIT Guwahati

Semester 1, 2010-11

Passing Arguments to **Function**

- Passing **values** (call by value)
- Passing **pointers** (call by reference)
- Passing a **pointer to an array**
- Passing a **pointer to a character string**
- Passing a **pointer to a character**

Passing **values**

- Passing values is known as **call by value.**
- You actually pass a copy of the variable to the function.
- If the function modifies the copy, the original remains unaltered. Following example demonstrated **call by value.**

Example:

```
int func(int, int);
main() {
        int i=1, j;
        printf("i before call the func %d", i);
        j= func(i, i);
        printf("i after the func is executed  %d and j = %d. \n", i, j);
}
int func(int m, int n){
        int i;
        i = m + n;
        return i;
}
```

Output:
i before call the func 1 i after the func is executed 1 and j= 2.

Passing **pointers**

- This is known as **call by reference.**
- We do not pass the data to the function, instead we pass a ***pointer*** *to the data*.
- This means that if the function *alters the data*, the *original is altered*.
- Following example demonstrated **call by reference.**

Cont.. Passing **pointers**

Example:

```
void func(int*);
main() {
        int i=4;
        int *ptri;
        ptri = &i;
        printf("i before call the func %d\n", i);
        printf(" *ptri is %d\n", *ptri);
        func(ptri);
        printf(" i after call the func %d\n", i);
}
void func(int *p) {
        ++*p;                    /* Add 1 to the value */
        return;
}
```

Output:

```
i before call the func 4
 *ptri is 4
 i after call the func 5
```

Passing a pointer to an **array**

```
// how to access values of a array using pointers
#define SIZE_ARRAY 2
void func(int*);                                    // Function declaration
main(){
    int A[SIZE_ARRAY]={14, 16};                     // array declaration
    int count=0;
        for (count=0; count<SIZE_ARRAY; count++)
            printf(" A before call the func %d ptri = %x\n", A[count], A+count);
        func(A);                                    // Function call
        for (count=0; count<SIZE_ARRAY; count++)
            printf(" A after call the func %d. Ptri = %x\n", A[count],  A+count);
}
void func(int *ptr) {
        ++*ptr;                                     // Add 1 to the first element in the array
        ++*(ptr+1);                                 // And the second element
return;
}
```

Output:

```
i before call the func14 Ptri = 22ccc0
i before call the func16 Ptri = 22ccc4
 i after call the func 15 Ptri = 22ccc0
 i after call the func 17 Ptri = 22ccc4
```

Passing ptr to a character **string**

```
int func(char *ptrc);
main(){
    char array_s[10]="987654321";      // initialization
    func(array_s);                     // function call
    printf("%s, %x\n", array_s, array_s);
}
int func(char *array){
  printf("%s %x\n", array, array);
  array +=4;                           // Modify the pointer
  *array = 'x';                        // Modify the data pointed to  by 'array‘
}
```

Output:
987654321 22ccb0
9876x4321, 22ccb0

Passing pointer to a **character**

```
int func(char *ptrc);
main(){
      char achar = 'h';                          // initialization
      func(&achar);                              // call function
      printf("%c, %x\n", achar, &achar);
}
func(char *ptrc){
   printf("%c\n", *ptrc);
  *ptrc = 'x';                                   // Modify the data
}
```

Output:
h
x, 22ccc7

A function can pointer to an array

Example:

```
void  func(int *p);
main(){
    static int a[5]={10,20,30,40,50};
    func(a + 3);
}
void func(int *p){
    int i, sum = 0;
    for(i=0; i<2; i++)
            sum += *(p+i);
    printf("sum = %d", sum);
return;
}
```

Example:

```
void  func(int p[]);
main(){
    static int a[5]={10,20,30,40,50};
    func(a + 3);
}
void func(int p[]){
    int i, sum = 0;
    for(i=0; i<2; i++)
            sum += *(p+i);
    printf("sum = %d", sum);
return;
}
```

A function can pointer to a multidimensional array (static)

```
void fun_add(int a[2][3], int row_sum[2]){
	int  i, j;
	for(i=0; i<2; i++) {
		row_sum[i]=0;
		for(j=0; j<3; j++)
			row_sum[i] +=  a[i][j];
	}
}
void fun_print(int row_sum[2]){
	int  i;
	for(i=0; i<2; i++)
		printf("row_sum[%d] = %d", i, row_sum[i]);
}
main(){
	int row_sum[2]={0,0}, a[2][3]={10,20,30,40,50,60};
	fun_add(a, row_sum);
	fun_print(row_sum);
}
```

Pointer & multidimensional arrays (dynamic)

x: 2-D array having 10 rows 20 columns

declare as:

int (*x)[20]; a ptr to a group of contiguous one dimensional, 20-element integer array.

which is same as

int x[10][20];

Similarly:

int (*b)[20][30]; a ptr to a group of contiguous two dimensional, 20 X 30 integer arrays.

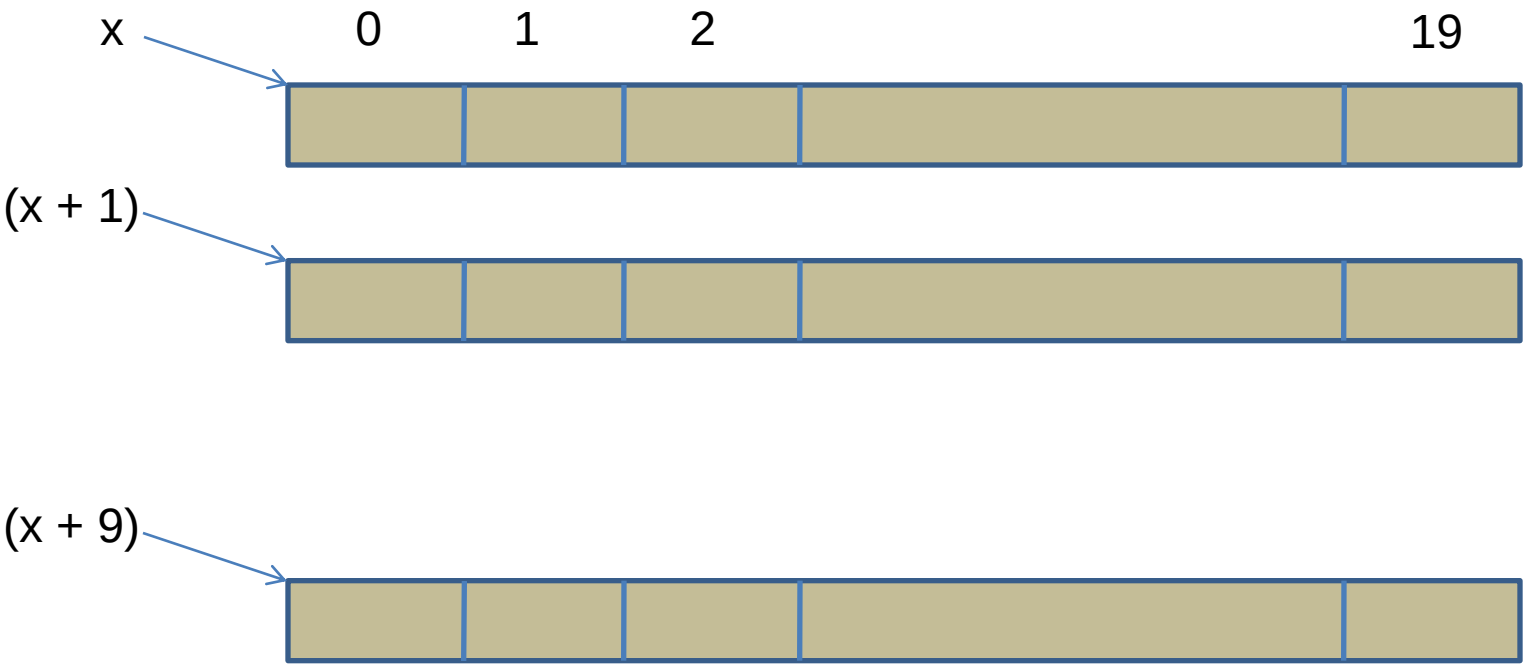
which is same as

int b[10][20][30];

Pointer & multidimensional arrays (dynamic)

int (*x)[20]; a ptr to a group of contiguous one dimensional, 20-element integer array.

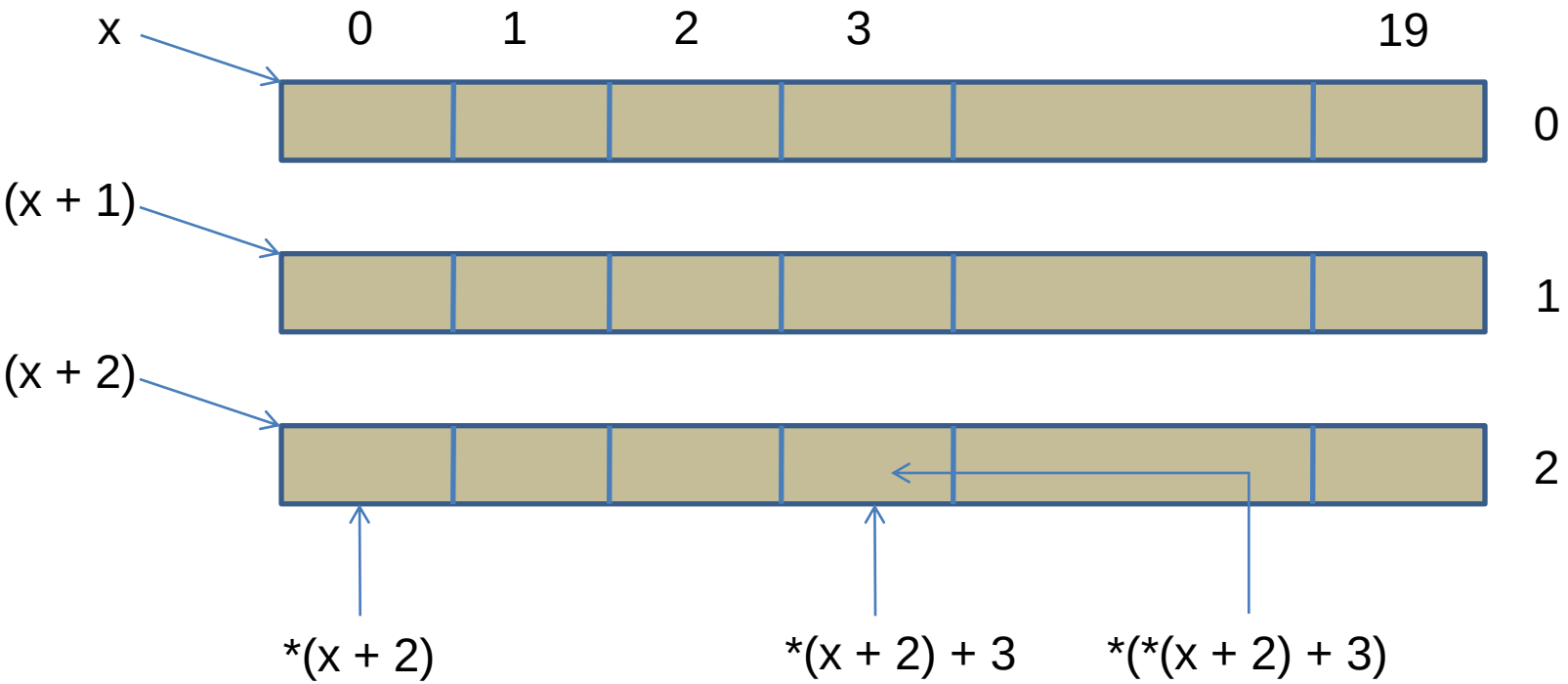
Or int x[10][20];



Pointer & multidimensional arrays (dynamic)

How to access the **item** in row 2 and column 3

x[2][3] or *(*(x+2) + 3)



Exercise

1. int X[8] ={10, 20, 30, 40, 50, 60, 70, 80}; What is
   a) X
   b) (x+2)
   c) *x
   d) *x+2
   e) *(x+2) ?

   a) Address of A[0]
   b) Address of A[2]
   c) 10
   d) 12
   e) 30

2. float table[2][3] = { {1.1, 1.2, 1.3}, {2.1, 2.2, 2.3} }; What is
   a) table,
   b) (table +1)
   c) *(table +1)
   d) *(table +1) + 1
   e) (*(table) +1)
   f) *(*(table +1)+1)
   g) *(*(table)+1)
   h) *(*(table +1))
   i) *(*(table) +1) + 1?

   a) Address of table[0][0]
   b) Address of 2nd row of the table
   c) Address of table[1][0]
   d) Address of table[1][1]
   e) Address of table[0][1]
   f) 2.2
   g) 1.2
   h) 2.1
   i) 2.2