CS 201
Array

Debzani Deb

Having trouble linking math.h?

- Link with the following option

  gcc –lm –o test test.o

Quick Review

```
int * ptr1, * ptr2;
int a[10];
ptr1 = &a[2];
ptr2 = a; // equivalent to ptr2 = &a[0];
```

- An array variable is actually a pointer to the first element of the array.
- ptr2 points to the first element of the array and get others by offset.
- Referring a[i] is same as referring *(a+i).

Memory Addresses

a	?	0
a+1	?	1
a+2	?	2
.	?	3
.	?	4	
	?	5
	?	6
	?	7

Array Elements as Function arguments

- If we want to print the ith element of the array x[i], then we can do the following:

  printf("%d\n", x[i]);

  scanf("%d", &x[i]);
- We can also pass array elements as arguments to functions that we write. For example:

  swap1(x[i], x[i+1]);

  swap2(&x[i], &x[i+1]);

Having arrays as function arguments

- Besides passing individual array elements to functions, we can write functions that take entire arrays as arguments.
- There are several ways of passing arrays to functions but in each case we only pass the **address of the array**.
- This is very similar to what we did during "passing variables by reference…"
- As we are not passing a copy of the array – any changes to the array made within the function will also effect the original array.
- When an array name with no subscript appears in the argument list of a function call, what is actually stored in the function's corresponding parameter is the address of the array. Example,

  int a[10];

  foo(a);

  foo(&a[0]); // same as above

Using arrays in formal parameter list

void foo(int x[10]); // sized array

- Store the address of the corresponding array argument (a in previous slide) to variable x and remember it as an array of 10 elements.

void foo(int x[]); // unsized array

- The length of the array is not specified. Since it is **not a copy** , the compiler does not need to allocate space for the array and therefore does not need to know the size of the array.
- With this, we can pass an array of any size to function foo.

void foo(int *x); // array pointer

- Function foo can take any integer array as argument.

C code Example

```
int main(void){
    int a[5]={1,2,3,4,5};
    int i;

    clear1(a,5);
    clear2(a,5);
    for(i=0; i<5; i=i+1)
          printf("%d ", a[i]);
    return 0;
}
```

```
void clear1(int x[], int size){
    int i;
    for(i=0; i<size; i=i+1)
          x[i] = 0;
}

void clear2(int *x, int size){
    int *p;
    for(p=x; p<(x+size); p=p+1)
     // for(p=&x[0]; p<&x[size]; p=p+1)
          *p = 0;
}
```


Passing array as function parameter

- Array names are pointers.
- The name of an array contain the address of its initial element, a = &a[0];
- Passing an array as function argument copies the value of the pointer to the formal array parameter and points to the same location.
- Where is the array actually located?
  - At the data area of the function that declared it.
  - For an array declared as formal parameter, space is allocated in the function data area only for the address of the initial array element of the array passed as actual argument.

Arrays as Input Arguments

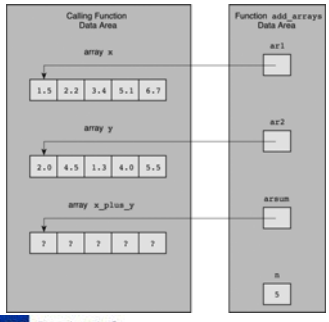
- ANSI C provides a qualifier that we can include in the declaration of the array formal parameter in order to notify the C complier that the array is only an input to the function and that the function does not intend to modify the array.
- This qualifier allows the compiler to mark as an error any attempt to change an array element within the function.

  void foo(const int x[], int size)
- The const tells C not to allow any element of the array x to change in the function foo.

Figure 8.9 Function to Add Two Arrays

```
1.  /*
2.   *  Adds corresponding elements of arrays ar1 and ar2, storing  the result in
3.   *  arsum. Processes first n elements only.
4.   *  Pre:  First n elements of ar1 and ar2 are defined. arsum's corresponding
5.   *        actual argument has a declared size >= n (n >= 0)
6.   */
7.  void
8.  add_arrays(const double ar1[],   /* input -                          */
9.             const double ar2[],   /*    arrays being added            */
10.            double       arsum[], /* output - sum of corresponding
11.                                      elements of ar1 and ar2          */
12.            int          n)       /* input -  number of element
13.                                      pairs summed                     */
14. {
15.     int i;
16.
17.     /* Adds corresponding elements of ar1 and ar2                     */
18.     for  (i = 0;  i < n;  ++i)
19.        arsum[i] = ar1[i] + ar2[i];
20. }
```


Figure 8.10 Function Data Areas for add_arrays(x, y, x_plus_y, 5);



Returning an Array from a function(1)

- Returning arrays should work too - right?

```
int a[5];
a = foo();
.....
int* foo() {
   int c[5] = {1,2,3,4,5};
   return c;
}
```

- Wrong! In C, it is not legal for a function's return type to be an array.

Returning an Array from a function(2)

- Type problem between int[] and int*
  - a is a constant pointer type, you can't change it.
- Array c is stored in function foo's stack frame.
  - Return just copies the pointer, not the array.
  - Memory where the array c resides may be overwritten once function foo finishes its execution.
- To do it properly
  - Pass an array as a function argument from your calling function and modify it inside the called function.

Partially Filled Arrays

- Frequently, a program will need to process many lists of similar data
- These list may not all be the same length
- In order to reuse an array for processing more than one data set, the programmer often declares an array large enough to hold the largest data set anticipated.
- This array can be used for processing shorter lists as well, provided that the program keeps track of how many array elements are actually in use.
- One way to do this is to have the original array size declared as the highest needed, and then store a sentinel value after the last value inputted.

Multidimensional Arrays

- A multidimensional array is an array with two or more dimensions.
- We will use two-dimensional arrays to represent tables of data, matrices, and other two-dimensional objects.
- Thus int x[3][3] would define a three by three matrix that holds integers.
- Initialization is bit different
  int x[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};
- Both indices starts at 0.
- Think of it as x[rows][cols].

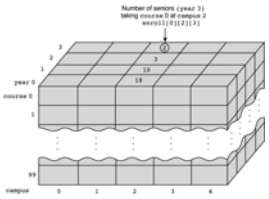
x[0][0]=1	x[0][1]=2	x[0][2]=3
x[1][0]=4	x[1][1]=5	x[1][2]=6
x[2][0]=7	x[2][1]=8	x[2][2]=9

Row Col inside a row

Three Dimensional Array

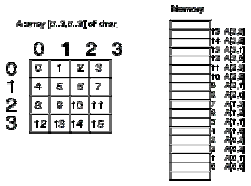
int enroll[100][5][4];

- The first dimension stores the course number. 100 of 2D matrices put one under another.
- The second dimension stores campus id. 5 vectors, each 4-element long.
- The last dimension stores the year number.



2D array in address space(1)

- How to store a multi-dimensional array into a one-dimensional memory space.
- Row major ordering assigns successive elements, moving across the rows and then down the columns, to successive memory locations.



Address of a[row][col] = Base_Address + (row * row_size + col)
Base_Address is the address of the first element of the array (A[0][0] in this case).

char *base = &A[0][0];

Referring A[i][j] is same as referring *(base+4*i+j);

2D array in address space(2)

int a[2][3]; // 2 rows and 3 cols

- Alternative interpretation: Three 1D array of 4 integers.
- The base address of the array is &a[0][0].
- The array name a by itself is equivalent to &a[0]. This time it is a pointer to an array of 3 elements.
- Different ways to access (i,j) th element:
  a[i][j]
  *(a[i] + j)
  (((a+i)) + j)
  *(&a[0][0] + 4*i + j)

a[1][2]	
a[1][1]	
a[1][0]	a+1
a[0][2]	
a[0][1]	
a[0][0]	a

Multi-dimensional Arrays.

Declaration	int a[N];	int b[M][N];	int c[L][M][N];
Array element	a[i]	b[i][j]	c[i][j][k]
Pointer to element	a	b[i]	c[i][j]
Pointer to 1D array of N element		b	c[i]
Pointer to 2D array of M*N element			c

Passing Multidimensional array (1)

```
a[7][9][2] = {0};
...
sum(arr,7);
...
int sum(int a[][9][2], int asize){
   int  i, j, k, sum = 0
        for( i = 0; i<asize; ++i) {
                for( j = 0; j<9; ++j) {
                        for( k = 0; k<2; ++k) {
                                sum += a[i][j][k];
   return sum;
}
```

For a multidimensional array we must declare all but the first dimension. Only 7 can be omitted.

Passing Multidimensional array (1)

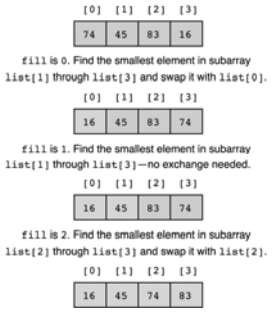
- For correct calculation of addresses, the array size must be specified for every dimension except the first one.
- Why?

Address of a[row][col] = b_add + (row * row_size + col)

Sorting an Array – Selection Sort

- Similar to sorting cards.

Figure 8.16 Trace of Selection Sort




```
/*
 *  Finds the position of the smallest element in the subarray
 *  list[first] through list[last].
 *  Pre:  first < last and elements 0 through last of array list are defined.
 *  Post: Returns the subscript k of the smallest element in the subarray;
 *        i.e., list[k] <= list[i] for all i in the subarray
 */
int get_min_range(int list[], int first, int last);


/*
 *  Sorts the data in array list
 *  Pre:  first n elements of list are defined and n >= 0
 */
void
select_sort(int list[],    /* input/output - array being sorted                */
            int n)         /* input - number of elements to sort               */
{
      int fill,            /* first element in unsorted subarray               */
          temp,            /* temporary storage                                */
          index_of_min;    /* subscript of next smallest element               */

      for (fill = 0; fill < n-1; ++fill) {
          /* Find position of smallest element in unsorted subarray */
          index_of_min = get_min_range(list, fill, n-1);

          /* Exchange elements at fill and index_of_min */
          if (fill != index_of_min) {
              temp = list[index_of_min];
              list[index_of_min] = list[fill];
              list[fill] = temp;
          }
      }
}
```

Common Programming Errors

- The most common error in using arrays is a subscript-range error.
- An out-of-range reference occurs when the subscript value is outside the range specified by the array declaration.
  - In some situations, **no run-time error message will be produced** – the program will simply produce incorrect results.
  - Other times, you may get a runtime error like "segmentation fault" or "bus error"
- Remember how to pass arrays to functions.
- Remember that the first index of the array is **0**