Lecture 6+7 Arrays

**Concept**

* Array is a collection of data values (elements) of the same data type
* Array elements are organized in linear order and stored in contiguous memory locations one after another.

**Declaring an Array**

Declare as: data\_type array\_name [length];

* The address of an array is the address of its first element
* As long as you assign one value to an array, the rest of them will be assigned to 0

**Initializing an array**

Int a[5]; //no initialization

Int a [5] = {1,2,3,4,5} //each array element gets a value

Int a[5] ={4, 1, 0} //first three elements get values the rest are 0

Int a[5] = {0} //all elements are set to 0

Int a[] = {1,2,3,4,5}//auto declare length 5 and initialization

**Creating arrays by dynamic method**

* Arrays created by declaration method are static arrays. Memory blocks of arrays are allocated in the stack region if they are declared in functions, or in data region if they are declared outside all functions
* Arrays can also be created by using the dynamic method, namely using the stdlib function malloc() function to allocate memory space for an array
* Memory block of dynamic arrays are allocated in the heap region at runtime

Example

Int length = 5;

Int \*a = (int\*) malloc(sizeof(int) \*length);

//this requests and allocates a memory block of 20 bytes in the heap region

Free(a); //release the memory block of a

**Array operations by pointers**

* Arrays can be operated on efficiently by using pointers. If pointer ptr holds the address of an array element, then ptr++ represents the address of the successive element.

Example

Int a[10];

Int \*ptr = &a[0]; //or int \*ptr = a;

Ptr++;

Printf(“The value of the second element in the array is %d”, \*ptr);

**Example of swapping array values using pointers**

T = \*(x+i);

\*(x+i) = \*(x+k);

\*(x+k) = t;

**Array of pointers**

* C allows array of pointers, where each element is a pointer

Example

Int \*ptr[5]; // declare an array of 5 int type pointer values (addresses of int data)

Int p=1,q=2,r=3,s=4,t=5;

Ptr[0] = &p;

Ptr[1]=&q;

Ptr[2]=&r;

Ptr[3]=&s;

Ptr[4]=&t;

**Array pointer**

* An array pointer is a pointer pointing to an array data object
  + An array can be viewed as a derived data type.
  + An array pointer can be declared by syntax data\_type (\*ptr\_name)[k];
  + Then ptr\_name is a pointer to an array of type data\_type of length k

Example

Int a[5], \*p;

Int (\*ptr)[5];//different from int \*ptr[5];

P=&a[0] //p points to a[0], p++ will increase by sizeof(int)

Ptr = &a; //ptr and p has the same value, but ptr++ will increase by sizeof(int)\*5

\*ptr points a[0]

\*ptr + I points to a[i]

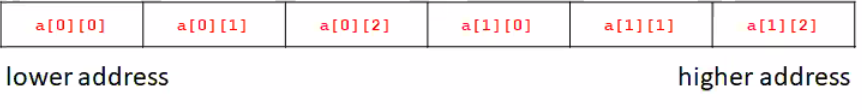
\*(\*ptr+i) gives a[i]

**Two dimensional arrays**

* A two-dimensional array is specified using two subscripts where the first subscript denotes row and the second denotes column
* Declared as:

Data\_type array\_name [row\_size][row\_size]

* Holds in memory in *row major order*, meaning it stores a whole row consecutively in memory and then moves to the next



**Using pointers with 2D arrays**

Syntax:

Int a[2][3];

Int \*p = &a[0][0];

**Row pointers**

* A row pointers is an array pointer pointing to a row of 2D array

Example:

Int mat[2][3] = {{1,2,3},{4,5,6}};

//mat[i] represents a row pointer pointing to row i

Int (\*ptr)[3]; //this declares array pointer ptr of type int[3]

Ptr = mat; //ptr points to the first row

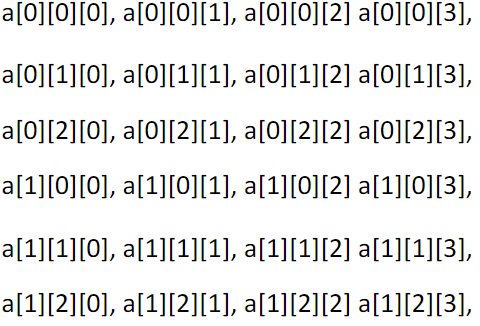
Ptr++; // now ptr points to the second row

//2D array elements can be represented by row pointers

Mat[i][j] <-> \*(\*(mat+i)+j) <-> \*(mat[i]+j);

**Multi dimensional arrays**

* Declared the same as a 2D array, C allows for n dimensions
* Elements are ordered by lexical-order of index



**Applications of arrays**

1. Arrays are commonly used to store a collection of data of the same type, where the maximum number of data items are known and random access are often used in the application.
2. Arrays are widely used to implement mathematical vectors, matrices and other kinds of rectangular tables
3. Arrays are also used to implement other data structures like string, queues, stack, hash tables and heaps
4. Arrays are used in databases to store data tables

**Array data structure operations**

1. Traversal: access each data element once
2. Searching an element by a search key
3. Inserting an element at a specific position
4. Deleting an element at a specific position
5. Merging two arrays
6. Sorting data elements in ascending or descending order