Lecture 7 : Arrays and Pointers

CST238 – Intro to Data Structures
YoungJoon Byun
ITCD

Lecture Objectives

- After completion of this lecture, you will be able to

Dynamic Allocation with **new** (reminder)

```
type * ptr = new type [capacity];
// capacity is an integer.
// It requests a block of memory dynamically.
// Then, assign the starting address of the allocated
// memory to a pointer
```

- Example

```
int n;
int * arrayPtr;
cin >> n;   // e.g., n has 6.
arrayPtr = new int[n];
```

Access an element of dynamic array

- Example

```
1. int * arrayPtr = new int[3];
2. arrayPtr[0] = 10;
3. *(arrayPtr+1) = 20;
4. *(arrayPtr+2) = 30;
5. for (int i=0; i < 3; i++)
6. {
7.   cout << *(arrayPtr) << endl;
8.   arrayPtr++;
9. }
```


Array of Pointers

- Pointers (as other variables) can be stored in an array
- Example
  - char * months [3] = {“Feb”, “Mar”, “Apr”};
  - int * p[100];

Example – Array of Pointers

```
1.  #include <iostream>
2.  using namespace std;
3.  int main() {
4.      int i, j;
5.      double * arrayPtr[10];
6.      for (i = 0; i < 10; i++) {
7.          arrayPtr[i] = new double [5];
8.          for (j = 0; j < 5; j++) {
9.              *(arrayPtr[i]+j) = j;
10.         }
11.     }
12. }
```

Chapter 3: Data Structures and Abstract Data Types

- 3.1 Data Structures, Abstract Data Types and Implementations
- 3.2 Static Arrays
- 3.3 Multidimensional Arrays
- 3.4 Dynamic Arrays
- **3.5 C-Style Structs (skip)**
- 3.6 Procedural Programming

Array Access through a Pointer

- In general, it is faster but somewhat harder to understand.
- Example
  1. int a[10];
  2. int * pa;
  3. pa = &a[0];
  4. *pa = 100;
  5. *(pa+1) refers to the contents of a[1]
  6. pa+i is the address of a[i],
  7. *(pa+i) is the content of a[i].

Array Indexing and Pointer Access

- There's close correspondence between array indexing and pointer arithmetic.
  - a[i] can be written *(a+i)
- In evaluating a[i], a C++ compiler converts it to *(a+i) immediately.
  - &a[i] and a+i are also identical.
- One difference
  - Array name is the location of the first element.
    - Thus, array name is a constant.
  - But a pointer is a variable.

Array name is a constant.

- int ints[100]
- int * p;
- int i;
- ints = NULL; // NO
- ints = &i; // NO
- ints++; // NO
- p = ints; // OK
- p++; // OK
- p = NULL; // OK
- p = &i; // OK
- foo(ints);

Array Parameter

- Caller: foo(ints);
- Callee: The following code are the exact same.

```
1. void foo(int arrayParam[ ]) {
2.     arrayParam = NULL;
3.     …
4. }

1. void foo(int * arrayParam) {
2.     arrayParam = NULL; // ditto
3.     …
4. }
```


Array Example – strcpy () (1 of 5)

```
1.  /* array index version 1 */
2.  void strcpy1(char dest[ ], char source[ ])
3.  {
4.      int i = 0;
5.      while (1) {
6.          dest[i] = source[i];
7.          if (dest[i] == '\0')
8.              break;
9.          i++;
10.     }
11. }
```

Array Example – strcpy () (2 of 5)

```
1. /* array index version 2
2. /* Move the assignment into the test */
3. void strcpy2(char dest[ ], char source[ ])
4. {
5.     int i = 0;
6.     while ((dest[i] = source[i]) != '\0') {
7.         i++;
8.     }
9. }
```


Array Example – strcpy () (3 of 5)

```
1. /* pointer version 1 */
2. void strcpy3(char * dest, char * source)
3. {
4.     while ((*dest = *source) != '\0') {
5.         dest++;
6.         source++;
7.     }
8. }
```


Array Example – strcpy () (4 of 5)

```
1. /* pointer version 2 */
2. /* Relies on the precedence of * and ++. */
3. void strcpy4(char * dest, char * source)
4. {
5.     while ((*dest++ = *source++) != '\0') ;
6. }
```

Array Example – strcpy () (5 of 5)

```
1. /* pointer version 3 */
2. /* '\0' is equivalent to FALSE */
3. void strcpy5(char * dest, char * source)
4. {
5.     while (*dest++ = *source++) ;
6. }
```


Pointer Arithmetic – Question

- What is this function for?

```
1. int guessWhat (char * s)
2. {
3.     char * p = s;
4.     while (*p != '\0')
5.         p++;
6.     return p – s;
7. }
```


Summary

- Arrays and pointers
- Reference
  - Essential C: http://cslibrary.stanford.edu/101/EssentialC.pdf