

Strings , arrays and functions, searching arrays

Computer Programming 143 – Lecture 14

Arrays II

Faculty of Engineering





6.5 Store and Manipulation of Strings

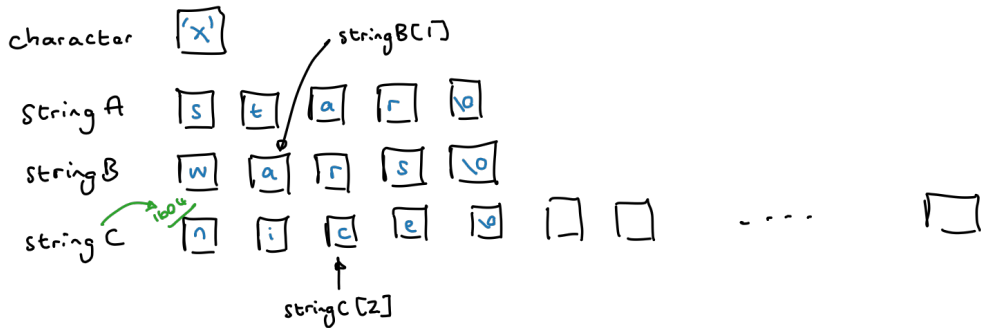
Character arrays

```
char str1[] = "first";
```

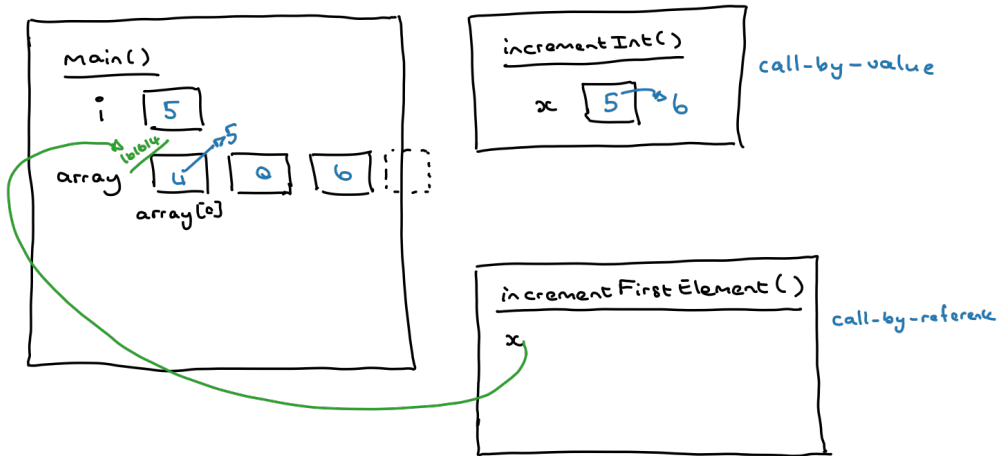
- String "first" is really a static array of characters
- Character arrays can be initialized using string literals
 - Null character '\0' terminates strings
 - **str1** actually has 6 elements
 - It is equivalent to

```
char str1[] = { 'f', 'i', 'r', 's', 't', '\0'};
```

- Can access individual characters
 - **str1[3]** is character 's'



Passing arrays to functions



6.7 Passing Arrays to Functions

Function prototype

- Prototype for a function that takes an array as argument:

```
void myFunction( int b[], int arraySize );
```

- Specifies that the first argument of function **myFunction** is an array of integers. The second argument is the size of the array.
- To prevent the function from modifying the array, use the keyword **const**:

```
void myFunction( const int b[], int arraySize );
```

- See program listing in p. 318 in Deitel & Deitel

6.7 Passing Arrays to Functions

Passing array elements

- Passed by call-by-value
- Pass subscripted name (i.e., **myArray[3]**) to function
- Value of the element is copied into the parameter of the function
- Original element in array is unaffected by function

6.10 Searching Arrays

Introduction

- When working with large amounts of data
- Search to see if it can match one of the array values to a **key value**
- Two searching techniques
 - Linear search
 - Binary search

6.10 Searching Arrays

Binary search

- Only for arrays **sorted** by key
- Compares **middle** element with **key**
 - If equal, match found
 - If **key** < **middle**, looks further in lower half of array
 - If **key** > **middle**, looks further in upper half of array
 - Repeat

6.10 Binary Search

searchKey = 11

c[0]	1	← low = 0
c[1]	3	
c[2]	4	
c[3]	7	
c[4]	11	
c[5]	12	
c[6]	13	
c[7]	16	
c[8]	23	
c[9]	25	
c[10]	31	
c[11]	36	
c[12]	37	
c[13]	40	
c[14]	41	
c[15]	47	← high = 15

6.10 Binary Search

searchKey = 11

c[0]	1	← low = 0
c[1]	3	
c[2]	4	
c[3]	7	
c[4]	11	
c[5]	12	
c[6]	13	
c[7]	16	← middle = (high + low) / 2 = 7
c[8]	23	
c[9]	25	
c[10]	31	
c[11]	36	
c[12]	37	
c[13]	40	
c[14]	41	
c[15]	47	← high = 15

6.10 Binary Search

c[0]	1	← low = 0	searchKey = 11
c[1]	3		
c[2]	4		
c[3]	7		
c[4]	11		
c[5]	12		
c[6]	13		
c[7]	16	← middle = 7	
c[8]	23	searchKey < c[middle]	
c[9]	25		
c[10]	31		
c[11]	36		
c[12]	37		
c[13]	40		
c[14]	41		
c[15]	47	← high = 15	

6.10 Binary Search

c[0]	1	← low = 0	searchKey = 11
c[1]	3		
c[2]	4		
c[3]	7		
c[4]	11		
c[5]	12		
c[6]	13	← high = middle - 1 = 6	
c[7]	16	← middle = 7	
c[8]	23		
c[9]	25		
c[10]	31		
c[11]	36		
c[12]	37		
c[13]	40		
c[14]	41		
c[15]	47		

6.10 Binary Search

c[0]	1	← low = 0	searchKey = 11
c[1]	3		
c[2]	4		
c[3]	7	← middle = (high + low) / 2 = 3	
c[4]	11		
c[5]	12		
c[6]	13	← high = 6	
c[7]	16		
c[8]	23		
c[9]	25		
c[10]	31		
c[11]	36		
c[12]	37		
c[13]	40		
c[14]	41		
c[15]	47		

6.10 Binary Search

c[0]	1	← low = 0	searchKey = 11
c[1]	3		
c[2]	4		
c[3]	7	← middle = 3	
c[4]	11		
c[5]	12		
c[6]	13	← high = 6	
c[7]	16		
c[8]	23		
c[9]	25		searchKey > c[middle]
c[10]	31		
c[11]	36		
c[12]	37		
c[13]	40		
c[14]	41		
c[15]	47		

6.10 Binary Search

searchKey = 11

c[0]	1	
c[1]	3	
c[2]	4	
c[3]	7	← middle = 3
c[4]	11	← low = middle + 1 = 4
c[5]	12	
c[6]	13	← high = 6
c[7]	16	
c[8]	23	
c[9]	25	
c[10]	31	
c[11]	36	
c[12]	37	
c[13]	40	
c[14]	41	
c[15]	47	

6.10 Binary Search

searchKey = 11

c[0]	1	
c[1]	3	
c[2]	4	
c[3]	7	
c[4]	11	← low = 4
c[5]	12	← middle = (high + low) / 2 = 5
c[6]	13	← high = 6
c[7]	16	
c[8]	23	
c[9]	25	
c[10]	31	
c[11]	36	
c[12]	37	
c[13]	40	
c[14]	41	
c[15]	47	

6.10 Binary Search

```
searchKey = 11
```

c[0]	1
c[1]	3
c[2]	4
c[3]	7
c[4]	11
c[5]	12
c[6]	13
c[7]	16
c[8]	23
c[9]	25
c[10]	31
c[11]	36
c[12]	37
c[13]	40
c[14]	41
c[15]	47

low = 4

```
middle = 5
```

high = 6

```
searchKey < c[ middle ]
```

6.10 Binary Search

searchKey = 11

c[0]	1	
c[1]	3	
c[2]	4	
c[3]	7	
c[4]	11	$\leftarrow \text{high} = \text{middle} - 1 = 4$
c[5]	12	$\leftarrow \text{low} = 4$
c[6]	13	$\leftarrow \text{middle} = 5$
c[7]	16	
c[8]	23	
c[9]	25	
c[10]	31	
c[11]	36	
c[12]	37	
c[13]	40	
c[14]	41	
c[15]	47	

6.10 Binary Search

c[0]	1
c[1]	3
c[2]	4
c[3]	7
c[4]	11
c[5]	12
c[6]	13
c[7]	16
c[8]	23
c[9]	25
c[10]	31
c[11]	36
c[12]	37
c[13]	40
c[14]	41
c[15]	47

searchKey = 11

high = 4

low = 4

middle = (high + low) / 2 = 4

6.10 Binary Search

c[0]	1
c[1]	3
c[2]	4
c[3]	7
c[4]	11
c[5]	12
c[6]	13
c[7]	16
c[8]	23
c[9]	25
c[10]	31
c[11]	36
c[12]	37
c[13]	40
c[14]	41
c[15]	47

searchKey = 11

high = 4
low = 4
middle = 4

searchKey = c[middle]
return middle

Example: binary Search

```
/* function to perform binary search of an array */
int binarySearch( const int b[], int searchKey, int low, int high ) {
    int middle;

    while (low <= high) {

        middle = ( low + high ) / 2; /* get middle element*/

        if ( searchKey == b[ middle ] ) {
            return middle;
        } else if ( searchKey < b[ middle ] ) {
            high = middle - 1; /* search low end of array */
        } else {
            low = middle + 1; /* search high end of array */
        }
    }
    return -1; /* searchKey not found */
}
```

See Fig. 6.15 in Deitel & Deitel for an example of binary search

How many steps does the search take in the worst case?

An array with 32 elements