

CS161: Introduction to Computer Science I

Week 8 – Arrays

12/2020

What is in CS161 today?

□ Arrays in C++

- Introduction
- Declaring Arrays
- Example of using Arrays

□ Strings

- What is a string in C++?
- How can I define strings?
- How can I read and write strings?
- Comparing and copying strings
- Accessing single characters in a string
- Write a program with strings

□ Arrays

- Structures of **related data items**
- **Static entity** (same size throughout program)

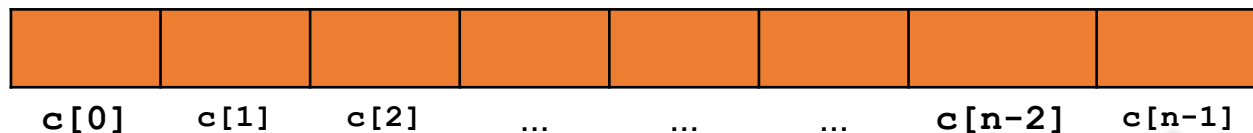
□ Array

- Consecutive group of memory locations
- Same name and type (**int**, **char**, etc.)

Introduction to Arrays

- ❑ To refer to an element
 - Specify array name and position number (index)
 - Format: **arrayname**[**position number**]
 - First element at position **0**

- ❑ N-element array c



n^{th} element as position $n-1$

Introduction to Arrays

- ❑ Array elements like other variables
 - Assignment, printing for an integer array **c**

```
c[0] = 3;  
cout << c[0];
```

- ❑ Can perform operations inside subscript

```
c[5-2] same as c[3]
```

Introduction to Arrays

-45	6	0	72	154	-89	0	62	-3	1
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]

Name of the array (note that all elements of this array have the same name, c)

Position number of the element within array c

Declaring Arrays

❑ When declaring arrays, specify

- Name
- Type of array (Any data type)
- Number of elements
- `type arrayName[arraySize] ;`

```
int c[10];    // array of 10 integers
float d[3284]; // array of 3284 floats
```

❑ Declaring multiple arrays of same type

- Use comma separated list, like regular variables

```
int b[100], x[27];
```

Declaring Arrays

❑ Initializing arrays: many ways

1. **for** loop: set each element

```
for(int i=0; i<n; i++)  
    c[i] = 0;
```

2. Initializer list: Specify each element when array declared

```
int c[5] = { 1, 2, 3, 4, 5 };
```

→ If not enough initializers, rightmost elements 0

→ Listing too many will result in a syntax error

3. To set every element to same value

```
int c[5] = { 0 };
```

4. If array size omitted, initializers determine size

```
int c[] = { 1, 2, 3, 4, 5 };
```

→ 5 initializers, therefore 5 element array

Declare & Initialize Arrays: Example

```
// Initializing an array.
```

```
#include <iostream>
```

```
#include <iomanip>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int n[10]; // n is an array of 10 integers
```

```
    // initialize elements of array n to 0
```

```
    for (int i = 0; i < 10; ++i)
```

```
        n[i] = 0; // set element at location i to 0
```

```
    cout << "Element" << setw(13) << "Value" << endl;
```

```
    // output contents of array n in tabular format
```

```
    for (int j = 0; j < 10; ++j)
```

```
        cout << setw(7) << j << setw(13) << n[j] << endl;
```

```
    return 0;
```

```
} //end main
```

Declare a 10-element array of integers.

Initialize array to 0 using a for loop. Note that the array has elements `n[0]` to `n[9]`.

Declare & Initialize Arrays: Example

□ Resulting Array:

Element	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Declare & Initialize Arrays: Example

```
// Initializing an array.
```

```
#include <iostream>
```

```
#include <iomanip>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    // use initializer list to initialize array n
```

```
    int n[10] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37
```

```
};
```

```
    cout << "Element" << setw(13) << "Value" << endl;
```

```
    // output contents of array n in tabular format
```

```
    for (int i = 0; i < 10; ++i)
```

```
        cout << setw(7) << i << setw(13) << n[i] << endl;
```

```
    return 0;
```

```
} //end main
```

Note the use of the initializer list.

Declare & Initialize Arrays: Example



fit@hcmus

□ Resulting Array:

Element	Value
0	32
1	27
2	64
3	18
4	95
5	14
6	90
7	70
8	60
9	37

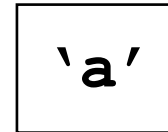
Introduction to Strings

- ❑ There is **NO** such thing as a **string data type** build into the language of C++
 - Although there is a string **“class”** that provides a standardized string type
- ❑ When I use the term “string” I mean some ***sequence of characters*** (such as a name, address, description, etc.)
- ❑ Strings are represented in C and C++ by **arrays of characters**
- ❑ We all know what a character is (a single byte), so what’s an array of characters?
 - a sequence of character stored sequentially in memory

How do I define an Array of Characters?

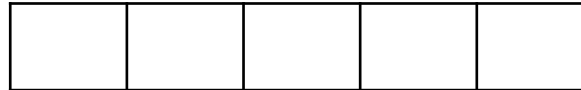
- ❑ We know how to define a single character:

```
char ch = 'a';
```



- ❑ But what about an array of characters?

```
char str[5];
```



- ❑ Since these are just characters stored sequentially in memory, we use a special character to indicate the end of a string: `'\0'`

How do I read in a string?

- ❑ There are **two ways** to read in strings
- ❑ If the string is a sequence of characters without any whitespace, then you can say:

```
cin >> str;
```

- ❑ If I enter "hi", this is what is stored:

'h'	'i'	'\0'		
-----	-----	------	--	--

What does `cin >> array_of_characters` do?

```
char str[5];  
cin >> str;
```

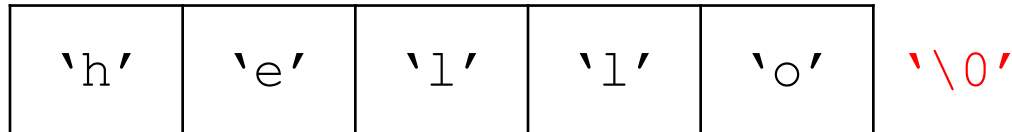
- ❑ When reading in an array of characters, `cin` and the extraction operator (`>>`) skip leading whitespace and read characters until a whitespace character is encountered.
- ❑ Then, it automatically stores a `'\0'` after the last character read in.

What do we need to be careful about?

- ❑ We need to be careful when working with arrays of characters...
- ❑ If we have an array of size 5
 - that means there are 5 bytes of memory allocated for our variable sequentially in memory
- ❑ This means that we can store four characters at most, since one spot needs to be reserved for the terminating null

So, What could happen???

- ❑ Using `cin >> str;`
- ❑ If I enter “hello”, this is what is stored:



- ❑ Notice we ended up storing the '\0' in memory that is **not allocated** for our variable
 - *this is extremely dangerous and can cause our programs to bomb! (segmentation fault or core dump when running...)*

What do we need to be careful about?



- ❑ What this means is that C++ does not check to make sure we stay within the bounds of our arrays
- ❑ C++ assumes that we know what we are doing!
- ❑ It is a powerful language...one that can even be used to design operating systems
- ❑ Therefore, if there is a chance that the user may type in too many characters, we need to read in our strings using a different approach

How do I read in a string safely?

- ❑ There is a `cin.get` function that is useful
- ❑ There are three ways to use this function:
 - it can be used to read a single character

```
char ch;  
ch = cin.get();  
Or cin.get(ch);
```

→ *this reads in the next character from the input buffer, even if that next character is whitespace!*

How do I read in a string safely?

- ❑ Or, we can use this function to read in a string using 2 or 3 arguments:

```
char str[5];  
cin.get(str, 5);  
Or cin.get(str, 5, '\n');
```

→ this reads in the next sequence of characters up until (size-1) characters are read or the delimiting character is encountered ('\n' by default)

How do I read in a string safely?

- ❑ The three argument version of `cin.get` has the following form:

```
cin.get(array_name, max_size, delimiting_character) ;
```

- ❑ A side benefit of this function is that it will allow us to read in sentences, our entire first/last name, a paragraph, etc. This is because the delimiting character need not be white space!

How do I read in a string safely?

- ❑ There is one “**gotcha**” with this function.
- ❑ While the three argument version of `cin.get` won't read in too many characters (so it will **never** store characters outside your array bounds),
 - it will **NOT** read in the delimiting character!
- ❑ Therefore, we must always “**eat**” the delimiting character, using either:
`cin.get() ;` or `while (cin.get() != '\n') ;`

Let's read another string, using `cin.get()`:

- ❑ Using `cin.get(str, 5);`
- ❑ If I enter “`hi !`”, this is what is stored:

<code>'h'</code>	<code>'i'</code>	<code>' '</code>	<code>'!'</code>	<code>'\0'</code>
------------------	------------------	------------------	------------------	-------------------

- ❑ Notice that room is left to store the `'\0'` at the end of the array, and there is no danger of writing outside of our array bounds.
- ❑ But, what is left in the input buffer? `'\n'`
- ❑ How do we “flush” this?
`cin.get();`

Let's read another string, using `cin.get`:

- ❑ Using `cin.get(str, 5);`
- ❑ If I enter “hello”, this is what is stored:

'h'	'e'	'l'	'l'	'0'
-----	-----	-----	-----	-----

- ❑ Notice that room is left to store the `'\0'` at the end of the array, and there is no danger of writing outside of our array bounds.
- ❑ But, what is left in the input buffer? `'o\n'`
- ❑ How do we “flush” this?

```
while(cin.get() != '\n');
```

How do I display a string?

- ❑ Luckily, displaying strings isn't as complicated.

```
cout << str;
```

- ❑ Simply by using `cout` followed by the insertion operator (`<<`), we can display as many characters as have been stored in the array until the terminating null (`'\0'`) is encountered.
- ❑ Notice, the `'\0'` is important so that we don't display “garbage” characters (i.e., memory that has not been set or used yet!)

More about strings

- ☐ Comparing and copying strings
- ☐ Accessing single characters in a string
- ☐ Write a program with strings

Operations on Strings

- ❑ There are very few operations that can be performed on array of characters (i.e., strings)

- ❑ We can read in string using:

```
cin >> array_of_characters;  
cin.get(array, size, delimiter);
```

- ❑ We display strings using:

```
cout << array_of_characters;
```

- ❑ But, there are no others...

Operations on Strings

- ❑ For example, we cannot compare two strings by saying:

```
char str1[10], str2[10];  
if (str1 == str2)
```

- ❑ This is because an array is really *the address of the first element in a sequentially allocated set of memory*.
- ❑ So, the `==` or `!=` operators would simply be comparing the memory addresses!
- ❑ Oops!

Comparing Strings

- ❑ Instead, to compare two strings we must include another library: `<cstring>`
- ❑ And, call the string compare function:

```
strcmp(first_array, second_array);
```

- ❑ The strcmp function returns:
 - 0 if first_array is equal to second_array
 - <0 if first_array is less than second_array
 - >0 if first_array is greater than second_array

Copying Strings

- ❑ We also cannot copy strings using the assignment operator:

```
char str1[10], str2[10];  
str1 = str2;
```

- ❑ This is **illegal** because an array is really *the constant address of the first element of the array.*
→ *We can't change the location in memory where your array is located!!!! And...that is what this assignment statement is attempting to do...*
- ❑ Instead, we call strcpy from cstring

```
strcpy(str1, str2); //str1=str2
```

Passing Arrays to Functions

- ❑ Specify name without brackets
 - To pass array `myArray` to `myFunction`

```
int myArray[ 24 ];  
myFunction( myArray );
```


Passing Arrays to Functions

- ❑ Arrays **passed-by-reference**
 - Functions can modify original array data
 - Value of name of array is address of first element
 - ✓ Function knows where the array is stored
 - ✓ Can change original memory locations
- ❑ Individual array elements **passed-by-value**
 - Like regular variables

```
square ( myArray [ 3 ] ) ;
```

Passing Arrays to Functions

- ❑ Functions taking arrays as parameters
 - Function prototype

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

```
void modifyArray( int [], int );
```

- Names are optional in prototype
- Both take an integer array and a single integer
- No need for array size between brackets
- Ignored by compiler

For example:

- ❑ Let's now put this to use by writing a function to read in two strings and displaying them in alphabetical order
- ❑ First, write the algorithm:
 - Get two strings (prompt, input, echo)
 - If the first string is less than the second
display the first string followed by the second
 - If the first string is greater or equal to the second
display the second string followed by the first

Writing a function to work with strings

```
#include <cstring>
void sort_two() {
    char first[20], second[20];
    cout << "Please enter two words: ";
    cin.get(first, 20, '\n');
    cin.get(); //don't forget this part!
    cin.get(second, 20, '\n');
    cin.get(); //eat the carriage return;
    if (strcmp(first, second) < 0)
        cout << first << ' ' << second << endl;
    else
        cout << second << ' ' << first << endl;
}
```

Change the function to have args



```
#include <cstring>
void sort_two(char first[], char second[]) {
    cout << "Please enter two words: ";
    cin.get(first, 20, '\n');
    cin.get();
    cin.get(second, 20, '\n');
    cin.get(); //eat the carriage return;
    if (strcmp(first, second) > 0) {
        char temp[20];
        strcpy(temp, first);
        strcpy(first, second);
        strcpy(second, temp);
    }
}
```

We'd call the function by saying:

```
#include <cstring>
void sort_two(char first[], char second[]);
int main() {
    char str1[20], str2[20];

    sort_two(str1, str2);
    cout << str1 << ' ' << str2 << endl;

    //what would happen if we then said:
    sort_two(str2, str1);
    cout << str1 << ' ' << str2 << endl;
    return 0;
}
```

Working with arrays, character at a time



- ❑ We can also work with strings an element at a time,
 - by indexing through the array
 - we begin by using subscripts that start at zero and then progress until the array size-1
- ❑ For example, we can read in a string by:
 - Read a character
 - If that character is not a carriage return
o save the character in the array

Reading a Character at a time:

```
char str[20];
char ch;
int index = 0;

ch = cin.get();
while (ch != '\n') {
    str[index] = ch;    //str[index] is a char
    ++index;
    ch = cin.get();
}
str[index] = '\0';    //why is this important?
```

❑ But, what if they type in too many characters?

A Better Approach?

```
const int MAX = 20;
char str[MAX];      char ch;
int index = 0;

ch = cin.get();
while (index < MAX-1 && ch != '\n') {
    str[index] = ch;    //str[index] is a char
    ++index;
    ch = cin.get();
}
str[index] = '\0';    //why is this important?
```

The Same Thing...Just Condensed:



fit@hcmus

```
const int MAX = 20;
char str[MAX];
int index = 0;

while (index < MAX-1 && (ch= cin.get()) != '\\n'))
    str[index++] = ch; //Remember postfix????

str[index] = '\\0';    //Still important
```

Or, going to an extreme!

```
const int MAX = 20;  
char str[MAX];  
int index = 0;  
  
while (index < MAX-1 &&  
      (str[index++] = cin.get()) != '\\n');  
  
str[index] = '\\0';
```