# More Arrays; Chars

ITP 165 – Fall 2015
Week 4, Lecture 2

# Array Initialization

- Initializing the array this way has a lot of repetition with several lines that just begin with `names[index]`:

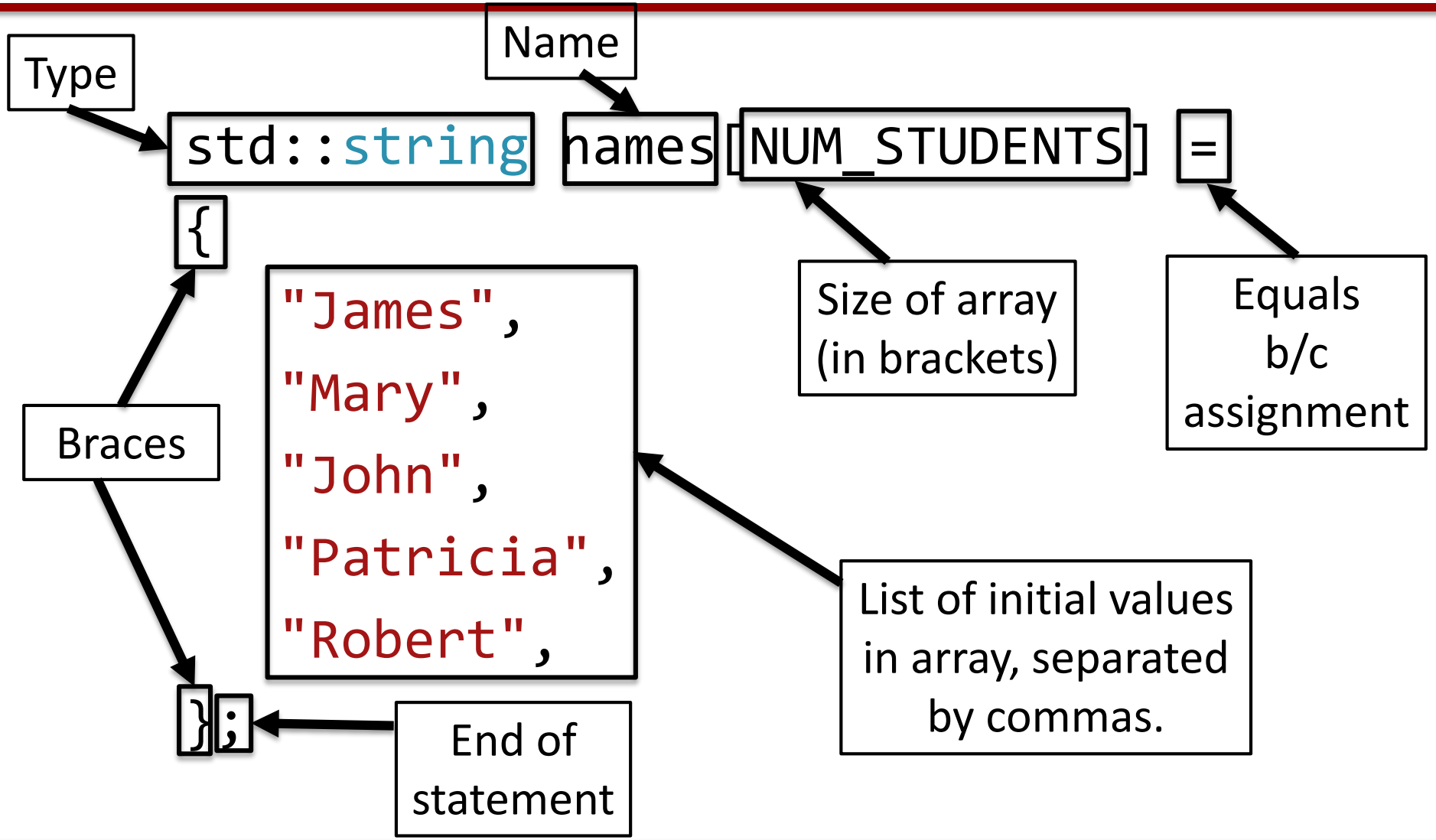
```
const int NUM_STUDENTS = 5;
std::string names[NUM_STUDENTS];
names[0] = "James";
names[1] = "Mary";
names[2] = "John";
names[3] = "Patricia";
names[4] = "Robert";
```

# Array Initialization, Cont'd

- The previous code could be rewritten using a shortcut:

```
const int NUM_STUDENTS = 5;
std::string names[NUM_STUDENTS] =
{
    "James",
    "Mary",
    "John",
    "Patricia",
    "Robert",
};
```

# Declaring an Array, w/ Initialization



# Array Initialization Examples, w/ Comments

- Just in case it's unclear which elements are being assigned in the array…

```
const int NUM_STUDENTS = 5;
std::string names[NUM_STUDENTS] =
{
    "James", // Sets index 0 to "James"
    "Mary", // Sets index 1 to "Mary"
    "John", // Sets index 2 to "John"
    "Patricia", // Sets index 3 to "Patricia"
    "Robert", // Sets index 4 to "Robert"
};
```

# Size Mismatch

- What happens if I say there are 5 elements, but I actually assign 6?

```
const int NUM_STUDENTS = 5;
std::string names[NUM_STUDENTS] =
{
    "James",
    "Mary",
    "John",
    "Patricia",
    "Robert",
    "Raymond", // Error: "too many initializers"
};
```

# Size Mismatch, Cont'd

- If I have too few, it won't complain…

```
const int NUM_STUDENTS = 5;
std::string names[NUM_STUDENTS] =
{
    "James",
    "Mary",
    "John",
    "Patricia",
    // I should really have 5 initial values...
};
```

- However…the value at `names[4]` would be uninitialized

# The char type

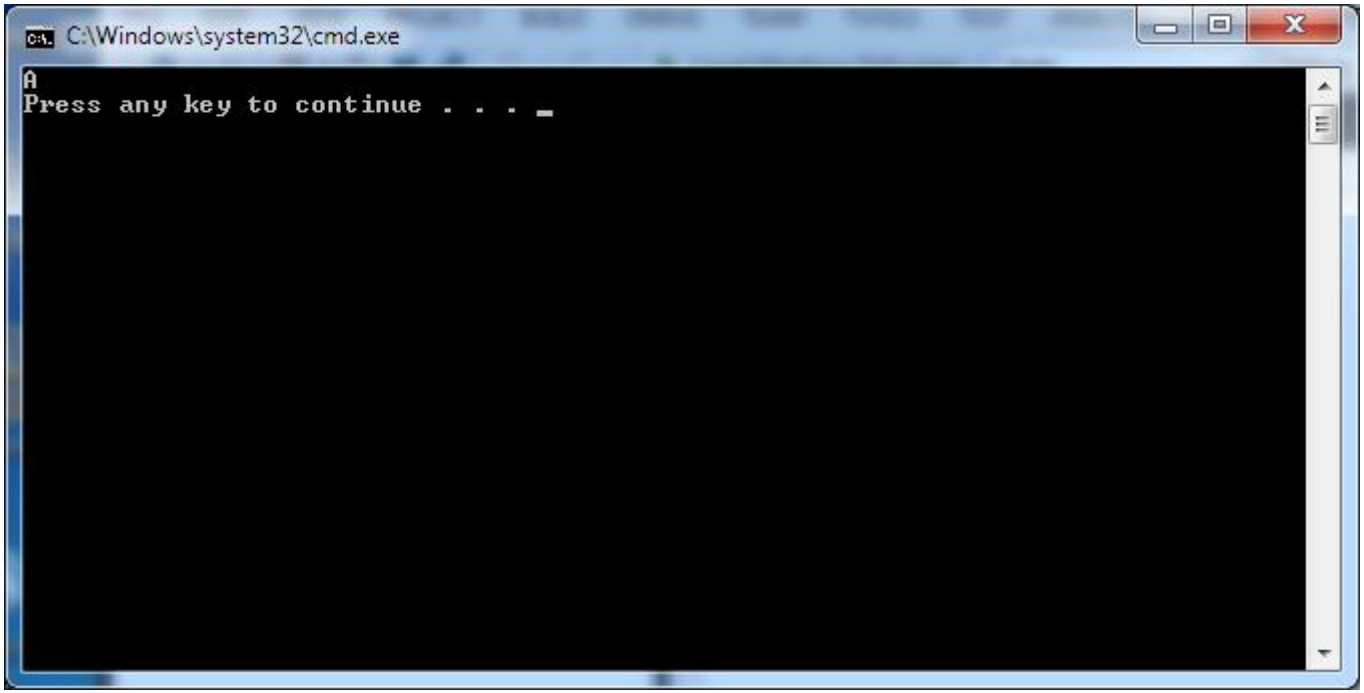
- The `char` type is a whole number type that ranges from -128 to 127

- However, the `char` type has another special use!

- What happens if I run this code?

```
char test = 65;
std::cout << test << std::endl;
```

# Char Code in Action




- ?!?!?!?!

# ASCII

- ***ASCII*** – A standardized number code where every number corresponds to a specific letter

- It turns out that the number 65 is the code for an uppercase A!

- By default, if you `cout` a `char`, it will output the ASCII letter corresponding to the number

- This is why we call it a `char` – it stands for ***character***!

# ASCII Table

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [ENG OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |

# You don't have to memorize the table!

- Rather than having to remember that 65 corresponds to the letter A, we can use a shortcut…

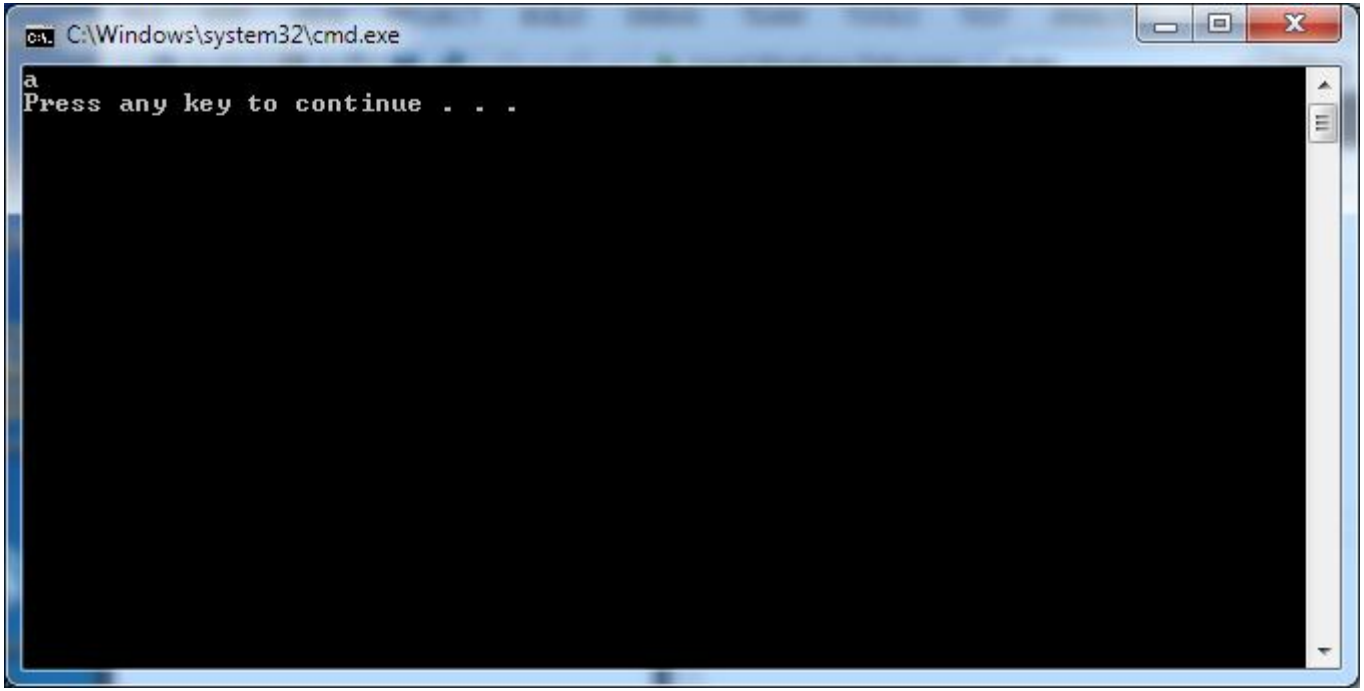
- Single quotes around a letter signifies it is a character:

```
// This has the same result as the prior example
char test = 'A';
std::cout << test << std::endl;
```

- Internally, C++ converts the `'A'` into the appropriate number (65)

# Another Example

```
char test2 = 'A' + 32;
std::cout << test2 << std::endl;
```



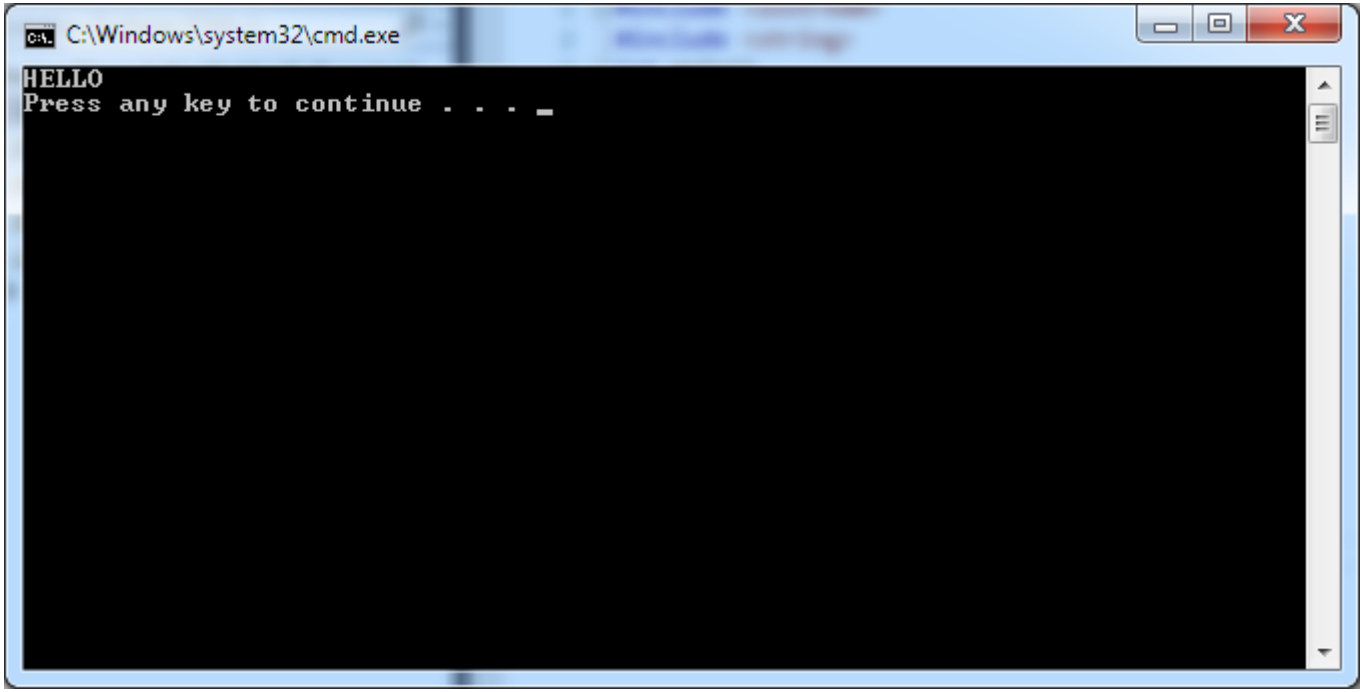
# Array of chars

- Just like with any other type, we can have an array of characters:

```
char word[6] =
{
    72,
    69,
    76,
    76,
    79,
    0
};
std::cout << word << std::endl;
```

# Array of chars, Cont'd

- Running the previous code yields:



# Array of chars, cont'd

- We could rewrite the code as:

```
char word[6] =
{
    'H',
    'E',
    'L',
    'L',
    'O',
    0
};
std::cout << word << std::endl;
```

- But why is the size of the array 6 instead of 5, and why is the value in the last index 0?

# C-Style Strings and Null Terminators

- An array of characters is also referred to as a ***C-style string***

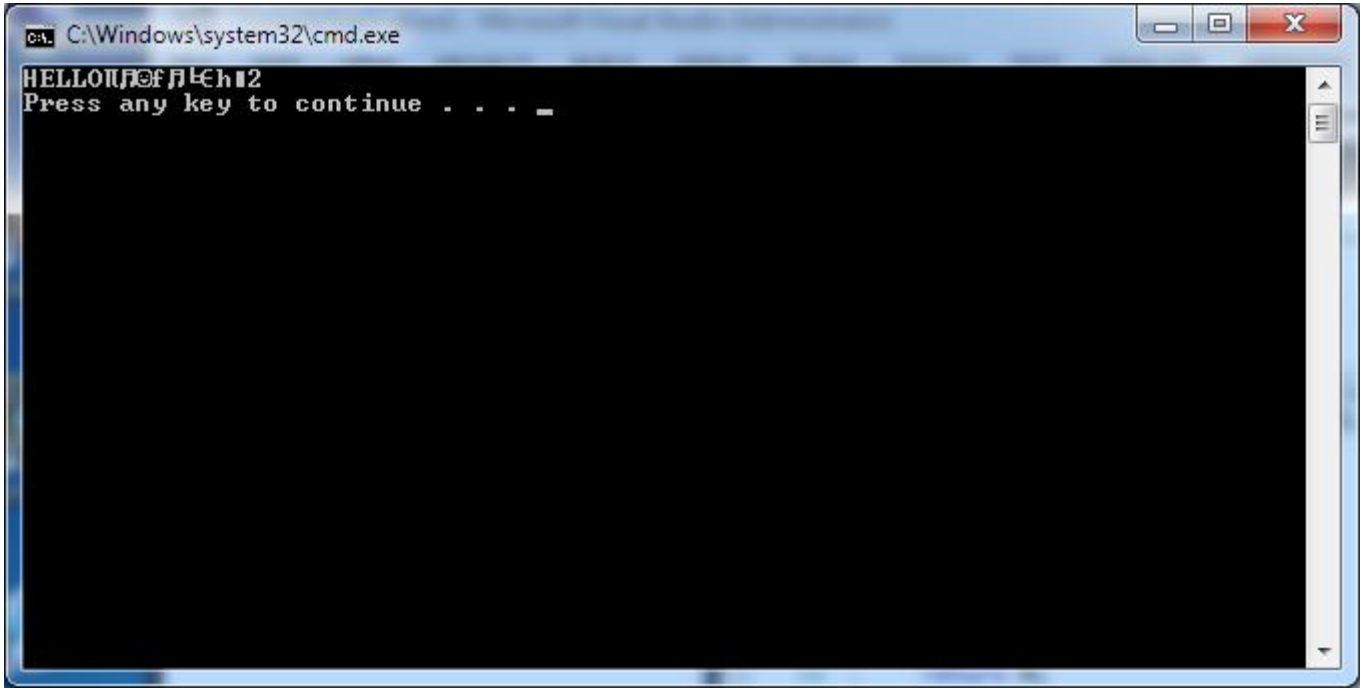
- A C-style string must end with a 0 after the last letter – the zero is the special ASCII code for the ***null terminator***

- If you don't have the 0, anything that operates on the C-style string (including `cout`) won't end a the proper point

- The null terminator is what signifies that the C-style string is over!!!

# What happens if we omit the null terminator?

```
char word[5] = { 'H', 'E', 'L', 'L', 'O' };
std::cout << word << std::endl;
```



# Or looking at it another way…

- Given the following C-style string:

```
char word[6] = { 'H', 'E', 'L', 'L', 'O', 0 };
```

- This code:

```
std::cout << word << std::endl;
```

- It is equivalent to:

```
int i = 0;
while (word[i] != 0)
{
    std::cout << word[i];
    i++;
}
std::cout << std::endl;
```

# Null Terminator

- Given the following C-style string:

```
char word[6] = { 'H', 'E', 'L', 'L', 'O', 0 };
```

Notice: the ***null terminator*** is not enclosed in single quotes (' ')

In C++:

`0 == '0'` is ***FALSE***

In reality, in C++:

`0 == '\0'`

# C-style string Initialization

- So of course, writing out a C-style string like this is annoying:

```
char word[6] = { 'H', 'E', 'L', 'L', 'O', 0 };
```

- To fix this issue, we can initialize it like so:

```
char word[6] = "HELLO";
```

- This will automatically add the null terminator, so we don't have to worry about adding on a zero.

# C-style strings, cont'd

- What this means is that when you say:

```
std::cout << "Hello world!" << std::endl;
```

- C++ is actually secretly using C-style strings!!!

- It's not using `std::string`

# Shortcuts upon shortcuts

- There's something slightly annoying about this initialization:

```
char word[6] = "HELLO";
```

- The problem is we have to remember that even though it's 5 letters, we need 6 spots in the array
- Because the last spot is the null terminator

- So another shortcut is to just omit the number of elements in the array:

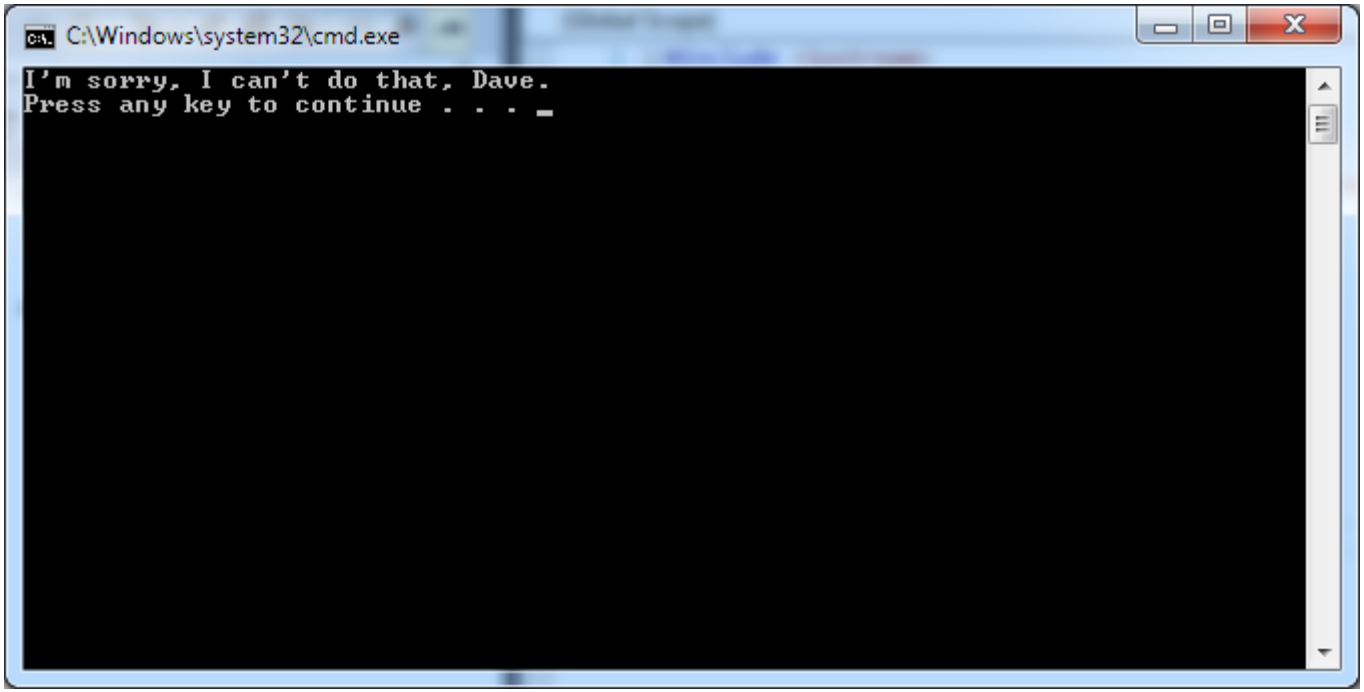
```
char word[] = "HELLO";
```

- C++ will automatically figure out that it's 5 letters plus a null terminator, so the size should be 6.

# Another Example

- This will work just fine:

```
char stuff[] = "I'm sorry, I can't do that, Dave.";
std::cout << stuff << std::endl;
```



# Lab Practical #6