12 - Strings and Objects

Dr. Robert Lowe

Division of Mathematics and Computer Science
Maryville College

Outline

1. Objects

2. Strings

Outline

1. Objects

2. Strings

What is an object?

What is an object?

- An **object** is an entity that has both state and behavior.

What is an object?

- An **object** is an entity that has both state and behavior.
- It’s a thing that “remembers” something and does something.

What is an object?

- An **object** is an entity that has both state and behavior.
- It's a thing that "remembers" something and does something.
- In C++, objects allow us to have complex types with lots of useful abstract behavior.

The “Black Box” View of Objects

- When using objects, we think of them as “black boxes”.

Image Source: `http://ebay.com`

The “Black Box” View of Objects

- When using objects, we think of them as “black boxes”.
- The object does what it does, and we don’t know or care how.

Image Source: `http://ebay.com`

The “Black Box” View of Objects

- When using objects, we think of them as “black boxes”.
- The object does what it does, and we don’t know or care how.
- Kind of like a soda machine.

Image Source: `http://ebay.com`

The “Black Box” View of Objects

- When using objects, we think of them as “black boxes”.
- The object does what it does, and we don’t know or care how.
- Kind of like a soda machine.
- By what arcane arts does the machine convert money into cold, bubbly, liquid candy?

Image Source: `http://ebay.com`

The “Black Box” View of Objects

- When using objects, we think of them as “black boxes”.
- The object does what it does, and we don’t know or care how.
- Kind of like a soda machine.
- By what arcane arts does the machine convert money into cold, bubbly, liquid candy?
  - Who knows?

Image Source: `http://ebay.com`

The “Black Box” View of Objects

- When using objects, we think of them as “black boxes”.
- The object does what it does, and we don’t know or care how.
- Kind of like a soda machine.
- By what arcane arts does the machine convert money into cold, bubbly, liquid candy?
    - Who knows?
    - Who cares? (After we drink this stuff, the inner workings of the machine are the least of our worries!)

Image Source: `http://ebay.com`

Accessing Object Functions

- Every object is a scope unto itself.

Accessing Object Functions

- Every object is a scope unto itself.
- Every object contains member functions or **methods**.

Accessing Object Functions

- Every object is a scope unto itself.
- Every object contains member functions or **methods**.
- Every object also contains member variables.

Accessing Object Functions

- Every object is a scope unto itself.
- Every object contains member functions or **methods**.
- Every object also contains member variables.
- The member methods operate on the member variables.

Accessing Object Functions

- Every object is a scope unto itself.
- Every object contains member functions or **methods**.
- Every object also contains member variables.
- The member methods operate on the member variables.
- We access members by using the “.” operator.

Accessing Object Functions

- Every object is a scope unto itself.
- Every object contains member functions or **methods**.
- Every object also contains member variables.
- The member methods operate on the member variables.
- We access members by using the "`.`" operator.
- For example, suppose we had an object `str` which has a method `length`. We would call this method like so:
  `str.length();`

Objects Types

- Like everything else in C++, objects have types.

Objects Types

- Like everything else in C++, objects have types.
- In C++, an object's type is its **class**.

Objects Types

- Like everything else in C++, objects have types.
- In C++, an object's type is its **class**.
- The class of an object determines what member methods and variables it has.

Objects Types

- Like everything else in C++, objects have types.
- In C++, an object's type is its **class**.
- The class of an object determines what member methods and variables it has.
- Later on, we will define our own classes. For now, we will simply make use of pre-existing classes.

Outline

1 Objects

2 Strings

String Literals and C-Strings

- Recall that a string literal has quotation marks around it. Example: “`Hello, world`”.

String Literals and C-Strings

- Recall that a string literal has quotation marks around it. Example: “`Hello, world`”.
- The actual type of this literal is `const char[]`.

String Literals and C-Strings

- Recall that a string literal has quotation marks around it. Example: “`Hello, world`”.
- The actual type of this literal is `const char[]`.
- That is, an array of constant characters.

String Literals and C-Strings

- Recall that a string literal has quotation marks around it. Example: “`Hello, world`”.
- The actual type of this literal is `const char[]`.
- That is, an array of constant characters.
- This is a sequence of characters stored in contiguous memory:

H	e	l	l	o	,		W	o	r	l	d	∅

String Literals and C-Strings

- Recall that a string literal has quotation marks around it. Example: “`Hello, world`”.
- The actual type of this literal is `const char[]`.
- That is, an array of constant characters.
- This is a sequence of characters stored in contiguous memory:

H	e	l	l	o	,		W	o	r	l	d	∅

- C-Strings can be difficult to work with.

String Literals and C-Strings

- Recall that a string literal has quotation marks around it. Example: “`Hello, world`”.
- The actual type of this literal is `const char[]`.
- That is, an array of constant characters.
- This is a sequence of characters stored in contiguous memory:

H	e	l	l	o	,		W	o	r	l	d	∅

- C-Strings can be difficult to work with.
- They cannot grow or shrink, and literal strings are immutable.

String Literals and C-Strings

- Recall that a string literal has quotation marks around it. Example: “`Hello, world`”.
- The actual type of this literal is `const char[]`.
- That is, an array of constant characters.
- This is a sequence of characters stored in contiguous memory:

H	e	l	l	o	,		W	o	r	l	d	∅

- C-Strings can be difficult to work with.
- They cannot grow or shrink, and literal strings are immutable.
- Fortunately, C++ gives us a better way!

C++ Strings

- C++ provides a string object which abstracts away the details of how stings work.

C++ Strings

- C++ provides a string object which abstracts away the details of how stings work.
- To use the string objects, you need to include the string header:

```
#include <string>
```

C++ Strings

- C++ provides a string object which abstracts away the details of how stings work.
- To use the string objects, you need to include the string header:

  ```
  #include <string>
  ```

- String objects are declared with the class type `string`:

  ```
  string str;
  ```

C++ Strings

- C++ provides a string object which abstracts away the details of how stings work.
- To use the string objects, you need to include the string header:

  ```
  #include <string>
  ```
- String objects are declared with the class type `string`:

  ```
  string str;
  ```
- String objects can also be assigned string literal:

  ```
  str = "Hello, world";
  ```

C++ Strings

- C++ provides a string object which abstracts away the details of how stings work.
- To use the string objects, you need to include the string header:

  ```
  #include <string>
  ```

- String objects are declared with the class type `string`:

  ```
  string str;
  ```

- String objects can also be assigned string literal:

  ```
  str = "Hello, world";
  ```

- C++ strings can grow and shrink as needed. They are much more convenient than C strings!

String Demonstration

- Go ahead and compile and and run `examples/12-Strings/string_demo.cpp`

String Demonstration

- Go ahead and compile and and run `examples/12-Strings/string_demo.cpp`
- First we have the `string` declaration and initialization:

```
string str = "Hello, World";
```

String Demonstration

- Go ahead and compile and and run `examples/12-Strings/string_demo.cpp`
- First we have the `string` declaration and initialization: `string str = "Hello, World";`
- Then we can see that strings interact with output streams: `cout « "The string is:  " « str « endl;`

String Demonstration

- Go ahead and compile and and run `examples/12-Strings/string_demo.cpp`
- First we have the `string` declaration and initialization: `string str = "Hello, World";`
- Then we can see that strings interact with output streams: `cout « "The string is:  " « str « endl;`
- Another handy method allows us to get the length of a string: `str.length()`

Indexing Characters

Character	H	e	l	l	o	,		W	o	r	l	d
Index	0	1	2	3	4	5	6	7	8	9	10	11

- Each character in a string occupies a numbered slot.

Indexing Characters

Character	H	e	l	l	o	,		W	o	r	l	d
Index	0	1	2	3	4	5	6	7	8	9	10	11

- Each character in a string occupies a numbered slot.
- The number of each position is called an **index.**

Indexing Characters

Character	H	e	l	l	o	,		W	o	r	l	d
Index	0	1	2	3	4	5	6	7	8	9	10	11

- Each character in a string occupies a numbered slot.
- The number of each position is called an **index.**
- In C++, indexes (alas) begin at zero.

Indexing Characters

Character	H	e	l	l	o	,		W	o	r	l	d
Index	0	1	2	3	4	5	6	7	8	9	10	11

- Each character in a string occupies a numbered slot.
- The number of each position is called an **index.**
- In C++, indexes (alas) begin at zero.
- The largest index in a string will be its `length() - 1`

Indexing Characters

Character	H	e	l	l	o	,		W	o	r	l	d
Index	0	1	2	3	4	5	6	7	8	9	10	11

- Each character in a string occupies a numbered slot.
- The number of each position is called an **index.**
- In C++, indexes (alas) begin at zero.
- The largest index in a string will be its `length() - 1`
- Hence the following loops through every index in the string:

```
//Loop over each character in the string
for(int i=0; i<str.length(); i++) {
    cout << i << ": " << str[i] << endl;
}
```

Indexing Characters

Character	H	e	l	l	o	,		W	o	r	l	d
Index	0	1	2	3	4	5	6	7	8	9	10	11

- Each character in a string occupies a numbered slot.
- The number of each position is called an **index.**
- In C++, indexes (alas) begin at zero.
- The largest index in a string will be its `length() - 1`
- Hence the following loops through every index in the string:

```
//Loop over each character in the string
for(int i=0; i<str.length(); i++) {
    cout << i << ": " << str[i] << endl;
}
```

- Note the use of the index operator `[]`.

Indexing Characters

Character	H	e	l	l	o	,		W	o	r	l	d
Index	0	1	2	3	4	5	6	7	8	9	10	11

- Each character in a string occupies a numbered slot.
- The number of each position is called an **index**.
- In C++, indexes (alas) begin at zero.
- The largest index in a string will be its `length() - 1`
- Hence the following loops through every index in the string:

```
//Loop over each character in the string
for(int i=0; i<str.length(); i++) {
    cout << i << ": " << str[i] << endl;
}
```

- Note the use of the index operator `[]`.
- Discuss: Why must an index be an integer?

Substrings

- Another handy feature of strings is the ability extract substrings.

Substrings

- Another handy feature of strings is the ability extract substrings.
- A **substring** is a segment within a larger string.

Substrings

- Another handy feature of strings is the ability extract substrings.
- A **substring** is a segment within a larger string.
- The `substr` function has two versions:

Substrings

- Another handy feature of strings is the ability extract substrings.
- A **substring** is a segment within a larger string.
- The `substr` function has two versions:
    - `substr(start)`

Substrings

- Another handy feature of strings is the ability extract substrings.
- A **substring** is a segment within a larger string.
- The `substr` function has two versions:
  - `substr(start)`
  - `substr(start, len)`

Substrings

- Another handy feature of strings is the ability extract substrings.
- A **substring** is a segment within a larger string.
- The `substr` function has two versions:
  - `substr(start)`
  - `substr(start, len)`
- Both are demonstrated in `string_demo.cpp`. What do they each do?

Strings and Extraction Operators

- Take a look at `examples/12-Strings/name.cpp`

Strings and Extraction Operators

- Take a look at `examples/12-Strings/name.cpp`
- Run the program. What can you say about the extraction operator and strings?

Strings and Extraction Operators

- Take a look at `examples/12-Strings/name.cpp`
- Run the program. What can you say about the extraction operator and strings?
- When you run the extraction operator, it only reads until the next space!

The `getline` Function

- The `getline` function allows us to read an entire line of text into a string.

The `getline` Function

- The `getline` function allows us to read an entire line of text into a string.
- Its function prototype is:

```
getline(istream &is, string &str);
```

The `getline` Function

- The `getline` function allows us to read an entire line of text into a string.
- Its function prototype is:
  `getline(istream &is, string &str);`
- Make a new directory: `labs/week8`

The getline Function

- The `getline` function allows us to read an entire line of text into a string.
- Its function prototype is:
  `getline(istream &is, string &str);`
- Make a new directory: `labs/week8`
- Copy `name.cpp` into `labs/week8`

The `getline` Function

- The `getline` function allows us to read an entire line of text into a string.
- Its function prototype is:
  `getline(istream &is, string &str);`
- Make a new directory: `labs/week8`
- Copy `name.cpp` into `labs/week8`
- Change the input line to the following:
  `getline(cin, name);`

The `getline` Function

- The `getline` function allows us to read an entire line of text into a string.
- Its function prototype is:
  ```
  getline(istream &is, string &str);
  ```
- Make a new directory: `labs/week8`
- Copy `name.cpp` into `labs/week8`
- Change the input line to the following:
  ```
  getline(cin, name);
  ```
- Discuss: Why isn't getline a member of string?

Lab Activity: Palindrome Detector

- A palindrome is a string that reads the same backwards and forwards.

Lab Activity: Palindrome Detector

- A palindrome is a string that reads the same backwards and forwards.
- For instance “racecar” is a palindrome.

Lab Activity: Palindrome Detector

- A palindrome is a string that reads the same backwards and forwards.
- For instance “racecar” is a palindrome.
- Let’s design and implement a program which reads in a line of text and then determines if it is a palindrome or not!

Lab Activity: Palindrome Detector

- A palindrome is a string that reads the same backwards and forwards.
- For instance “racecar” is a palindrome.
- Let’s design and implement a program which reads in a line of text and then determines if it is a palindrome or not!
- Now, let’s make our palindrome program ignore the following:

Lab Activity: Palindrome Detector

- A palindrome is a string that reads the same backwards and forwards.
- For instance “racecar” is a palindrome.
- Let’s design and implement a program which reads in a line of text and then determines if it is a palindrome or not!
- Now, let’s make our palindrome program ignore the following:
  - Ignore Case

Lab Activity: Palindrome Detector

- A palindrome is a string that reads the same backwards and forwards.
- For instance “racecar” is a palindrome.
- Let’s design and implement a program which reads in a line of text and then determines if it is a palindrome or not!
- Now, let’s make our palindrome program ignore the following:
    - Ignore Case
    - Ignore Punctuation

Lab Activity: Palindrome Detector

- A palindrome is a string that reads the same backwards and forwards.
- For instance “racecar” is a palindrome.
- Let’s design and implement a program which reads in a line of text and then determines if it is a palindrome or not!
- Now, let’s make our palindrome program ignore the following:
  - Ignore Case
  - Ignore Punctuation
  - Ignore Spaces

The cctype Library

- The cctype library (#include<cctype>) contains lots of handy functions to help us deal with characters. Some of these that may come in handy are:

The cctype Library

- The cctype library (#include<cctype>) contains lots of handy functions to help us deal with characters. Some of these that may come in handy are:
  - toupper(c)

The `cctype` Library

- The cctype library (`#include<cctype>`) contains lots of handy functions to help us deal with characters. Some of these that may come in handy are:
    - `toupper(c)`
    - `tolower(c)`

The cctype Library

- The cctype library (#include<cctype>) contains lots of handy functions to help us deal with characters. Some of these that may come in handy are:
  - toupper(c)
  - tolower(c)
  - isalpha(c)

The cctype Library

- The cctype library (`#include<cctype>`) contains lots of handy functions to help us deal with characters. Some of these that may come in handy are:
  - `toupper(c)`
  - `tolower(c)`
  - `isalpha(c)`
  - `isspace(c)`

The cctype Library

- The cctype library (`#include<cctype>`) contains lots of handy functions to help us deal with characters. Some of these that may come in handy are:
  - `toupper(c)`
  - `tolower(c)`
  - `isalpha(c)`
  - `isspace(c)`
- Let's finish the palindromes!