19 - Classy Programming

Dr. Robert Lowe

Division of Mathematics and Computer Science
Maryville College

Outline

1. Objects and Classes
2. Classes and Objects in C++

Outline

1 Objects and Classes

2 Classes and Objects in C++

Bundling Variables

- `struct`s provide us with a way to “bundle” several fields.

Bundling Variables

- `struct`s provide us with a way to “bundle” several fields.
- This is useful for maintaining **state** of a composite type.

Bundling Variables

- `struct`s provide us with a way to “bundle” several fields.
- This is useful for maintaining **state** of a composite type.
- But what if there was another layer of abstraction?

Objects

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

Objects

- An **object** is an entity with both state and behavior.
- In addition to fields, objects have their own functions.

Objects

- An **object** is an entity with both state and behavior.
- In addition to fields, objects have their own functions.
- The basic idea is to have something in a program that both "remembers" and "acts".

Objects

- An **object** is an entity with both state and behavior.
- In addition to fields, objects have their own functions.
- The basic idea is to have something in a program that both "remembers" and "acts".
- Objects provide a way to model real world entities within a program.

Object Example

- Suppose we encountered a soda machine.

Object Example

- Suppose we encountered a soda machine.
- What could we do with it?

Object Example

- Suppose we encountered a soda machine.
- What could we do with it?
  - Insert Money

Object Example

- Suppose we encountered a soda machine.
- What could we do with it?
  - Insert Money
  - Push Buttons

Object Example

- Suppose we encountered a soda machine.
- What could we do with it?
  - Insert Money
  - Push Buttons
  - Pull Change Return

Object Example

- Suppose we encountered a soda machine.
- What could we do with it?
  - Insert Money
  - Push Buttons
  - Pull Change Return
- What sort of internal state does the machine have?

Object Example

- Suppose we encountered a soda machine.
- What could we do with it?
  - Insert Money
  - Push Buttons
  - Pull Change Return
- What sort of internal state does the machine have?
  - Money Deposited

Object Example

- Suppose we encountered a soda machine.
- What could we do with it?
  - Insert Money
  - Push Buttons
  - Pull Change Return
- What sort of internal state does the machine have?
  - Money Deposited
  - Money in the Vault

Object Example

- Suppose we encountered a soda machine.
- What could we do with it?
  - Insert Money
  - Push Buttons
  - Pull Change Return
- What sort of internal state does the machine have?
  - Money Deposited
  - Money in the Vault
  - Price, Quantity, and Brand of Each Soda

Classes

- Classes define objects.

Classes

- Classes define objects.
- An object is an instance of a class.

Classes

- Classes define objects.
- An object is an instance of a class.
- The basic elements of a class are:

Classes

- Classes define objects.
- An object is an instance of a class.
- The basic elements of a class are:
  - Constructors

Classes

- Classes define objects.
- An object is an instance of a class.
- The basic elements of a class are:
  - Constructors
  - Member Variable Declarations (*state*)

Classes

- Classes define objects.
- An object is an instance of a class.
- The basic elements of a class are:
  - Constructors
  - Member Variable Declarations (*state*)
  - Member Functions (*behavior*)

Classes

- Classes define objects.
- An object is an instance of a class.
- The basic elements of a class are:
  - Constructors
  - Member Variable Declarations (*state*)
  - Member Functions (*behavior*)
- Note that sometimes, member variables are called **attributes** and member functions are called **methods**.

Constructors

- A constructor is a block of code that is executed when we make an object.

Constructors

- A constructor is a block of code that is executed when we make an object.
- The job of the constructor is to initialize the class.

Constructors

- A constructor is a block of code that is executed when we make an object.
- The job of the constructor is to initialize the class.
- In C++ a constructor has the same name as its class.

Constructors

- A constructor is a block of code that is executed when we make an object.
- The job of the constructor is to initialize the class.
- In C++ a constructor has the same name as its class.
- Constructors are declared like functions, but they have no return type.

Constructors

- A constructor is a block of code that is executed when we make an object.
- The job of the constructor is to initialize the class.
- In C++ a constructor has the same name as its class.
- Constructors are declared like functions, but they have no return type.
- A constructor can take parameters, just like a function.

Constructors

- A constructor is a block of code that is executed when we make an object.
- The job of the constructor is to initialize the class.
- In C++ a constructor has the same name as its class.
- Constructors are declared like functions, but they have no return type.
- A constructor can take parameters, just like a function.
- A class may have multiple constructors (more on this later).

Member Variables

- Member variables maintain the state of objects.

Member Variables

- Member variables maintain the state of objects.
- Each object has its own set of member variables.

Member Variables

- Member variables maintain the state of objects.
- Each object has its own set of member variables.
- These are somewhat analogous to the fields in a struct.

Member Variables

- Member variables maintain the state of objects.
- Each object has its own set of member variables.
- These are somewhat analogous to the fields in a struct.
- Member variables should be accessible only to the class's code.

Member Functions

- Member functions provide the behavior of objects.

Member Functions

- Member functions provide the behavior of objects.
- Member functions operate on the member variables of an object.

Member Functions

- Member functions provide the behavior of objects.
- Member functions operate on the member variables of an object.
- The member variables are always in scope within a member function.

Controlling Access

- Classes also allow us to control who has access to the members of the class.

Controlling Access

- Classes also allow us to control who has access to the members of the class.
- In C++, the access levels are:

Controlling Access

- Classes also allow us to control who has access to the members of the class.
- In C++, the access levels are:

  public - Public members are accessible by all code.

Controlling Access

- Classes also allow us to control who has access to the members of the class.
- In C++, the access levels are:

  public - Public members are accessible by all code.

  private - Private members are accessible only by code within the class.

Controlling Access

- Classes also allow us to control who has access to the members of the class.
- In C++, the access levels are:

  public - Public members are accessible by all code.

  private - Private members are accessible only by code within the class.

  protected - Protected members are accessible only by code within the class or any subclass (more about this later).

Controlling Access

- Classes also allow us to control who has access to the members of the class.
- In C++, the access levels are:

  public - Public members are accessible by all code.

  private - Private members are accessible only by code within the class.

  protected - Protected members are accessible only by code within the class or any subclass (more about this later).
- As a general rule, all member variables should be **private**.

Controlling Access

- Classes also allow us to control who has access to the members of the class.
- In C++, the access levels are:

  public - Public members are accessible by all code.

  private - Private members are accessible only by code within the class.

  protected - Protected members are accessible only by code within the class or any subclass (more about this later).
- As a general rule, all member variables should be **private**.
- Constructors and member functions should usually be **public**.

Controlling Access

- Classes also allow us to control who has access to the members of the class.
- In C++, the access levels are:

  public - Public members are accessible by all code.

  private - Private members are accessible only by code within the class.

  protected - Protected members are accessible only by code within the class or any subclass (more about this later).
- As a general rule, all member variables should be **private**.
- Constructors and member functions should usually be **public**.
- Why do you think this is?

Outline

1 Objects and Classes

2 Classes and Objects in C++

Class Definitions in C++

```
class Name
{
public:
    //Public Members go here
private:
   //Private Members go here
};
```

Class Definitions in C++

```
class Name
{
public:
    //Public Members go here
private:
   //Private Members go here
};
```

- Class definitions typically go in header files.

Class Definitions in C++

```
class Name
{
public:
    //Public Members go here
private:
   //Private Members go here
};
```

- Class definitions typically go in header files.
- They contain function prototypes, constructor prototypes, and member variable declarations.

Class Definitions in C++

```
class Name
{
public:
    //Public Members go here
private:
   //Private Members go here
};
```

- Class definitions typically go in header files.
- They contain function prototypes, constructor prototypes, and member variable declarations.
- Class names should begin with an upper case letter to set them apart from variable names.

Class Definitions in C++

```
class Name
{
public:
    //Public Members go here
private:
   //Private Members go here
};
```

- Class definitions typically go in header files.
- They contain function prototypes, constructor prototypes, and member variable declarations.
- Class names should begin with an upper case letter to set them apart from variable names.
- Defining a class creates a new type (just like with `struct`.)

examples/19-Classy/soda-machine.h

```
#ifndef SODA_H
#define SODA_H

class Soda_Machine
{
public:
    //constructor
    Soda_Machine();

    //deposit money into the machine
    void insert_money(double amount);

    //pull the change return, returning the deposited change
    double change_return();

    //push a button, the return value is any message the machine gives
    //in response
    std::string push_button(int button);

private:
    std::vector<std::string> brand; //brands for the buttons
    std::vector<double> price;      //prices of each soda
    std::vector<int> quantity;      //the quantity of each soda
    double deposit;
    double vault;

    //dispense a soda
    void vend(int slot);
};
#endif
```

Conditional Compilation

```
#ifndef SODA_H
#define SODA_H
...
#endif
```

- Multiple definition of classes (and structs) is not allowed.

Conditional Compilation

```
#ifndef SODA_H
#define SODA_H
...
#endif
```

- Multiple definition of classes (and structs) is not allowed.
- A header file may be included more than once.

Conditional Compilation

```
#ifndef SODA_H
#define SODA_H
...
#endif
```

- Multiple definition of classes (and structs) is not allowed.
- A header file may be included more than once.
- Wrapping the header contents as above makes the preprocessor include the contents only one time.

Conditional Compilation

```
#ifndef SODA_H
#define SODA_H
...
#endif
```

- Multiple definition of classes (and structs) is not allowed.
- A header file may be included more than once.
- Wrapping the header contents as above makes the preprocessor include the contents only one time.
- Always do this with C/C++ header files for safety!

Implementation of Classes

```
//deposit money into the machine
void Soda_Machine::insert_money(double amount)
{
    deposit += amount;
}
```

- Class methods are typically implemented in a cpp file.

Implementation of Classes

```
//deposit money into the machine
void Soda_Machine::insert_money(double amount)
{
    deposit += amount;
}
```

- Class methods are typically implemented in a cpp file.
- Method names are prefixed with the name of the class and the scope resolution modifier.

Implementation of Classes

```
//deposit money into the machine
void Soda_Machine::insert_money(double amount)
{
    deposit += amount;
}
```

- Class methods are typically implemented in a cpp file.
- Method names are prefixed with the name of the class and the scope resolution modifier.
- The same is true of constructors.

Implementation of Classes

```
//deposit money into the machine
void Soda_Machine::insert_money(double amount)
{
    deposit += amount;
}
```

- Class methods are typically implemented in a cpp file.
- Method names are prefixed with the name of the class and the scope resolution modifier.
- The same is true of constructors.
- Take a look in `soda-machine.cpp` to see this in action.

Using Objects

- Take a look at `sodasim.cpp` to see how we use a class.

Using Objects

- Take a look at `sodasim.cpp` to see how we use a class.
- First, we must make an object. This is called **instantiation** of the class.

```
Soda_Machine machine;  //the machine
```

Using Objects

- Take a look at `sodasim.cpp` to see how we use a class.
- First, we must make an object. This is called **instantiation** of the class.

```
Soda_Machine machine;  //the machine
```

- We can interact with the object using its member functions:

```
machine.insert_money(money);
```

Using Objects

- Take a look at `sodasim.cpp` to see how we use a class.
- First, we must make an object. This is called **instantiation** of the class.

```
Soda_Machine machine;  //the machine
```

- We can interact with the object using its member functions:

```
machine.insert_money(money);
```

- Take a look at the rest of the `main` function to see how it interacts with our machine. Isn't the realism thrilling?

Using Objects

- Take a look at `sodasim.cpp` to see how we use a class.
- First, we must make an object. This is called **instantiation** of the class.

```
Soda_Machine machine;  //the machine
```

- We can interact with the object using its member functions:

```
machine.insert_money(money);
```

- Take a look at the rest of the `main` function to see how it interacts with our machine. Isn't the realism thrilling?
- Compile the program using the following command:

```
g++ sodasim.cpp soda-machine.cpp -o sodasim
```

Using Objects

- Take a look at `sodasim.cpp` to see how we use a class.
- First, we must make an object. This is called **instantiation** of the class.
  ```
  Soda_Machine machine;  //the machine
  ```
- We can interact with the object using its member functions:
  ```
  machine.insert_money(money);
  ```
- Take a look at the rest of the `main` function to see how it interacts with our machine. Isn't the realism thrilling?
- Compile the program using the following command:
  ```
  g++ sodasim.cpp soda-machine.cpp -o sodasim
  ```
- Play with it and see what it does.

Best Practices Recap

- A class name should begin with a capital letter.

Best Practices Recap

- A class name should begin with a capital letter.
- Each class should be created in two files:

Best Practices Recap

- A class name should begin with a capital letter.
- Each class should be created in two files:
  - A header file (definition)

Best Practices Recap

- A class name should begin with a capital letter.
- Each class should be created in two files:
  - A header file (definition)
  - A cpp file (implementation)

Best Practices Recap

- A class name should begin with a capital letter.
- Each class should be created in two files:
  - A header file (definition)
  - A cpp file (implementation)
- Member variables should be private.

Best Practices Recap

- A class name should begin with a capital letter.
- Each class should be created in two files:
  - A header file (definition)
  - A cpp file (implementation)
- Member variables should be private.
- Member methods should usually be public.

Best Practices Recap

- A class name should begin with a capital letter.
- Each class should be created in two files:
  - A header file (definition)
  - A cpp file (implementation)
- Member variables should be private.
- Member methods should usually be public.
- Constructors should usually be public.

Best Practices Recap

- A class name should begin with a capital letter.
- Each class should be created in two files:
  - A header file (definition)
  - A cpp file (implementation)
- Member variables should be private.
- Member methods should usually be public.
- Constructors should usually be public.
- Always provide a constructor.

Finishing the Program

1. Make the directory `labs/week12`
2. From your `cs1` directory, copy all of the files like this:
   `cp examples/19-Classy/* labs/week12`
3. Implement the rest of the class.
4. Implement button pushing in `sodasim.cpp`

Button Pushing Pseudocode

```
push_button(button)
    if deposit >= cost of the soda
        if that soda is sold out
          return "Sold Out"
        else
          vend the soda
    else if soda is sold out
      return "Sold Out"
    else
      return The brand and the cost of soda
```

Vending Pseudocode

```
vend(soda)
    Subtract 1 from the available quantity of soda
    move the cost of the soda to the vault
    print a message indicating vending and brand
```