(5 - 1) Object-Oriented Programming (OOP) and C++

Instructor - Andrew S. O'Fallon

CptS 122 (February 17, 2021)

Washington State University

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Key Concepts

- Object-Oriented Design
- Object-Oriented Programming (OOP)
- Class and Objects
- Data Encapsulation
- Abstraction/Information Hiding
- C++ I/O
- References and Reference Parameters
- Unary Scope Resolution Operator
- Function Overloading

Object-Oriented Design (OOD)

- Model software in ways that are similar to how people view/describe real-world objects
- Descriptions and designs include properties or attributes of the real-world objects
- The Unified Modeling Language (UML) provides a specification for illustrating properties of objects along with interactions between them

Object-Oriented Programming (OOP) (I)

- Programming language model which institutes mechanisms to support implementing object driven software systems
  - C++, C#, Java
- Procedural programming, such as instituted by C, is action oriented
- In C, the unit of programming is a function
- In C++ the unit is a `class`

Object-Oriented Programming (OOP) (II)

- We'll explore OOP with classes, encapsulation, objects, operator overloading, inheritance, and polymorphism
- We'll also explore generic programming with function templates and class templates

Classes and Objects

- What is a `class`?
  - A user defined type or data structure
  - Contains data members (attributes) and member functions (operations)
  - A blueprint for an object
- What is an object?
  - An instantiation of a class
  - The class is the type and the object is the variable with allocated memory for that type

Data Encapsulation (I)

- A way of organizing or wrapping of data/attributes and methods/operations into a structure (or capsule)
  - Demonstrated by objects
- Objects naturally impose encapsulation – attributes and operations are closely tied together
- How does making a function or class a `friend` of another class impact encapsulation?

Abstraction/Information Hiding (I)

- A design principle which states a design decision should be hidden from the rest of the system
- In other words, objects should communicate with each other through well-defined interfaces, but not know how other objects are implemented

Abstraction/Information Hiding (II)

- Prevents access to data aside from the methods specified by the object
- Guarantees integrity of data
- Access specifiers in C++ control the access to information
  - `public`, `protected`, and `private`

Programming in C++

- When programming in an object-oriented language, we'll be exposed to encapsulation, abstraction, and information hiding in action
- We need to start thinking in an object-oriented way so that we can leverage the software design benefits of objects and the richness of C++!
- Always remember, objects contain data and associated operations!

Basics of C++ and I/O (I)

- In C++, just like in C, every program begins execution with function main ()
- To perform input and output (I/O) we need to include the C++ Standard Library `<iostream>`
  - Essentially replaces `<stdio.h>`, but with even more richness and convenience

Basics of C++ and I/O (II)

- In tandem with including `<iostream>`, we'll need to use the following:
  - A standard output stream object (std::cout) and stream insertion operator (<<) to display information on the screen
    - Replaces the need for printf ()
  - A standard input stream object (std::cin) and the stream extraction operator (>>) to read data from the keyboard
    - Replaces the need for scanf ()

Basics of C++ and I/O Example

```
#include <iostream>
using std::cin; // replaces need for std:: in front of cin
using std::cout; // replaces need for std:: in front of cout
using std::endl; // replaces need for std:: in front of endl
int main (void)
{
        int n1 = 0;
        cout << "Enter a number: ";
        cin >> n1; // Notice no address of (&) required!

        int n2 = 0, sum = 0; // Can declare variables right
                             // before their use in C++!
        cout << "Enter a second number: ";
        cin >> n2;
        sum = n1 + n2;
        cout << "The sum is: " << sum << endl; // endl outputs a
                                               // newline, then flushes buffer
        return 0;
} A. O'Fallon, J. Hagemeister
```

References and Reference Parameters (I)

- There are two ways to pass arguments to functions in C++
  - Pass-by-value (PBV) – a copy of the contents/value of each argument is made and passed (on the function call stack) to the called function
    - One disadvantage of pass-by-value is copying the contents of a large data item or object introduces longer execution times and memory space
    - In general, should only be used with simple types
    - Passing-by-pointer falls under this category
  - Pass-by-reference (PBR) – NO copy of the contents/value of each argument is made
    - The called function can access the caller's data directly, and modify the data

References and Reference Parameters (II)

- Thoughts: we don't use pass-by-reference strictly so that we can modify the data in an object directly, in many cases we use it so that the overhead of copying data is circumvented
- We use the ampersand (&) to represent pass-by-reference
  - i.e. void cube (int **&n**); // this is a prototype
  - Don't confuse with the address of (&) operator! Context determines which one's in play!
- Check out: http://www.cplusplus.com/articles/z6vU7k9E/

References and Reference Parameters (III)

- We can return a reference to a variable as well – however we have to be very careful!
  - i.e. **int &** someFunction (int &n);
- If we return a reference to an automatic local variable, the variable becomes "undefined" when the function exits; unless the variable is declared as "static" (keyword)
  - References to undefined variables are called *dangling* references
  - Note: dangling references and dangling pointers are NOT the same!

References and Reference Parameters Example

```
…
void cubeByRef (int &n);
void cubeByPtr (int *pN);
int main (void)
{
        int n = 5;
        cubeByRef (n); // Don't need &, the formal parameter list indicates PBR
        cubeByPtr (&n); // Need address of (&) operator to satisfy pointer; applying PBV
…
}
void cubeByRef (int &n)
{
        n = n * n * n; // We have direct access to n, don't need to dereference;
                       // changes are retained
}
void cubeByPtr (int *pN)
{
        *pN = (*pN) * (*pN) * (*pN); // Need to dereference to indirectly change value
}
```


Unary Scope Resolution Operator

- It's possible to declare local and global variables of the same name
  - Unary Scope Resolution Operator (::) allows a global variable to be accessed without confusing it with a local variable

```
...
int num = 42; // global variable
int main (void)
{
    double num = 100.25; // local variable
    cout << num << endl; // displays 100.25
    cout << ::num << endl; // displays 42
}
```


Function Overloading (I)

- The ability to define multiple functions with the same name
  - Requires that each function has different types of parameters and/or different number of parameters and/or different order of parameters
  - i.e. `int cube (int n);`
    `double cube (double n);`
- The C++ compiler selects the appropriate function based on the *number*, *types*, and *order* of arguments in the function *call*

Function Overloading (II)

- We use function overloading to increase readability and understandability
  - Of course, we only want to overload functions that perform similar tasks

C++ Standard Template Library (STL) Class Vector

- STL class `vector` represents a more robust array with many more capabilities
- May operate with different types of data because they're templated!
  - i.e.

```
vector<int> v1(10); // declares a 10 element
                    // vector of integers

vector<double> v2(5); // declares a 5 element vector
                      // of doubles
```


Closing Thoughts

- OOP and C++ opens us up to an entirely different world!
- We need to start thinking more in terms of data and “capsules” instead of just actions and logic
- Learning C++ is a challenge, but provides features that will increase levels of production!

References

- P.J. Deitel & H.M. Deitel, *C++: How to Program* (9th ed.), Prentice Hall, 2014.

Collaborators

- Jack Hagemeister