Queens College, CUNY, Department of Computer Science

Object-oriented programming in C++

CSCI 211 / 611

Summer 2018

Instructor: Dr. Sateesh Mane

© Sateesh R. Mane 2018

Project 3, part 1

due date Monday August 13, 2018, 11:59 pm

- **NOTE: It is the policy of the Computer Science Department to issue a failing grade to any student who either gives or receives help on any test. A student caught cheating on any question in an exam, project or quiz will fail the entire course.**
- **Students who form teams to collaborate on projects *must inform the lecturer of the names of all team members* ahead of time, else the submissions will be classified as cheating and will receive a failing grade.**
- **Any problem to which you give two or more (different) answers receives the grade of zero automatically.**
- Please submit your solution via email, as a file attachment, to `Sateesh.Mane@qc.cuny.edu`.
- Please submit one zip archive with all your files in it.
  1. The zip archive should have either of the names (CS211 or CS611):

     `StudentId_first_last_CS211_project3_pt1_Aug2018.zip`
     `StudentId_first_last_CS611_project3_pt1_Aug2018.zip`
  2. The archive should contain one "text file" named "Project3.[txt/docx/pdf]" (if required) and one cpp file per question named "Q1.cpp" and "Q2.cpp" etc.
  3. Note that not all questions may require a cpp file.
  4. A text file is not always required for every project.
- **In all questions where you are asked to submit programming code, programs which display any of the following behaviors will receive an automatic F:**
  1. Programs which do not compile successfully (non-fatal compiler warnings are excluded).
  2. Array out of bounds, reading of uninitialized variables (including null pointers).
  3. Operations which yield NAN or infinity, e.g. divide by zero, square root of negative number, etc. *Infinite loops.*
  4. Programs which do NOT implement the public interface stated in the question.
- **In addition, note the following:**
  1. All debugging statements (for your personal testing) should be commented out.
  2. Program performance will be graded solely on the public interface stated in the questions.

1 Templated class `Vec`: method `erase`

- **Write a public method `erase` for the templated `Vec` class.**

```
template<typename T> class Vec {
public:
  void erase(int n);
  // etc
};
```

- We require this method to be able to erase (delete) messages from anywhere in a folder, not just the end.
- The functionality is obvious, we want to erase the element at the location n.
- Test test if $n \geq 0$ and `n < _size`.
  1. If `false`, then do nothing and exit.
  2. If `true`, then copy `_vec[n+1]` to position `_vec[n]`, copy `_vec[n+2]` to position `_vec[n+1]`, etc., until the end of the array `_vec`.
  3. Then set `_size = _size-1` and exit the function.
- Notice that `pop_back` is a special case of `erase`.
- However, `pop_back` can be implemented much more efficiently than `erase`.
- Do not change `pop_back`.

2 Class Name

- **Write a class Name.**

```
class Name {
public:
  Name() {}
  Name(string s) { set(s); }

  void set(string s);                              // mutator
  string name() const { return _name; }            // accessor
  string address() const { return _address; }      // accessor

private:
  string _name;
  string _address;
};
```

- The string "_name" is the "short name" that appears in a message display.
- The string "_address" is the full email address, with the domain name included.
- The accessors are obvious.
- We require a default constructor because draft messages can be created with no names.
- **We require the header <sstream> for use below.**
- **Write a mutator set as follows.**

1. ```
   void set(string s) {
     const string _domain = "@qc.cuny.edu";
     istringstream iss(s);
     iss >> _name;                 // remove leading and trailing blank spaces
     // see below
   }
   ```
2. If _name is blank, **set _address = "" (blank) and exit the function.**
3. If _name is not blank, **set _address = _name + _domain.**
4. **Also convert _address to lower case.**
   ```

3 Operator overload

- **Overload `operator==` and `operator<` and for two `Name` objects `n1` and `n2`.**
- **Compare only the addresses `n1.address()` and `n2.address()` in the operators.**

4 Class Message

4.1 Class declaration

- **Write a class Message.**
- It is similar to Message1 from Project 2, but we use Name objects for _from and _to.
- The "addDomain" method is gone. It has effectively been transferred to the Name class.
- Also the date is filled in.
- **Include the headers <ctime> and <chrono>.**

```
class Message {
public:
  Message(string f);
  Message(string f, string t);
  Message(string f, string t, string s);
  Message(string f, string t, string s, string txt);

  const Name& from() const { return _from; }
  const Name& to() const { return _to; }
  string subject() const { return _subject; }
  string text()    const { return _text; }

  string date()    const;                          // see below
  void send();                                     // see below

  void setRecipient(string t) { _to.set(t); }
  void setSubject(string s) { _subject = s; }
  void setText(string txt) { _text = txt; }

  void prependText(string t);                      // see Message1 class
  void appendText(string t);                       // see Message1 class
  void print() const;                              // see Message1 class


private:
  void setDate();                                  // see below

  Name _from;                                      // use "Name" class
  Name _to;                                        // use "Name" class
  string _subject;
  string _text;
  time_t _date;                                    // see below
};
```

4.2 Constructors

- **Change all the "`_from`" and "`_to`" fields to use the "`set`" method of the `Name` class.**

```
_from.set(f);
_to.set(t);
```

- **Set `_date=0` in all the constructors.** Use memberwise initialization if desired.
- Note also the "`setRecipient`" method.

```
void setRecipient(string t) { _to.set(t); }
```

4.3 Print

- **The "print" method must be modified because we are using `name` objects.**
- We must invoke the "name()" and "address()" methods of the `Name` class.

```
void print() const {
  cout << "From: " << _from.name() << "    <" << _from.address() << ">" << endl;
  cout << "To:   " << _to.name()   << "    <" << _to.address() << ">" << endl;
  cout << "Subject: " << _subject << endl;
  cout << "Date: " << date() << endl;
  cout << _text << endl;
  cout << endl;
}
```

4.4 Date

- The data member `_date` is a "`time_t`" struct.
- **Write the private method `setDate()` as follows.**

```
void setDate() {
  auto t_now = std::chrono::system_clock::now();
  _date = std::chrono::system_clock::to_time_t(t_now);
}
```

- Do not worry if you do not understand. It is complicated.
- **Write the public method `date()` as follows.**

```
string date()    const {
  if (_date > 0)
    return std::ctime(&_date);
  else
    return "";
}
```

- Do not worry if you do not understand. It is complicated.
- However, this will format the date correctly in a nice readable format.
- **Write the public method `send()` as follows.**

```
void send() { setDate(); }
```

- This will also be explained later.
- Basically, when a draft message is sent, that is when we set the date of the message.

5 Summary

- You should now have a templated `Vec` class which is suitable for our needs.
- You also have a `Name` class and overloaded comparison operators, which will be usful.
- You also have a `Message` class, with a completed functionality for the date.
- There is much more work to be done.
- We require an email account class and also folder classes to contain messages.
- But this is the first step, the messages.