Queens College, CUNY, Department of Computer Science

Object-oriented programming in C++

CSCI 211 / 611

Summer 2018

Instructor: Dr. Sateesh Mane

Project 3, part 3

due date Tuesday August 14, 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_Aug2018.zip`
     `StudentId_first_last_CS611_project3_Aug2018.zip`
  2. The archive should contain one "text file" named "Part3.[txt/docx/pdf]" (if required) and one cpp file per question named "Pt3_Q1.cpp" and "Pt3_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 Polymorphic folder classes `BaseFolder`, `Inbox`, `Outbox`

- The class declarations are as follows.
- The base class contains protected data as follows.
  1. `Vec<const Message*> _msg` of a Vec of `const` pointers to messages.
  2. **You may employ STL `vector<const Message*> _msg`, but for reduced credit.**
  3. A pointer `EmailAccount *_ac` to the email account which owns the folder.
  4. All folder objects have a parent `EmailAccount` pointer.
- The list of non-virtual functions is also shown (all are public).
- The virtual functions are all protected methods. Make them pure virtual if you wish..

```
class BaseFolder {
public:
  int size() const { return _msg.size(); }

  void display() const;
  void erase(int n);
  void forward(int n) const;
  void print(int n) const;
  void receive(const Message *m);
  void reply(int n) const;

protected:
  virtual string type() const;                          // pure virtual if desired
  virtual const Name& tofrom(const Message *m) const;  // pure virtual if desired

// data
  Vec<const Message*> _msg;
  EmailAccount *_ac;
};
```

- **Write a virtual destructor.**
  Recall that the vector contains pointers to messages.
  Run a loop and invoke `operator delete` to release all dynamic memory.
- **Write functions to (i) make a deep copy, else (ii) disable copies by making the relevant functions private.** It is your choice.

- **Write the constructor (non-default) as follows.**

```
BaseFolder(EmailAccount *ac) : _ac(ac) { _msg.push_back(NULL); }
```

  1. We push back a NULL message onto `_msg` in the constructor.
  2. This is so that the displayed messages are numbered 1,2,... which looks prettier than 0,1,2,...

- **To make the class abstract, do either of the following (or both).** It is your choice.

  1. Make the constructor protected.
  2. Declare the virtual functions as pure virtual.
  3. If you do not make them pure virtual, then make the functions perform nothing.

- **The declarations of the derived classes are as follows.**

  1. *The derived classes are easy and we shall write them first.*
  2. This is the way things are done in real life. The base class is not always written first.
  3. The derived classes have public constructors and override the virtual function `tofrom`.

```
// Inbox
class Inbox : (inherit) {
public:
  Inbox(EmailAccount *ac);                      // public constructor
 protected:
  virtual string type() const;                             // override
  virtual const Name& tofrom(const Message *m) const;    // override
};

// Outbox
class Outbox : (inherit) {
public:
  Outbox(EmailAccount *ac);                     // public constructor
protected:
  virtual string type() const;                             // override
  virtual const Name& tofrom(const Message *m) const;    // override
};
```

- **Write the constructors of both derived classes to invoke the non-default constructor of the base class.**

- The virtual function `tofrom` is required to specify whether we wish to print "to" or "from" in displays, etc. **Override the virtual functions as follows.**

```
// Inbox
string type() const { return "inbox"; }
const Name& tofrom(const Message *m) const { return m->from(); }

// Outbox
string type() const { return "outbox"; }
const Name& tofrom(const Message *m) const { return m->to(); }
```

- *That's it. We have completed the derived classes.*

- We now complete the base class.
- **Write the non-virtual method `receive` as follows.**
  Push back m **onto** `_msg`**.** That's all.
- **Write the non-virtual method `print(int n)` as follows.**
  1. If `n < 1` or `n >= (size of vector)` then do nothing.
  2. If `n >= 1` and `n < (size of vector)` then invoke `print()` for message n.
- **Write the non-virtual method `erase(int n)` as follows.**
  1. If `n < 1` or `n >= (size of vector)` then do nothing.
  2. If `n >= 1` and `n < (size of vector)` then do the following.

     ```
     if // tests
     {
       // release memory  (operator delete)
       _msg.erase(n);
     }
     ```

     (a) Release memory for message n (operator `delete`).
     (b) Erase the entry in the vector.

- **Write the non-virtual function `display` as follows.**

  1. Print the account owner's name (and the type of the in/outbox).
  2. Test if there are any messages to display.

     ```
     cout << _ac->owner().name() << "  " << type() << endl;
     if (_msg.size() <= 1) {
       cout << "no messages to display" << endl;
     }
     ```

  3. *Comment out the "account" line for now, because the accounts class is not written yet.*
  4. If there are messages to display, run a loop over the messages, **starting from $i = 1$ to skip the initial NULL message.**
  5. In each line, print the following:
     (a) Print the value of i.
     (b) Declare a temporary variable `const Name &tmp = tofrom(_msg[i])`.
     (c) Print `tmp.name()`.
     (d) Print `_msg[i]->subject()`.
  6. Hence the overall display is as follows.

     ```
     for (int i = 1; i < _msg.size(); ++i) {
       // tmp = ...
       // print  i,   tmp.name(),   _msg[i]->subject()
     }
     ```

- The non-virtual functions `reply` and `forward` are similar.
- **Write the non-virtual method `forward(int n)` as follows.**
  1. If `n < 1` or `n >= (size of vector)` then do nothing.
  2. If `n >= 1` and `n < (size of vector)` then do the following.
  3. Create a temporary string `fwd_subject = "Fwd:  " + m->subject()`.
  4. The "to" field is a blank string.

     ```
     const Message *m = _msg[n];
     string fwd_subject = ...
     Message *ptr = new Message(_ac->owner().name(), "", fwd_subject, m->text());
     _ac->insert( ptr );
     ```

  5. The last line is to tell the account object to insert the message into the Drafts folder.
  6. *Comment out the lines involving the account for now, because the accounts class is not written yet.*
- **Write the non-virtual method `reply(int n)` as follows.**
  1. If `n < 1` or `n >= (size of vector)` then do nothing.
  2. If `n >= 1` and `n < (size of vector)` then do the following.
  3. Create a temporary string `subject = "Re:  " + m->subject()`.
  4. Declare temporary `const Name &tmp = tofrom(m)`.

     ```
     const Message *m = _msg[n];
     string subject = ...
     const Name& tmp = tofrom(m);
     Message *ptr = new Message(_ac->owner().name(), tmp.name(), subject, m->text());
     _ac->insert( ptr );
     ```

  5. The last line is to tell the account object to insert the message into the Drafts folder.
  6. *Comment out the lines involving the account for now, because the accounts class is not written yet.*