Queens College, CUNY, Department of Computer Science

Object-oriented programming in C++

CSCI 211 / 611

Summer 2018

Instructor: Dr. Sateesh Mane

Project 3, part 2

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 "Part2.[txt/docx/pdf]" (if required) and one cpp file per question named "Pt2_Q1.cpp" and "Pt2_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 Forward declarations, non-inline code, static initialization

- The C++ classes described below make references to each other.
- **It is your responsibility to write all forward declarations correctly.**
- **It is your responsibility to write all non-inline function definitions correctly.**
- **It is your responsibility to write all static initialization statements correctly.**

2 Project overview

- A schematic is as follows.
- The class `ISP` ("internet service provider") has a map of email accounts.

ISP		
Account	Account	Account

- Each account has three folders (Drafts, Inbox, Outbox).

Account		
Drafts	Inbox	Outbox
message	message	message
⋮	⋮	⋮

- The messages in Drafts are read/write.
- The messages in Drafts support a "send" function.
- The messages in Inbox and Outbox are read-only.
- The messages in Inbox and Outbox support "reply" and "forward" functions.
- A new message can be created in Drafts only.
- Messages in any folder can be deleted, not only at the end of the folder but anywhere.
- All folders support a "display" command to print the list of messages in that folder.
- A message object supports a "print" command to print the contents of the message.

3 Sending a message

- The Drafts folder supports a function "`send(int n)`" to send message n in the folder.
- If the value of n is not valid, an error message is printed.
- If the value of n is valid, several things happen.
  1. The date is set in the message object (the code to do this is given to you).
  2. The Account object transfers the message from the Drafts to the Outbox folder.
  3. The Account object passes a pointer to the message to the ISP.
  4. The ISP reads the "to" field of the message and searches for a match in its map of accounts.
  5. If there is no match, an error message "Delivery failed" is printed.
  6. If there is a match, a copy of the message is given to the target account.
  7. The target account adds the incoming message to its Inbox folder.

4 Reply and forward

- The messages in the Inbox and Outbox support the “reply” and “forward” commsnds.
- The “reply” command places a copy of the message in the Drafts folder.
    1. The “from” and “to” fields of the original message are interchanged.
    2. The subject string is changed to “Re: (original subject)” in the copy in Drafts.
    3. It is allowed to invoke “reply” for the same message multiple times. Each function call creates a new copy in the Drafts.
- The “forward” command places a copy of the message in the Drafts folder.
    1. The “from” field is set to the owner of the account and the “to” field is blank.
    2. The subject string is changed to “Fwd: (original subject)” in the copy in Drafts.
    3. It is allowed to invoke “forward” for the same message multiple times. Each function call creates a new copy in the Drafts.

5 List of classes and polymorphism

- This is the list of classes in the project.
    1. BaseFolder
        (a) Inbox
        (b) Outbox
    2. Drafts
    3. EmailAccount
    4. ISP
    5. Message
    6. Name
    7. Vec (templated).
- The Inbox and Outbox both derive from a polymorphic abstract base class `BaseFolder`.
- The Inbox and Outbox override virtual functions.
- The Drafts folder has different functionality and is a separate class.
- The Drafts folder does not derive from `BaseFolder`.