```
// Email, Shwitter and Inheritance
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

class Message
{
private:
[TAB]string message;
[TAB]string author;
[TAB]static bool isValidAuthor(string author);
[TAB]static bool isValidMessage(string message);

public:
[TAB]static const string DEFAULT_MESSAGE;
[TAB]static const int MAX_EMAIL_ADDRESS_LENGTH = 60;
[TAB]static const int MIN_EMAIL_ADDRESS_LENGTH = 1;
[TAB]static const string DEFAULT_EMAIL_ADDRESS;
[TAB]static const string DEFAULT_TO_EMAIL_ADDRESS;
[TAB]static const int MAX_MSG_LENGTH = (1000 * 1000);
[TAB]static const int MIN_MSG_LENGTH = 0;
[TAB]static const string DEFUALT_AUTHOR;
[TAB]static const int MAX_AUTHOR_LENGTH = 50;
[TAB]static const int MIN_AUTHOR_LENGTH = 2;
[TAB]static const int MAX_SHWITTER_ID_LENGTH = 15;
[TAB]static const int MIN_SHWITTER_ID_LENGTH = 1;
[TAB]static const int MAX_SHWITTER_LENGTH = 140;
[TAB]static const string DEFAULT_USER_ID;

[TAB]Message();
[TAB]Message(const string theAuthor, const string theMessage);

[TAB]string getMessage() const { return message; }
[TAB]string getAuthor() const { return author; }
[TAB]string toString();

[TAB]bool setMessage(string newMessage);
[TAB]bool setAuthor(string newAuthor);
};
const string Message::DEFUALT_AUTHOR = "UNKNOWN AUTHOR";
const string Message::DEFAULT_MESSAGE = "SPARKLING DEFAULT MESSAGE";
const string Message::DEFAULT_EMAIL_ADDRESS = "EMAIL_DEFAULT@GMAIL.COM";
const string Message::DEFAULT_TO_EMAIL_ADDRESS = "TO_DEFAULT_EMAIL@GMAIL.COM";
const string Message::DEFAULT_USER_ID = "@DEFAULT_ID";

class Email : public Message
{
private:
[TAB]string fromAddress;
[TAB]string toAddress;
[TAB]static bool fromAddressValid(string fromAddress);
[TAB]static bool toAddressValid(string toAddress);
[TAB]static bool validAddress(string validAdrress);

public:
[TAB]Email();
[TAB]Email(const string theAuthor, const string theMessage,
[TAB][TAB]  const string theFromAdress, const string theToAddress);

[TAB]string getFromAddress() const { return fromAddress; }
```

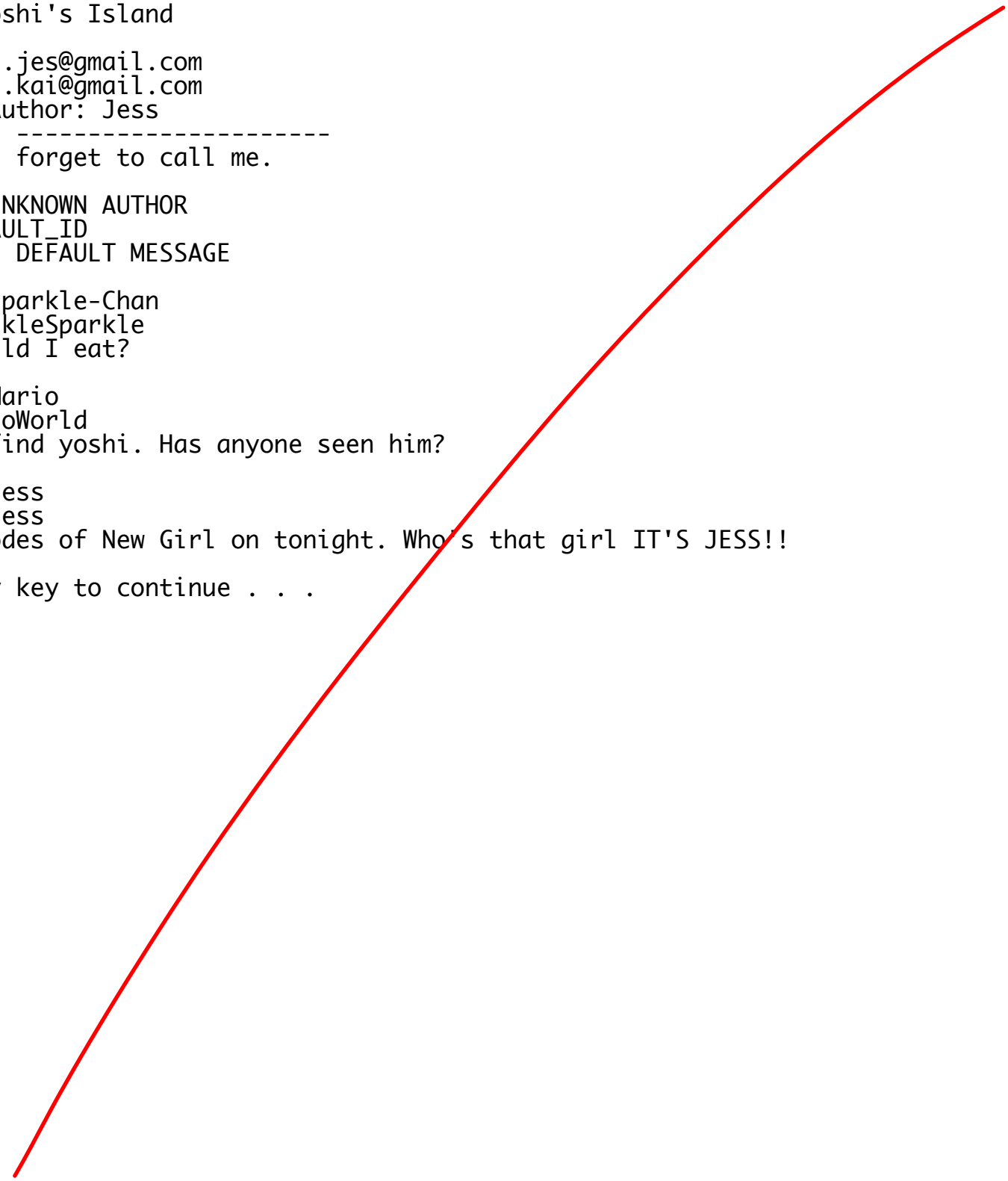
```
[TAB]string getToAdress() const { return toAddress; }
[TAB]string toString();

[TAB]bool setFromAddress(string newFromAddress);
[TAB]bool setToAddress(string newToAddress);
};

class Shweet : public Message
{
private:
[TAB]string fromId;
[TAB]static bool fromIdValid(string fromId);
[TAB]static bool validID(string validID);

public:
[TAB]Shweet();
[TAB]Shweet(const string theAuthor, const string theMessage,
[TAB][TAB]   const string theFromId);

[TAB]string getFromId() const { return fromId; }
[TAB]string toString();

[TAB]bool setFromId(string newFromId);
};

int main()
{
[TAB]Message message1;
[TAB]Message message2("Kai-Kun", "Don't forget to study your Japanese.");
[TAB]Message message3("Sparkle-Chan", "You are so sparklely ");
[TAB]Message message4("Yoshi", "Where is Mario?");
[TAB]
[TAB]cout << message1.toString() << message2.toString() << message3.toString() <
< message4.toString();
[TAB]
[TAB]Email email1;
[TAB]Email email2("Sparkle-Chan", "Hi Girl, I can't for summer to go to the beac
h!!", "lilly@gmail.com", "beach.girl@gmail.com");
[TAB]Email email3("Mario", "I'm on Yoshi's Island", "the.original.green.yoshi@gm
ail.com", "mario.world@gmail.com");
[TAB]Email email4("Jess", "Hi, Don't forget to call me.", "ocean.kai@gmail.com",
 "its.jes@gmail.com");

[TAB]cout << email1.toString() << email2.toString() << email3.toString() << emai
l4.toString();


[TAB]Shweet shweet1;
[TAB]Shweet shweet2("sparkle-Chan", "What should I eat?", "@sparkleSparkle");
[TAB]Shweet shweet3("Mario", "I can't find yoshi. Has anyone seen him?", "@mario
World");
[TAB]Shweet shweet4("Jess", "New episodes of New Girl on tonight. Who's that gir
l IT'S JESS!!", "@itsJess");

[TAB]cout << shweet1.toString() << shweet2.toString() << shweet3.toString() << s
hweet4.toString();

}
Message::Message() : author(DEFUALT_AUTHOR), message(DEFAULT_MESSAGE){}

Message::Message(const string theAuthor, const string theMessage) : message(theM
essage), author(theAuthor){}
```

```
bool Message::setAuthor(const string newAuthor)
{
[TAB]if (!isValidAuthor(newAuthor))
[TAB][TAB]return false;
[TAB]author = newAuthor;
[TAB]return true;
}

bool Message::isValidAuthor(string newAuthor)
{
[TAB]if (newAuthor.length() >= MIN_AUTHOR_LENGTH && newAuthor.length() <= MAX_AU
THOR_LENGTH)
[TAB][TAB]return true;
[TAB]return false;
}
bool Message::setMessage(const string newMessage)
{
[TAB]if (!isValidMessage(newMessage))
[TAB][TAB]return false;
[TAB]message = newMessage;
[TAB]return true;
}
bool Message::isValidMessage(string newMessage)
{
[TAB]if (newMessage.length() >= MIN_MSG_LENGTH && newMessage.length() <= MAX_MSG
_LENGTH)
[TAB][TAB]return true;
[TAB]return false;
}
string Message::toString()
{
[TAB]string results;

[TAB]results =
[TAB][TAB]"Author: " + getAuthor() + "\n  Message ----------------------\n"
[TAB][TAB]+ getMessage() + " \n\n";
[TAB]return results;
}
Email::Email() : Message(), fromAddress(DEFAULT_EMAIL_ADDRESS), toAddress(DEFAUL
T_TO_EMAIL_ADDRESS) {}

Email::Email(const string theAuthor, const string theMessage,
[TAB]const string theToAddress, const string theFromAdress) : Message(theAuthor,
 theMessage),
[TAB]toAddress(theToAddress), fromAddress(theFromAdress){}

bool Email::setFromAddress(string newFromAddress)
{
[TAB]if (!fromAddressValid(newFromAddress))
[TAB][TAB]return false;
[TAB]fromAddress = newFromAddress;
[TAB]return true;
}

bool Email::setToAddress(string newToAddress)
{
[TAB]if (!toAddressValid(newToAddress))
[TAB][TAB]return false;
[TAB]toAddress = newToAddress;
[TAB]return true;
}
```

```
bool Email::fromAddressValid(string newFromAddress)
{
[TAB]if (newFromAddress.length() >= MIN_EMAIL_ADDRESS_LENGTH && newFromAddress.l
ength() <= MAX_EMAIL_ADDRESS_LENGTH)
[TAB][TAB]return true;
[TAB]return false;
}

bool Email::validAddress(string validAddress)
{
[TAB]bool hasAt = false;
[TAB]bool hasDot = false;
[TAB]for (int i = 0; i < validAddress.length(); i++)
[TAB]{
[TAB][TAB]if (hasAt && hasDot)
[TAB][TAB][TAB]if (i = '@')
[TAB][TAB][TAB]{
[TAB][TAB][TAB][TAB]hasAt = true;
[TAB][TAB][TAB]}
[TAB][TAB]if (i = '.')
[TAB][TAB]{
[TAB][TAB][TAB]hasDot = true;
[TAB][TAB]}
[TAB][TAB]return true;
[TAB]}
[TAB]return false;
}
bool Email::toAddressValid(string newToAddress)
{
[TAB]if (newToAddress.length() >= MIN_EMAIL_ADDRESS_LENGTH && newToAddress.lengt
h() <= MAX_EMAIL_ADDRESS_LENGTH)
[TAB][TAB]return true;
[TAB]return false;
}
string Email::toString()
{
[TAB][TAB]string results;

[TAB][TAB]results =
[TAB][TAB][TAB]"From: " + getFromAddress() + "\nTo: " + getToAdress() + "\nAutho
r: "
[TAB][TAB][TAB]+ Message::toString();
[TAB][TAB]/*getAuthor() + "\n  Message ----------------------\n"
[TAB][TAB]+ getMessage() + " \n\n";*/
[TAB]return results;
}

Shweet::Shweet() : Message(), fromId(DEFAULT_USER_ID){}

Shweet::Shweet(const string theAuthor, const string theMessage,
[TAB]const string theFromId) : Message(theAuthor, theMessage), fromId(theFromId)
{}

bool Shweet::setFromId(string newFromId )
{
[TAB]if (!fromIdValid(newFromId))
[TAB][TAB]return false;
[TAB]fromId = newFromId;
[TAB]return true;
}
bool Shweet::fromIdValid(string newFromId)
```

```
{
[TAB]if (newFromId.length() >= MIN_SHWITTER_ID_LENGTH && newFromId.length() <= M
AX_SHWITTER_ID_LENGTH)
[TAB][TAB]return true;
[TAB]return false;
}
bool Shweet::validID(string validID)
{
[TAB]bool hasAt = false;
[TAB]
[TAB]for (int i = 0; i < validID.length(); i++)
[TAB]{
[TAB][TAB]if (hasAt)
[TAB][TAB][TAB]if (i = '@')
[TAB][TAB][TAB]{
[TAB][TAB][TAB][TAB]hasAt = true;
[TAB][TAB][TAB]}
[TAB][TAB]return true;
[TAB]}
[TAB]return false;
}
string Shweet::toString()
{
[TAB]string results;

[TAB]results =
[TAB][TAB]"Shweet: " + getAuthor() + "\nID: " + getFromId() + "\n"
[TAB][TAB]+ getMessage() + " \n\n";
[TAB]return results;
}

/*
Author: UNKNOWN AUTHOR
  Message ----------------------
SPARKLING DEFAULT MESSAGE

Author: Kai-Kun
  Message ----------------------
Don't forget to study your Japanese.

Author: Sparkle-Chan
  Message ----------------------
You are so sparklely.

Author: Yoshi
  Message ----------------------
Where is Mario?

From: EMAIL_DEFAULT@GMAIL.COM
To: TO_DEFAULT_EMAIL@GMAIL.COM
Author: Author: UNKNOWN AUTHOR
  Message ----------------------
SPARKLING DEFAULT MESSAGE

From: beach.girl@gmail.com
To: lilly@gmail.com
Author: Author: Sparkle-Chan
  Message ----------------------
Hi Girl, I can't for summer to go to the beach!!

From: mario.world@gmail.com
To: the.original.green.yoshi@gmail.com
```

```
Author: Author: Mario
  Message ----------------------
I'm on Yoshi's Island

From: its.jes@gmail.com
To: ocean.kai@gmail.com
Author: Author: Jess
  Message ----------------------
Hi, Don't forget to call me.

Shweet: UNKNOWN AUTHOR
ID: @DEFAULT_ID
SPARKLING DEFAULT MESSAGE

Shweet: sparkle-Chan
ID: @sparkleSparkle
What should I eat?

Shweet: Mario
ID: @marioWorld
I can't find yoshi. Has anyone seen him?

Shweet: Jess
ID: @itsJess
New episodes of New Girl on tonight. Who's that girl IT'S JESS!!

Press any key to continue . . .
*/
```



```
// Lab 04 Option A - Instructor Solution:
// Original - Prof. Loceff, Updates, Edits, Annotations: &
//
//Notes:
//- Correct adherence to guidelines learned in CS2A
//- Correct access qualifiers (private/public/protected)
//- Correct use of getters/setters
//- Proper bounds checking on array accesses (Serious error if not)
//- Not using non-recommended naming strategies for variables/methods
//- Correct use of global consts
//- Correct way of reading input (using getline())
//- Use of symbolic consts rather than literals (magics)
//- No output in interior methods
//- Correct formatting
//- Correct qualifications on variables, params and methods/functions.
//- Faithfulness to spec
//- Code must be readable

#include <iostream>
#include <string>
using namespace std;

class Message {
protected:
    // immutable constants
    static const int MAX_MSG_LEN = 500*1000; // not realistic;  20M often
    static const int MIN_MSG_LEN = 1;           // should be > 0
    static const int MIN_AUTH_LEN = 2;          // < 2 is unreasonable
    static const int MAX_AUTH_LEN = 50;
    static const string DEFAULT_MSG;
    static const string DEFAULT_AUTH;

private:
    // main data
    string message;
    string author;

public:
    // constructors, mutators, etc.
    Message();
    Message(const string& theAuth, const string& theMsg);

    bool setAuthor(const string& auth);
    bool setMessage(const string& msg);
    string getAuthor() const { return author; }
    string getMessage() const { return message; }
    string toString() const;

private:
    // helpers, etc.
    static bool isValidMsg(const string& msg);
    static bool isValidAuth(const string& auth);
};
```

```
const string Message::DEFAULT_MSG = "(no message)";
const string Message::DEFAULT_AUTH = "(no author)";

class EMail : public Message {
public:
    // immutable constants
    static const int MAX_EADDR_LEN = 100;  // real world: 254
    static const int MIN_EADDR_LEN = 5;  // a@b.c
    static const string DEFAULT_E_ADDR;

private:
    // additional data
    string fromAddress;
    string toAddress;

public:
    // constructors, mutators, etc.
    EMail();
    EMail(const string& theAuth, const string& theMsg,
          const string& fromAddr, const string& toAddr);

    bool setFromAddress(const string& eAddr);
    bool setToAddress(const string& eAddr);
    string getFromAddress() const { return fromAddress; }
    string getToAddress() const { return toAddress; }
    string toString() const;

private:
    // helpers, etc.
    static bool isValidAddress(const string& eAddr);
};

const string EMail::DEFAULT_E_ADDR = "no_user_defined@error.err";

class Shweet : public Message {
public:
    // immutable constants
    static const int MAX_SHWITTER_ID_LEN = 15;
    static const int MAX_SHWEET_LEN = 140;
    static const string DEFAULT_USER_ID;

private:
    // additional data
    string fromID;

public:
    // constructors, mutators, etc.
    Shweet();
    Shweet(const string& theAuth, const string& theMsg, const string& theID);

    bool setFromID(const string& theID);
    bool setMessage(const string& theMsg);  // overrides
```

```
    string getFromID() const { return fromID; }
    string toString() const;

private:
    // helpers, etc.
    static bool isValidShwitterID(const string& theID);
    static bool isValidShweet(const string& theMsg);
    static bool isAlphaOrNumOrUnderscore(const string& theString);
};

const string Shweet::DEFAULT_USER_ID = "no_user_id";

// Client
int main() {
    Message msg1, msg2("loceff", "hello world"), msgDflt;

    // ----- base class testing -----
    cout << "Base Class Testing **********************************\n";

    msg1.setAuthor("Kinnard");
    msg1.setMessage("Some messages just aren't worth sending.");
    cout << endl;

    // I kinda like this way of formatting long cout statements.
    cout << msgDflt.toString() << endl
         << msg1.toString()    << endl
         << msg2.toString()    << endl
    ;

    cout << "testing Message accessors:" << endl
         << msg1.getAuthor()  << endl
         << endl
         << msg2.getMessage() << endl
         << endl
    ;

    cout << "testing Message mutators:" << endl;
    if (!msg1.setAuthor("LONG STRING abcde abcde abcde abcde  abcde abcde"
                        " abcde  abcde  abcde  abcde  abcde  abcde abcde"
                        " abcde abcde abcde"))
        cout << "too long (as expected)" << endl;
    else
        cout << "acceptable length (should not be)" << endl;
    cout << msg1.toString() << endl;

    if (!msg1.setMessage("LONG STRING abcde abcde abcde abcde  abcde abcde"
                         " abcde  abcde  abcde  abcde  abcde  abcde abcde"
                         " abcde abcde abcde"))
        cout << "too long (unexpected - debuging required)" << endl;
    else
        cout << "acceptable length (should  be)" << endl;
    cout << msg1.toString() << endl;
```

```
cout << "----- EMail Derived Class Testing -----\n";

EMail emailDflt, emailMsg1, emailMsg2("chloe", "bark bark",
                                      "chloe123@gmail.com",
                                      "lili999@gmail.com");

emailMsg1.setAuthor("lili koi");
emailMsg1.setFromAddress("lili999@gmail.com");
emailMsg1.setToAddress("chloe123@gmail.com.com");
emailMsg1.setMessage("Arf, arf, arf, arf ........ arf");
cout << endl;

cout << emailDflt.toString() << endl
     << endl
     << emailMsg1.toString() << endl
     << emailMsg2.toString()
     << endl
;

cout << "testing EMail accessors:" << endl
     << emailMsg1.getFromAddress() << endl
     << endl
     << emailMsg2.getToAddress() << endl
     << endl
;

cout << "testing EMail mutators:" << endl;
if (!emailMsg1.setFromAddress("LONG STRING abcde abcde abcde abcde abcde"
                              "abcde abcde  abcde  abcde  abcde  abcde"
                              "abcde abcde  abcde abcde abcde"))
    cout << "too long (as expected)" << endl;
else
    cout << "acceptable length (should not be)" << endl;
cout << emailMsg1.toString() << endl;

if (!emailMsg1.setToAddress("No_At_Character.com"))
    cout << "missing @ char (as expected)" << endl;
else
    cout << "good email address (unexpected - debuging required)" << endl;
cout << emailMsg1.toString() << endl;

if (!emailMsg1.setFromAddress("No_Dot_Character@somewhere_com"))
    cout << "missing DOT char (as expected)" << endl;
else
    cout << "good email address (unexpected - debuging required)" << endl;
cout << emailMsg1.toString() << endl;

cout << "----- Shweet Derived Class Testing -----\n";

Shweet tweetMsg1, tweetMsg2("Katy Perry", "It's a verb and an adjective.",
                            "katyperry"), tweetDflt;

tweetMsg1.setAuthor("Kim Kardashian");
```

```
    tweetMsg1.setFromID("kimkardashian");
    tweetMsg1.setMessage("Oh Deer https://www.keek.com/!PYjCdab");
    cout << endl;

    cout << tweetDflt.toString() << endl
         << endl
         << tweetMsg1.toString()
         << endl
         << tweetMsg2.toString()
         << endl
    ;

    cout << "testing Shweet accessors:" << endl
         << tweetMsg1.getFromID() << endl
         << endl
    ;

    cout << "testing Shweet mutators:" << endl;
    if (!tweetMsg1.setFromID("a_space and ."))
        cout << "bad shwitter ID (as expected)" << endl;
    else
        cout << "acceptable shwitter ID (should not be)" << endl;
    cout << tweetMsg1.toString() << endl;

    if (!tweetMsg1.setFromID("a_good_user99"))
        cout << "bad shwitter ID (unexpected - debuging required)" << endl;
    else
        cout << "acceptable shwitter ID (as expected)" << endl;
    cout << tweetMsg1.toString() << endl;

    return 0;
}

// ----- Base class Message method definitions -----
// constructors, mutators, etc.
Message::Message() {
    author = DEFAULT_AUTH;
    message = DEFAULT_MSG;
}

Message::Message(const string& auth, const string& msg) {
    if (!setMessage(msg))
        message = DEFAULT_MSG;
    if (!setAuthor(auth))
        author = DEFAULT_AUTH;
}

bool Message::setAuthor(const string& theAuth) {
    if (!isValidAuth(theAuth))
        return false;
    author = theAuth;
    return true;
}
```

```
bool Message::setMessage(const string& theMsg) {
    if (!isValidMsg(theMsg))
        return false;
    message = theMsg;
    return true;
}

// helpers
bool Message::isValidMsg(const string& msg) {
    return msg.length() >= MIN_MSG_LEN && msg.length() <= MAX_MSG_LEN;
}

bool Message::isValidAuth(const string& auth) {
    return auth.length() >= MIN_AUTH_LEN && auth.length() <= MAX_AUTH_LEN;
}

string Message::toString() const {
    string retString;

    retString = "Author: " + author + '\n'
              + " Message ---------------------- \n"
              + message
              + '\n'
    ;
    return retString;
}

// ----- Derived class EMail method definitions -----
// constructors, mutators, etc.
EMail::EMail() : Message() {
    fromAddress = DEFAULT_E_ADDR;
    toAddress = DEFAULT_E_ADDR;
}

EMail::EMail(const string& author, const string& msg, const string& fromAddr,
             const string& toAddr) : Message(author, msg) {
    if (!setFromAddress(fromAddr))
        fromAddress = DEFAULT_E_ADDR;
    if (setToAddress(toAddr))
        toAddress = DEFAULT_E_ADDR;
}

bool EMail::setFromAddress(const string& eAddr) {
    if (!isValidAddress(eAddr))
        return false;
    fromAddress = eAddr;
    return true;
}

bool EMail::setToAddress(const string& eAddr) {
    if (!isValidAddress(eAddr))
        return false;
```

```
    toAddress = eAddr;
    return true;
}

string EMail::toString() const {
    string retString;

    retString = "From: " + fromAddress + '\n'
              + "To: " + toAddress + '\n'
              + Message::toString()
    ;
    return retString;
}

bool EMail::isValidAddress(const string& eAddr) {
    // basic test -- not as rigorous as it could be:
    // length is good, and has an @ and a dot (.)

    if (eAddr.length() < MIN_EADDR_LEN || eAddr.length() > MAX_EADDR_LEN)
        return false;

    if (eAddr.find("@") == string::npos)
        return false;

    if (eAddr.find(".") == string::npos)
        return false;

    return true;
}

// ----- Derived class Shweet method definitions -----
// constructors, mutators, etc.
Shweet::Shweet() : Message() {
    fromID = DEFAULT_USER_ID;
}

// chain to DEFAULT constructor otherwise we may end up setting a too-long
// message. We assume that a default Message msg/auth are legal Tweet msg/auth
Shweet::Shweet(const string& theAuth, const string& theMsg,
               const string& theID) : Message()
{
    if (!setFromID(theID))
        fromID = DEFAULT_USER_ID;

    // no need to test, since default constructor was chained
    setMessage(theMsg);
    setAuthor(theAuth);
}

bool Shweet::setFromID(const string& theID) {
    if (!isValidShwitterID(theID))
        return false;
    fromID = theID;
```

```
    return true;
}

bool Shweet::setMessage(const string& theMsg) {
    // this enforces Shweet limits
    if (!isValidShweet(theMsg))
        return false;

    // we also have to conform to base class limits; done inside base class
    return Message::setMessage(theMsg);
}

// helpers, etc.
// overrides Message::isVAlidMsg()
bool Shweet::isValidShweet(const string& theMsg) {
    if (theMsg.length() < MIN_MSG_LEN || theMsg.length() > MAX_SHWEET_LEN)
        return false;
    return true;
}

bool Shweet::isValidShwitterID(const string& theID) {
    return theID.length() > 0 && theID.length() <= MAX_SHWITTER_ID_LEN
      && isAlphaOrNumOrUnderscore(theID)
    ;

}

bool Shweet::isAlphaOrNumOrUnderscore(const string& theString) {
    for (int k = 0; k < theString.length(); k++) {
        // if it is not a letter or number, last hope is an underscore
        if(!isalnum(theString[k]) && theString[k] != '_')
            return false;
        // otherwise fine; go on to next char
    }
    return true;
}

string Shweet::toString() const {
    string retString;

    retString = "Shweet: " + getAuthor() + " @" + fromID + '\n'
              + getMessage()
              + "\n"
    ;

    return retString;
}

/* ----- Run -----
 Base Class Testing **********************************

 Author: (no author)
 Message ----------------------
```

```
(no message)

Author: Kinnard
Message ----------------------
Some messages just aren't worth sending.

Author: loceff
Message ----------------------
hello world

testing Message accessors:
Kinnard

hello world

testing Message mutators:
too long (as expected)
Author: Kinnard
Message ----------------------
Some messages just aren't worth sending.

acceptable length (should  be)
Author: Kinnard
Message ----------------------
LONG STRING abcde abcde abcde abcde  abcde abcde abcde  abcde  abcde  abcde  abcde  abcde
abcde abcde abcde abcde

----- EMail Derived Class Testing -----

From: no_user_defined@error.err
To: no_user_defined@error.err
Author: (no author)
Message ----------------------
(no message)


From: lili999@gmail.com
To: chloe123@gmail.com.com
Author: lili koi
Message ----------------------
Arf, arf, arf, arf ........ arf

From: chloe123@gmail.com
To: no_user_defined@error.err
Author: chloe
Message ----------------------
bark bark

testing EMail accessors:
lili999@gmail.com

no_user_defined@error.err
```

```
testing EMail mutators:
too long (as expected)
From: lili999@gmail.com
To: chloe123@gmail.com.com
Author: lili koi
Message ----------------------
Arf, arf, arf, arf ........ arf

missing @ char (as expected)
From: lili999@gmail.com
To: chloe123@gmail.com.com
Author: lili koi
Message ----------------------
Arf, arf, arf, arf ........ arf

missing DOT char (as expected)
From: lili999@gmail.com
To: chloe123@gmail.com.com
Author: lili koi
Message ----------------------
Arf, arf, arf, arf ........ arf

----- Shweet Derived Class Testing -----

Shweet: (no author) @no_user_id
(no message)


Shweet: Kim Kardashian @kimkardashian
Oh Deer https://www.keek.com/!PYjCdab

Shweet: Katy Perry @katyperry
It's a verb and an adjective.

testing Shweet accessors:
kimkardashian

testing Shweet mutators:
bad shwitter ID (as expected)
Shweet: Kim Kardashian @kimkardashian
Oh Deer https://www.keek.com/!PYjCdab

acceptable shwitter ID (as expected)
Shweet: Kim Kardashian @a_good_user99
Oh Deer https://www.keek.com/!PYjCdab

Program ended with exit code: 0
------------------------------------------------------------------------ */
```