Assessment Data Samples

(CPSC 121, Spring 2018)
Paul Salvador Inventado
## CPSC 121 Performance Indicators (Spring 2018)

IDE: Demonstrate ability to develop software using an Integrated Development Environment (IDE).

<table>
<thead>
<tr>
<th>Performance Indicator</th>
<th>0: Unsatisfactory</th>
<th>1: Developing</th>
<th>2: Satisfactory</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE</td>
<td>Unable to develop, build or debug software using an integrated Development Environment (IDE)</td>
<td>Able to develop, build, and/or debug software using an Integrated Development Environment (IDE), lacking proficiency in all areas.</td>
<td>Able to develop, build, and debug software using an Integrated Development Environment (IDE).</td>
</tr>
</tbody>
</table>

CMNT: Write source code with clear and informative comments.

<table>
<thead>
<tr>
<th>Performance Indicator</th>
<th>0: Unsatisfactory</th>
<th>1: Developing</th>
<th>2: Satisfactory</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMNT</td>
<td>Unable to write source code with clear and informative comments.</td>
<td>Write source code with somewhat informative comments.</td>
<td>Write source code with clear and informative comments.</td>
</tr>
</tbody>
</table>

OOP: Make appropriate use of object-oriented concepts such as classes, encapsulation, interfaces, inheritance, and/or polymorphism

<table>
<thead>
<tr>
<th>Performance Indicator</th>
<th>0: Unsatisfactory</th>
<th>1: Developing</th>
<th>2: Satisfactory</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOP</td>
<td>Unable to write classes.</td>
<td>Able to write code decomposed into classes, but little to no evidence of cohesive use of object- oriented paradigms.</td>
<td>Able to write code, making appropriate use of object-oriented paradigms.</td>
</tr>
</tbody>
</table>
IDE: Satisfactory

All students successfully used the Codelite++ IDE to answer lab exercises and projects.
Project: TweetSplitter (pg. 1 of 2)

Tweet Splitter

Twitter is an online news and social media service that allows users to interact using messages, which are called tweets. Each tweet has a limit of 280 characters, so users need to keep track of the number of characters they use. This can be annoying when users need more than 280 characters for their message.

Our goal is to create a program that automatically splits messages with more than 280 characters so that users will not need to worry about character limits. You will need to create a program using three files. These files have been partially created so you only need to complete its contents. It will be your responsibility to do some research to implement the necessary features.

```
twitter.cpp
```

This file will contain two function implementations that are needed to process tweets.

- `splitTweet` - An implementation of this function should accept the user’s message, a string array that will store the split-tweets, and the number of splits. The user’s message will be split into a maximum of 200 characters wherein a split should only be performed on spaces or punctuation (`,`, `.`, `?`, `!`) Words should not be split. Use the cases below for reference: `...` indicates that there are other words before or after the word.

- `splitTweet` - An implementation of this function should accept the user’s message, a string array that will store the split-tweets, and the number of splits. The user’s message will be split into a maximum of 200 characters wherein a split should only be performed on spaces or punctuation (`,`, `.`, `?`, `!`) Words should not be split. Use the cases below for reference: `...` indicates that there are other words before or after the word.

Less than 200 characters

```
Tweet: Hello
```

Greater than 200 characters, 280th character is a space or punctuation

```
Tweet: a b c d
```

Greater than 200 characters, 280th character is not a space or punctuation

```
Tweet: a b c d e
```

It is possible to have messages that are split into more than two messages. For example, a message with 1000 characters could be split into 4.

Each split needs to be stored in the string array parameter. For example, split string 1 will be stored in index 0, split string 2 in index 1, and so forth. The number of splits should be stored in the parameter for storing the number of splits. In the case of 2 string splits, for example, the value of this variable should be 2.

Notes:
- The string array and number of splits parameter are variables declared in the main function. After calling the `splitTweet` function, the values of the string array and number of splits parameters in the main function should be updated.
- Explore the following string object functions to split tweets: `at`, `length`, `find`, and `substr`.
- The `splitTweet` function only splits messages. It does not delete any characters so it is expected to have spaces before or after words. Use the `trim` method to ensure you are generating the correct output.
Project: TweetSplitter (pg. 2 of 2)

displaySplitTweets: This function should accept as parameters a string array, the number of split tweets (string in the array), and a username. For each split tweet, this function should display the username and the split content. See the example below for reference ... indicates that there are other words before or after the tweet.
@Source: where you got it (D3)
@Source: It's all ... (10)

Whenever the original message is less than 200 characters (only 1 split), the split count is not displayed. See the example below for reference ... indicates that there are other words before or after the text.
@Source: Hello

twitter.hpp
This file should only contain function prototypes. Specifically, splitTweets and displaySplitTweets.

tweetsplit.cpp
This file will include twitter.hpp, and contain the main function that will initiate the tweet splitter program. As a minimum, the main function should ask the user to input their username and message. It should also contain at least two variables: an array with a size of 100 to store the split tweets, and the number of splits.

The user’s message should be passed to the splitTweets function together with the variables referring to the string array and number of splits. The username, updated string array and updated number of splits variables should be passed to the displaySplitTweets function to show the message on the screen.

I encourage you to be creative in your user interface. Although this is not required, you can, for example, create a user menu that will ask the user to enter their message until they input a sentinel value (e.g., "exit").

Sample run 1

Please enter your username: Jacqueline
Please enter your message: okay
@Jacqueline: okay

Sample run 2

Please enter your username: Blackbear

Compilation, execution, and testing cheat sheet

<table>
<thead>
<tr>
<th>Change the current working directory to the appropriate folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd Desktop</td>
</tr>
<tr>
<td>cd lab 1 [lab]</td>
</tr>
<tr>
<td>Compilation</td>
</tr>
<tr>
<td>g++ -std=c++11 twitter.cpp tweetsplit.cpp -o tweetsplit</td>
</tr>
<tr>
<td>Execution</td>
</tr>
<tr>
<td>tweetsplit</td>
</tr>
<tr>
<td>Testing</td>
</tr>
<tr>
<td>ok test1</td>
</tr>
</tbody>
</table>

void displaySplitTweet(string arr [], int count, string username) {
    //counter is set to one
    int x = 1;
    //if statement for if count is greater than one
    if (count > 1) {
        //for loop to print out the @ symbol with username and the tweet in the array
        //Also prints out the counter
        for (int i = 0; i < count; i++) {
            cout << "@" << username << ": " << arr[i] << " (" << x << "/" << count << ")";
            cout << endl;
            x++;
        }
    } else {
        //else prints out the entire tweet
        cout << "@" << username << ": " << arr[0];
        cout << endl;
    }
}
void displaySplitTweet(string array[], int splits, string username) // displays the tweet
{
    if(splits>1)
    {
        for(int counter = 0; counter < splits; counter++) // displays the array
        {
            cout << "@" << username << ": ";
            cout << array[counter];
            cout << " (" << counter +1 << "/" << splits << ")" << endl;
        }
    }
    else
    {
        cout << "@" << username << ": ";
        cout << array[0] << endl;
    }
}
void displaySplitTweet(string* ptrSplit, int numSplits, string n)
{
    if (numSplits == 1)
    {
        cout << "@" << n << ": " << ptrSplit[0] << endl;
    }
    else
    {
        for (int i = 0; i < numSplits; i++)
        {
            cout << "@" << n << ": " << ptrSplit[i] << " (" << i + 1 << "/" << numSplits << ")" << endl;
        }
    }
}
Lab exercise: Inheritance (instructions)

Student (functions.hpp and functions.cpp)
Implement the Student class that inherits from the Person class whose code is already provided for you. The Student class contains two additional member variables that represent their school and their gradeLevel.

Provide getters and setters for both the school and gradeLevel variables. Create a default constructor that sets the school to “California State University Fullerton” and the gradeLevel to 1. It should use the Person class’ default constructor so that it’s name and birthday is set to default values. Provide a second constructor that requires school and gradeLevel as parameters, but calls the Person’s constructor to assign default values to name and birthday. Finally, create a third constructor that accepts a name, birthday, school, and gradeLevel that assigns all values using the appropriate Person constructor.

Create a display function that will display the name, birthday, school, and gradeLevel of a Student. Use method overloading to reuse Person’s display function.

Compile the main function in student.cpp and kindly ensure that your program follows, as closely as possible, the sample run shown below. Take note, however, that this is only one example and the values will change with different user input.

functions.hpp should only contain class definitions and the function prototypes of all functions that are not getters or setters. functions.cpp should contain all the implementations of the class member functions.

Sample run 1
Please provide student name: Lily Luna Potter
Please provide student birth month (1-12): 6
Please provide student birth date: 1
Please provide student birth year: 2007
Please provide student school: Hogwarts
Please provide student grade level: 6

Student enrolled!
Name: Lily Luna Potter
Birthday: September 1, 2006
School: Hogwarts
Grade level: 6

Compilation and execution cheat sheet
Change the current working directory to the appropriate folder
```
cd Desktop

cd lab [ Tab ]
```
Compilation
```
g++ -std=c++11 functions.cpp student.cpp -o student
```
Execution
```
./student
```
Lab exercise: Inheritance (Github repository)
OOP: Satisfactory (Lab exercise)

Proper code for overriding a parent class’ method and reusing the parent’s implementation of that method.

```cpp
// Insert code for the display function of Student
void Student::display()
{
    Person::display();
    cout << "School: " << school << endl;
    cout << "Grade level: " << gradeLevel << endl;
}
```
OOP: Developing (Lab exercise)

Code correctly overrides the parent class’ method, but it should be unable to access its private attributes.

```cpp
// Insert code for the display function of Student
void Student::display()
{
    cout << "Name: " << name;
    cout << "Birthday: " << birthday;
    cout << "School: " << school;
    cout << "Grade level: " << gradeLevel;
}
```