

A Heap of students (A6)

100 Possible Points

2/24/2023

Attempt 1



2/24/2023

Next Up: Review Feedback

Attempt 1 Score:
N/A

Add Comment

Unlimited Attempts Allowed▼ **Details**

Overview

The purpose of this lab is to give you a deeper understanding of memory management by forcing you to utilize the standard C++ heap structure as much as possible.

Build a normalized student record system. Create a class to represent student data. You will also create classes to handle other specific types of data including dates and addresses. Your system will include all the necessary header files, cpp files, Makefile, and a main program to incorporate the larger system.

Details

Imagine a university wants to keep track of the following data about each student:

- First name
- Last name
- address line 1
- address line 2
- city
- state
- zip code
- date of birth month
- date of birth day
- date of birth year
- anticipated completion month
- anticipated completion day

<https://iu.instructure.com/courses/2131288/modules/items/28790603>

Again

<https://iu.instructure.com/co>

It is obvious that we will need a student class to contain all this data. However, the principles of data design indicate that the address should be its own class, as should the date. Your program should then have at least three classes: address, date, and student. Student will have two instances of date and one of address among its data members.

Process

Build all the classes you will need for this project. Each class should have a separate header file. This file contains only the class definition. Ensure your headers use the `#ifndef` structure described in lecture, and include whatever other headers they need.

Build a cpp file for each class which contains the code implementation of the class. This code file will include the class header file and any other headers and libraries needed.

Build a main program that imports all the needed classes and tests them. You'll then modify this program.


Create a makefile to control the compilation process. The instructor will test the program by running the make utility, so be sure your makefile works. If you use a visual editor like code::blocks, you'll still need to create and test a makefile for your project. Note IDE's have tools to build make files automatically, but they are often terrible. Make your own. It's must include to include clean and run builds in your makefile along with a command to run Valgrind. **Look at Andy's example for guidance on how to create it, but the command should be called valgrind.**

Use git to manage your versions. There is no way you can prove you used git, but learn how to use it now. It's ideal for a larger project like this with multiple files.

All instances of your classes should be stored separately on the heap. That is to say, all should make use of the 'new' keyword. Your program will be tested using the Valgrind tool to ensure you have responsibly deallocated memory when your are done with it.

The complete project

When the classes are done, create a main program which does the following:

- **Load up student data from a text file.** All the needed information for the students should be in a text file, with each student's information on one line. We will provide you with a file with student data. There will be 50 students. The file is available here: [students.dat](https://www.cs.iupui.edu/~ajharris/240/students.dat) 
(<https://www.cs.iupui.edu/~ajharris/240/students.dat>)
- **Store all instances of your classes on the heap** As stated above, all instances of your custom classes should be created on the heap.



(<https://iu.instructure.com/courses/2131288/modules/items/28790603>)

(<https://iu.instructure.com/coi>)

- **List all data for all students in a report format** Create a method of the Student class to print a report about every student *into a separate text file called "fullReport.txt"*.
- **Create a simpler list** that prints only the last and first name of each student *into a separate text file called "shortReport.txt"*.
- **(optional) Output a list of student names in alphabetical order** Print the list in alphabetical order *into a separate text file called "alphaReport.txt"*. You can use your own sort algorithm, or the one from `std::sort` for this. You can consider this a blackbelt requirement, so wait until you get other things done before adding this.
- **Note:** You may not use any pre-constructed classes from the c++ standard template library (vectors, sets, etc). You MAY construct your own (if you reallllly wanna go there). Additionally, any instances of the classes you write need to be stored separately on the heap by utilizing the keyword `new`.
- Your program needs to test for memory leaks using Valgrind. We won't use your valgrind file, but the program will be tested for memory leaks so check your Valgrind
- Use C++98 for the base assignment, smart pointers and other features of C++11 are great, but they are not the point of the assignment. Use them in the blackbelt or future classes.
- Use a standard c array not a vector in this program.

Here is the format you may use for a line of student data in your text file:

Surname,GivenName,StreetAddress,Address2,City,State,ZipCode,Birthday,Graduation,GPA,Credit Hours Complete<end of line>

Note that the text file provided by your recitation leader might be slightly different. Use the data you're given.

Here is an example of what it may look like:

Fry,Lock,123 Hillside Drive,,North Brunswick,NJ,57237,08/22/1970,05/15/2012,4.00,90

Turning it in

Algorithm

As always, begin with an algorithm in markdown. This document should begin by describing the main purpose of the program using the Goals - Input - Output - Steps technique we've been using. For



(<https://iu.instructure.com/courses/2131288/modules/items/28790603>)

(<https://iu.instructure.com/coi>)

You should get mostly through the algorithm step together in the recitation center, but definitely try to write this on your own as well.

Your algorithm file must be in a .md file named: algorithm.md; and the file should be in the base folder.


Points will be taken off if it is not named this exactly, or if it is in another file type (including but not limited to; .rtf, .docx, .doc, .pdf, ... etc.)

UML Diagram

You will need to turn in a UML diagram displaying all classes used in your program. The UML diagram should be in a standard image file (.png, .jpeg, .jpg, .jiff).

Include the file in your github repo under the base folder, and if you have more classes for your blackbelt, you should make another one that reflects these changes.

Turning in the project

- This assignment will be turned in through iu github. Please name your repo CSCI24000_spring23_A6
- If your repo has another name, it will not be graded.
- After you've submitted your assignment in Github, come back to this assignment page and submit the full Github URL (<https://github.iu.edu/username/reponame> ) (<https://github.iu.edu/username/reponame>) for your repo here.
- Your repo must be private. We will not grade projects in public repos, and we will require you to take down any homework assignments in a public repo.
- Please ensure to add all the following are listed as collaborators:
- **[Git Collaborators \(https://iu.instructure.com/courses/2131288/pages/assignments-submission-and-grading\)](https://iu.instructure.com/courses/2131288/pages/assignments-submission-and-grading)**

Note that your code will be tested with an online plagiarism tool. Please DO NOT turn in work that is not yours. We will know, and we will be displeased.

Black Belt Extension

- C++ 11 or greater, vectors



(<https://iu.instructure.com/courses/2131288/modules/items/28790603>)

(<https://iu.instructure.com/coi>)

https://github.iu.edu/rajaali/CSCI24000_spring23_A6



<https://iu.instructure.com/courses/2131288/modules/items/28790603>

<https://iu.instructure.com/courses/2131288/modules/items/28790603>