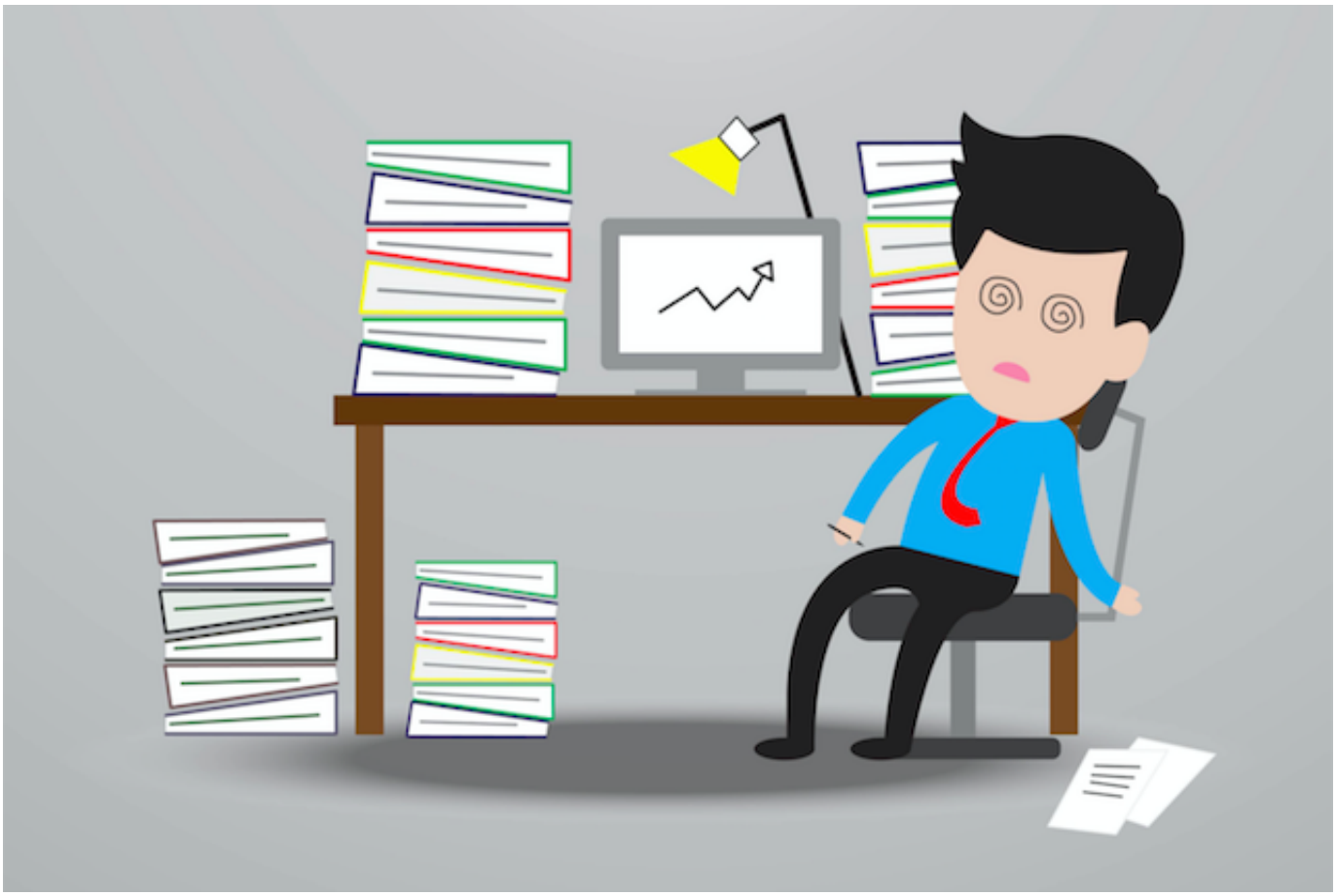
Project 1: Event Planning

Instructor: Sadab Hafiz

Due: June 17, 2024



Introduction

We have talked about different Object-Oriented Programming(OOP) concepts in class. We also looked at how templates can help in defining abstract data types. It is time to put your skills to the test. The more code you write, the better you get. If you understand what each line of your code does after successfully completing the assignment, you will be prepared for midterm. This project is an introduction to submitting your code on Gradescope through GitHub. It is also a chance for you to make sure you have a working `C++` environment. You will also get used to documenting and formatting your code appropriately. These skills will be fundamental to learn right now so that you can focus more on the coding part in the future assignments. In this project, you will implement the `Event` class. After that, you will implement a class template called `Plan`, which uses dynamic memory allocation.

Setting up GitHub

This is the first step necessary. Without finishing this step, you won't be able to submit your code. To get started on GitHub, you first need to create an account on the platform. Visit the GitHub website and sign up for a new account. If you stick with Computer Science, you will most likely use GitHub professionally in the future, so pick a name that you are comfortable with.
After creating your account, I highly encourage you watch this video by Brian Yu which covers the basic of Git and GitHub. After that, watch this video which shows the entire process, from accepting an assignment to submitting it. Although the video is about a different course, the instructions are the same (with different repo and file names). The only difference is that we will not add a distribution branch, so you can ignore the part where it says to execute the two git commands in the readme file. There are no extra instructions in the readme file on our repo). **The link to accept the GitHub Classroom assignment can be found on Blackboard**. The above video will also show you how to submit to Gradescope via GitHub. Make sure to refer back to these instructions when it's time to submit.

Documentation and Design

Documentation and design is worth 20% of each project like spring semester. The requirements are:

Documentation Requirements (15%)

- All files should start with block comments with at least your name, date, and a brief description of the code implemented in that file (a rough idea of what this document is).
- All functions in the `.hpp` files should have a comment describing the function:

```
/**
 * @brief: describe the function in general and what it does
 *
 * @pre: describes any precondition
 * @param: one for each parameter the function takes
 * @return: describes the return type
 * @post: describes any postconditions
 * @throws: describe any exceptions that the function may throw
 */
```

  For the required functions in each project, these will be provided to you in the instructions. However, you have to write them for any additional helper functions that you add and implement.

- While function comments are descriptive, they are intended to let users of the class know how to use the functions. Thus, they don't go into the details of the implementation. In order to explain the implementation, you should add inline comments where necessary (loops, conditionals, etc.):

```
// comment explaining what is happening in the for-loop
for (DataType x: some_arr_){
    // comment explaining the if-else statements
    if (...x meets some condition...){
        ...some code in here...
    }
    else{
        ...some code in here...
```

```
    }
}
```

If something feels necessary to be explained to someone looking at your code, you should leave an inline comment explaining it.

- Avoid commenting every single line. Things that are self-explanatory should not be commented.

Design Requirements (5%)

- You must follow the naming convention discussed in class.

```
string my_variable;
class MyClass{};
MyClass class_instance;
string my_data_member_;
void myFunction();
const int MY_CONSTANT;
```

- Any decisions you make, make sure to be consistent. This applies to the choice of include guards (`#pragma` or `#ifndef`), the order of things in classes, comments, code formatting, etc.
- Avoid making decision choices that violate OOP principles. For example, public data members and global variables are to be avoided unless explicitly mentioned.
- For any helper functions, the keyword `const` and `&` should be used where appropriate. For example, if your function doesn't modify the private data members, the functions should be defined as `const`.
- Do not include `using namespace std;` in the header files.
- As long as you follow the standard practices that are demonstrated in class, you will not lose points.

Helpful Tips

If you are using vscode, there are extensions that can help you. The documentation generated in the header files I share use an extension that generate comments compatible with Doxygen. Doxygen is a tool that provides robust support for documenting `C++` code. Here are the extensions that I highly recommend you to use to automate some documentation generation:

- Doxygen Documentation Generator : Generate Doxygen compatible comment templates
- `C/C++` for VSCode : VSCode language support for `C/C++` editing and debugging

If you don't have a working `C/C++` environment, contact me as soon as possible. The starter `Event` header file shows an example of how you can document your header files.

Plagiarism and Citing

Your code in all the `.cpp` files should not 100% match with another student. If that happens, I will assume it is a violation of academic integrity. It can happen with the use of generative AI, it can also happen if you both copy code from the same source. If you use tools or functions that are not mentioned in the slides, or I didn't teach in class, you have to cite them with an inline comment. You should also be able to explain such lines of code if I ask you to explain what you wrote.

Task A: Event Class

Your first task is to implement a class called `Event`. The header file is in GitHub, **which you can access through the GitHub classroom link in Blackboard**. Carefully go through each of the function comments to figure out how to implement them. Some functions can be used to implement the other ones, so I encourage you to go over the header entirely first before starting to code. The `.cpp` file is also provided with the implementation of one of the helper functions. Do not modify the implementation of this helper function. I have tested it and it works as intended.
In order to show you an example of how you can test different functions of the `Event` class, there is a `main.cpp` file provided with some code that tests a few functions of the `Event` class. This file is not complete as it doesn't test everything. You should add to it or modify it as you see fit to test everything in your class while implementing it. Furthermore, the output of this file is also provided in `example_output.txt` file. You can use this output as a basis to implement the output operator overload.

Task B: Plan Class

Unlike Task A, I am not providing the header file for `Plan` class. I want you all to implement it from scratch. Note that the `Plan` class doesn't require the `Event` class to be implemented. Therefore, you can work on Task B independently of Task A. Here is what you need to know to implement it:

- `Plan` class is a class template. Remember to separate the implementation and interface into `Plan.hpp` and `Plan.cpp` files. Remember to include the `Plan.cpp` at the end of `Plan.hpp`.
- `Plan` class is an ordered ADT where we insert at the end and maintain order of insertion.
- You will implement the plan class using dynamic array. Here is a quick example of the syntax.

- The data members that you need in your class are described as follows:
  - An integer that keeps track of the capacity of your dynamically allocated array
  - An integer that keeps track of the number of items in the array
  - A pointer to the dynamically allocated array
- There are no `private` member functions. You can add some helper functions if you want.
- `Plan` class has a default constructor which allocates a dynamic array of capacity 2 and sets the data members accordingly. `Plan` class doesn't have any parameterized constructors.
- The `public` interface contains the five things in the rule of five. Take a look at the code we discussed in class for guidance and implement: destructor, copy constructor, assignment operator, move constructor, and move assignment operator.
- The other `public` functions of `Plan` class are described below, assuming `template <class T>`:

```
/**
 * @brief Add an event to the Plan
 *
 * @param event The event to be added to the Plan
 * @post If the array is full, the capacity of the array will be
   doubled and the event will be added
 */
void addEvent(const T& event);

/**
 * @brief Remove an event from the plan
 *
 * @param event The event to be removed
 * @return true if the event is in the array and is successfully
   removed
 * @return false if the event is not in the array and can't be
   removed
 * @post If the event exists in the Plan, it will be removed while
   retaining the order
 */
bool removeEvent(const T& event);

/**
 * @brief Get the pointer to the Plan array
 *
 * @return T* Pointer to the dynamically allocated array where Plan
   is stored
 */
T* getEvents() const;

/**
 * @brief Get the size of the Plan
 *
 * @return int Number of events in the plan
```

```
 */
int getSize() const;

/**
 * @brief Get the capacity of the Plan array
 *
 * @return int The maximum number of items that can be stored in the
   currently allocated Plan array
 */
int getCapacity() const;

/**
 * @brief Get the index of the object
 *
 * @param event Object whose index is being queried
 * @return int Index of the object if it exists, -1 otherwise
 */
int getIndexOf(const T& event);
```

- The `removeEvent` function doesn't free the space that is dynamically allocated.
- Don't forget to update the size after adding or removing from `Plan`.

Compiling and Testing

Since the two parts of the project are independent of each other, you can compile them separately or together, depending on how you test it. Assuming you implement everything, you can compile everything with `g++ --std=c++11 Event.cpp main.cpp`. Note that we don't compile the `Plan.cpp` as it is a class template. Including the `Plan.hpp` in `main.cpp` will cause it to be compiled when we compile `main.cpp`. Make sure you test everything to match the instructions. Write test cases for all the functions you implement and all the edge cases. Testing is a very important skill that you need to learn by trial and error. If you get an error, look at the error message and find out which line it is coming from. I recommend implementing each function and testing it before implementing the next one.

Submissions

You will submit your code to Gradescope through GitHub Classroom. Go to the beginning of this document to look at the videos that explain the basics of GitHub if you haven't done so already. The autograder will grade: `Event.hpp`, `Event.cpp`, `Plan.hpp`, and `Plan.cpp`. Although Gradescope allows multiple submissions, it is not a place to test your code every time you make changes. You have to test it locally before uploading to Gradescope. To prevent the abuse of autograder and to encourage testing, **the autograder output will be vague**. Your code needs to compile on Gradescope in order to get any feedback from the autograder. If it doesn't compile, make sure you fulfill all the requirements mentioned in the project specifications. Once you pass the test cases on Gradescope, make sure to verify the documentation and design requirements before making the final submission.