Task Agenda Program

C++ Project Documentation

By Tasviq Hossain

Dr. Noukhovitch
ICS4U1-02
October 30, 2020

Table of Contents

Introduction

Design Synopsis

Purpose of Program: The Task Agenda is a program that enables users to add, create, delete, and view events in a created agenda. With guided prompts from the console, the user inputs information corresponding to a task with names, dates, and times. The Task Agenda stores the events and displays based on a given date or lists all the events stored. These functionalities help the user with universal skills such as time scheduling, organization, and punctuality.

Inputs and Outputs of Program: This program is structured using prompts that collect inputs to display corresponding outputs. *Table 1.1* below shows the general inputs and outputs:

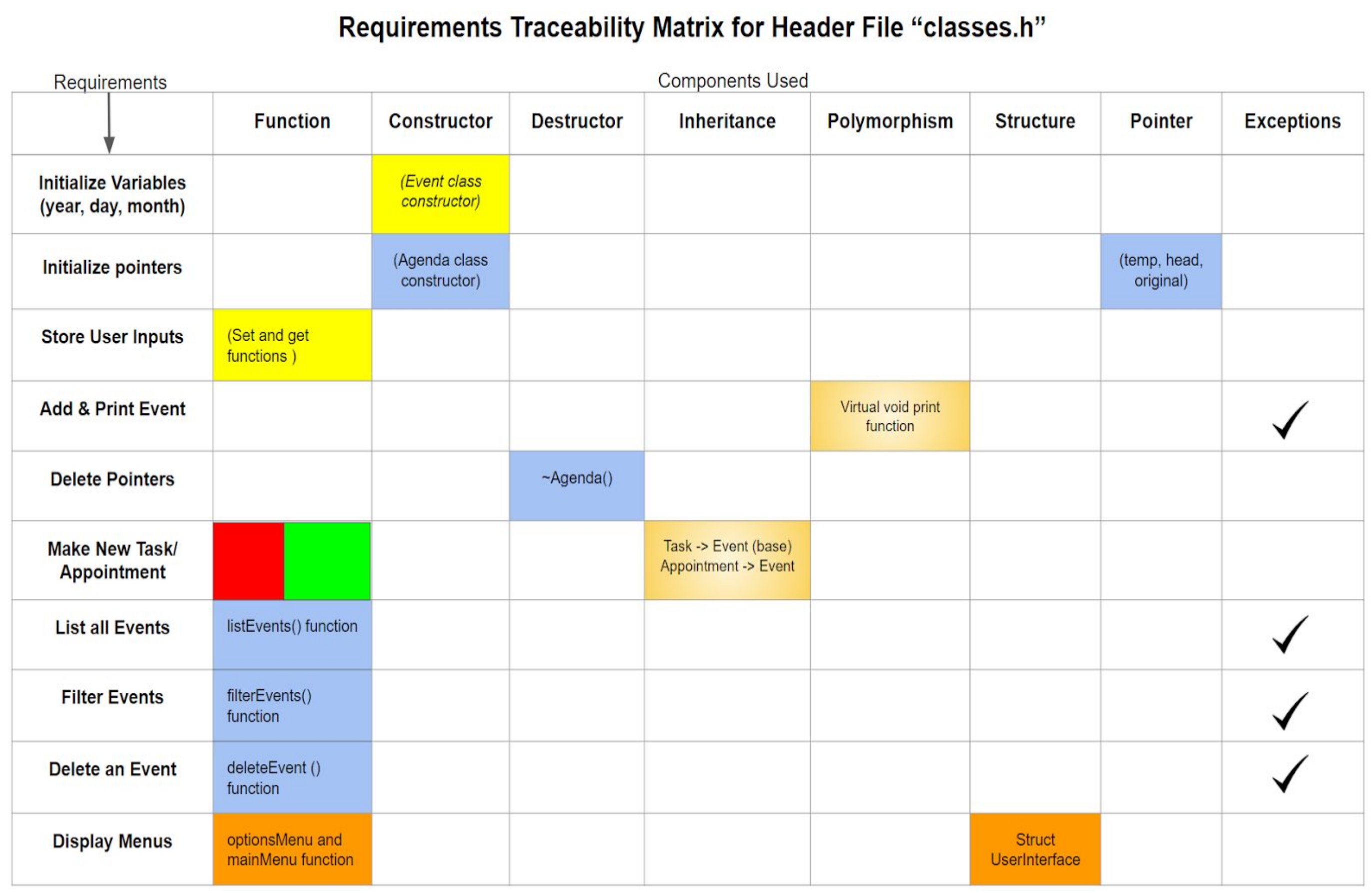
User Input	Console Output
Choice of option from Main Menu *(Ex: Add Event)*	Further prompts from the corresponding function *(Ex: addEventPrompts())*
Input another choice *(Ex: Enter 0 to add a task or 1 to add an appointment)*	Executes the next set of prompts accordingly *(Ex: Enter the name of task)*
Fill out all the data and information *(Ex: Event name, dates, times, etc)*	Display message that confirms the event *(Ex: "Event added.")*

Table 1.1: A run-through of user inputs and console outputs in the program. (Read left to right)

C++ Components Used in the Program: This program is a composition of all the concepts and syntax learned in this course. These include:

- Classes
- Inheritance
- Polymorphism
- Pointers
- Friends
- Strings
- Constructors & Destructors
- Exceptions
- Objects
- Concepts learned in the Nodes lab
- C++ Project (Header files, main.cpp)
- Functions

Requirements Traceability Matrix: The code for this project is compartmentalized into three files: two header files and a main.cpp file. The first header file contains the prototypes and the other header file manages the functionality. The main.cpp file contains the main function. *Figure 2.1, 2.2, and 2.3* are matrices that trace the requirements alongside the components used.

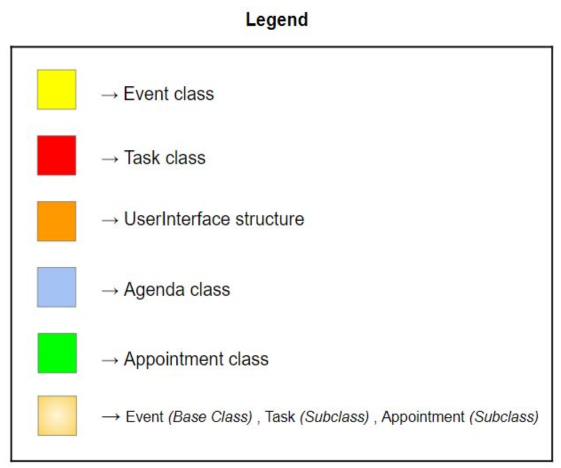


Requirements Traceability Matrix for Header File "classes.h"

Requirements ↓ Components Used

	Function	Constructor	Destructor	Inheritance	Polymorphism	Structure	Pointer	Exceptions
Initialize Variables (year, day, month)		*(Event class constructor)*						
Initialize pointers		(Agenda class constructor)					(temp, head, original)	
Store User Inputs	(Set and get functions)							
Add & Print Event					Virtual void print function			✓
Delete Pointers			~Agenda()					
Make New Task/ Appointment	🟥🟩			Task -> Event (base) Appointment -> Event				
List all Events	listEvents() function							✓
Filter Events	filterEvents() function							✓
Delete an Event	deleteEvent () function							✓
Display Menus	optionsMenu and mainMenu function					Struct UserInterface		

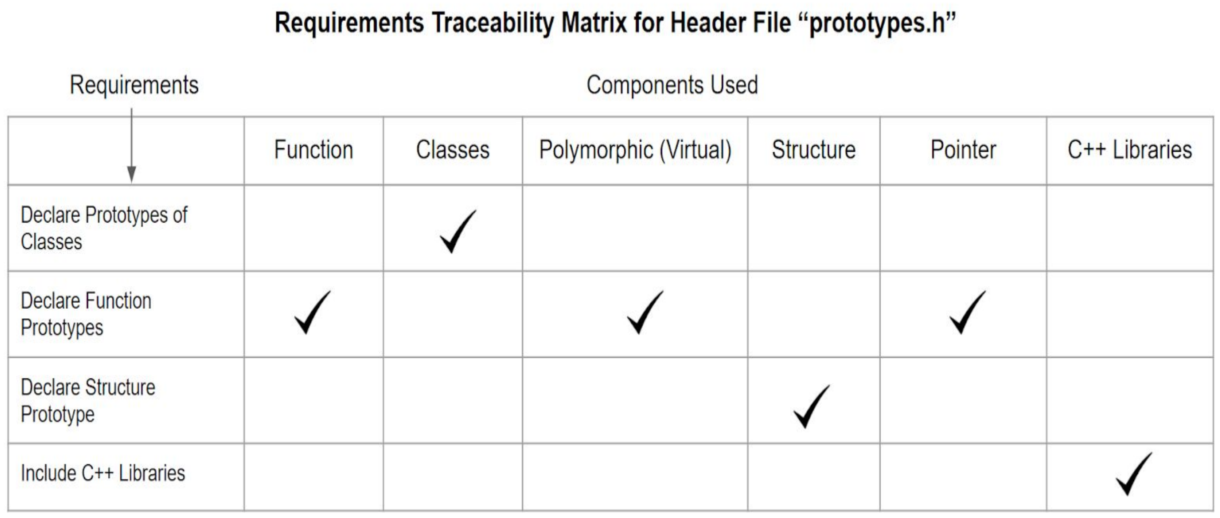
Figure 2.1 is the Requirements Traceability Matrix for the "classes.h" header file.



Legend

🟨	→ Event class
🟥	→ Task class
🟧	→ UserInterface structure
🟦	→ Agenda class
🟩	→ Appointment class
🟨	→ Event *(Base Class)* , Task *(Subclass)* , Appointment *(Subclass)*

Figure 2.11: The legend to interpret the matrix in further detail.

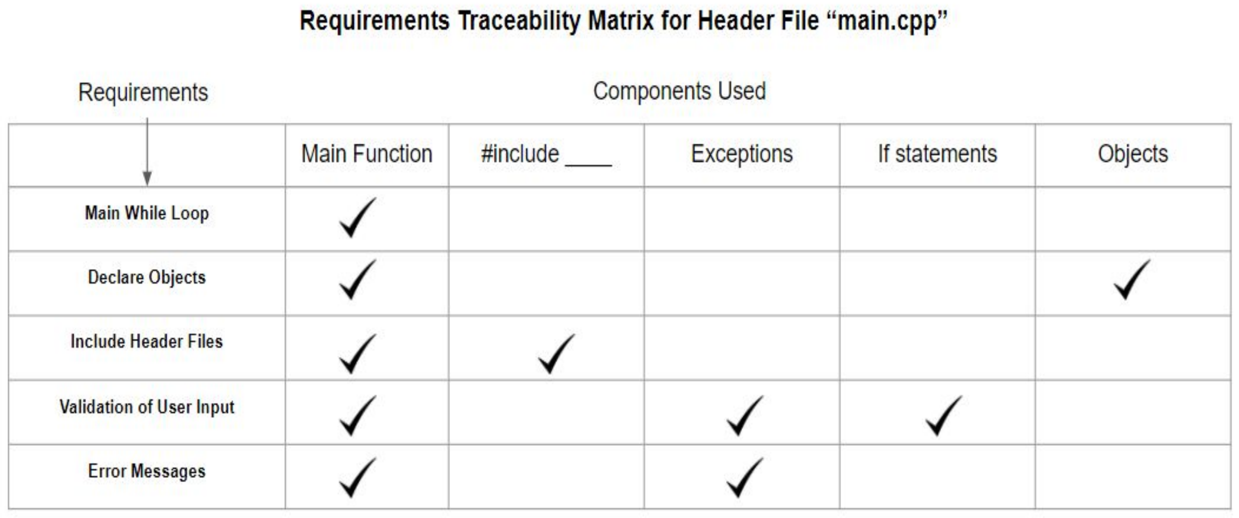


Requirements Traceability Matrix for Header File "prototypes.h"

Requirements ↓ / Components Used

	Function	Classes	Polymorphic (Virtual)	Structure	Pointer	C++ Libraries
Declare Prototypes of Classes		✓				
Declare Function Prototypes	✓		✓		✓	
Declare Structure Prototype				✓		
Include C++ Libraries						✓

Figure 2.2 is the Requirements Traceability Matrix for the "prototypes.h" header file.



Requirements Traceability Matrix for Header File "main.cpp"

Requirements ↓ / Components Used

	Main Function	#include ____	Exceptions	If statements	Objects
Main While Loop	✓				
Declare Objects	✓				✓
Include Header Files	✓	✓			
Validation of User Input	✓		✓	✓	
Error Messages	✓		✓		

Figure 2.3 is the Requirements Traceability Matrix for the "main.cpp" file.

System Architectural Design

Chosen System Architecture

As mentioned in the Requirements Traceability Matrix, the code is structured with the use of three files, all part of a C++ project folder. There are two header files: "prototypes.h" contains the prototypes of all the functions and classes and it is auxiliary to the second header file which contains the actual execution of them. The third file is the "main.cpp" which contains the main function and directs the program based on the initial user input of option choice. This system design most resembles the **Layered Pattern** discussed in the article, "10 Common Software Architectural Patterns in a Nutshell" by Vijini Mallawaarachchi. The incorporation of a C++ project "that can be decomposed into groups of subtasks, each of which is at a particular level of abstraction," mirrors the architectural pattern of my Task Agenda Program code.

<u>Header File #1 “prototypes.h”:</u>

- Including C++ Libraries
- Class Prototypes
- Structure Prototype
- Function prototypes

<u>Header File #2 “classes.h”: *(Ordered based on Class Hierarchy)*</u>

- Class Event (Base Class/Superclass)
  - Private Variables
  - Constructors
  - Setter & Getter Functions **(Data Access Layer of the Layered Pattern)**
  - Initialization of polymorphic function (virtual void print())
- Class Task and Class Appointment (Derived Classes/Subclasses)
  - Constructor
  - Print Function
- Class Agenda **(Application Layer in the Layered Pattern)**
  - Constructor
    - Assign head pointer to NULL
  - Deconstructor
    - Delete “new” pointers
  - Pointers
  - Functions **(Business Logic Layer of the Layered Pattern)**
    - addEvent, listEvents, filterEvents, deleteEvent
    - addEventPrompts, filterPrompts, deleteEventPrompts
  - Friend class
    - Friends of class Task, Appointment, Agenda
- Structure UserInterface **(Presentation Layer in the Layered Pattern)**
  - Functions
    - OptionsMenu, mainMenu, closingMenu
    - setOption, getOption

<u>CPP File “main.cpp”:</u>

- Main Function

- ➢ Declaring objects of classes
- ➢ While loop that iterates till user exits
- ➢ Exceptions: Try and Catch block for user input of choice
- ➢ Calling corresponding functions according to user input
- ➢ Display Menus

Discussion of Alternate Designs

Alternatively, the design of this program could have been structured under one C++ file. Though this would have made it more challenging to navigate through the code, a systemic layout would have made it organized. For instance, the base class would be described first. Next, the subclasses and structures would have been described followed by the main function at the end. With this design, the hierarchy of the classes as well as the various unique functions would have been sequentially ordered from top to bottom.

System Interface Design

In order to blueprint the design of this program's code, *Figure 3.1* is a flow chart that encapsulates the processes, functionalities, hierarchy, and the systematic order.

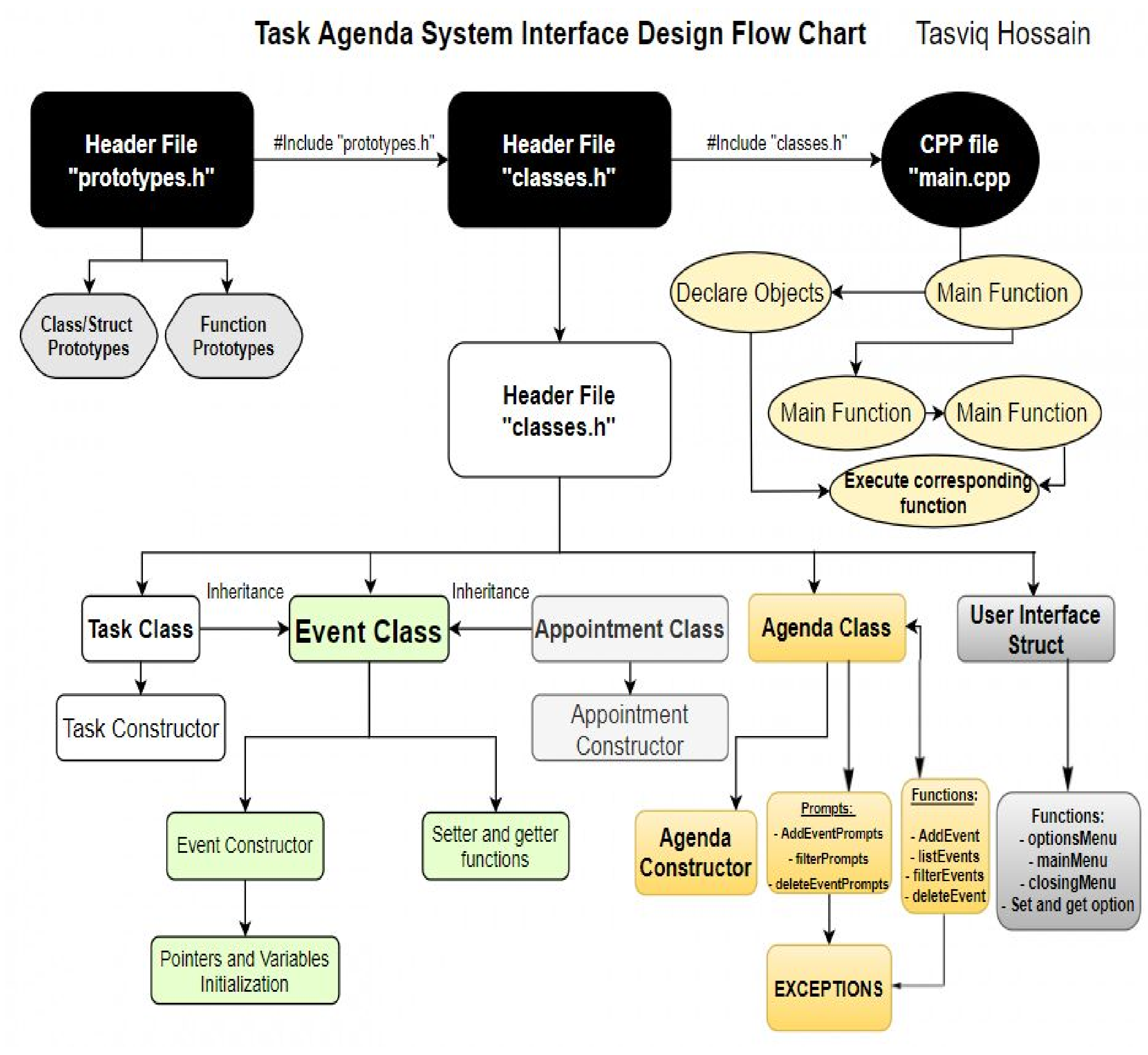


Figure 3.1 illustrates the flow chart for the system interface design of the Task Agenda Program.

Components

1. Classes
   - Event Class (Base Class)
   - Task Class (Derived Class)

  - Appointment Class (Derived Class)
  - Agenda Class

2. Structures
   - User Interface Struct

3. Constructors
   - Event Constructor
     - Declare variables to store year, month, day, hour, and id
   - Task Constructor
     - Assign the task from user input to string variable “taskname”
   - Appointment Constructor
     - Assign the name from user input to string variable “aptName”
   - Agenda Constructor
     - Initialize the pointer “head” as NULL

4. Destructors
   - Agenda Destructor
     - Deletes the “new” pointers and the head, temp, original pointers

5. Functions
   - addEvent () → Adds an event (task or appointment)
   - addEventPrompts() → Prompts to get user inputs
   - listEvents() → List the added events
   - filterEvents() → Checks for event on a specific date
   - filterPrompts() → Prompts to get user inputs of date for filterEvents()
   - deleteEvent() → Deletes an event given the ID # from user
   - deleteEventPrompts() → Prompts to get user input for event ID
   - int setOption() & getOption() → Takes in user input, stores it, returns it for use
   - mainMenu(), optionsMenu(), closingMenu() → Displays the menus
   - getYear(), getMonth(), getDay(), getHour, getId() → Get user inputs and store

6. Inheritance, Polymorphism, Friends
   - Polymorphism
     - virtual print(): Polymorphic function used in other classes to print event
   - Inheritance
     - Task class inherits from Event class
     - Appointment class inherits from Event class
     - Agenda class is friends with Task, Appointment, and Event classes which gives Agenda class access to all private variables in the respective classes

7. Variables
    - int year, int month, int day, int hour, int id → From user inputs
    - int numEvents → Event counter, used for id #
    - Bool eventAdded → Ensures that user does not choose delete event (option 4) before adding any event.
    - Bool running → Loop control variable

8. Pointers
    - Event *next → Pointer to object of class Event
    - Event *head → Pointer to object head of class Event
    - Event *temp → "Temporary" pointer object of class Event
    - Event *original → Stores the original/main data

9. Exceptions (Try and catch blocks)
    - Main Menu → to validate user input, throws exception when invalid input
    - addEventPrompts()
        - Validates user input when adding an event
        - Checks if input for dates are in realistic range, throws exception otherwise
    - filterEvents()
        - Throws exception if no events are scheduled on the chosen date
    - deleteEvent
        - Throws exception if desired event to delete is not found

10. Loops
    - Main while loop
        - Iterates each time the user does not choose to exit program
        - Controlled by the boolean running variable

11. Conditional Statements
    - If statements: Sorts the user input for choices and executes corresponding output functions/commands

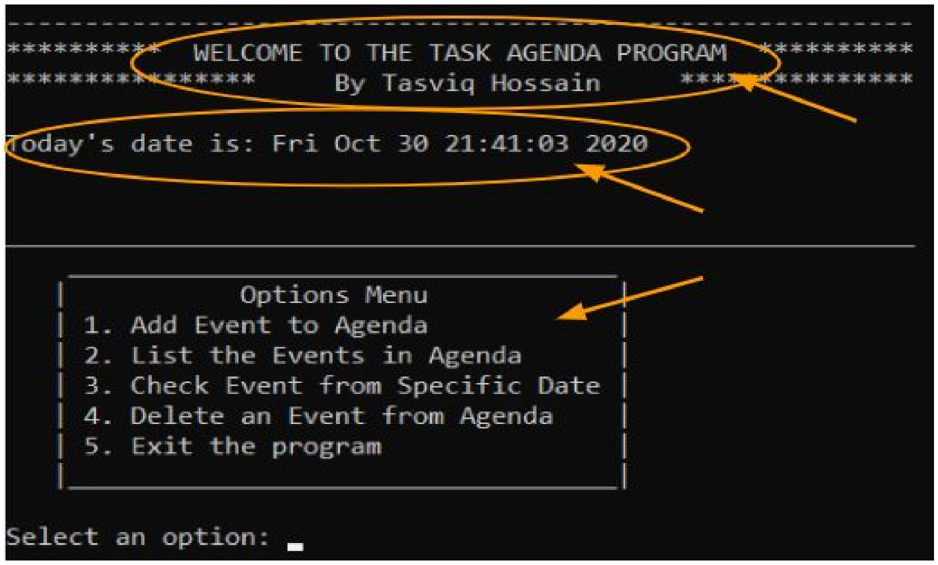
User Interface Design

Description of the User Interface

The user interface of this program is designed to achieve a user friendly platform. With the implementation of Main Menus, Option Menus, Command Prompts, Error Messeges, and an organized layout, the user is immersed into a program that is easy to navigate and use.

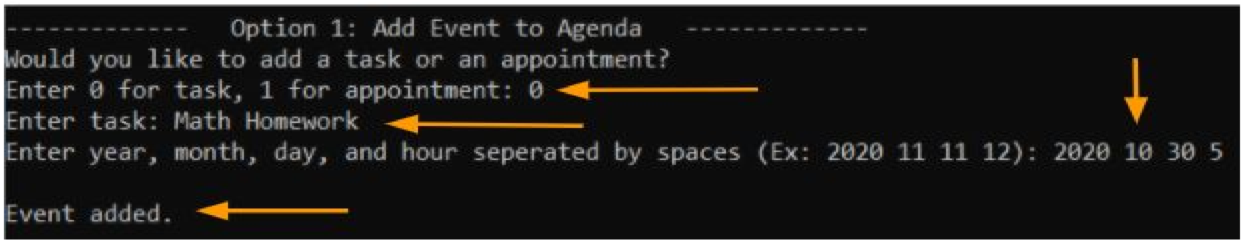
Screen Image

Upon launching the program, the user is welcomed with the Main Menu that contains the title, options menu, and the current date in the system.



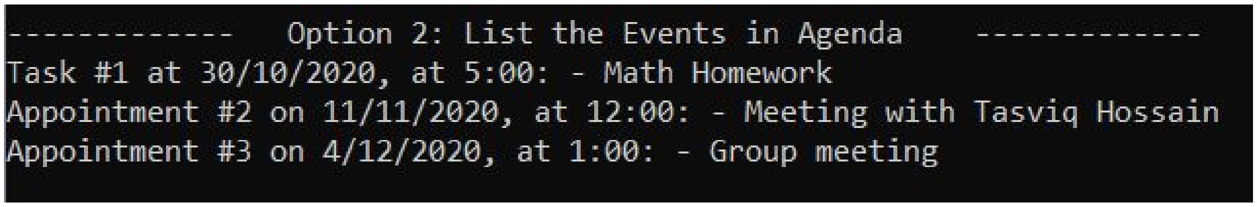
```
-----------------------------------------------------------
**********   WELCOME TO THE TASK AGENDA PROGRAM   **********
***************        By Tasviq Hossain     ***************

Today's date is: Fri Oct 30 21:41:03 2020


____________________________________________________________

   _____________________________________
   |            Options Menu             |
   | 1. Add Event to Agenda              |
   | 2. List the Events in Agenda        |
   | 3. Check Event from Specific Date   |
   | 4. Delete an Event from Agenda      |
   | 5. Exit the program                 |
   |_____________________________________|

Select an option: _
```

Figure 4.1 shows the Main Menu with the Options Menu, upon launch of program



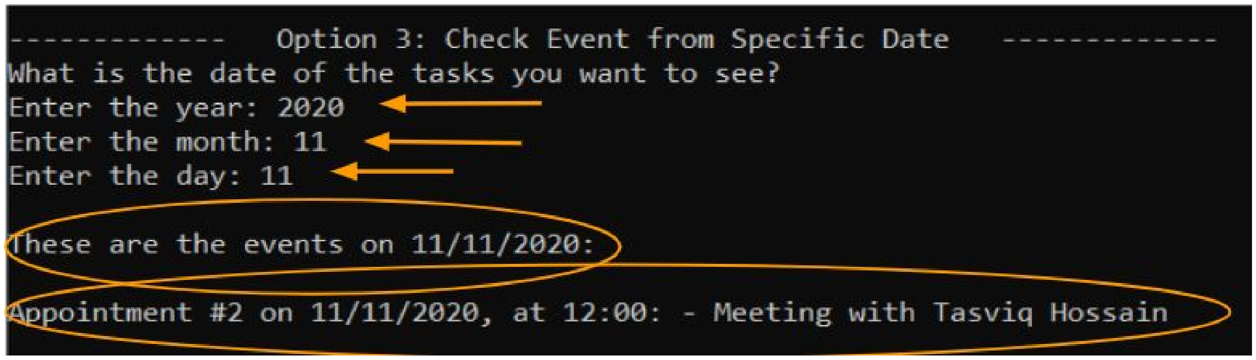
```
------------   Option 1: Add Event to Agenda   ------------
Would you like to add a task or an appointment?
Enter 0 for task, 1 for appointment: 0
Enter task: Math Homework
Enter year, month, day, and hour seperated by spaces (Ex: 2020 11 11 12): 2020 10 30 5

Event added.
```

Figure 4.2 shows Option 1 and the locations of user inputs are marked with arrows. The bottom arrow indicates a success message



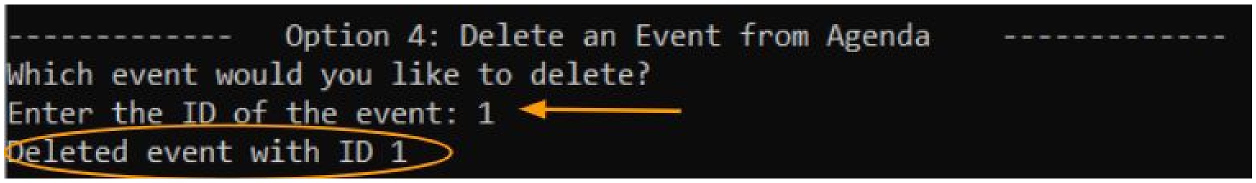
```
-------------   Option 2: List the Events in Agenda   -------------
Task #1 at 30/10/2020, at 5:00: - Math Homework
Appointment #2 on 11/11/2020, at 12:00: - Meeting with Tasviq Hossain
Appointment #3 on 4/12/2020, at 1:00: - Group meeting
```

Figure 4.3 shows the output for Option 2. The events added are listed on the basis of ID # (the order in which the events were added)



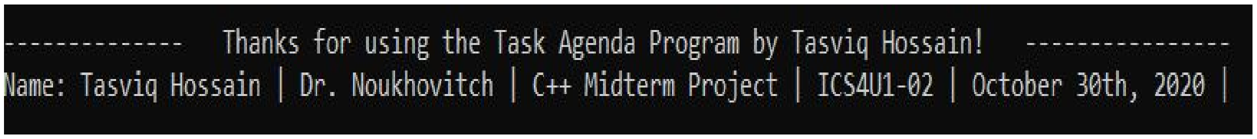
```
-------------    Option 3: Check Event from Specific Date   -------------
What is the date of the tasks you want to see?
Enter the year: 2020
Enter the month: 11
Enter the day: 11

These are the events on 11/11/2020:

Appointment #2 on 11/11/2020, at 12:00: - Meeting with Tasviq Hossain
```

Figure 4.4 shows the output for Option 3. The user is guided by prompts to enter the date. The console outputs the events on that date.



```
-------------    Option 4: Delete an Event from Agenda    -------------
Which event would you like to delete?
Enter the ID of the event: 1
Deleted event with ID 1
```

Figure 4.5 shows the output for Option 4. The user inputs the ID of the event after the prompt, and the console outputs a successful message.

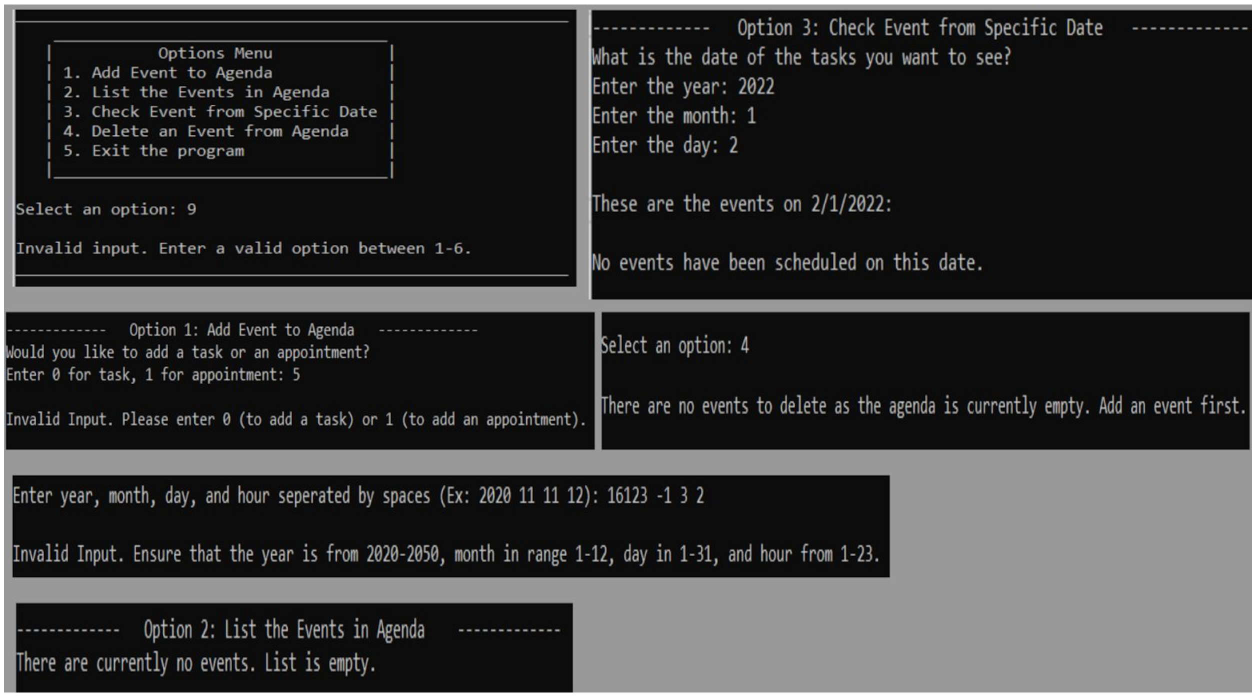


```
-------------    Thanks for using the Task Agenda Program by Tasviq Hossain!   ---------------
Name: Tasviq Hossain | Dr. Noukhovitch | C++ Midterm Project | ICS4U1-02 | October 30th, 2020 |
```

Figure 4.6 shows the output for Option 5. The closing menu is executed and the program terminates.

Testing - Exceptions & Error Messages

For each user input, there are sets of if statements that validate the user input. If an input by the user is invalid, exceptions with corresponding error messages are thrown.



```
 ____________________________________
|           Options Menu             |
| 1. Add Event to Agenda             |
| 2. List the Events in Agenda       |
| 3. Check Event from Specific Date  |
| 4. Delete an Event from Agenda     |
| 5. Exit the program                |
|____________________________________|

Select an option: 9

Invalid input. Enter a valid option between 1-6.
```

```
------------   Option 3: Check Event from Specific Date   ------------
What is the date of the tasks you want to see?
Enter the year: 2022
Enter the month: 1
Enter the day: 2

These are the events on 2/1/2022:

No events have been scheduled on this date.
```

```
------------   Option 1: Add Event to Agenda   ------------
Would you like to add a task or an appointment?
Enter 0 for task, 1 for appointment: 5

Invalid Input. Please enter 0 (to add a task) or 1 (to add an appointment).
```

```
Select an option: 4

There are no events to delete as the agenda is currently empty. Add an event first.
```

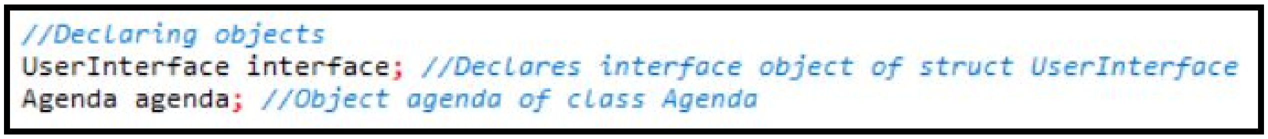
```
Enter year, month, day, and hour seperated by spaces (Ex: 2020 11 11 12): 16123 -1 3 2

Invalid Input. Ensure that the year is from 2020-2050, month in range 1-12, day in 1-31, and hour from 1-23.
```

```
------------   Option 2: List the Events in Agenda   ------------
There are currently no events. List is empty.
```

Figure 4.7 shows all the possibilities of exceptions that can be thrown.

Objects and Action

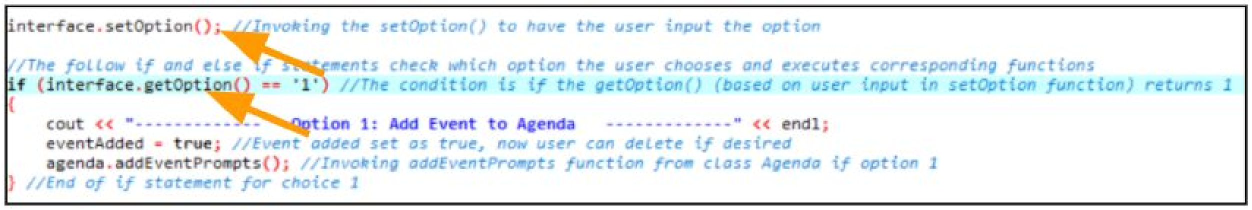
Objects: In the “main.cpp” file, the main function declares objects conventionally.



```
//Declaring objects
UserInterface interface; //Declares interface object of struct UserInterface
Agenda agenda; //Object agenda of class Agenda
```

Figure 5.1 shows the general code for declaring objects of classes.

Action: When invoking functions from the classes, the object of the class is used and the code looks like the conventional syntax.



```
interface.setOption(); //Invoking the setOption() to have the user input the option

//The follow if and else if statements check which option the user chooses and executes corresponding functions
if (interface.getOption() == '1') //The condition is if the getOption() (based on user input in setOption function) returns 1
{
    cout << "-------------   Option 1: Add Event to Agenda   -------------" << endl;
    eventAdded = true; //Event added set as true, now user can delete if desired
    agenda.addEventPrompts(); //Invoking addEventPrompts function from class Agenda if option 1
} //End of if statement for choice 1
```

Figure 5.2 is a snippet of code that showcases the syntax used when invoking functions from objects of classes.

Points of Improvement & Summary

As a next step in programming, I would like to include more structure to the code. An important reminder is that the overdependence on the main function to do all the programming is something to avoid. Though this project was a massive stepping stone in my experience with object oriented programming, I would like to continue experimenting with objects and pointers. The potential of creating programs with the inclusion of pointers and class objects is endless, and I want to use this as the beginning of my experimentation with all these concepts. Throughout this course, the labs and exercises paved a solid foundation in implanting the important concepts I used in this project.

For this particular program, I can improve the Task Agenda by implementing a system where a user can save the events into a file and the program can input and output from the file. This would add a real-life use to this program. Nonetheless, the functionality and the execution of the current program is useful and convenient.

All in all, this Project as well as the Documentation opened doors to many interesting topics such as System Architectural Designs, Requirements Traceability Matrix, use of Classes and objects, working with pointers, Inheritance, Polymorphism, and various other concepts.

Tasviq Hossasin | ICS4U1-02
Dr. Noukhovitch
October 30, 2020
C++ Project & Documentation: Task Agenda