

Professor Tuka Alhanai

Object Oriented Programming

ENGR-UH 2510, Fall 2022

Student Flavia Trotolo

05-04-2022

Final Project

X-Nerd Calendar in C++

The aim of the course project is to give you an opportunity to synthesize topics you have learned during the Object-oriented Programming Course into a single system embedded in a real-world engineering challenge. your program should be implemented to solve in a simplistic way your chosen project topic. It should not need to be overly complicated, but it should demonstrate optimizations for speed and memory, as well as object oriented programming paradigms. The code should be readable, with a clear output to the console

Introduction

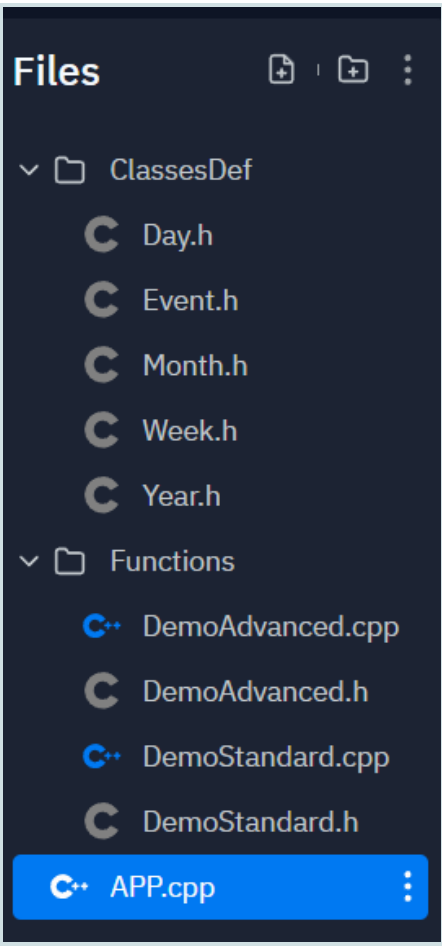
The Project is a program which creates a customizable **calendar** for the user.

The Program has the following **functionalities**:

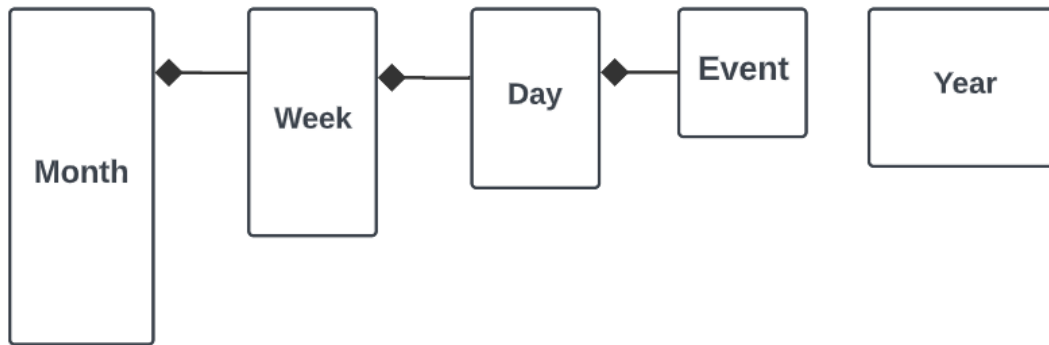
- Allows users to create **objects** of class Event, Day, Week, Month, Year. User can display all created objects on console using public functions for print in objects
- Display the calendar for year in table format (year will be chosen by the user) Original code by (1)
- User can create empty objects of class Week or Month and secondly fill them with events
- User can create objects of class Week given vector of Day objects previously created (advanced feature)
- User can create objects of class Month given vector Week objects which were previously created (advanced feature)

Approach

The program has the following structure:

	<p>An header file is created for each class (OO)</p> <p>Folder ClassesDef: contains distinct header files for each class):</p> <ul style="list-style-type: none">• Day.h: contains class Day• Event.h: contains class Event• Month.h: contains class Month• Week.h: contains class Week• Year.h: contains class Year <p>Folder Functions:</p> <ul style="list-style-type: none">• DemoAdvanced.cpp: Include sample functions for advanced features: CreateMonth(), SummerTerm()• DemoStandard.cpp: Include samples functions for standard features: CreateDay(), CreateCostumWeek(), CalendarPrint(int &&theYear), CreateEmptyWeek() <p>APP.cpp: contains main function. In App.cpp the standard functionalities of the program are demonstrated:</p> <ul style="list-style-type: none">• CalendarPrint(2020+1): prints the calendar for year 2021 in table form• CreateDay(): Create objects of type Day, create Events for that Day and prints Day object• CreateCostumWeek(): Create Week object, fills it with events and prints it• CreateCostumMonth(): Create object of type Month, fills it with events and prints it
--	---

Simplified UML Diagram is shown below:



Note: (In this code composition is used rather than inheritance) composition shown by solid diamond arrows. Complete UML Diagram for each class shown in Solution Section

Solution

*(some but not all of the private variables are mentioned in the Explanation Right column for the sake of conciseness). OP1, OP2, OP3, OP4, O1, O2, O3, O4 are codes which connect each point to further explanation in **Project Evaluation Scheme** Section*

Event.h: The file contains the class Event.

<pre> Event EventName: string Month: int Day:int Duration: double year: int StartingTime: double Event(); Event(name:string, theDay:int, theMonth: int, double theStartingTime: double theDuration, theYear:int): Print(): void friend SortEvents(events: vector<Event>): *Event </pre>	<p>Event stores private variables (OP1) for the name of the event, day, month and year of the event, and the duration.</p> <p>Event has the following public functions:</p> <ul style="list-style-type: none"> • Event(): default constructor • parameterized constructor which completes information of private variables as per user input • void print(): Prints information of the Event • Event * SortEvents(vector<Event> events): sorts an array of events in ascending order for a given day on the base of their starting time. Global friend function accessible by other classes (OP2) (O1)
---	--

Day.h: contains class Day

<div data-bbox="203 279 581 342">Day</div> <div data-bbox="203 342 581 535"> WeekDay int; Dayy: int; Month: int; Year: int; DayName: string; EventsDay vector<Event> ; EventsBlock[Time::hoursDay]: string </div> <div data-bbox="203 535 581 955"> Day() Day(int theWeekDay) getWeekDay(): int; setEvent(name:string, theDay:int, theMonth: int, double theStartingTime: double theDuration, theYear:int): void; printDay(): void; printTitle():void; printEvent():void; </div>	<p>Day Contains private variables:</p> <ul style="list-style-type: none"> • WeekDay: is the index of a day (varies from 0 to 6, 0 corresponds to Sunday, 6 to Saturday) • DayName: is a string containing the name of the day • EventsDay: A vector of objects Event which stores events for a given day • EventsBlock: Array storing an event. Maximum number of events for a day is 24, the number of hours in the day <p>Public Function:</p> <ul style="list-style-type: none"> • Day() : Default constructor creates an empty day • Parameterized constructor creates day based on the index • getWeekDay(): Returns the index of that day; (OP1) • setEvent uses parameterized constructor of function Event to create a new event for the Day object, Stores name of event in EventsBlock • getMatrixEvent() Returns Pointer to copy of EventsBlock in heap: (O1) • printDay() Sort events calling global friend function SortEvents (O2) delete dynamically allocated memory in heap for array of events • void printDay() Prints Day and its Events.
---	--

Week.h: contains class Week

<div data-bbox="186 1150 634 1213">Week</div> <div data-bbox="186 1213 634 1350"> Week vector <Day> ; WeekId int; Monthz: int, Year: int, MatrixEvents[Time::hoursDay]: string title: string, date: string; startHour: int </div> <div data-bbox="186 1350 634 1759"> Week() Week(vector: vector<Day>) CreateEvent(name:string, index:int, theStartingTime: theDuration: double,) setWeekDate(day: int, month: int, year: int): void Void setWeekId(ID: int) Void setWeekTitle(theTitle: string) getWeekID() int printWeek(): void printEvents():void printWeek():void </div>	<p>Private Variables:</p> <ul style="list-style-type: none"> • MatrixEvents: 2d Array of strings containing name of event at a time for a specific day of week • Week: vector of Days (OP3) • WeekId: index which signifies the position of the week in a month. Last week of month has position 3. If WeekId > 4, object of type Month becomes longer than 4 weeks • startHour: the starting time for which the calendar should be printed <p>Public Functions:</p> <ul style="list-style-type: none"> • Week(): Default constructor, creates a default week of seven empty Days • Week(vector<Day> vector): Parameterized constructor, accepts a vector of Days, to fill the object of type Week with non-empty Day objects. Calls getMatrixEvent() for each day in Week to obtain a pointer to copy of in heap EventsDay, fills MatrixEvents (O2) (O3). • Setters and getters: setWeekId,setWeekDate,setWeekTitle, getWeekID (OP1) • printWeek(): Prints the schedule for a week in tabular format (library iomanip), by calling function returnDay for each Day object in vector Week. Then prints MatrixEvents; (OP3)
---	--

Month.h: contains class Month

<div>Month</div> <div> theMonth: vector <Week> ; Month: int; Year: int; dateMonth: string; titleMonth: string </div> <div> Month() Month(vector: vector<Week>) Month(length: int) printMonth() void SetTitleMonth(titleMonth: string): void; setMonthDate(month: int, year: int) CreateEvent(name:string, Weekldd: int, DayIndex: int, theStartingTime: int, theDuration: int) </div>	Private Variables: <ul style="list-style-type: none"> dateMonth: string which represents the date of the month (in format month/year) theMonth: is a vector of objects of class Week (OP3) Public Functions: <ul style="list-style-type: none"> Month(): Default constructor, creates an object of class Week which is empty, fills vector theMonth with the empty object Week four times. Default month should have four weeks Month(vector<Week> vector): Parameterized constructor replaces empty objects of type week in theMonth, with non-empty objects of type Week contained in vector of objects Week which is the parameter. The number of objects of Week in Vector can exceed 4 to create an expanded month like the SummerTerm in main SetTitleMonth and setMonthDate are setters for dateMonth and titleMonth respectively. (OP1) printMonth(): displays the month in tabular form. This is done by calling printWeek for each Week object contained in theMonth.
--	--

Year.h

The class for Year adapts code of Techsane Girl to Object Oriented Programming. Original code at <https://github.com/debasree888/calendar-in-c/blob/master/Calendar-in-C-master/main.c> some changes were included to the code for the sake of this calendar application

<div>Year</div> <div> year: int, month: int day,: int MonthLength: int theDayStart: int startYear: int Monthss: vector <pair<string, double>> </div> <div> Year(time: int) setYear(time: int) :int print():void </div>	Private variables: (OP1) <ul style="list-style-type: none"> Monthss:vector <pair<string, double>> : map which we created on top of the original code by TechSane to facilitate association between a given month and number of week. (OP3) Information of the objects of class Year shall not be accessed outside the objects, hence private variables are created. Hence, we can improve TechSane code by allowing for Encapsulation. Public Functions: <ul style="list-style-type: none"> Parameterized constructor creates the year as per user input. setYear is a setter that we created to possible modify the Year. This could be useful for a loop which increments on multiple years (OP1) print(): displays calendar for year in tabular form
---	--

Project Evaluation Scheme: The program that I described is worth 14/15 points.

The program is correct, readable. OP1, OP2, OP3, OP4, O0, O1, O2, O3, O4 are codes which connect each point of explanation below to tables **Solution** Section.

Object Oriented Programming Paradigms

1. **OP1: Creation of classes** (Year in Year.h, Month in Month.h, Day in Day.h, Week in Week.h, Event in Event.h) and we instantiated objects for each of these classes. **Encapsulation** was achieved through creation of **private variables** and **public functions**. We used **getters and setters** to retrieve private variables only when necessary. We improved the code by Techsane Girl by adapting her code to a class Year which has encapsulated private variables and public functions for constructor and print which allow for greater abstraction, setters were added to her code.
2. **OP2: Global Friend Function** SortEvents in Event class, is a global friend function. The function can be accessed safely by objects of class Day, Week, Month, to arrange the events in ascending order by event property startingTime. Position: ClassesDef/Event.h; line 38-46

```
ClassesDef/Event.h ×
38 //GLOBAL FRIEND FUNCTION can be used by all classes safely to organize events
39 ▼ friend Event* SortEvents(vector<Event> events) { // Orders events in ascending order given
vector of events
40     Event* EventPointer = new Event[10]; //create an empty array of events in heap
41
42     sort(events.begin(), events.end(), [](const Event& lhs, const Event& rhs) {return
lhs.StartingTime < rhs.StartingTime; }); //sorts by object property StartingTime
43 ▼     for (int i = 0; i < events.size(); i++) {
44         EventPointer[i] = events[i]; }
45     return EventPointer; //returns pointer pointing to array of events in order by starting
time
46     }
47
```

3. **OP3: Usage of STL Library** to use data structures vector and map. io manip used extensively:
 - Vector: user in constructor for classes Week, Month
 - Map: user in constructor for class Year
 - Iomanip library: used in function printWeek(), in ClassesDef/Week.h lines 100 to 123. Library used to display schedule in tidy tabular form
4. **OP4: Namespace** Time created in class Day with constant variables which can be used across classes. Position: in ClassesDef/Day.h line 10

```
ClassesDef/Day.h ×
9 //namespace Time include constants which are relevant in Day and Week class
10 namespace Time { const int hoursDay = 24; const int DayWeek = 7;}
11
```

optimizations for speed and memory:

1. **Organization of classes and functions into header files (O0)** optimizes the logic of repository, although this may relent speed in the absence of multithreading
2. Allocation of **dynamic memory in public functions (O1)** speeds up transferring information between classes by returning pointers to memory blocks in heap, as opposed to declaring dynamic memory connected to a private variable in class. In addition we create dynamic memory only if the function is called, while the other method allocates memory always when the constructor is called (position ClassesDef/Event.h; lines: 38-39; line 45)

```

ClassesDef/Event.h x
38 //GLOBAL FRIEND FUNCTION can be used by all classes safely to organize events
39 friend Event* SortEvents(vector<Event> events) { // Orders events in ascending order given
vector of events
40     Event* EventPointer = new Event[10]; //create an empty array of events in heap
41
42     sort(events.begin(), events.end(), [](const Event& lhs, const Event& rhs) {return
lhs.StartingTime < rhs.StartingTime; }); //sorts by object property StartingTime
43 for (int i = 0; i < events.size(); i++) {
44     EventPointer[i] = events[i]; }
45     return EventPointer; //returns pointer pointing to array of events in order by starting
time
46 }
47

```

3.

4. **Deallocation of dynamic memory to prevent leakages**(O2) Unless we deallocate dynamic memory the program will keep allocating memory in heap which is not efficient (memory leakage) . position: ClassesDef/Week.h line 63

```

ClassesDef/Week.h x
56 if (day >= 0 && day < Time::DayWeek) {
57     week[day] = vector[i];
58
59     string*eventsCopy = vector[i].getMatrixEvent(); //take the address to string
array in heap with name of events
60 for (int k = 0; k < Time::hoursDay; k++) {
61     MatrixEvents[k][day] = eventsCopy[k];
62 }
63 delete[]eventsCopy; //deallocate dynamic memory
64 }

```

5. **Use of pointers**(O3) rather than creating another array or vector to copy values saved in heap memory which are retrieved with getMatrixEvent (picture above, position: ClassesDef/Week.h line 59)
6. **Passing by Rvalue** to function: we pass temporary value to Year constructor as we only want to print calendar, no need to allocate memory to keep track of year printed. position: Functions/DemoStandard.cpp line 38)

```

Functions/DemoStandard.cpp x
38 void CalendarPrint(int &&theYear) { //shows how external code of Techsane Girl can used to
increase functionalities of our program. Original code at https://github.com/debasree888/calendar-
in-c/blob/master/Calendar-in-C-master/main.c
39     Year Freshman(theYear);
40     Freshman.print();
41     // Freshman.setYear(2021);
42     //Freshman.print();
43 }
44

```

Nonetheless, it is essential in the future to add multithreading to the code, as very long computationally heavy functions are called in main. This was not done as I failed to learn advanced multithreading on my own to face interactions of header files and classes. Hence, I would rate the program 14/15 as there is space for improvement regarding the speed of the application.

Bibliography

1. GitHub - talhanai/public--OOP-Spring22: Lecture materials for ENGR-UH 2510 Object-oriented Programming taught at NYUAD in Spring 2022. (2022). Retrieved 12 May 2022, from <https://github.com/talhanai/public--OOP-Spring22>.

2. Techsane Girl: calendar-in-c/main.c at master · debasree888/calendar-in-c. (2022). Retrieved 12 May 2022, from <https://github.com/debasree888/calendar-in-c/blob/master/Calendar-in-C-master/main.c>

ENGR-UH 2510, Fall 2022 | New York University Abu Dhabi

Student's Evaluation:

Instructor's Evaluation:

Code: Readability: 3 / 3 / 3

Optimization: 2 / 3 / 3

Concept #1: pass by r value_ Line numbers(s): Functions/DemoStandard.cpp line 38

Concept #2: pointer Line numbers(s): ClassesDef/Week.h line 59

Concept #3: passing dynamic memory in public functions Line numbers(s): ClassesDef/Event.h: lines: 38-39; line 45

Concept #4: dynamic memory deallocation_ Line numbers(s): ClassesDef/Week.h line 63

(Concept #5: Multi-File code_ Line numbers(s): Read *Approach* Section)

OOP Paradigms: 3 / 3 / 3

Concept #1: Encapsulation Line numbers(s): ClassesDef/Year.h: line 13-14

Concept #2: **Global Friend Function** Line numbers(s): ClassesDef/Event.h: line 38-46

Concept #3: **STL Library** Line numbers(s): ClassesDef/Week.h lines 100 to 123

Concept #4: **Namespace** Line numbers(s): Position: in ClassesDef/Day.h line 10

Output: 3 / 3 / 3

Correctness: 3 / 3 / 3

Report: Engaging: 3 / 3 / 3

Grammar: 3 / 3 / 3

Structure: 3 / 3 / 3

Visual: 3 / 3 / 3

Content: 3 / 3 / 3

References: 3 / 3 / 3

TOTAL: / 33