BTP305 Fall 2016 – Term Project

ARAIG Evaluation Control Software

Learning Outcomes

Upon successful completion of this term project, you will have demonstrated the abilities to:

- Use the std::string class to manage character data
- Parse a field-delimited std::string record into a set of std::string tokens
- File I/O using ifstream
- Access data common to all objects of a class using class variables and class functions
- Code a copy and move constructor and operator
- Handle overloaded operators
- Implement composition and aggregations
- Display information in aligned tabular form
- Use STL containers
- Implement Polymorphic objects
- Handle dynamic memory allocation/deallocation

Introduction to the Project

The final project has been designed to help develop some evaluation code for a cross-disciplinary applied research project between the School of Aviation and the School of Information and Communications Technology. The purpose of the project is to design and develop evaluation software to control a commercial Virtual Reality suite called ARAIG (As Real As It Gets). You are **NOT** required to interface directly with the suit, you will be developing an object oriented shell that could be adapted to control the suit.

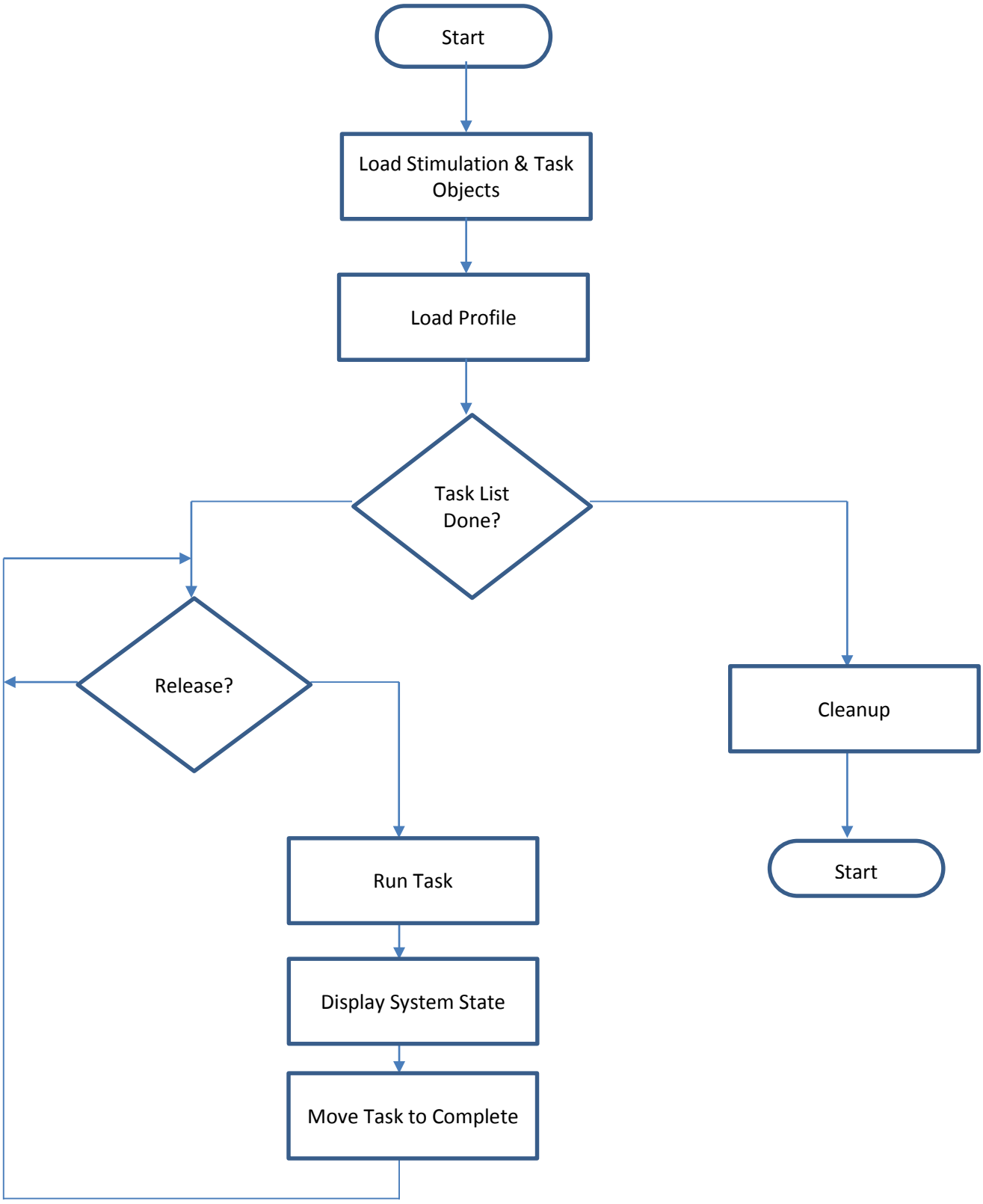
The complete solution is built in four milestones defined as follows:

1. Stimulations – handles the functionality of all the stimulation interfaces of the ARAIG suit
2. Tasks – handles the definitions of tasks, which include 1 or more stimulations
3. ARAIG_Sensors – An object that will read, create and store stimulation and Task objects
4. Profile – handles the definition of the flight plan, execution and user related control of the suit

The project has been developed to provide the School of Aviation with a means to configure flight scenarios and execute the scenarios on students flying the FRASCA simulators. The software shall provide the following functionality:

- Instantiate ARAIG Sensor objects based on a configuration file of definitions
- Instantiate a set of pre-defined Task objects based on a configuration file of definitions
- Instantiate a Profile object to control the execution and release of stimulus based on a flight plan file and user input
- Write execution status of the simulation to a log file throughout the run

The figure below illustrates the flow of the system.


Start
Load Stimulation & Task Objects
Load Profile
Task List Done?
Release?
Cleanup
Start
Run Task
Display System State
Move Task to Complete

Milestone #1

Design and implement the ***Stimulation***, ***Stims*** and ***Exoskeleton*** class definitions. These classes define a mapping between the required information from the flight instructors to the technical capabilities of the ARAIG suit.

Class Stimulation

The ***Stimulation*** class is an *abstract base class* that will be used to create *polymorphic objects*. The purpose of the ***Stimulation*** class is to provide a common data type between the different inputs the ARAIG suit provides. Your ***Stimulation*** class should store the ***name*** of the defined ***Stimulation*** and have a *pure virtual function display* that will allow all inheriting concrete classes to have a means to the output of the requested class to the *ostream&* provided.

Class Stims

The ***Stims*** class is a concreate class that inherits ***Stimulation***. It defines the interface to the ARAIG suit "Stims" functionality. Your ***Stims*** class should contain the following information:

- The *name* of the ***Stims*** object
- The *location* of the ***Stims*** on the suit, which can be *{abs, front, back, traps}*
- The *intensity* value of the ***Stims*** ranging from 0% - 100%
- The *frequency* value of the ***Stims***
- The *duration* (in seconds) of how long the ***Stims*** stimulation should be active

Appropriate constructors/destructors and member functions shall be created to support the instantiation and configuration of the ***Stims*** object.

Class Exoskeleton

The ***Exoskeleton*** class is a concreate class that inherits ***Stimulation***. It defines the interface to the ARAIG suit "Exoskeleton" functionality. Your ***Exoskeleton*** class should contain the following information:

- The *name* of the ***Exoskeleton*** object
- The *intensity* value of the ***Exoskeleton*** ranging from 0% - 100%
- The *duration* (in seconds) of how long the ***Exoskeleton*** stimulation should be active

Appropriate constructors/destructors and member functions shall be created to support the instantiation and configuration of the ***Exoskeleton*** object.

Milestone #2

Design and implement a ***Task*** module that contains a STL container *List* of ***Stimulations*** to be executed.

Class Task

A ***Task*** is an object that consist of one or more ***Stimulations*** in the order in which they would be executed. Your ***Task*** object should allow for instantiation in a safety state with no defined ***Stimulation*** objects or with a *List* of ***Stimulations***. Once created your object should allow for the addition and subtraction of ***Stimulations***. A ***Task*** should contain a unique *name* provided at creation and a *List* of ***Stimulations***.

As a minimum, your ***Task*** class should contain the following functionality:

- A default constructor that puts a newly created ***Task*** in a safe state
- A copy constructor and operator to allow for ***Task*** object copy operations
- A move constructor and operator to allow for ***Task*** object move operations
- An overloaded operator to add ***Stimulation*** objects using +=
- An overloaded operator to access ***Stimulation*** objects using []
- A function that allows the removal of a ***Stimulation*** based on ***Stimulation*** *name*
- A *dump* function that writes the ***Task*** configuration to an *ostream&*
- An *execute* function that takes an *ostream&* and activates each ***Stimulation*** that has been configured in the **Task** *List*
    - NOTE: Execution *displays* the details of each of the ***Stimulation*** objects to the *ostream&* provided

Milestone #3

Design and develop an ***ARAIG_Sensors*** module that loads all the pre-defined ***Tasks*** and ***Stimulations*** from configuration files.

Class ARAIG_Sensors

An ***ARAIG_Sensors*** module performs the initial configuration of the system using information provided in *ASCII delimited text files*. Your ***ARAIG_Sensors*** class should *dynamically allocate memory* to create and store ***Stimulation*** and ***Task*** objects defined in the input files. As a minimum, your ***ARAIG_Sensors*** class should contain the following functionality:

- A constructor that takes *two filenames* containing the ***Stimulation*** and ***Task*** definitions, reads the input file and dynamically creates objects based on the parsed information from the file
- A *dump* member function that will write the contents of memory (all the dynamically created ***Stimulation*** objects information) to an *ostream&*
- A destructor that cleans up all dynamically allocated memory at end of life

The stimulations configuration file will contain one ***Stimulation*** per line with parameter information delimited by a comma (,). An example of a stimulations configuration file is as follows:

Example File: stim, frontShoulder, front, 90, 55,5 exoskeleton, backvib,75,10

The task configuration file will contain the definition of a ***Task*** on a single line, following by 1 or more lines defining the ***Stimulations*** that should be used. Each ***Task*** definition will start with the key word *"TASK"* to determine the transition between two ***Task*** definitions. An example of a task configuration file is as follows:

```
TASK SampleTask
frontShoulder
backvib
TASK SampleTask2
backvib
TASK SampleTask3
….
```

Milestone #4

Design and develop a ***Profile*** module used to define a student and load a flight plan of tasks to be executed for the given student.

Class Profile

Your ***Profile*** class should contain, as a minimum, the *students first* and *last name*, *student number*, flight *instructors first* and *last name* and instructors *employee number* and a *structure* that defines the *calibration* *max* and *min intensity* values for the given student. Upon creation your ***Profile*** module should take the *name of the file* that contains the student information, instructor information and the flight plan to be executed. The flight plan defines the list of ***Tasks*** that will be executed. At startup your ***Profile*** class should *copy* the ***Tasks*** into a *ToRun* STL vector container. Once as ***Task*** is completed the ***Task*** should be *moved* from the *ToRun* container over to a *Completed* STL vector container.

As a minimum, your ***Profile*** class should contain the following functionality:

- A constructor that takes a *filename*, *ostream&*, *ARAIG_Sensors&* and creates the profile based on the information parsed from the file
- A function that displays all the ***Tasks*** currently in the *ToRun* container to the screen
- A function that displays all the ***Tasks*** currently in the *Completed* container to the screen
- A function that displays the next ***Task*** to be executed to the screen
- A function that displays the last completed ***Task*** to the screen
- A *run* function that executes the flight plan. Based on user input, this function will execute the next ***Task*** in the *ToRun* container and then move the ***Task*** to the *Completed* Container.

*NOTE: The execution of a **Task** simply calls the display function of the **Stimulations** objects to show the **Task** has been executed.*

The profile configuration file will use the following format:

```
Student First Name, Last Name, Student Number
Instructors First Name, Last Name, Employee Number
CalMax, CalMin
Task 1 Name
Task 2 Name
…
Task N Name
```

Main

Your main program should take in the following command line arguments

- Filename of the profile configuration file.
- Filename of the Stimulation configuration file
- Filename of the Tasks configuration file
- Output Filename

The main program should then perform the following operations:

- Create an ***ARAIG_Sensors*** object using the Stimulation and Tasks configuration files provided
- Open a *ostream* object using the output filename provided
- Create a ***Profile*** object using the profile configuration file and *ostream* object
- Run the ***Profile***

Due Dates

The milestones for this project as due as follows:

- Milestone #1 Due Friday Nov 18th @ 11:59pm
- Milestone #2 Due Friday Nov 24th @ 11:59pm
- Milestone #3 Due Friday Dec 2nd @ 11:59pm
- Milestone #4 Due Friday Dec 9th @ 11:59pm

Submissions

All submissions shall be through the course Blackboard site. All submissions should be a single ZIP file containing the following information:

- All source files (*.h and *.cpp)
- Input ASCII text files created for testing purposes

For the Milestone #4 you should submit all the above information, but also include the following:

- Output ASCII text files created
- Screenshots showing the program working using standard out

Marking Rubric

This is a problem based assignment. Meaning I don't have a solution programmed to compare your solutions against. I will be evaluating your design and implementation based on the learning outcomes and application of the topics taught in class over the semester.

Milestones 1, 2 and 3 will be given a mark based on completion and whether or not it was submitted on time.

Milestone 4 will be looking at the quality of your code, have you followed the specifications and implemented the minimum requirements and have you followed the topics taught in class and demonstrated you understand the Learning Outcomes specified at the beginning of this document.

Description	Max Marks	Summary
Use of std::string	2	Proper use of the string class/member functions to perform the string parsing
File I/O	5	Proper use of the ostream and istream objects for handing input and output of the system
Data Access	5	Have you encapsulated and protected the data within all the defined objects. Are you interfacing with the objects using the appropriate member function calls
Copy/Move	8	Have you implemented the copy/move constructors and operators correctly? Have you used them appropriately within the execution of the Profile object
Overloaded Operators	2	Have all the overloaded operators been implemented correctly. Were they used to manipulate the objects data throughout the system
STL Containers	3	Have you used vector and list correctly where required. Did you use iterators to manipulate the data within the containers
Polymorphic Object	2	Did you create the polymorphic objects correctly? Is your abstract and concrete base classes implemented appropriately
Dynamic Allocation	2	Have you correctly handled dynamical allocation. Do you objects create safe empty states? Do they deallocate the data at the appropriate time? No memory leaks are visible in the design.
Execution	1	Do the code compile and execute correctly?
TOTAL MARKS	30	