Part B – Foundations

Dynamic Memory

Workshop 2 V0.6 (in-lab draft only)

In this workshop, you use references to modify content of variables in other scopes, overload functions and allocate memory at run-time and deallocate that memory when it is no longer required.

Learning Outcomes

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

- allocate and deallocate dynamic memory for an array;
- allocate and deallocate dynamic memory for a single variable
- overload a function;
- Create and use references

Submission Policy

The workshop is divided into 3 sections;
in-lab - 30% of the total mark

To be completed before the end of the lab period and submitted from the lab.

at-home - 35% of the total mark

To be completed within 2 days after the day of your lab.
DIY (Do It yourself) – 35% of the total mark

To be completed within 3 days after the at-home due date.

The *in-lab* section is to be completed after the workshop is published, and before the end of the lab session. The *in-lab* is to be submitted during the workshop period form the lab.

If you attend the lab period and cannot complete the *in-lab* portion of the workshop during that period, ask your instructor for permission to complete the *in-lab* portion after the period. You must be present at the lab in order to get credit for the *in-lab* portion.

If you do not attend the workshop, you can submit the *in-lab* section along with your *at-home* section (see penalties below). The *at-home* portion of the lab is due on the day that is 2 days after your scheduled *in-lab* workshop (23:59) (even if that day is a holiday).

The DIY (Do It Yourself) section of the workshop is a task that utilizes the concepts you have done in the in-lab + at-home section. This section is completely open ended with no detailed instructions other than the required outcome. You must complete the DIY section up to 3 days after the at-home section.

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to back up your work regularly.

Ask your professor if there are any additional requirements for your specific section.

CITATION AND SOURCES

When submitting the DIY part of the workshop, Project and assignment deliverables, a file called sources.txt must be present. This file will be submitted with your work automatically.

You are to write either of the following statements in the file "sources.txt":

I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.

Then add your name and your student number as signature

OR:

Write exactly which part of the code of the workshops or the assignment are given to you as help and who gave it to you or which source you received it from.

You need to mention the workshop name or assignment name and also the file name and the parts in which you received the code for help.

Finally add your name and student number as signature.

By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrong doing.

LATE SUBMISSION PENALTIES:

-*In-lab* portion submitted late, with *at-home* portion:

0 for *in-lab*. Maximum of **DIY+at-home**/10 for the workshop.

-at-home or DIY submitted late:

1 to 2 days, -20%, 3 to 7 days -50% after that submission rejected.

-If any of *the at-home* or DIY portions is missing, the mark for the whole workshop will be **0**/10

WORKSHOP DUE DATES

You can see the exact due dates of all assignments by adding -due after the submission command:

Run the following script from your account (use your professor's Seneca userid to replace `profname.proflastname`, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

```
~profname.proflastname/submit 244/NXX/WS02/in_lab -due<ENTER>
~profname.proflastname/submit 244/NXX/WS02/at_home -due<ENTER>
~profname.proflastname/submit 244/NXX/WS02/DIY -due<ENTER>
```

IN-LAB (30%)

Create an Empty module called “Subject” to keep track of student enrollment in a Subject in a School.

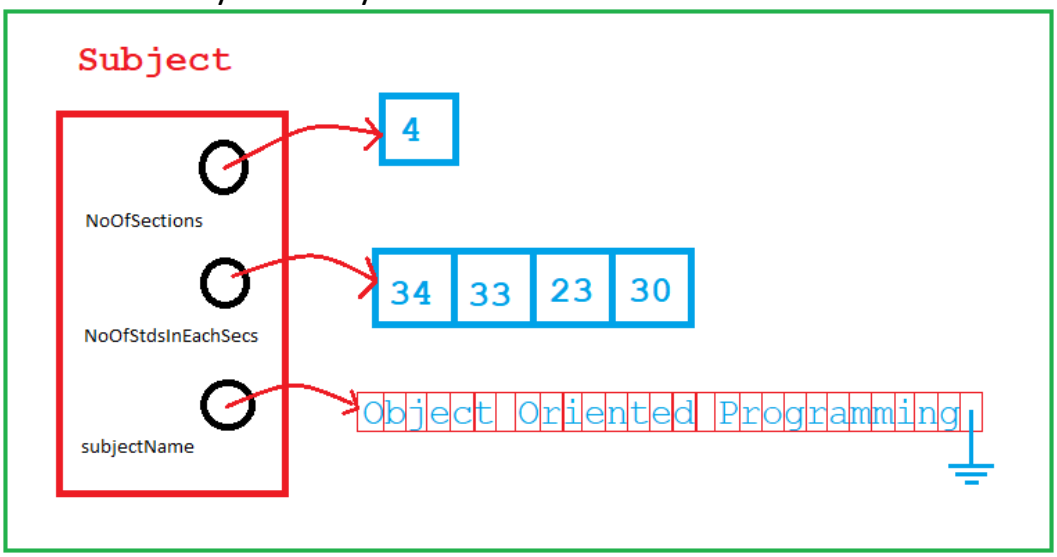
> *Reminder:*
> *This is how to create an empty module called Subject before you start coding:*
> *Create a header file called Subject.h, add the compilation safeguards and the sdds namespace. Then create a source file called Subject.cpp, include Subject.h and add the sdds namespace.*

Design and code a structure (struct) named `Subject` in the namespace `sdds`.

> *To practice dynamic memory allocation, we will have all the member variables as pointers and dynamically allocated memory for them.*

Your structure should have three pointer data members:

- `m_noOfSections`: An integer pointer for a dynamically allocated integer value that will hold the number of sections of a subject in a school.
- `m_noOfStdsInSecs`: An integer pointer for a dynamically allocated array of integers that will store the number of students for each section.
- `m_subjectName`: A character pointer to hold the name of the subject dynamically.



`read(......)` function overloads.

Add three overloads of read(…) function to the Subject module in `sdds` namespace as follows:

1- `void read(char*)` A read function that returns void and receives a char pointer parameter.
This function first prints the following prompt:
`"Enter subject name: "`
and then calls the read function (already coded in utils module) to read up to 70 characters and prints
`"Name is too long, only 70 characters allowed!\nRedo Entry: "`
if there is an error in data entry.

2- `void read(int&)` A read function that returns void and receives a reference to an integer number.
This function first prints the following prompt:
`"Enter number of sections: "`
and then calls the read function (already coded in utils module) to read an integer between 1 and 10 (inclusive) and prints
`"Invalid Number of sections, 1<=ENTRY<=10\nRedo Entry: "`
if there is an error in data entry.

3- `void read(int[], int)` A read function that returns void and receives an integer array to be read from the console. It also receives an integer argument as the number of elements of the array.

This function first prints the following prompt:
`"Enter the number of students in each one of the X sections:"`
where X is replaced by the number of elements in the array.
Then in a loop that runs to the number of elements in an array, read an integer for each element of the array between 5 and 35.
For errors print the following message:
`"Invalid Number of students, 5<=ENTRY<=35\nRedo Entry: "`
Make sure you prompt the user with the row of entry at the beginning of each line. (ie, if the third number is being received, prompt will be `3:`)

`void read(Subject&)`

This function utilizes the last three read function overloads and DMA to read and store data for the entire Subject instance.

Add one more overload of read function to get a Subject from the console. The function will return void and receives a reference to a Subject object (let's call it **Sub**). This function will use the first three read functions and dynamic memory allocation to read and setup a subject structure as follows:
- Create a local array of 71 characters and use the read function (*`void read(char*)`*) to read the name of the subject into it. Then dynamically allocate memory in *`Sub.m_subjectName`*, large enough to hold the name and copy the name into it.
- Dynamically allocate an integer and keep its address in *`Sub.m_noOfSections`* and call the read function (`void read(int&)`)passing the reference of the allocated integer to it, (this will receive the number of subjects)

- Dynamically allocate an array of integers to the size of the integer you just read (number of subjects) and call the read function (`void read(int[], int)`)for reading the number of students.

`read(Subject&)` **EXECUTION EXAMPLE:**

```
Enter subject name: An obviously long text to type so the read function gives me an error
for subject name
Name is too long, only 70 characters allowed!
Redo Entry: Intro to OOP using C++
Enter number of sections: 100
Invalid Number of sections, 1<=ENTRY<=10
Redo Entry: 4
Enter the number of students in each one of the 4 sections:
1: 340
Invalid Number of students, 5<=ENTRY<=35
Redo Entry: 34
2: 33
3: 23
4: 30
```

`int report(const Subject&)`

Create a function called report, that returns an integer and receives a constant Subject reference.

Report subject should print a subject as follows:
At the first line it should print a comma separated list of the enrollment of all the sections.
At second line it should print the name of the subject and then the total enrollment.

Then it should return the total enrollment.

EXECUTION EXAMPLE:

```
34, 33, 23, 30
Intro to OOP using C++: 120
```

`void freeMem(Subject&)`

Finally Create a function called freeMem that returns void, receives a reference of a Subject and deletes all three dynamically allocated memories pointed by member variables of the Subject.

TESTER PROGRAM:

subjectTester.cpp

```
/*************************************************************************
// OOP244 Workshop 2: DMA and overloading
// File subjectTester.cpp
// Version 1.0
// Date       2019/09/15
// Author     Fardad Soleimanloo
// Description
// tests Subject data entry and report
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
// Fardad
/////////////////////////////////////////////////////////////////
*************************************************************************/
#include <iostream>
using namespace std;
#include "Subject.h"
#include "Subject.h"  // intentional

using namespace sdds;
int main() {
   // Create a Subject
   Subject S;
   // Read Subject from console and place the data
   // in Dynamically allocated memory
   read(S);
   // print the data kept in Subject S
   report(S);
   // free the memory allocations pointed by Subject S
```

```
   freeMem(S);
   return 0;
}
```

OUTPUT SAMPLE:

```
Enter subject name: Intro to OOP using C++
Enter number of sections: 12
Invalid Number of sections, 1<=ENTRY<=10
Redo Entry: 4
Enter the number of students in each one of the 4 sections:
1: 340
Invalid Number of students, 5<=ENTRY<=35
Redo Entry: 34
2: 33
3: 23
4: 30
34,33,23,30
Intro to OOP using C++: 120
```

To complete your coding

a) remove all unnecessary comments and debugging statements from your code
b) Include in each file an appropriate header comments uniquely identify the file (as shown above)
c) Preface each function definition in the implementation file with a function header comment explaining what the function does in a single phrase (as shown above).
d) Make sure your function prototypes have meaningful argument names to help understand what the function does.

IN-LAB SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload `utils and Subject modules` and the `subjectTester.cpp` program to your `matrix` account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account during the lab(use your professor's Seneca userid to replace `profname.proflastname`, and your section ID to replace `NXX`, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS02/in_lab<ENTER>

and follow the instructions generated by the command and your program.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

At home and DIY: Under Construction