

Dynamic Memory

Workshop 2 V0.91 (Clarified DMA in *noOfStdInSecs*)

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 from 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>
```

COMPILING AND TESTING YOUR PROGRAM

All your code should be compiled using this command on matrix:

g++ -Wall -std=c++11 -o ws (followed by your .cpp files)

After compiling and testing your code, run your program as follows to check for possible memory leaks: (assuming your executable name is "ws")

valgrind ws <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`.

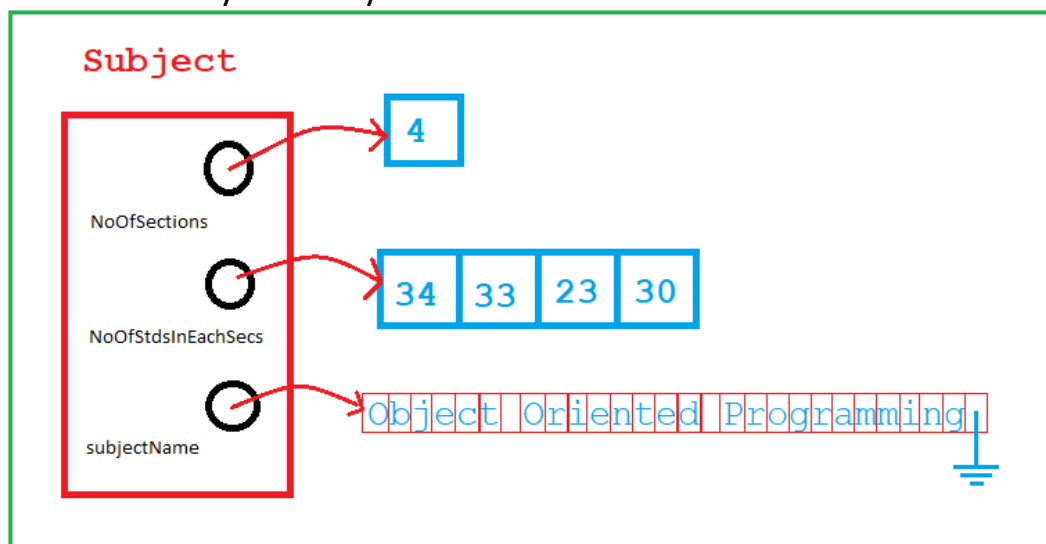
To practice dynamic memory allocation, we will have all the member variables as pointers and dynamically allocated memory for them.

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_noOfStdInSecs`: 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 sections)
- Dynamically allocate an array of integers in `Sub.m_noOfStdsInSecs` to the size of the integer you just read (number of sections) and call the read function (`void read(int[], int)`) for reading the number of students in each sections.

`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

- remove all unnecessary comments and debugging statements from your code
- Include in each file an appropriate header comments uniquely identify the file (as shown above)
- 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).
- 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 (35%)

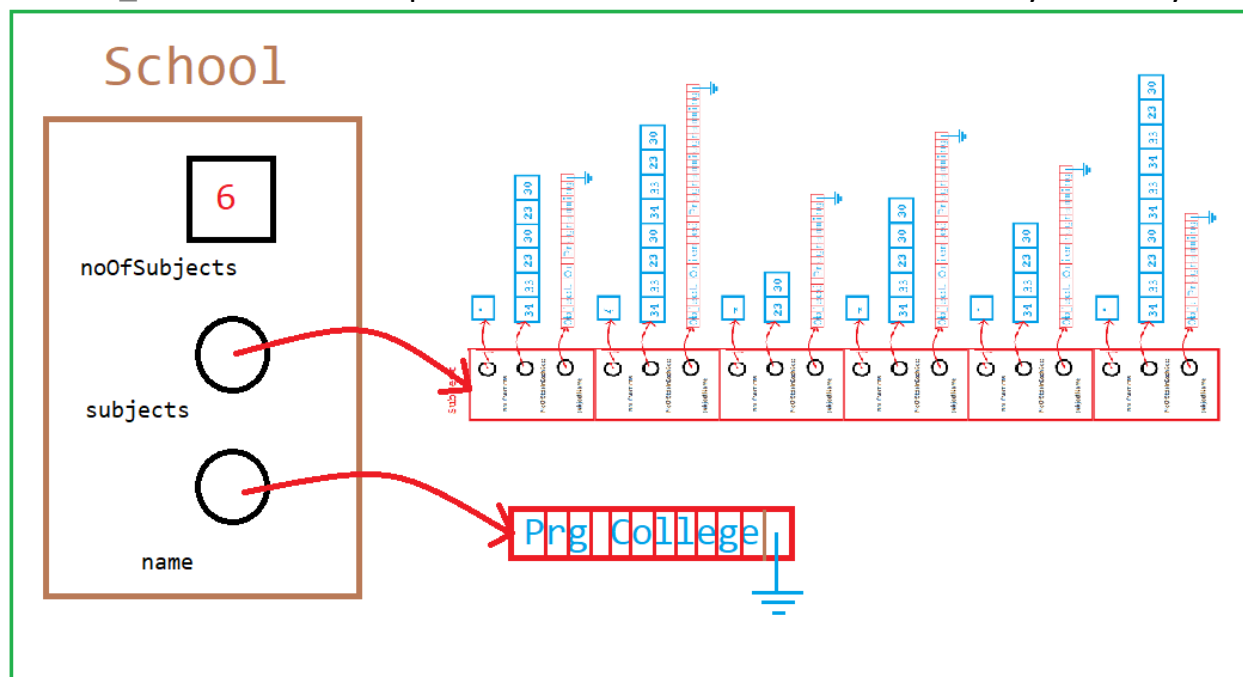
Like what you have done in the in_lab section create an Empty module called “School” to keep track of student enrollment in all the subjects offered in a school a School.

Design and code a structure (struct) named `School` in the namespace `sdds`. Your structure should have one integer and two pointer data members:

`m_noOfSubjects`: An integer variable that will hold the number of subjects offered in a school.

`m_subjects`: A Subject pointer for a dynamically allocated array of Subjects that will store all the subjects offered by the School.

`m_name`: A character pointer to hold the name of the School dynamically.



```
void read(School&)
```

Implement an overload of read function to get all the enrollment information of a Subject from the console. The function will return void and receives a reference to a School object (let's call it **Sch**).

- Create a local array of 61 characters and print the following prompt:

```
"Please enter the name of the school:\n> "
```

use the read function (`void read(char*, int, const char*)`) to read the name of the School.

Use this message in case of error in entry:

```
"Name is too long, only 60 characters allowed!\nRedo Entry: "
```

Then dynamically allocate memory in `Sch.m_name`, large enough to hold the name and copy the name into it.

- print the following message:

```
"Please enter the number of subjects offered by XXXXX: " where XXXXX is replaced by the name of the school.
```

Then call the read function (`void read(int&)`) passing the reference of `Sch.m_noOfSubjects` integer to it, this will receive the number of subjects to allocate next.

Use this message in case of error in entry:

```
"Invalid Number of subjects, 2<=ENTRY<=50\nRedo Entry: "
```

- Dynamically allocate an array of Subjects to the size of the integer you just read (number of subjects), then in a loop that repeats to the number of Subjects, call the read function (`void read(Subject&)`) on each element for reading each Subject.

Make sure you print a sequence number and a separator before each read function call to make data entry clearer:

```
"XX) -----" XX is replace by the sequence (loop) number.
```

read(School&) **EXECUTION EXAMPLE:**

```
Please enter the name of the school:
```

```
> Prg School
```

```
Please enter the number of subjects offered by Prg School: 2
```

```
1) -----
```

```
Enter subject name: 00 Programming 1
```

```
Enter number of sections: 2
```

```

Enter the number of students in each one of the 2 sections:
1: 20
2: 25
2) -----
Enter subject name: 00 Programming 2
Enter number of sections: 3
Enter the number of students in each one of the 3 sections:
1: 30
2: 35
3: 25

```

```
int report(const School&)
```

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

Report subject should print the School enrollment report as follows:

At the first line it should print the name of the School

At the second line it should print the title:

"Subject Enrollments"

Then it should loop through the Subjects, print their report one by one and add their return value to a total enrollment variable.

Then it should print this message:

"Total enrollment: " and print the total enrollment value and return it.

EXECUTION EXAMPLE:

```

Prg School
Subject Enrollments
20,25
00 programming 1: 45
30,35,25
00 Programming 2: 90
Total enrollment: 135

```

```
void freeMem(School&)
```

Finally Create a function called freeMem that returns void, receives a reference of a School and deletes all the dynamically allocated memories pointed by member variables of the School as follows:

-First it should deallocate the name

-Then loop through all the subjects and use `void freeMem(Subject&)` to free the memories used by individual Subject elements.

-Finally delete the dynamically allocated array pointed by the m_subjects member variable.

TESTER PROGRAM:

```

/*****
// OOP244 Workshop 2: DMA and overloading
// File schoolTester.cpp
// Version 1.0
// Date      2019/09/15
// Author    Fardad Soleimanloo
// Description
// tests school data entry and report
//
// Revision History
// -----
// Name      Date      Reason
// Fardad
////////////////////////////////////
*****/

#include "School.h"
using namespace sdds;

int main() {
    School S;
    read(S);
    report(S);
    freeMem(S);
    return 0;
}

```

EXECUTION EXAMPLE:

```

Please enter the name of the school:
> Prg School
Please enter the number of subjects offered by Prg School: 2
1) -----
Enter subject name: 00 Programming 1
Enter number of sections: 2
Enter the number of students in each one of the 2 sections:
1: 20
2: 25
2) -----
Enter subject name: 00 Programming 2
Enter number of sections: 3
Enter the number of students in each one of the 3 sections:
1: 30
2: 35
3: 25
Prg School
Subject Enrollments
20,25

```

```
00 programming 1: 45
30,35,25
00 Programming 2: 90
Total enrollment: 135
```

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.

AT-HOME 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`, `Subject` and `School` modules and the `schoolTester.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/at_home<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.

DIY (Do It Yourself)

For the DIY part of this workshop copy the `utils` module of the `in_lab` section and add two modules called `ParkingSpot` and `ParkingLot`.

These modules are to provide the core logic of a Valet Parking lot. The main program of the parking lot is already provided in the file `parking.cpp`.

`ParkingSpot` module:

Create a structure called `ParkingSpot` with two member variables:

A C-style string to hold an 8 letter license plate of a car.

A character pointer to dynamically hold make and model of a car.

`ParkingSpot` module functions:

`void setEmpty(ParkingSpot&)`

sets the license plate of the `ParkingSpot` to an empty string and the Make and Model of the `ParkingSpot` to a null value (`nullptr`).

`bool isEmpty(const ParkingSpot&)`

returns true if make and model of the `ParkingSpot` is null.

`void freeSpot(ParkingSpot&)`

deletes the memory allocated in the make and model of the `ParkingSpot` and then calls the `setEmpty` function.

`void print(const ParkingSpot&)`

Print the make and model followed by:

", plate number: "
and then the license plate.

```
void parkCar(ParkingSpot&)
```

Prints the message: "Make and Model: "

Reads a string up to 60 characters and the dynamically stores it in the make and model.

If the length is too long it will show the following error message:

"Too long, Make and model must be shorter than 60 characters\nRedo Entry: "

ParkingLot module:

Create a structure called ParkingLot with two member variables:

An integer holding the number of ParkingSpots in the ParkingLot.

A ParkingSpot pointer to dynamically hold an array of ParkingSpots

ParkingLot module functions:

```
void setEmpty(ParkingLot&)
```

sets the number of ParkingSpots to zero and the ParkingSpot pointer to a null value (nullptr).

```
bool isEmpty(const ParkingLot&)
```

returns true the ParkingSpot pointer is null.

```
void openParking(ParkingLot& , int noOfSpots)
```

dynamically allocates "noOfSpots" ParkingSpots and keeps the address in the ParkingSpot pointer of the ParkingLot structure.

the sets all the elements to empty using the setEmpty function;

```
int parkCar(ParkingLot&)
```

Looks for the first empty ParkingSpot in the Parking lot. If the spot is found, it will Park the Car in the spot using parkCar(ParkingSpot&) and returns the index of the found spot, otherwise it returns -1.

```
bool returnCar(ParkingLot&, int spotNo)
```

if the spotNo is a valid number (between 0 and number of the spots) and the Spot is not empty:

1- it will print: "Returning "

2- it will print the Spot using void print(const ParkingSpot&)

3- It will free the spot using void freeSpot(ParkingSpot&)

4- it will return true

Otherwise it will return false

```
void closeParking(ParkingLot&)
```

Goes through all the ParkingSpots and if any of them are is empty, it will return the car. Then it will delete the dynamic array of ParkingSpots and empty the ParkingLot.

Your module should work with this parking application:

```
/******
// OOP244 Workshop 2: SenePark application
// File      parking.cpp
// Version 1.0
// Date      2019/09/15
// Author     Fardad Soleimanloo
// Description
// This is a parking application using the ParkingLot and
// ParkingSpot modules of DIY section of the workshop.
// Revision History
// -----
// Name      Date      Reason
// Fardad
////////////////////////////////////
*****/
#include <iostream>
using namespace std;
#include "ParkingLot.h"
#include "utils.h"
using namespace sdds;
int menu();
int main(void) {
    ParkingLot SenePark;
    int noOfSpots=0;
    int spotNumber=0;
    int selection = -1;
    setEmpty(SenePark);
    cout << "SenePark Valet Parking" << endl << endl;
    while (selection) {
```



```

switch (selection = menu()) {
case 0: // close parking
    if (!isEmpty(SenePark)) {
        closeParking(SenePark);
        cout << "Parking is closed now! Goodbye..." << endl;
    }
    else {
        cout << "Parking is already closed!" << endl;
    }
    break;
case 1: // open parking
    if (isEmpty(SenePark)) {
        cout << "Please enter the number of available spots.\n> ";
        read(noOfSpots, 1, 100, "Invalid Number of spots.\n> ");
        openParking(SenePark, noOfSpots);
    }
    else {
        cout << "The parking is already open!" << endl;
    }
    break;
case 2: // park a car
    if (isEmpty(SenePark)) {
        cout << "Parking is closed!" << endl;
    }
    else {
        spotNumber = parkCar(SenePark);
        if (spotNumber < 0) {
            cout << "Sorry Parking is full!" << endl;
        }
        else {
            cout << "Your car is parked in spot number " << spotNumber + 1 << endl;
        }
    }
    break;
case 3: // return a car
    if (isEmpty(SenePark)) {
        cout << "Parking is closed!" << endl;
    }
    else {
        cout << "What is your spot number?\n> ";
        read(spotNumber, 1, noOfSpots, "Invalid Spot Number!\n> ");
        if (!returnCar(SenePark, spotNumber-1)) {
            cout << "There is no car parked at spot " << spotNumber << endl;
        }
    }
    break;
}
}

int menu() {
    int option;
    cout << "1- Open Parking" << endl
        << "2- Park A Car" << endl
        << "3- Return A Car" << endl
        << "0- Close Parking" << endl
        << "> ";
    read(option, 0, 3, "Invalid Option.\n> ");
}

```

```
    return option;
}
```

SAMPLE EXECUTION:

SenePark Valet Parking

```
1- Open Parking
2- Park A Car
3- Return A Car
0- Close Parking
> 2
Parking is closed!
1- Open Parking
2- Park A Car
3- Return A Car
0- Close Parking
> 1
Please enter the number of available spots.
> 3
1- Open Parking
2- Park A Car
3- Return A Car
0- Close Parking
> 3
What is your spot number?
> 2
There is no car parked at spot 2
1- Open Parking
2- Park A Car
3- Return A Car
0- Close Parking
> 2
Make and Model: BMW 320
License Plate: ABC123
Your car is parked in spot number 1
1- Open Parking
2- Park A Car
3- Return A Car
0- Close Parking
> 2
Make and Model: Tesla Model 3
License Plate: GVXT123
Your car is parked in spot number 2
1- Open Parking
2- Park A Car
3- Return A Car
0- Close Parking
> 2
Make and Model: Honda Civic
License Plate: QWE123
Your car is parked in spot number 3
1- Open Parking
2- Park A Car
3- Return A Car
0- Close Parking
> 2
```

```
Sorry Parking is full!
1- Open Parking
2- Park A Car
3- Return A Car
0- Close Parking
> 3
What is your spot number?
> 2
Returning Tesla Model 3, plate number: GVXT123
1- Open Parking
2- Park A Car
3- Return A Car
0- Close Parking
> 0
Returning BMW 320, plate number: ABC123
Returning Honda Civic, plate number: QWE123
Parking is closed now! Goodbye...
```

DIY 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`, `ParkingLot` and `ParkingSpot` modules and the `parking.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/diy<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.