

# 1 Introduction

Each exercise should be its own separate project.

**Remember to follow the coding and documentation standards for the class listed on the MyClasses pages.**

When you are ready to submit your work create a folder called Homework10 in that folder have separate folders for each project, one folder per project. Put all the code files needed for that project in its respective folder. Do not include the files that the IDE creates, I just want the code files. Zip the entire Homework10 folder up into a single zip file and submit it.

## 2 Exercises

1. Write a program that stores the following data about a soccer player in a structure:

- Player's Name
- Player's Number
- Points Scored by Player

The name of the structure must be `Player`. The program should keep a vector of players, called `team`. Each element is for a different player on the team. The main that you must use for the program is below. There is also an example run below. You will create the following functions.

- `int menuselect()` : This will print out the following menu to the screen and take in (and return) the selection number. Validate the user input to make sure that they type in a number between 1 and 6 inclusively.

```
Options
-----
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection:
```

- `void addPlayer(vector<Player> &)` : This will take the player information from the user, create a `Player` with this information, and add that player to the team vector. Validate user input so that the number of scored points is not negative and make sure the player number is larger than zero and does not match any other player number.
- `void removePlayer(vector<Player> &)` : This will print out a list of players on the team and allows the user to select which player to remove from the

team. The selection is to be by a list number and there should be an option 0 to skip the deletion and not delete anyone. For example,

```
Team List
-----
0: Skip
1: Don Spickler
2: John Doe
3: Sam Spade
Remove Player:
```

- `Player findMVP(const vector<Player> &) :` This should find the MVP of the team, the player with the most points scored, and return a `Player` struct of that person. If there is more than one player with a maximum score you may simply return any of them with the maximum score.
- `int getTotalPoints(vector<Player> &) :` Calculates and returns the total points scored by the team.
- `void printPlayerInfo(Player) :` Prints the player information in the struct. For example,

```
John Doe
Number: 23
Points Scored: 15
```

- `void printTeamInfo(vector<Player> &) :` This will print out the list of team members (does not need to be sorted in any way), the total points scored by the team, and the name of the MVP. For example,

```
Don Spickler
Number: 12
Points Scored: 5
```

```
John Doe
Number: 23
Points Scored: 15
```

```
Sam Spade
Number: 25
Points Scored: 10
```

```
Total Team Points = 30
MVP is John Doe
```

The main must be the following with no alterations.

```
int main() {
    vector<Player> team;
    bool running = true;

    while (running) {
        int selection = menuselect();

        switch (selection) {
            case 1:
                addPlayer(team);
                break;
            case 2:
                removePlayer(team);
```

```
        break;
    case 3:
        if (team.size() > 0)
            printPlayerInfo(findMVP(team));
        else
            cout << "No players on the team." << endl;
        break;
    case 4:
        cout << "Total Team Points = " << getTotalPoints(team) << endl;
        break;
    case 5:
        printTeamInfo(team);
        break;
    case 6:
        running = false;
    }
}

return 0;
}
```

### Example Program Run:

Options

-----

```
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 1
```

```
Input Player Name: Don Spickler
Input Player Number: 12
Input Player's Points: 5
```

Options

-----

```
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 1
```

```
Input Player Name: John Doe
Input Player Number: -5
Invalid input, try again.
Input Player Number: 12
Invalid input, try again.
Input Player Number: 23
Input Player's Points: 15
```

Options

-----

```
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
```

```
Selection: 1

Input Player Name: Sam Spade
Input Player Number: 23
Invalid input, try again.
Input Player Number: 25
Input Player's Points: 10

Options
-----
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 3

John Doe
Number: 23
Points Scored: 15

Options
-----
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 4

Total Team Points = 30

Options
-----
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 5

Don Spickler
Number: 12
Points Scored: 5

John Doe
Number: 23
Points Scored: 15

Sam Spade
Number: 25
Points Scored: 10

Total Team Points = 30
MVP is John Doe

Options
-----
1: Add Player
2: Remove Player
3: Find MVP
```

```
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 2
```

Team List

-----

```
0: Skip
1: Don Spickler
2: John Doe
3: Sam Spade
Remove Player: 0
```

Options

-----

```
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 5
```

Don Spickler

Number: 12

Points Scored: 5

John Doe

Number: 23

Points Scored: 15

Sam Spade

Number: 25

Points Scored: 10

Total Team Points = 30

MVP is John Doe

Options

-----

```
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 2
```

Team List

-----

```
0: Skip
1: Don Spickler
2: John Doe
3: Sam Spade
Remove Player: 1
```

Options

-----

```
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
```

```
Selection: 5

John Doe
Number: 23
Points Scored: 15

Sam Spade
Number: 25
Points Scored: 10

Total Team Points = 25
MVP is John Doe

Options
-----
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 3

John Doe
Number: 23
Points Scored: 15

Options
-----
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 4

Total Team Points = 25

Options
-----
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 5

John Doe
Number: 23
Points Scored: 15

Sam Spade
Number: 25
Points Scored: 10

Total Team Points = 25
MVP is John Doe

Options
-----
1: Add Player
2: Remove Player
```

```
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 2
```

```
Team List
-----
```

```
0: Skip
1: John Doe
2: Sam Spade
Remove Player: 1
```

```
Options
-----
```

```
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 3
```

```
Sam Spade
Number: 25
Points Scored: 10
```

```
Options
-----
```

```
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 4
```

```
Total Team Points = 10
```

```
Options
-----
```

```
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
Selection: 5
```

```
Sam Spade
Number: 25
Points Scored: 10
```

```
Total Team Points = 10
MVP is Sam Spade
```

```
Options
-----
```

```
1: Add Player
2: Remove Player
3: Find MVP
4: Calculate Total Team Points
5: Print Team Information
6: Quit
```

Selection: 6

2. This exercise is to create a structure that acts like an STL vector. This one will simply store integers and have only a fraction of the functionality but it will require you to code several functions that the vector performs with respect to memory management.

Create a struct called `IntList` that will store a size, capacity, and a pointer to an integer (array), as below. The capacity is the amount of space in the array that is pointed to by `list`. The size is how many elements are currently in the list. So if your array pointed to by `list` has 25 cells and you are using just the first 13 of those cells then the size is 13 and the capacity is 25.

```
struct IntList {  
    int size = 0;  
    int capacity = 0;  
    int *list = nullptr;  
};
```

Now create the following functions to be used on the `IntList`. The prototypes are below and descriptions of these functions follow.

```
void set(IntList&, int, int);  
int get(const IntList&, int);  
int get_size(const IntList&);  
int get_capacity(const IntList&);  
void push_back(IntList&, int);  
void push_front(IntList&, int);  
int pop_back(IntList&);  
int pop_front(IntList&);  
void concat(IntList&, const IntList&);  
void increase_capacity(IntList&);  
void destroy(IntList&);  
void sort(IntList&);  
void print(IntList&);
```

- `set`: Will take the list and two integers, the first is the position to place the element and the second is the element to put into the array. If the position is out of bounds the function should display an error that you are out of bounds and do nothing to the array.
- `get`: Will take the list and a position and return the element in the array at that position. If the position is out of bounds the function should display an error that you are out of bounds and return 0.
- `get_size`: Will return the size of the list, that is, the number of elements in the list.
- `get_capacity`: Will return the capacity of the list, that is, the number of cells available for storage.
- `push_back`: This will insert the integer onto the back of the list. If the capacity of the array is not large enough for the entry to be added the `increase_capacity` function should be called to adjust the size of the array.



- `push_front`: This will insert the integer onto the front of the list. If the capacity of the array is not large enough for the entry to be added the `increase_capacity` function should be called to adjust the size of the array.
- `pop_back`: This will remove the integer from the back of the list and return it.
- `pop_front`: This will remove the integer from the front of the list and return it.
- `concat`: This will concatenate the two lists together and store the result in the first list parameter. The new capacity and size are to be the sum of the sizes of the two input lists. That is, they should be the smallest size that will accommodate the two lists.
- `increase_capacity`: This will increase the capacity of the list. If the list is empty, size 0, then the new capacity is to be 1. Otherwise the capacity should be doubled. When the capacity is increased the values in the array and the size should no be altered. So if the current capacity is 8 with a size of 3 storing the numbers 5, 2, and 9 then when the capacity is increased to 16 it should still have a size of 3 storing the numbers 5, 2, and 9.
- `destroy`: This will delete the array storage from memory.
- `sort`: This will sort the list from lowest to highest. Use either the bubble sort, selection sort, or insertion sort to do this. Do not use or even include the algorithm library.
- `print`: This will print the list on a single line with a space or two between the elements.

Once these functions are written the following main program will produce the following output.

```
int main() {
    IntList list1, list2;

    push_back(list1, 15);
    push_back(list1, -4);
    push_back(list1, 3);
    push_back(list1, 7);
    print(list1);
    cout << get_size(list1) << " " << get_capacity(list1) << endl;
    push_back(list1, 17);
    print(list1);
    cout << get_size(list1) << " " << get_capacity(list1) << endl;

    for (int i = 1; i <= 10; i++)
        push_back(list2, i);

    print(list2);
    cout << get_size(list2) << " " << get_capacity(list2) << endl;

    cout << get(list2, 5) << endl;
    cout << get(list2, 15) << endl;

    set(list2, 5, 1234);
    set(list2, 15, -5);
    print(list2);

    push_front(list2, 3);
```

```

    print(list2);
    cout << get_size(list2) << " " << get_capacity(list2) << endl;

    cout << pop_front(list2) << endl;
    cout << pop_front(list2) << endl;
    cout << pop_back(list2) << endl;

    print(list2);
    cout << get_size(list2) << " " << get_capacity(list2) << endl;

    cout << endl;
    print(list1);
    print(list2);

    cout << endl;
    concat(list1, list2);
    print(list1);
    cout << get_size(list1) << " " << get_capacity(list1) << endl;
    print(list2);
    cout << get_size(list2) << " " << get_capacity(list2) << endl;

    cout << endl;
    push_back(list1, 101);
    print(list1);
    cout << get_size(list1) << " " << get_capacity(list1) << endl;

    sort(list1);
    print(list1);
    cout << get_size(list1) << " " << get_capacity(list1) << endl;

    destroy(list1);
    destroy(list2);

    return 0;
}

```

### Output:

```

15 -4 3 7
4 4
15 -4 3 7 17
5 8
1 2 3 4 5 6 7 8 9 10
10 16
6
List bounds error, returning 0.
0
List bounds error.
1 2 3 4 5 1234 7 8 9 10
3 1 2 3 4 5 1234 7 8 9 10
11 16
3
1
10
2 3 4 5 1234 7 8 9
8 16

15 -4 3 7 17
2 3 4 5 1234 7 8 9

15 -4 3 7 17 2 3 4 5 1234 7 8 9
13 13
2 3 4 5 1234 7 8 9

```

```
8 16

15 -4 3 7 17 2 3 4 5 1234 7 8 9 101
14 26
-4 2 3 3 4 5 7 7 8 9 15 17 101 1234
14 26
```

3. This exercise is a simple use of the structure and functions you created in the previous exercise. Create a new project and copy all the code used for the `IntList` structure over to the new project. Now write a main that uses `IntList` and its functions to roll a pair of dice the number of times the user selects. Store the sum of the two die for each individual roll into a `IntList` structure. Then go through the rolls and count all of the 2's, 3's, 4's, ..., 12's. These counts should be stored into another `IntList` structure. Finally print out a list of the counts and a list of the probabilities for each roll. You may only use the `IntList` structure and its functions to store the data, no arrays, no vectors, not any other type of data structure. Also, in the main you may not access any of the fields of the `IntList` structure directly, all access is to be done through the support functions.

### Example Program Run:

```
Input the number of rolls: 1000000
```

```
Counts
```

```
=====
```

```
2: 27552
3: 55946
4: 83232
5: 111158
6: 138927
7: 166512
8: 138379
9: 111200
10: 83569
11: 55633
12: 27892
```

```
Probabilities
```

```
=====
```

```
2: 0.027552
3: 0.055946
4: 0.083232
5: 0.111158
6: 0.138927
7: 0.166512
8: 0.138379
9: 0.1112
10: 0.083569
11: 0.055633
12: 0.027892
```