Assignment 3: Overview

Modify the skeleton code and complete the implementation of `Ordered` and `OrderedSet`

- Do not modify the provided class and member function declaration.
- You can not use any C/C++ library functions and classes, other than malloc, new, free, delete.
- Do not include any other header/source files. All of your implementations should be included in the two files, “hw3.cpp” and “hw3.h”
- All the implementations should be executed without any error if there is no assumption, e.g., your implementation should check all corner cases that may cause segmentation faults.

Where to implement?

- You can implement anything, but do not change the parts wrapped with the “DO-NOT-TOUCH” section. You may want to implement your assignment in the parts specified with the “IMPLEMENT HERE” comment in hw3.h. You can implement anything in hw3.cpp (if needed.)
- You can use additional static functions and forward declaration;
  however, it is NOT ALLOWED to use global/static variables.

Due

~~Submit your implementation before Nov 11, Wednesday, 11:59:59pm, to LMS.~~
Submit your implementation before Nov 15, Sunday, 11:59:59pm, to LMS.
We DO NOT allow late submission.

Submission

Submit the source(.cpp) and header (.h) files that have your implementation.

- Compress two files, “hw3.cpp” and “hw3.h”, in a single zip file, naming “hw3_YOURSTUDENTID.zip”
- DO NOT submit the “main.cpp” file

Class Specification

What do the classes do?

Implement two classes, called "`Ordered`" and "`OrderedSet`"

1. `Ordered` performs simple list functionalities (again!) for the integer variables.
    a. We can add new values to the list as new elements, remove an element, access an element, and check if a value exists in the list
    b. More importantly, when adding a new element, you should position the element in a correct, increasing order.
        i. For example, if you add "10", "30", and "20" in order, your list should maintain the list elements with "10", "20", and "30".
2. `OrderedSet` performs simple set functionalities for the integer variables.
    a. It provides the same functionalities to `Ordered`; however, it does not allow any duplicated element existing in the container.
        i. For example, if you add "30", "30", and "10" in order, your list should maintain the set elements with "10", and "30".
    b. More importantly, you should implement this class by deriving `Ordered`.
    c. Hint: Probably, you may not need to implement many things again for `OrderedSet`. You can selectively implement some required functions and complete it by reusing the implementation existing in `Ordered`. (i.e., You do not need to implement all member functions again although they are virtual.)

Member functions to implement: Ordered

1. Constructor and Destructor

```
// Constructors & destructors
Ordered();
virtual ~Ordered();
```

   a. You should initialize your member variables in constructors.
   b. The destructor should delete all allocated memory.
      i. Note: it's virtual since it should be executed when `OrderedSet` is cast to `Ordered`.

2. Adding/Removing element(s)

```
virtual void add(int v);
virtual void add(int* arr, int size);
virtual void remove(int index);
```

   a. The first member function adds a new element to the list. You may need to implement something rather than simply adding it to the end of the list to keep the increasing order of list elements.
   b. The second function gets an array of integer variables and adds them to the list while maintaining the correct order.
   c. The third function removes an element at the given index (of the ordered list.)

3. Returning the number of elements

```
virtual int size() { return m_size; }
```

   a. Do not need to reimplement this function. Use it as it stands -- which means that you need to manipulate the `m_size` variable in `add` / `remove` functions.

4. Check if a value exists

```
virtual bool operator>>(int v);
```

   a. For example, after adding "30", "20", and "10" to "`Ordered o`", "`o >> 10`" returns true.

5. Return the value at a given index **(Added at 2020/11/02)**

```
virtual int operator[](int index);
```

a. Return the value of the list at the given index
b. If out-of-range, return the minimum integer (refer to the skeleton code to see how to get the minimum integer.)

6. Removing an element from the current instance

```
bool remove(const int idx);
```

a. Remove the element at the given index, i.e., `idx`
b. You also want to delete the memory space that turns to be unused.
c. Return `false` if out-of-index; `true` otherwise.

7. Returning the number of elements in the list

```
unsigned int getSize() const;
```

Member functions to implement: OrderedSet

1. You MUST derive `Ordered` to implement `OrderedSet`.
2. It provides the same interfaces to `Ordered`. All member functions should be supported.
3. Note that, using the class inheritance, you do not need to implement all the member functions again. A few tweaks for some member functions would be enough to complete this class. So, think thoroughly before diving into the coding!

Some extra comments

- You do not need to implement class copy operators. The skeleton already deletes the default copy operators to prevent mistakes:

```
Ordered(const Ordered&) = delete;
Ordered& operator=(const Ordered&) = delete;
```

Sample

Notes

- We will check with other example inputs for grading.

Sample Code 1

```
#include "hw3.h"

#include <iostream>

void testcase(Ordered* o) {
	o->add(30);
	o->add(20);
	o->add(20);
	o->add(10);
	o->add(15);
	o->add(15);
	o->add(25);
	int vals[] = { 10, 20, 35, 35, 10 };
	o->add(vals, sizeof(vals) / sizeof(int));

	o->remove(2);

	for (int i = 0; i < o->size(); ++i) {
		std::cout << (*o)[i] << ", ";
	}
	std::cout << std::endl;

	std::cout << std::boolalpha << (*o >> 15) << std::endl;
	std::cout << std::boolalpha << (*o >> 40) << std::endl;
}

int main()
{
	Ordered orderedList;
	testcase(&orderedList);

	OrderedSet orderedSet;
	testcase(&orderedSet);

	return 0;
}
```

Outputs of Sample code 1

```
10, 10, 15, 15, 20, 20, 20, 25, 30, 35, 35,
true
false
10, 15, 25, 30, 35,
true
false
```

Sample Code 2 (10.31 추가)

- 참고: OrderSet이 Order에 존재하는 member 함수를 전부 지원하는지 테스트하는 코드입니다.
- Sample Code 1과 매우 유사하게 생겼지만, 약간의 차이가 있습니다.
- 그 차이에 관련한 내용에 대해서는 11월 2일 강의 때 설명드리도록 하겠습니다.

```
#include "hw3.h"

#include <iostream>

void testcase(OrderedSet * o) {
	o->add(30);
	o->add(20);
	o->add(20);
	o->add(10);
	o->add(15);
	o->add(15);
	o->add(25);
	int vals[] = { 10, 20, 35, 35, 10 };
	o->add(vals, sizeof(vals) / sizeof(int));

	o->remove(2);

	for (int i = 0; i < o->size(); ++i) {
		std::cout << (*o)[i] << ", ";
	}
	std::cout << std::endl;

	std::cout << std::boolalpha << (*o >> 15) << std::endl;
	std::cout << std::boolalpha << (*o >> 40) << std::endl;
}

int main()
{
	OrderedSet orderedSet;
	testcase(&orderedSet);

	return 0;
}
```

Outputs of Sample code 2

```
10, 15, 25, 30, 35,
true
false
```