Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Spring 2020

Homework 5 – Operator overloading for integer set class

Due: 15/04/2020, Wednesday, 21:00

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You can NOT collaborate with your friends and discuss solutions. You have to write down the code on your own. Plagiarism will not be tolerated!

Introduction

In this homework, you are asked to implement a class for *integer sets* and overload some operators. The details of these operators will be given below. Our focus in this homework is on the class design and implementation. The usage of this class in the main function will be given to you.

A *set* is mathematically defined as a collection of distinct elements. So, in an *integer set,* each element must be a distinct integer. Here by distinctness, we mean that a particular element can exist at most once in a set. Thus, when you want to add an existing element to a set, its content does not change.

Integer Set Class Design

There should be two private data members in your class: (i) you have to keep *number of elements* (a.k.a. *size*) that exist in the set as a private data member of the class, and (ii) you have to keep the elements of the set in a **dynamic array of integers**, whose size is always equal to *number of elements*.

Your class implementation must include the following basic class functions:

- Default constructor: creates an *integer set* object with *number of elements* = 0. This constructor does not allocate memory for the elements of the set since there are not elements.
- Constructor with *number of elements* parameter: creates an *integer set* object and allocates enough memory (according to given parameter) for the dynamic array of integer. Then fill in the set such that it contains all the integer between and including 0 and *number of elements* -1. For instance, if the *number of elements* parameter is 6, then the constructed set contains {0, 1, 2, 3, 4, 5}.
- Deep copy constructor: takes an *integer set* object as const-reference parameter and creates a new *integer set* object as the deep copy of the parameter.
- Destructor: By definition, destructor deletes all dynamically allocated memory and returns them back to the heap.

In this homework, you will overload several operators for the *integer set* class. These operators and the details of overloading are given below. You can overload operators as member or free functions. However, in order to test your competence in both mechanisms, we request you to have at least three member and three free functions for these operators (the remaining three is up to you).

<< This operator will be overloaded such that it will put an *integer set* on an output stream (`ostream`) in mathematical format. See the sample runs for the required output format. By definition, the order of elements to be displayed is not important; so, the order of elements in your program may differ than the ones in the sample runs. Since the left hand side is an output stream, this function must be a free function.

Note that + operator will be overloaded in two different ways for set union and element addition to sets. See the following three paragraphs below for the details.

+ This operator will be overloaded such that it will add a single integer element to an *integer set* object. The integer element is the **right hand side** operand of the + operator, and the *integer set* is on the **left hand side**.

+ This operator will be overloaded such that it calculates and returns the union of two *integer set* operands. By definition, the union of two sets includes the elements that exist in at least one of the operands. If a particular element exists in both operands, of course, you will include it only once in the union set.

Here remark that we do not have any assumption about the size of a set. While implementing the + operators, the resulting set that you are going to return might have a size greater than the size of the operand(s). In such a case, you will need to allocate new memory area with a greater size. In such cases, (i) do not allocate memory more than needed; i.e. the amount of integers in the memory allocated for the returning set must be the size of it, and (ii) please be careful and do not make memory leak.

* This operator will be overloaded such that it will take the set intersection of two *integer set* operands and return the resulting *integer set*. The intersection set of two *integer set* operands includes the elements, which exist both in the left hand side set and in the right hand side set (i.e. common elements of both operands). You have to be careful here about the size of the returning set as well; the resulting set's size and memory allocation for it must be aligned with each other.

= This operator will be overloaded such that it will assign the *integer set* object on the right hand-side of the operator to the *integer set* object on the left hand-side. This function must be implemented in a way to allow cascaded assignments. Due to C++ language rules, this operator must be a member function (cannot be free function).

+= This operator will be overloaded such that it will unionize two *integer set* objects and assign the resulting *integer set* object to the *integer set* object on the left hand-side of the += operator. This function must be implemented in a way to allow cascaded assignments.

!= This operator will be overloaded such that it will check two *integer set* objects for inequality. This operator returns true when they are not equal; returns false otherwise. Two sets are said to be equal if they contain exactly the same elements (order of the elements does not matter).

Note that <= operator will be overloaded in two different ways for subset control and membership check. See the following two paragraphs below for details.

<= This operator will be overloaded such that it will check whether the *integer set* object on the left hand side is subset of the *integer set* object on the right hand side. If so, the operator returns true; otherwise returns false. The set A is the subset of set B, if every element of A is also an element of B.

<= This operator will be overloaded such that it will check whether a single integer element on the left hand side is a member of the *integer set* object on the right hand side. If so, the operator returns true; otherwise returns false. Since the left hand side is an integer, this function must be a free function.

In order the see the usage and the effects of these operators, please see **sample runs** and the provided **main.cpp**.

In this homework, you are allowed to implement some other helper member functions, if needed.

In this homework, you are allowed to implement some other helper member functions, accessors (getters) and mutators (setters), if needed. However, you are **not allowed to use friend functions and friend classes**.

A life-saving advice: Any member function that does not change the private data members needs to be **`const` member function** so that it works fine with const-reference parameters. If you do not remember what "`const` member function" is, please refer to CS201 notes or simply Google it!

You have to have separate header and implementation files, in addition to the main.cpp. Make sure that the header file name that you submit and #include are the same; otherwise your program does not compile.

We have seen a set class in lectures (LinkStringSet). Since this class uses linked list as container, and you will not use linked lists in this homework, it is not useful for you. Please do not make use of this class and any other standard or non-standard set classes in your homework; you will implement your own class from scratch.

main.cpp

In this homework, **main.cpp** file is given to you within this homework package and we will test your codes with this main function with different inputs. You are not allowed to make any modifications in the main function (of course, you can edit to add `#includes`, etc. to the beginning of the file). Inputs are explained with appropriate prompts so that you do not get confused, also you can assume that there won't be any wrong inputs in test cases; in other words, you do not need to do input checks. We strongly recommend you to examine main.cpp file and sample runs before starting your homework.

Some Important Rules

In order to get a full credit, your programs must be efficient and well presented, presence of any redundant computation or bad indentation, or missing, irrelevant comments are going to decrease your grades. You also have to use understandable identifier names, informative introduction and prompts. Modularity is also important; you have to use functions wherever needed and appropriate. Since using classes is mandated in this homework, a proper object-oriented design and implementation, and following the rules mentioned in this homework specification will also be considered in grading.

Since you will use dynamic memory allocation in this homework, it is very crucial to properly manage the allocated area and return the deleted parts to the heap whenever appropriate. Inefficient use of memory may reduce your grade.

When we grade your homework we pay attention to these issues. Moreover, in order to observe the real performance of your codes, we may run your programs in *Release* mode and **we may test your programs with very large test cases**. Of course, your program should work in *Debug* mode as well.

Please do not use any non-ASCII characters (Turkish or other) in your code.

You are allowed to use sample codes shared with the class by the instructor and TAs. However, you cannot start with an existing .cpp or .h file directly and update it; you have start with an empty file. Only the necessary parts of the shared code files can be used and these parts must be clearly marked in your homework by putting comments like the following. Even if you take a piece of code and update it slightly, you have to put a similar marking (by adding "`and updated`"to the comments below.

```
/* Begin: code taken from ptrfunc.cpp */

...

/* End: code taken from ptrfunc.cpp */
```

Sample Runs

Some sample runs are given below, but these are not comprehensive, therefore you have to consider **all possible cases** to get full mark. Especially try different set contents (other than the ones given in the sample runs) and extreme cases. In order to finish the input entry, we used `^Z`. This character can by typed as Ctrl-Z using keyboard.

Sample Run 1:

```
Please enter elements of set 1. Press CTRL + Z after all elements are entered
-99
-5
-3
-1
-3
^Z
intSet1
{-99, -5, -3, -1}

Please enter elements of set 2. Press CTRL + Z after all elements are entered
-99
-99
-88
-5
-4
-3
-1
^Z
intSet2
{-99, -88, -5, -4, -3, -1}

intSet3 after IntegerSet intSet3(intSet1)
{-99, -5, -3, -1}
```

```
intSet3{-99, -5, -3, -1} is equal to intSet1 {-99, -5, -3, -1}

intSet1 {-99, -5, -3, -1} is not equal to intSet2 {-99, -88, -5, -4, -3, -1}

intSet3 after intSet3 = Union(intSet1, intSet2)
{-99, -88, -5, -4, -3, -1}

intSet4 after intSet4 = intSet1 + intSet2
{-99, -88, -5, -4, -3, -1}

intSet3 {-99, -88, -5, -4, -3, -1} is equal to intSet4 {-99, -88, -5, -4, -3, -1}

intSet1 after intSet1 = intSet1 + 22
{-99, -5, -3, -1, 22}

intSet2 after intSet2 = intSet2 + -22
{-99, -88, -5, -4, -3, -1, -22}

intSet4 after intSet4 = intSet3 * intSet2
{-99, -88, -5, -4, -3, -1}

intSet4 after intSet4 = intSet3 * intSet1
{-99, -5, -3, -1}

intSet3 after intSet3 = intSet1 + intSet2 + intSet4
{-99, -5, -3, -1, 22, -88, -4, -22}

intSet4 after intSet4 += intSet1
{-99, -5, -3, -1, 22}

3 is not element of intSet1 {-99, -5, -3, -1, 22}

intSet4 {-99, -5, -3, -1, 22} is a subset of intSet3 {-99, -5, -3, -1, 22, -88, -4, -
22}

intSet2 after intSet1 = intSet2 += intSet3
{-99, -5, -3, -1, 22, -88, -4, -22}

intSet1 after intSet1 = intSet2 += intSet3
{-99, -5, -3, -1, 22, -88, -4, -22}

intSet5 after IntegerSet intSet5(11)
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

After intSet1 = intSet2 = intSet3 += intSet4 += intSet5

intSet1: {-99, -5, -3, -1, 22, -88, -4, -22, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intSet2: {-99, -5, -3, -1, 22, -88, -4, -22, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intSet3: {-99, -5, -3, -1, 22, -88, -4, -22, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intSet4: {-99, -5, -3, -1, 22, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intSet5: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Sample Run 2:

```
Please enter elements of set 1. Press CTRL + Z after all elements are entered
-7
-12
45
102
^Z
intSet1
{-7, -12, 45, 102}

Please enter elements of set 2. Press CTRL + Z after all elements are entered
-7
```

```
-12
45
102
^Z
intSet2
{-7, -12, 45, 102}

intSet3 after IntegerSet intSet3(intSet1)
{-7, -12, 45, 102}

intSet3{-7, -12, 45, 102} is equal to intSet1 {-7, -12, 45, 102}

intSet1 {-7, -12, 45, 102} is equal to intSet2 {-7, -12, 45, 102}

intSet3 after intSet3 = Union(intSet1, intSet2)
{-7, -12, 45, 102}

intSet4 after intSet4 = intSet1 + intSet2
{-7, -12, 45, 102}

intSet3 {-7, -12, 45, 102} is equal to intSet4 {-7, -12, 45, 102}

intSet1 after intSet1 = intSet1 + 22
{-7, -12, 45, 102, 22}

intSet2 after intSet2 = intSet2 + -22
{-7, -12, 45, 102, -22}

intSet4 after intSet4 = intSet3 * intSet2
{-7, -12, 45, 102}

intSet4 after intSet4 = intSet3 * intSet1
{-7, -12, 45, 102}

intSet3 after intSet3 = intSet1 + intSet2 + intSet4
{-7, -12, 45, 102, 22, -22}

intSet4 after intSet4 += intSet1
{-7, -12, 45, 102, 22}

3 is not element of intSet1 {-7, -12, 45, 102, 22}

intSet4 {-7, -12, 45, 102, 22} is a subset of intSet3 {-7, -12, 45, 102, 22, -22}

intSet2 after intSet1 = intSet2 += intSet3
{-7, -12, 45, 102, 22, -22}

intSet1 after intSet1 = intSet2 += intSet3
{-7, -12, 45, 102, 22, -22}

intSet5 after IntegerSet intSet5(11)
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

After intSet1 = intSet2 = intSet3 += intSet4 += intSet5

intSet1: {-7, -12, 45, 102, 22, -22, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intSet2: {-7, -12, 45, 102, 22, -22, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intSet3: {-7, -12, 45, 102, 22, -22, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intSet4: {-7, -12, 45, 102, 22, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intSet5: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Sample Run 3:

```
Please enter elements of set 1. Press CTRL + Z after all elements are entered
^Z
```

```
intSet1
{}

Please enter elements of set 2. Press CTRL + Z after all elements are entered
1
-1
^Z
intSet2
{1, -1}

intSet3 after IntegerSet intSet3(intSet1)
{}

intSet3{} is equal to intSet1 {}

intSet1 {} is not equal to intSet2 {1, -1}

intSet3 after intSet3 = Union(intSet1, intSet2)
{1, -1}

intSet4 after intSet4 = intSet1 + intSet2
{1, -1}

intSet3 {1, -1} is equal to intSet4 {1, -1}

intSet1 after intSet1 = intSet1 + 22
{22}

intSet2 after intSet2 = intSet2 + -22
{1, -1, -22}

intSet4 after intSet4 = intSet3 * intSet2
{1, -1}

intSet4 after intSet4 = intSet3 * intSet1
{}

intSet3 after intSet3 = intSet1 + intSet2 + intSet4
{22, 1, -1, -22}

intSet4 after intSet4 += intSet1
{22}

3 is not element of intSet1 {22}

intSet4 {22} is a subset of intSet3 {22, 1, -1, -22}

intSet2 after intSet1 = intSet2 += intSet3
{22, 1, -1, -22}

intSet1 after intSet1 = intSet2 += intSet3
{22, 1, -1, -22}

intSet5 after IntegerSet intSet5(11)
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

After intSet1 = intSet2 = intSet3 += intSet4 += intSet5

intSet1: {22, 1, -1, -22, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intSet2: {22, 1, -1, -22, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intSet3: {22, 1, -1, -22, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intSet4: {22, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intSet5: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

What and where to submit (PLEASE READ, IMPORTANT)

You should prepare (or at least test) your program using MS Visual Studio 2012 C++. We will use the standard C++ compiler and libraries of the abovementioned platform while testing your homework. It'd be a good idea to write your name and last name in the program (as a comment line of course).

Submissions guidelines are below. Some parts of the grading process might be automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.

Name your cpp file that contains your main program using the following convention:

"SUCourseUserName_YourLastname_YourName_HWnumber.cpp"

Your SUCourse user name is your SUNet user name which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:

Cago_Ozbugsizkodyazaroglu_Caglayan_hw5.cpp

In some homework assignments, you may need to have more than one .cpp or .h files to submit. In this case, add informative phrases after the hw number (e.g. Cago_Ozbugsizkodyazaroglu_Caglayan_hw5_myClass.cpp or Cago_Ozbugsizkodyazaroglu_Caglayan_hw5_myClass.h). However, do not add any other character or phrase to the file names. Sometimes, you may want to use some user defined libraries (such as strutils of Tapestry); in such cases, you have to provide the necessary .cpp and .h files of them as well. If you use standard C++ libraries, you do not need to provide extra files for them.

Make sure that you #include correct header file names. If you rename your header file names just before submission and does not update `#include` statements accordingly, the file name you use in your program would not match with the real file name and your program does not compile. This causes getting zero in your homework.

These source files are the ones that you are going to submit as your homework. However, even if you have a single file to submit, you have to compress it using ZIP format. To do so, first create a folder that follows the abovementioned naming convention ("SUCourseUserName_YourLastname_YourName_HWnumber"). Then, copy your source file(s) there. And finally compress this folder using WINZIP or WINRAR programs (or another mechanism). Please use "zip" compression. "rar" or another compression mechanism is NOT allowed. Our homework processing system works only with zip files. Therefore, make sure that the resulting compressed file has a zip extension. Check that your compressed file opens up correctly and it contains all of the files that belong to the latest version of your homework.

You will receive zero if your compressed zip file does not expand or it does not contain the correct files. The naming convention of the zip file is the same. The name of the zip file should be as follows:

SUCourseUserName_YourLastname_YourName_HWnumber.zip

For example, zubzipler_Zipleroglu_Zubeyir_hw5.zip is a valid name, but

Hw5_hoz_HasanOz.zip, HasanOzHoz.zip

are **NOT** valid names.

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!

Albert Levi, Vedat Peran