CS246 Winter 2013 Assignment 5 V1.0

You are to re-implement the SetInt class (from Assignment 4). The new implementation will replace vectors with linked lists. You will also be implementing the following new functions: a copy constructor, a destructor and an overload assignment operator "=". Your program should, at least in principle, behave just like your program from Assignment 4. However, your new implementation of the SetInt class may be somewhat more efficient.

Goals of assignment. This assignment illustrates and gives examples of a number of concepts. It illustrates abstraction – the SetInt class represents a high level concept (sets of integers) which can be implemented many different ways. It illustrates the use of copy constructors and destructors as well as overloading the assignment operator "=". It illustrates how a driver (driver.cpp) can be largely independent from the implementation of the programs it utilizes. It gives practice in using C and C++ pointers with dynamic allocation and how they can replace higher level mechanisms such as C++ vectors.

Replace your earlier implementation of your SetInt class. You will develop a new implementation of the same data abstraction (sets of integers) but with vectors replaced by linked lists. You are to write your own linked lists (do not use the C++ STL container). You are to use a minimally modified version of your existing header file setint.h to include the prototype for the copy constructor and the destructor as well as overloaded assignment =, now naming the file newsetint.h. You will also need to modify it to replace the old data structure (vector of integers) with the new data structure (the linked list as described below). You will program a new implementation file newsetint.cpp which will replace the old implementation setint.cpp. The new executable file for the SetInt driver should be called *newdriver*. The old driver was called *driver*.

Ideally, you will only change two files: setint.h and setint.cpp (renaming these to newsetint.h and newsetint.cpp) as well as changing any files that contain #include "setint.h" to #include "newsetint.h". Each file other than newsetint.h and newsetint.cpp that you modify should include documentation explaining how you changed the file. This should be done as comments in the file, beginning as follows:

```
// How this file was modified.
  … more comment lines here …
```

Use the same *main* function (perhaps minimally changed) and the same *help* function (in help.h and help.cpp) from the last assignment.

Use linked lists In your new implementation of the SetInt class, you are to use linked lists instead of C++ "vectors". At runtime, generally when an integer is added to a set, a node is allocated to store the integer and the node is linked to the list of nodes, which together represent the set. Each node in the linked list will consist of (1) an integer value and (2) a link (a pointer) to the next node. For each set, you are to use a singly linked list. The list should be ordered, with small values coming earlier in the list. The implementation will need to use **new** and **delete** statements to create and destroy space

for nodes. You are to use an *anchor* node in each set object; the anchor is a dummy node whose next pointer locates the first integer in the list (if any --- if the set is empty, that pointer is NULL). This anchor node makes programming simpler when dealing with the first node in the list.

Your implementation should make sure that no space is lost (no memory leakage) and that all pointers are followed only when they actually locate legitimate data (no access using dangling pointers).

Ordering and performance. Your linked lists should be ordered according to integer values, e.g., 3 would come before 7. Your operations such as + and - should be efficient in the following sense. For an integer set P1 with L1 members and an integer set P2 with L2 members, an operation such as set subtraction should use at most (or about) L1 + L2 machine level (double) operations. Put technically, these operations should run in time O(L1 + L2), i.e., order of L1 + L2.

Copy constructor and destructor. For your new implementation of the SetInt class to work correctly, you need to implement a "copy constructor" as well as a destructor and overloaded assignment = operator. Note that the copy constructor is implicitly invoked for value parameter passing of objects. Hint: The Savitch text book gives a detailed example using copy constructors. It also gives examples of overloading operators.

Keep it short. As a rule, in this assignment, shorter, more elegant solutions to this assignment are preferable to more complex solutions. But you are still expected to meet the performance requirements given above.

Coverage testing. You are to write a *testing harness* that tests one operator (the union + operator). Do this as follows. Write a file called testunion.cpp which can be used in place of newdriver.cpp. The testunion.cpp file will #include newsetint.h (but not help.h). With newdriver.cpp replaced by testunion.cpp, create an executable file called testunion. When testunion runs, it should execute all code (except error handling code) in your routine that implements set union. Your testunion.cpp file should have the following form:

```
// File testunion.cpp -- Exercise union routine in newsetint.cpp
#include "newsetint.h"
#include <iostream>
using namespace std;
int main (){
    // Lots of comments explaining what is being tested
    SetInt s, t, u;  // An example – declaring sets
    s = t + u;  // Include output stating what is being tested.
    s.output();
    // … more here …
}
```

Provided files. Files provided for this assignment can be found on the student Linux system in the directory:

```
~holt/cp-for-students/cs/246/2013/asgn05
```

Makefile. You should use the provided makefile to produce *newdriver*.

Master solution. You will be given a compiled (master) solution to this assignment. You can run this under your student account to see if your solution gives the same answers. Your output (except when there are user errors) from your program should be the same, character for character, as the master solution.

Files to submit for marking.
Files for a5p1:

- newsetint.h

Files for a5p2:

- newsetint.h
- newsetint.cpp
- newdriver.cpp

Files for a5p3:

- testunion.cpp

Note: a5p3 will be hand marked. Ideally the testing harness should test all cases without being repetitive, as grading will be based on the thoroughness, breadth and depth of your testing NOT on the quantity of tests you provide.

MARKS
a5p1 – 10 marks
a5p2 – 70 marks
a5p3 – 40 marks
Style – 20 marks
TOTAL = 140 marks