Course: CpSc 212: Algorithms and Data Structures Spring 2014
Lab 05: Integer Set Data Structure Using Binary Search Trees

1 The Integer Set Data Structure

In this lab, you will be building an integer set data structure using binary search trees. Remember that a *set* is a collection of distinct elements. In lab 03, we represented a set using a sorted linked list. In this lab, a set will be represented using a binary search tree. Remember the binary search tree property: An element with key k is at the root. All elements lesser and greater than k are in the left and right subtrees of the root respectively. Every node in the binary search tree satisfies this property. Since a set contains only distinct elements, the binary search tree will also contain only distinct elements.

1.1 Downloading and Extraction

Please first download the *lab05.tar.gz* file from the following webpage:

`http://people.cs.clemson.edu/~rmohan/course/cpsc2120-sp14/`

Next, please extract the files using the following command.

```
tar -zxvf lab05.tar.gz
```

This command will extract all the files in the directory named *lab05*. Please change your directory using the command:

```
cd lab05
```

1.2 IntSet specifications

Please open *intset.h* to see the specifications of the class. The class *Node* represents a node in the binary search tree. It contains the integer data *key* and pointers to the left and right children which are *left* and *right* respectively. A parent pointer *parent* points to the parent node in the binary search tree. Though the parent pointer is not significant in this lab, they will be useful when balancing the binary search tree during lab06. So please make sure the parent pointers are correctly assigned.

Note the public interface to the *IntSet* class. The pointer *root* points to the root element of the binary search tree. Open *intset.cpp* and implement the following functions.

1. ***find(key):*** This function returns a pointer to the node in the binary search tree with the required key. Returns *NULL* if the key is not found.

2. ***insert(key):*** This function inserts an integer key into the set. It is an error to call this function with a value of a key that is already present in the set.

3. ***print():*** This prints the contents of the set in increasing order.

The constructors for classes *Node* and *IntSet* are already provided for you. Please note the *assert* statement at the begining of funciton *insert*. The *assert* statement allows execution of the next line of code only if its corresponding condition is *true*.

The *insert*, *find* and *print* functions in the public section of class *IntSet* are wrapper functions. Wrapper functions call other functions and make user code simple and clean.

1.3 Testing your IntSet class

You are to write your own code to test the IntSet class. No additional code is provided for this task. If you named your test file *testIntSet.cpp*, then you can compile using the following command.

g++ testIntSet.cpp -o testIntSet intset.cpp

You can enter your input in a file named *testIntSetInput* and redirect your input while executing.

./testIntSet < testIntSetInput

You have to thoroughly test the IntSet class. Make sure your program works for all boundary cases. You will receive full credit only if your code works for all cases.

2 Ordering of Network Packets

In the field of computer networks, a sender process in the server sends data to a client receiver process. It normally waits for a certain period of time for the receiver to send back an acknowledgement (ACK). If the sender times out waiting for the ACK, it resends its packets to the receiver. Thus, the receiver may receive many duplicate packets which it discards.

Packets may also arrive out of order at the receiver. The receiver needs to order the packets to the desired application process. Ideally, each packet comes with a packet starting address followed by the length of the packet. The receiver re-orders its packets based on this information. In this exercise, we will only be re-ordering based on packet IDs.

Given a sequence of pakcet IDs, please show how duplicate elements can be discarded and all the packets re-ordered. Please write a stand alone program which takes as input, a sequence of packet IDs and re-orders distinct packets. Your program should make use of the integer set data structure you implemented in the previous section.

2.1 Input

The input to your program is a sequence of packet IDs represented as itnegers. Please accept the packet IDs from standard input. You can enter the sequence in a file and redirect the input.

2.2 Output

Output the packet IDs in order. All duplicate packet IDs should be discarded. A sample I/O is given below.

Sample I/O

Test case#	Input	Output
1.	2 3 10 5 6 2 1 1 3 9 2 3 10	1 2 3 5 6 9 10
2.	5 6 7 8 4 2 3 1 5 6 4 1 10 9 4 6 7 8	1 2 3 4 5 6 7 8 9 10

2.3 Compiling and Testing

You are to write a stand alone program that takes as input a sequence of packet ids and re-orders them. Name your program as *main.cpp* and compile it using the command:

g++ main.cpp -o main intset.cpp

You can put your input in a file named *in* and then execute your program using the command:

./main < in

Please thoroughly test your program and make sure it works for all cases.

3 Submission

Please submit your solutions through handin.cs.clemson.edu. The due date for this assignment is Friday February 21st at 11:59pm.

4 Grading

Each section in this assignment is graded out of ten points. No points will be awarded to a program that does not compile. You will recieve 1 point for successful compilation and 1 point for successful execution (no seg faults and infinite loops). 3 points is awarded for correctness and efficiency each and 2 points awarded for good code: readability, comments, simplicity and elegance, etc.