Assignment 5

Due: December 4th, 2015 before 11:59pm

Objectives

- Introduction to the Binary Search Tree ADT
- More practice with templates
- Introduction to tree traversals
- Introduction to the Dictionary ADT.
- Practice problem solving using ADTs.

Introduction

In this assignment you will implement a binary search tree that is slightly different from the simple example we've been discussing in lecture. In this assignment, each node in the binary search tree will contain both a key and a value associated with that key. The elements in the tree will be ordered by the key stored at each node.

After you've completed implementing the binary search tree, we will use that implementation to implement the Dictionary ADT.

Finally, after completing the Dictionary implementation, you will use the Dictionary to implement a contact list.

Quick Start:

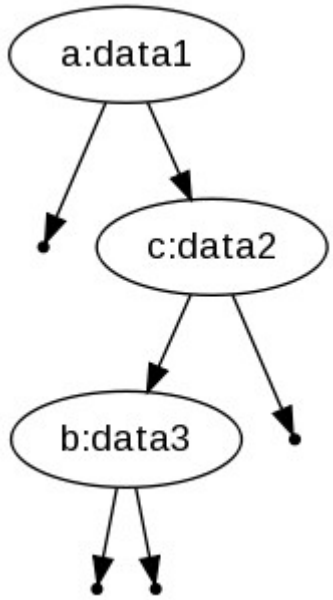
1. Read this entire document
2. Read bst`.h` carefully and implement functions in `bst.h` until all tests passed in `bst_tester.cpp`
3. Read `dictionary.h` carefully and implement functions in `dictionary.h` until all tests passed in `dictionary_tester.cpp`
4. Read `contactlist.h` carefully and implement functions in `contactlist.h` until all tests passed in `contactlist_tester.cpp`

Binary Search Trees

Your binary search tree implementation will use templates to allow users of the tree to specify the type for both the key and the value.

For example, the following code produces the tree shown below:

```
BinarySearchTree<string, string>  t1;
t1.insert("a", "data1");
t1.insert("c", "data2");
t1.insert("b", "data3");
```



Since it is helpful to be able to visualize trees, you've been provided with the `tree_view` class, which takes a tree as a parameter. If you use the following code:

```
tree_view<string,string> v1(t1);
v1.dot_print(cout);
```

It will produce a `dot` file[1] to standard output. You can copy and paste that code into the following web page to get a picture of your tree like that shown above:

http://sandbox.kidstrythisathome.com/erdos/

1 The .dot files are used as input into a program called `dot` which is part of the graphviz project. `dot` performs automatic layout of directed graphs. Trees are restricted versions of directed graphs, so we can use the tool to draw our trees.

Pairs

It is often helpful to be able to get a list of all the key/value pairs stored in the Binary Search Tree. In your assignment, you have to implement this method:

```
list<pair<K,V> > key_value_pairs()
```

Which is to return a list of all the key/value pairs stored in the tree. Notice that this function is returning an instance of list from the standard template library. However, the list can only store objects of a single type, and we need to return key/value pairs.

Fortunately, C++ includes a simple data structure named `pair`, whose only purpose is to store two items. The following code shows how to declare and create instances of pairs:

```
#include <iostream>
#include <utility>
using namespace std;
int main()
{
	pair<int,int>		p1 = make_pair(10,20);
	pair<string, string>	p2 = make_pair("abc", "def");

	cout << "pair p1 is: " << p1.first << ":" << p1.second << endl;
	cout << "pair p2 is: " << p2.first << ":" << p2.second << endl;
}
```

Be careful when using nested templates, you must type:

```
 list<pair<K,V> >
```

not

```
list<pair<K,V>>
```

In this instance, the space between the two > is important.

When testing your code, you may find it useful to look at the trees the tester uses. A visual representation of each is provided at the end of this PDF.

Dictionary

Once you've finished the Binary Search Tree, you will use it to implement a Dictionary ADT. The primary difference between the binary search tree interface and the dictionary interface is that the dictionary supports the use of the [] operator.

For example:

```
Dictionary<string,int>  d1;
d1["foo"] = 7;
if (d1["foo"] == 7)
      d1["foo"]+= 1;
cout << d1["foo"] << endl;
```

The [] operator returns a reference to the value stored at the key. There are two scenarios when implementing operator[]:

1. The key specified is not currently in the tree. In this case, a new node is inserted into the tree with a default value and a reference to that default value is returned.
2. The key specified is currently in the tree. In this case, a reference to the value is returned.

Contact List

The third part of the assignment is to use the Dictionary to create a simple contact list, like one that would be used in your phone. See the comments in `contactlist.h` for details.

Multi-file C++ Programs

As in previous assignments, you've been given multiple files.

This assignment has three "main" programs: `bst_tester.cpp, dictionary_test.cpp` and `contactlist_tester.cpp`

In order to create the test program for binary search trees, type:

```
g++ -Wall bst_tester.cpp -o bst
```

To create the tester for dictionary, type:

```
g++ -Wall dictionary_tester.cpp -o dictionary
```

To create the tester for phone book, type:

```
g++ -Wall contactlist_tester.cpp -o contact_list
```

Part I

Implement functions in bst.h

Read the comments in bst.h and implement the functions.

When you've finished your implementation, the output of `bst_tester.cpp` should be:

```
test_insert_size_height passed.
test_insert_find passed.
tree1_test passed.
tree2_test passed.
tree3_test passed.
tree4_test passed.
Passed: 6
```

Part II

When your implementation is complete, the output of `dictionary_tester.cpp` should be:

```
test_one passed.
test_two passed.
Passed: 2
```

Part III

When your implementation is complete, the output of `contact_list_tester.cpp` should be:

```
test_add_lookup passed.
test_lookup_update passed.
Passed: 2
```

Submission

Submit your `bst.h, dictionary.h` and `contactlist.h` using connex.

A reminder that it is OK to talk about your assignment with your classmates, and you are encouraged to design solutions together, but each student must implement their own solution.

We will be using plagiarism detection software on your assignment submissions.

Grading

If you submit something that does not compile, you will receive a grade of 0 for the assignment. It is your responsibility to make sure you submit the correct files.

Part I

Requirement	Marks
Your code passes the test cases in `bst_tester.cpp`	Up to 6

Part II

Requirement	Marks
Your code passes the test cases in `dictionary_tester.cpp`	Up to 2

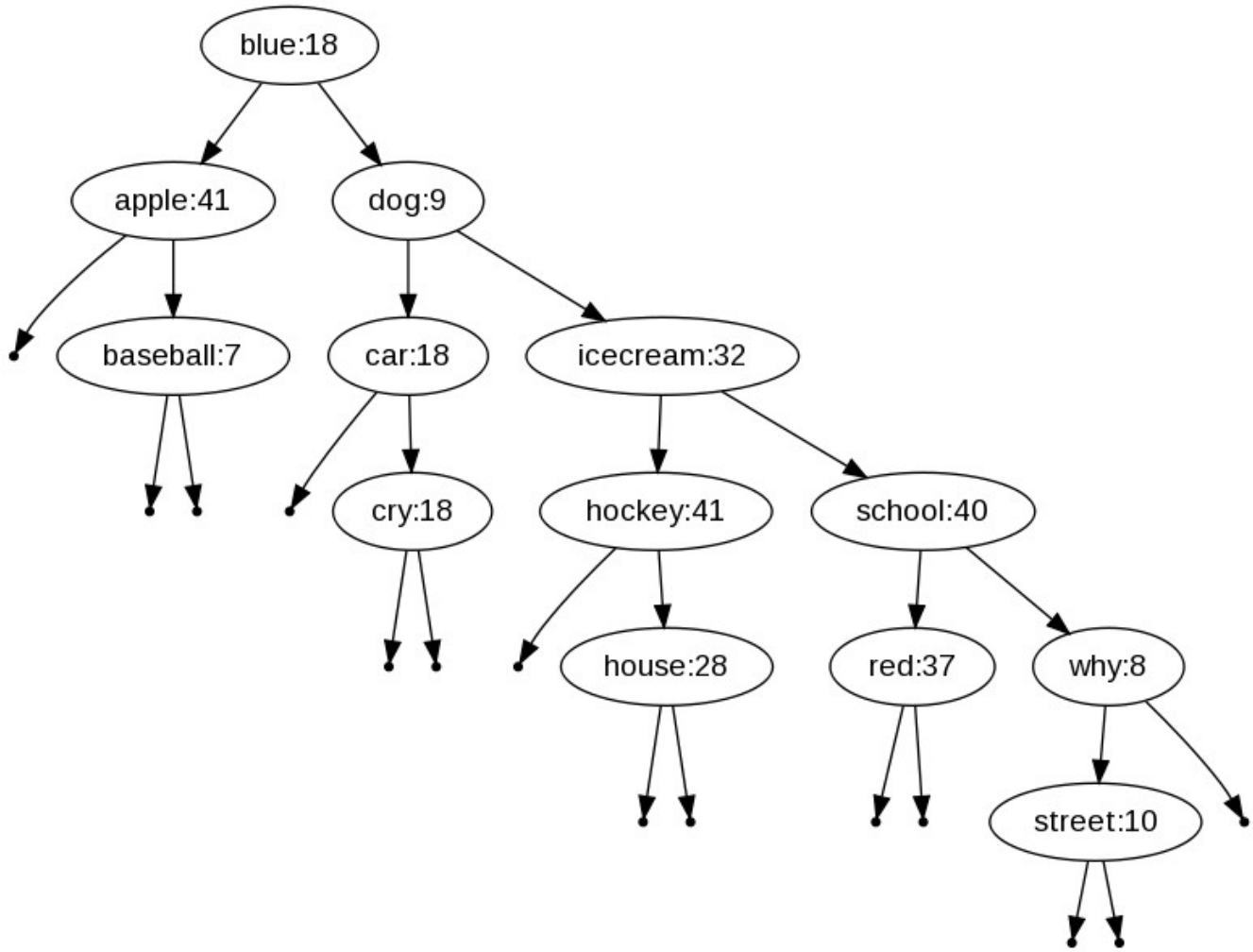
Part III

Requirement	Marks
Your code passes the test cases in `contactlist_tester.cpp`	Up to 2

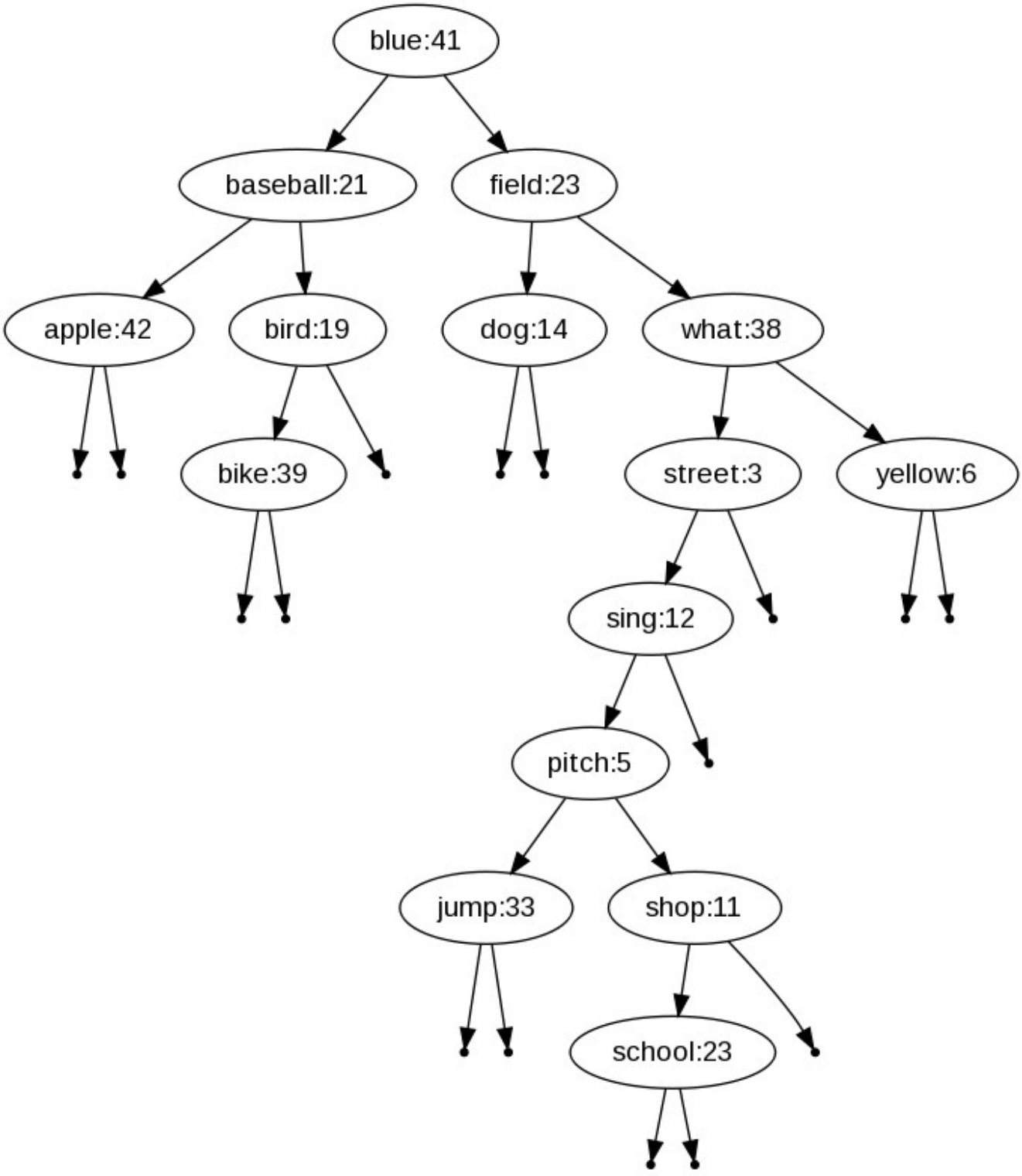
Total 10

Trees used in testing

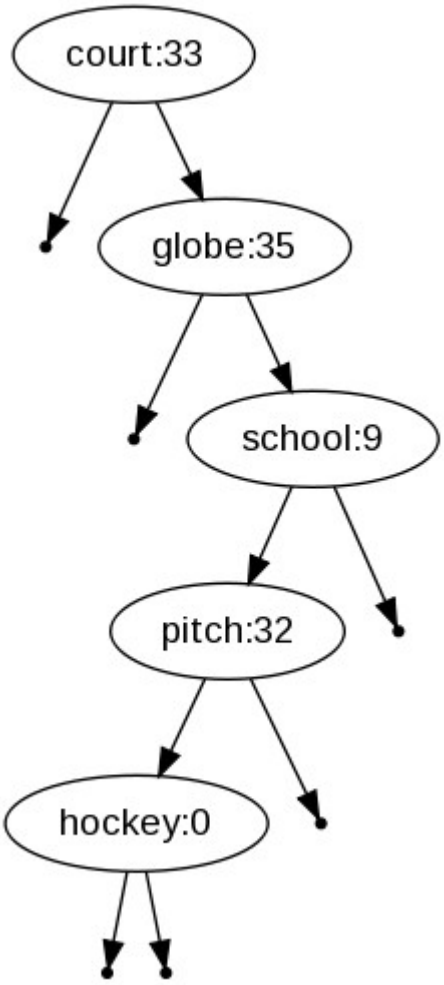
Tree1:



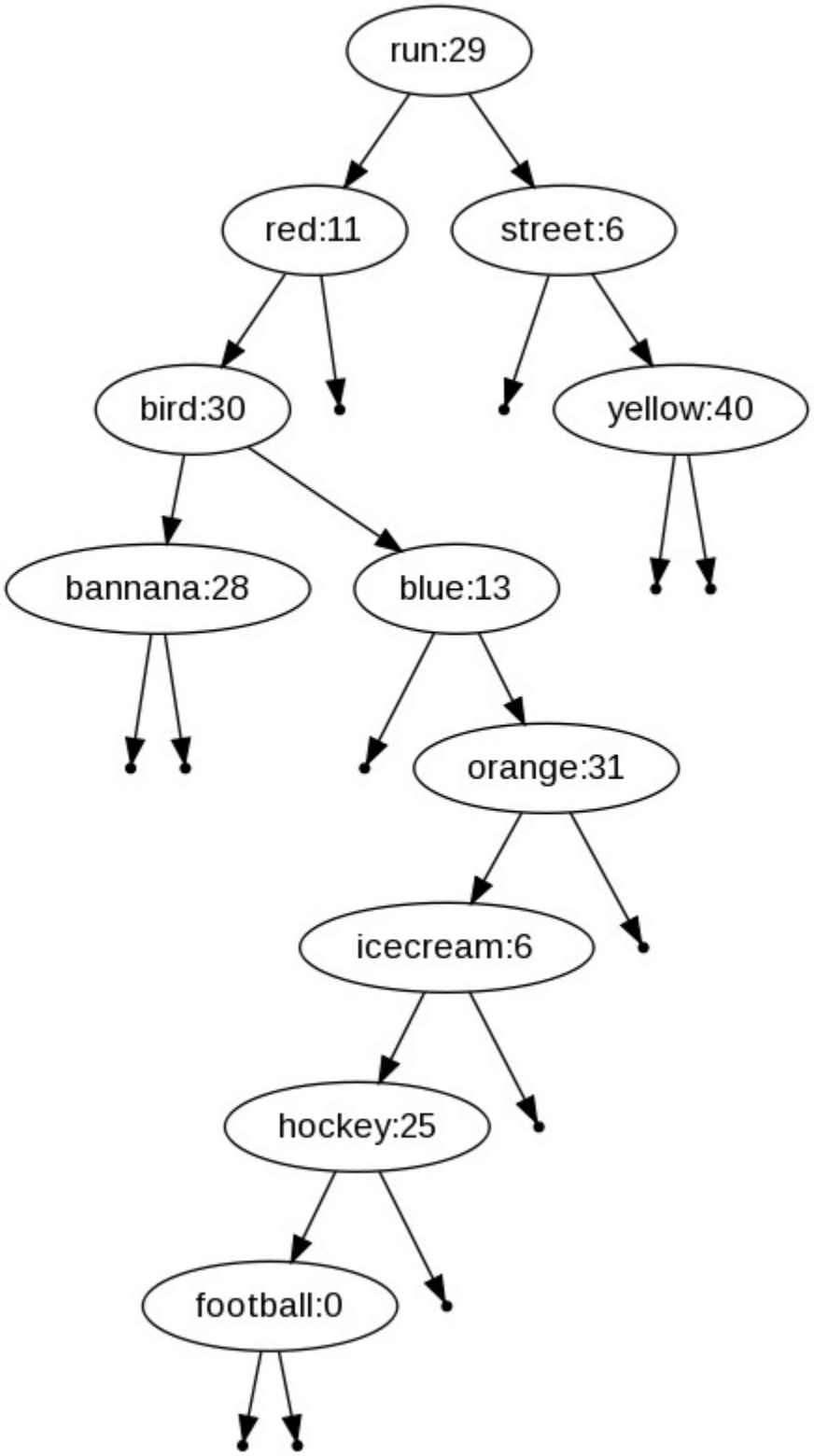
Tree2:


blue:41
baseball:21
field:23
apple:42
bird:19
dog:14
what:38
bike:39
street:3
yellow:6
sing:12
pitch:5
jump:33
shop:11
school:23

Tree3:



Tree4:


run:29
red:11
street:6
bird:30
yellow:40
bannana:28
blue:13
orange:31
icecream:6
hockey:25
football:0