Binary Search Trees

1. Sorting Numbers using a Tree
   - a sorting algorithm
   - using a tree of integer numbers
2. Header Files
   - defining a node struct
   - defining a tree class
3. Definition of Methods
   - selectors, methods `insert()` and `to_string()`
   - the depth of a tree and a membership function

MCS 360 Lecture 24
Introduction to Data Structures
Jan Verschelde, 9 March 2020

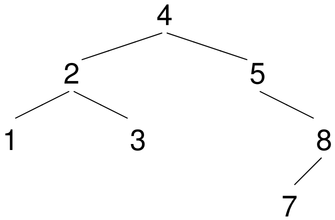
Binary Search Trees

1. Sorting Numbers using a Tree
   - a sorting algorithm
   - using a tree of integer numbers
2. Header Files
   - defining a node struct
   - defining a tree class
3. Definition of Methods
   - selectors, methods `insert()` and `to_string()`
   - the depth of a tree and a membership function

Sorting Numbers using a Tree

Consider the sequence 4, 5, 2, 3, 8, 1, 7

Insert the numbers in a tree:



Rules to insert x at node N:

- if N is empty, then put x in N
- if $x < N$, insert x to the left of N
- if $x \geq N$, insert x to the right of N

Recursive printing: left, node, right sorts the sequence.

running the program

We generate a sequence of random numbers.

```
$ /tmp/inttree
Give n : 8
inserting 36
inserting 471
inserting 297
inserting 453
inserting 142
inserting 917
inserting 259
inserting 123
tree inorder string :
 36  123  142  259  297  453  471  917
$
```

Binary Search Trees

1. Sorting Numbers using a Tree
   - a sorting algorithm
   - using a tree of integer numbers
2. Header Files
   - defining a node struct
   - defining a tree class
3. Definition of Methods
   - selectors, methods `insert()` and `to_string()`
   - the depth of a tree and a membership function

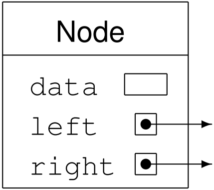
the main program

```
#include "mcs360_integer_tree.h"
using namespace mcs360_integer_tree;

int main()
{
   cout << "Give n : ";
   int n; cin >> n;
   srand(time(0));
   Tree T;
   for(int i=0; i<n; i++)
   {
      int r = rand() % 1000;
      cout << "inserting " << r << endl;
      T.insert(r);
   }
   cout << "tree inorder string : " << endl;
   cout << T.to_string() << endl;
   return 0;
}
```

Binary Tree of Nodes

A binary tree consists of nodes, a node has a data field and two pointers, to left and right child.



If both left and right point to NULL, then the node is a leaf.

The root of the tree is a pointer to a node.

In a binary *search* tree, for every node: all elements in the left subtree are smaller than the data element in the node and larger elements are stored in the right subtree.

binary tree ADT

```
abstract <typename T> binary_tree;
/* A binary tree is either empty or it has a data element
   and at most two subtrees, a left and a right subtree. */

abstract bool empty ( binary_tree t );
postcondition: empty(t)
   == true if t is empty,
   == false if t is not empty;

abstract T data_element ( binary_tree t );
precondition: not empty(t);
postcondition: data_element(t) is the data element of t;

abstract binary_tree left_subtree ( binary_tree t );
precondition: not empty(t);
postcondition: left_subtree(t) is the left subtree of t;

abstract binary_tree right_tree ( binary_tree t );
precondition: not empty(t);
postcondition: right_subtree(t) is the right subtree of t;
```

binary search tree ADT

A binary search tree stores items of a type
for which the comparison operation $<$ is defined.

A binary search tree is a binary tree with an *insert* operation.

```
abstract <typename T> binary_search_tree;
/* A binary search tree is a binary tree where everything
   less than the data element is in the left subtree
   and everyting else is in the right subtree. */

abstract void insert ( binary_search_tree t, T item );
postcondition: if empty(t), then after insert(t, item)
  we have item = data_element(t), it not empty(t), then:
  if item < data_element(t), item is in left_subtree(t)
  if item >= data_element(t), item is in right_subtree(t);

abstract bool member ( binary_search_tree t, T item );
postcondition: true if the item belongs to t,
  false otherwise.
```

Binary Search Trees

1. Sorting Numbers using a Tree
   - a sorting algorithm
   - using a tree of integer numbers
2. Header Files
   - defining a node struct
   - defining a tree class
3. Definition of Methods
   - selectors, methods `insert()` and `to_string()`
   - the depth of a tree and a membership function

a node struct

The file `mcs360_integer_tree_node.h` contains

```
#ifndef __TREE_NODE_H__
#define __TREE_NODE_H__
#include <sstream>

struct Node
{
   int data;     // numbers stored at node in tree
   Node *left;   // pointer to left branch of tree
   Node *right;  // pointer to right branch of tree

   Node(const int& item, Node* left_ptr = NULL,
                         Node* right_ptr = NULL) :
      data(item),
      left(left_ptr), right(right_ptr) {}
```

virtual methods

```
    virtual ~Node() {}

    virtual std::string to_string() const
    {
        std::ostringstream os;
        os << data;
        return os.str();
    }
};

#endif
```

By the `virtual` we can override later.

Binary Search Trees

1. Sorting Numbers using a Tree
   - a sorting algorithm
   - using a tree of integer numbers

2. Header Files
   - defining a node struct
   - defining a tree class

3. Definition of Methods
   - selectors, methods `insert()` and `to_string()`
   - the depth of a tree and a membership function

defining a Tree class

The file `mcs360_integer_tree.h` contains

```
#ifndef __MCS360_INTEGER_TREE_H__
#define __MCS360_INTEGER_TREE_H__
#include "mcs360_integer_tree_node.h"
#include <string>

namespace mcs360_integer_tree
{
   class Tree
   {
      private:

         Node *root;  // data member

         // construct tree from a node
         Tree(Node *r) : root(r) {}
```

public methods

```
public:

   Tree() : root(NULL) {}

   Tree(const int& item,
        const Tree& left = Tree(),
        const Tree& right = Tree() ) :
      root(new Node(item,left.root,right.root)) {}

   Tree get_left() const;
   Tree get_right() const;

   void insert(int item);
   std::string to_string();
```

Binary Search Trees

1. Sorting Numbers using a Tree
   - a sorting algorithm
   - using a tree of integer numbers

2. Header Files
   - defining a node struct
   - defining a tree class

3. Definition of Methods
   - selectors, methods `insert()` and `to_string()`
   - the depth of a tree and a membership function

get_left() and get_right()

The file mcs360_integer_tree.cpp contains

```
#include "mcs360_integer_tree.h"

namespace mcs360_integer_tree
{
   Tree Tree::get_left() const
   {
      return Tree(root->left);
   }

   Tree Tree::get_right() const
   {
      return Tree(root->right);
   }
```

inserting a number

```
void Tree::insert(int item)
{
   if(root == NULL)
      root = new Node(item);
   else if(item < root->data)
   {
      Tree L = this->get_left();
      L.insert(item);
      root->left = L.root;
   }
   else
   {
      Tree R = this->get_right();
      R.insert(item);
      root->right = R.root;
   }
}
```

writing the tree inorder

```
std::string Tree::to_string()
{
   using std::string;
   if(root == NULL)
      return "";
   else
   {
      string L = this->get_left().to_string();
      string d = root->to_string();
      string R = this->get_right().to_string();
      return L + " " + d + " " + R;
   }
}
```

tree traversals

Three kinds of tree traversals:

- preorder: visit node, left tree, and right tree
- inorder: visit left tree, node, and right tree
- postorder: visit left tree, right tree, and node

Binary Search Trees

1. Sorting Numbers using a Tree
   - a sorting algorithm
   - using a tree of integer numbers

2. Header Files
   - defining a node struct
   - defining a tree class

3. Definition of Methods
   - selectors, methods `insert()` and `to_string()`
   - the depth of a tree and a membership function

the Depth of a Tree

The depth of a node is zero is it is the root, otherwise it is 1 + the depth of its parent.

Need two new selector methods:

```
bool Tree::is_left_null() const
{
   return (root->left == NULL);
}
bool Tree::is_right_null() const
{
   return (root->right == NULL);
}
```

Need extra method to get data field:

```
int Tree::get_data() const
{
   return root->data;
}
```

a function depth()

```
int depth ( Tree t )
{
   int L = 0;
   if(!t.is_left_null())
      L = 1 + depth(t.get_left());

   int R = 0;
   if(!t.is_right_null())
      R = 1 + depth(t.get_right());

   return (L > R) ? L : R;
}
```

Note: not a member of the class.

visualizing the depth

Inserting 9 numbers:

```
 49 919 366 307 332 665 955 614 525
tree inorder string :
 49  307  332  366  525  614  665  919  955
depth of tree : 5
49
  919
    366
      307
        332
      665
        614
          525
    955
```

Insert 2 spaces for every level.

write with depth

```
void write_with_depth ( int k, Tree t );
// writes the tree t keeping track of the depth
// with the parameter k, call initially with k = 0

void write_with_depth ( int k, Tree t )
{
   for(int i=0; i<k; i++) cout << "  ";
   cout << t.get_data() << endl;
   if(!t.is_left_null())
      write_with_depth(k+1,t.get_left());
   if(!t.is_right_null())
      write_with_depth(k+1,t.get_right());
}
```

navigating the tree

Does an integer number belong to the tree?

Recursive algorithm `is_in(T,e)`:

- if `e` is in current node, return true
- if `e` $<$ data in current node,
  then return `is_in(T.get_left(),e);`
  else return `is_in(T.get_right(),e).`

the function is_in()

```
bool is_in ( Tree t, int e )
{
   if(e == t.get_data())
      return true;
   else if(e < t.get_data())
   {
      if(t.is_left_null())
         return false;
      else
         return is_in(t.get_left(),e);
   }
   else
   {
      if(t.is_right_null())
         return false;
      else
         return is_in(t.get_right(),e);
   }
}
```

Summary + Exercises

We started with binary search trees covered in §8.4.

Exercises:

1. Adjust the `get_left()` method to throw an exception in case the node is `NULL`.
2. Change the `to_string()` method so that the numbers are printed in decreasing order.
3. Modify `mcs360_integer_tree.h` and `.cpp` to define a templated class for any numeric type.
4. Use the code of this lecture to sort a vector of integer numbers. Write a function that takes on input an STL `vector<int>` and that returns the sorted vector.