CS 331 Group Project  
(Phase 2)

Brandon Theisen  
Jason Pederson  
Kelvin Shultz  
Chris Scholl  
Jared Kariniemi

**Introduction (Both Components):**  
This program implements a two heap, back-to-back, replacement selection algorithm that returns a vector list containing sorted vectors. The vectors inside the list of vectors contains either integers or strings. The vector list is passed into a tournament sort that will sort the data from least too largest. An unsorted file containing the data will be the input for the program, while the output file will contain the sorted data. The output file of the sorted data will then be inserted into a B+ Tree, where the data will be assigned a key and place into the bottom of the tree in *leaf\_nodes*. We are able to insert, remove, search, and print the values in the tree.

**Data Structure (Both Components):**

Component A: *HeapClass*uses a back to back heap replacement sort to put records into multiple lists. As a collection, each list will be placed into a final list vector that will be used for the *tournamentSort*. The *HeapClass* uses an active and pending heap which they will maintain themselves by heapifying up or down depending on the case, for example if an element is inserted or removed to the active heap.

*tournamentSort* accepts a vector list of vectors. Within each vector, is a sorted list containing either integers or strings. *sortedVector* is the final returned vector containing all sorted data. *tempVector* is a temporary vector for data to be put into the final sorted vector. Vectors allow us to use the functions for each vector, which includes size() to return the size of a vector, at() which allows us to look at any element inside the vector, pop\_back(), to pop the last item off the vector, and push\_back() which allows us to insert a vector at the end of the vector. If the vector lists contains 2 or more lists, it compares the lists and gets the smallest element of the two, which is the winner. The smallest winner will advance along the tournament tree and be stored into a buffer vector before being written to the file. If only one list remains, the rest of the list will be stored into the buffer then written to the output file. Once all lists are empty, the sort ends. The output file will be the final stored list of data.

(Component B)  
The B+ tree is comprised of two different types of nodes, *InteriorNode* and *LeafNode*. Our *Node* class is our base class that *InteriorNode* and *LeafNode* will be derived from. Out *Node* class contains 3 data members, a vector of keys, a pointer to its parent node, and also the capacity in which the node we be able to contain. The *InteriorNode* class contains only one data member, which is a vector of pointers to its children. Our B+ tree will be created using these nodes and the functions of the base class, as well as any different functions that the *InteriorNode* and *LeafNode* need to function correctly. The *bTree* class is the main class of the tree, including all the tree functions, which include the insertion, deletion, check to see if the tree is balanced, and balanced the tree if not balanced functions. Not all functions are correctly working, work in progress.

**User Document (Component A):**The program files can be found under the account **CS331128** in the folder called *ComponentA* on the desktop

Top compile:  
 **main.cpp**: g++ main.cpp  
 **testRun.cpp**: g++ testRun.cpp

All data files are included in the folder

**User Document (Component B):**The program files can be found under the account **CS331128** in the folder called *ComponentB* on the desktop

Top compile:  
 **g++ test.cpp**

All data files are included in the folder called **ComponentB** on the desktop

**Main Program Code (Both Components):  
HeapClass.cpp (Component A)**/\*\* @file HeapClass.cpp @brief Replacment Selection Sort @author Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared\*/

#include <vector>

#include <iostream>

#include <algorithm>

#include <stdexcept>

#include "tournamentSort.cpp"

#include <fstream>

#include <iomanip>

using namespace std;

template <class T>

void activeHeapUp(vector<T> &vect);

template <class T>

void activeHeapDown(vector<T> &vect);

template <class T>

void pendingHeapUp(vector<T> &vect);

void emptyActiveHeap();

template <class T>

void printVector(vector<T> &vec);

template <class T>

void printVectorList(vector<vector<T> > &vectList);

template <class T>

void emptyActiveHeap(vector<T> &outVect, vector<T> &contVect, vector<vector<T > > &outVectList);

int active\_heap\_size = 0;

bool working = true;

template <class T>

void replacementSort(vector<T> &inputVector, string outputName)

{

int CONTAINER\_SIZE = 10;

vector<T> containerVector;

vector<T> outputVector;

vector<vector<T> > outputVectorList;

reverse(inputVector.begin(), inputVector.end());

// INITIALIZE THE CONTAINER VECTOR UNTIL IT IS FULL

while((!inputVector.empty()) && containerVector.size() < CONTAINER\_SIZE)

{

containerVector.push\_back(inputVector.at(inputVector.size()-1));

active\_heap\_size++;

//Heapify element at end of active heap

activeHeapUp(containerVector);

//Heapify elemnt at end of pending heap

inputVector.pop\_back();

}

//-----INITIAL STATE END-----\\

T prev, nextValue;

bool popNextFromActive = true;

while(working)

{

if(active\_heap\_size == 0)

{

reverse(containerVector.begin(), containerVector.end());

//ADD NEW VECTOR HERE

if(outputVector.size() > 0)

{

outputVectorList.push\_back(outputVector);

outputVector.clear();

active\_heap\_size = containerVector.size();

}

}

// CHECK IF TEST INPUT IS EMPTY \\

if(inputVector.empty())

{

break;

}

// IF THERE IS A SPACE IN OUR CONTAINER, popNextFromActive will be TRUE \\

if(popNextFromActive)

{

prev = containerVector.at(0);

outputVector.push\_back(prev);

}

// GET NEXT INPUT ITEM \\

nextValue = inputVector.at(inputVector.size()-1);

inputVector.pop\_back();

// IF NEXT\_VALUE == PREV \\

if(nextValue == prev)

{

outputVector.push\_back(nextValue);

popNextFromActive = false;

}

// IF NEXT\_VALUE > PREV \\

else if(nextValue > prev)

{

containerVector.at(0) = nextValue;

activeHeapDown(containerVector);

popNextFromActive = true;

}

// IF NEXT\_VALUE < PREV \\

else

{

containerVector.at(0) = nextValue;

swap(containerVector.at(0), containerVector.at(active\_heap\_size - 1));

active\_heap\_size--;

activeHeapDown(containerVector);

pendingHeapUp(containerVector);

popNextFromActive = true;

}

}

// OUR INPUT STREAM IS EMPTY, SO WE WILL EMPTY THE CONTAINER VECTOR

emptyActiveHeap(outputVector, containerVector, outputVectorList);

//Check the size of each run to calculate execution details

size\_t numberOfRuns = 0;

size\_t runSmallest = 0;

size\_t runLargest = 0;

size\_t runMean = 0;

float mean = 0;

//Catch an empty input list

if(outputVectorList.size() == 0)

runSmallest = 0;

else

runSmallest = outputVectorList.at(0).size();

//Iterate through each output list to find smallest, largest and mean

for(int i=0; i < outputVectorList.size(); i++)

{

if(outputVectorList.at(i).size() > runLargest)

runLargest = outputVectorList.at(i).size();

if(outputVectorList.at(i).size() < runSmallest)

runSmallest = outputVectorList.at(i).size();

runMean = runMean + (outputVectorList.at(i)).size();

}

//Convert runMean data-type into a displayable float data-type

numberOfRuns = outputVectorList.size();

mean = runMean;

mean = mean/numberOfRuns;

//Output the calculated execution details

cout << "Amount of Memory for Runs: " << CONTAINER\_SIZE << endl;

cout << "Number of Runs: " << numberOfRuns << endl;

cout << "Smallest Run: " << runSmallest << endl;

cout << "Largest Run: " << runLargest << endl;

cout << "Arithmetic Mean: " << setprecision(2) << fixed << showpoint << mean << endl;

cout << "Output File: " << outputName << endl;

// TAKE OUTPUT VECTOR LIST, SORT IT, AND EXPORT IT TO A FILE

vector<T> finalVector;

finalVector = tournamentSort(outputVectorList);

ofstream outFile;

printToFile(finalVector, outFile, outputName);

}

/\*\*Heapify the active heap so insert smallest elment on the top

@pre Accepts a vector containing integers or strings

@param vect: vector containing integers or strings

@post Inserts the smallest element to the top of the active heap\*/

template <class T>

void activeHeapUp(vector<T> &vect)

{

int index = active\_heap\_size-1;

int parent = (index-1)/2;

while(vect.at(index) < vect.at(parent) && index != 0)

{

swap(vect.at(index), vect.at(parent));

index = parent;

parent = (index - 1)/2;

}

}

/\*\*Heapify the active heap

@pre Accepts a vector containing integers or strings

@param vect: vector containing integers or strings

@post Sorts the element downwards\*/

template <class T>

void activeHeapDown(vector<T> &vect)

{

int index = 0;

int leftChild, rightChild, smallest;

while(index < active\_heap\_size ) //Index should never reach beyond the heap size

{

leftChild = (2\*index) + 1;

rightChild = (2\*index) + 2;

smallest = index;

if(leftChild < active\_heap\_size)

{

if(vect.at(leftChild) < vect.at(smallest))

smallest = leftChild;

}

else

{

break; //There is no left child, and subsequently, no right child

}

if(rightChild < active\_heap\_size)

{

if(vect.at(rightChild) < vect.at(smallest))

smallest = rightChild;

}

if(smallest != index)

{

swap(vect.at(index),vect.at(smallest));

index = smallest;

}

else //There are no smaller children, so there is no more swapping possible

break;

}

}

/\*\*Heapify the pending heap

@pre Accepts a vector containing integers or strings

@param vect: vector containing integers or strings

@post Inserts last element to the top of the pending heap\*/

template <class T>

void pendingHeapUp(vector<T> &vect)

{

int index = active\_heap\_size;

int parent = vect.size() - ((vect.size() - index)/2);

while(index < vect.size()-1 && vect.at(index) < vect.at(parent))

{

swap(vect.at(index), vect.at(parent));

index = parent;

parent = vect.size() - ((vect.size() - index)/2);

}

}

/\*\*Heapify the active heap so insert smallest elment on the top

@pre Active heap contaings zero elements\*/

template <class T>

void emptyActiveHeap(vector<T> &outVect, vector<T> &contVect, vector<vector<T > > &outVectList)

{

while(active\_heap\_size !=0)

{

outVect.push\_back(contVect.at(0));

swap(contVect.at(0), contVect.at(active\_heap\_size - 1));

contVect.erase(contVect.begin() + active\_heap\_size - 1);

active\_heap\_size--;

activeHeapDown(contVect);

}

if(outVect.size() > 0)

outVectList.push\_back(outVect);

if(contVect.size() > 0)

{

outVect.clear();

reverse(contVect.begin(), contVect.end());

active\_heap\_size = contVect.size();

while(active\_heap\_size !=0)

{

outVect.push\_back(contVect.at(0));

swap(contVect.at(0), contVect.at(active\_heap\_size - 1));

contVect.erase(contVect.begin() + active\_heap\_size - 1);

active\_heap\_size--;

activeHeapDown(contVect);

}

outVectList.push\_back(outVect);

}

}

/\*\*Prints the vector to output screen

@pre Accepts a vector containing integers or strings

@param vect: vector containing integers or strings\*/

template <class T>

void printVector(vector<T> &vec)

{

for(int i=0; i < vec.size(); i++)

{

cout << " " << vec.at(i);

}

cout << endl << endl;

}

/\*\*Prints the list of vectors to the output screen

@pre Accepts a list of vectors containing integers or strings

@param vect: vector containing integers or strings\*/

template <class T>

void printVectorList(vector<vector<T> > &vectList)

{

for(int i=0; i < vectList.size(); i++)

{

printVector(vectList.at(i));

}

}

**tournamentSort.cpp (Component A)**

/\*\*

@file tournamentSort.cpp

@brief Tournament Sort and Test Program

@author Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

\*/

#include <iostream> //Input/Output stream objects

#include <vector> //Vecotr libaray

#include <fstream> //Input and Output file streams

using namespace std;

template <class T>

vector<T> tournamentSort(vector< vector<T> >& v);

template <class T>

void printToFile(vector<T> v, ostream& outputFile, string outputName);

/\*\*Uses tournament sort to sort a list of vectors

@pre Accepts a list of vectors that contains sorted integers or strings

@param v: vector of vectors that contain sorted integers or strings

@post returns a vector of sorted strings or integers\*/

template <class T>

vector<T> tournamentSort(vector< vector<T> >& v)

{

vector<T> sortedVector; //final sorted vector to return

vector<T> tempVector; //temp vector for putting at end of list

size\_t toBeSortedIndex; //how many vectors to be still sorted

bool done = false; //true when fully sorted

toBeSortedIndex = v.size(); //size of final returned value

//while vector list is not completely sorted

while(!done)

{

//checks if input vector list is empty

if(v.size() == 0)

{

done = true;

} //end if

//Only one vector left to sort in list

else if(v.size() == 1)

{

tempVector = v.at(0);

v.pop\_back();

//put the last list onto the final sorted list

while(!tempVector.empty())

{

sortedVector.push\_back(tempVector.at(0));

tempVector.erase(tempVector.begin());

} //end while

}//end else if

//two or more vectors remain in list

else

{

int searchIndex = 0; //current vector in list to compare smallest element with

T smallestItem = v.at(0).at(0); //smallest element in vector list

int vectorContainingSmallestItem = 0; //vector in list containing smallest element

while(searchIndex < v.size()) //while current vector in list is less than list size

{

//if current element in current vector is less than smallest item

if(v.at(searchIndex).at(0) < smallestItem)

{

smallestItem = v.at(searchIndex).at(0); //sets smallestItem to new smallest element

vectorContainingSmallestItem = searchIndex; //new vector with smallest element

searchIndex++;

} //end if

//Current vector to compare goes to the next vector in the list

else

{

searchIndex++;

} //end else

} //end while

sortedVector.push\_back(smallestItem); //push smallest element onto final vector

//if vector in list has only one element

if(v.at(vectorContainingSmallestItem).size() == 1)

{

v.erase(v.begin()+(vectorContainingSmallestItem)); //delete vector from list

} //end if

//erase smallest item in vector list

else

{

v.at(vectorContainingSmallestItem).erase(v.at(vectorContainingSmallestItem).begin());

} //end else

} //end else

} //end else

return sortedVector; //return final sorted vector

} //end tournamentSort

/\*\*Outputs sorted list to file

@pre Accepts a a vector of a sorted complete list

@param v: sorted vector of a complete list of vectors or integers

@param outputFile: output file to be written to

@post return is void\*/

template <class T>

void printToFile(vector<T> v, ofstream& outputFile, string outputName)

{

//Create output file for sortedlist

outputFile.open(outputName.c\_str());

//Print sorted list to output file

for(int j=0; j<v.size(); j++)

{

//outputFile << v.at(j) << endl;

if(j % 10 == 0 && j != 0)

{

outputFile << v.at(j) << endl;

}

else

{

outputFile << v.at(j) << " ";

}

} //end for

//Close output file

outputFile.close();

} //end printToFile **main.cpp (Component A)**

/\*\*

@file main.cpp

@brief Main program for replacement heap sort and tournament sort

@author Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

\*/

#include <vector>

#include "HeapClass.cpp"

using namespace std;

int main()

{

ifstream inputFile; //input file stream

vector<int> test\_input\_ints; //vector for integers

vector<string> test\_input\_strings; //vector for strings

string input\_string; //input for strings

int input\_int; //input for integers

string filename; //input for user desired filename

string choice; //user input for type of data to sort

cout << "Please enter file name to be sorted: ";

getline( cin, filename );

cout << "What type of data do you wish to test (integers or strings)?: ";

cin >> choice;

cout << endl << "Input File: " << filename << endl;

if(choice == "integers")

{

inputFile.open(filename.c\_str());

while(inputFile >> input\_int)

{

test\_input\_ints.push\_back(input\_int);

}

replacementSort(test\_input\_ints, "Sorted\_Ints.txt");

}

else if(choice == "strings")

{

inputFile.open(filename.c\_str());

while(inputFile >> input\_string)

{

test\_input\_strings.push\_back(input\_string);

}

replacementSort(test\_input\_strings, "Sorted\_Strings.txt");

}

return 0;

}

**Run of Main.cpp**

**data.txt**

24 29 11 1 2 64 1097 228 399 476 37 476 112 3 497

2937 408 390 4862 498 298 384 3 87 49 38 398 398

**Output**  
Please enter file name to be sorted: data.txt

What type of data do you wish to test (integers or strings)?: integers

Input File: data.txtAmount of Memory for Runs: 10Number of Runs: 2  
Smallest Run: 9

Largest Run: 19Arithmetic Mean: 14.00Output File: Sorted\_Ints.txt

**Node.h (Component B)**/\*\*

@file Node.h

@brief Base Node Class

@author Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

\*/

#ifndef \_NODE

#define \_NODE

#include <vector>

using namespace std;

template<class T>

class Node

{

protected:

vector<int> keys; //Container for the keys within the node

Node<T>\* parentPtr; //Points to the parent node

int capacity; //Block size for all nodes (Passed through constructor)

//----- TO DO -----\\

//Remove virtual from functions and remove generic functions from Leaf and Interior Nodes

//If the function is not defined in Leaf Node, it will simply call Node implementation.

//As long as the implementation is not different between Leaf and Interior Nodes, do it that way.

//Split and Merge are different because of the need to split/merge children, values, \*next, \*prev, etc.

//A-Level will need to be defined and have operators overloaded.

public:

Node();

Node(int cap);

//Copy Constructor

Node(const Node& newCopy);

//GET KET AT

int getKeyAt(int position);

//SEARCH KEY

virtual int searchKey(int key);

//ADD KEY

virtual int addKey(int newKey) = 0;

//GET PARENT

Node<T>\* getParent();

//SET PARENT

void setParent(Node<T>\* newParentPtr);

//GET SIZE

int getSize();

//GET CHILD SIZE

virtual int getChildSize();

//GET CHILD

virtual Node<T>\* getChild(int key) = 0;

//SET CHILD

//virtual void setChild(Node<T>\* newChild, int position) = 0;

//CONTAINS

bool contains(int key);

//SPLIT

virtual void split(Node<T>\* &newNodePtr) = 0;

//MERGE

virtual void mergeNodes(Node<T>\* &otherNodePtr) = 0;

//Destructor

virtual ~Node();

};

#include "Node.cpp"

#endif // \_NODE

**Node.cpp (Component B)**/\*\*

@file Node.cpp

@brief Implementation of base Node Class

@author Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

\*/

#include "Node.h"

#include <vector>

#include <iostream>

using namespace std;

//-----CONSTRUCTOR-----\\

/\*\*Default Constructor for Node

@post Creates a base Node for either a leaf node or a interior node\*/

template<class T>

Node<T>::Node()

{

capacity = 6; //6 is default

//Default Constructor

}

//-----CONSTRUCTOR WITH CAPACITY-----\\

/\*\*Default Constructor for Node with capacity

@pre Accepts a capacity for a node, depending on what type of node is being created

@param cap: capacity of the size in the node

@post Creates a base Node for either a leaf node or a interior node with a capacity\*/

template<class T>

Node<T>::Node(int cap)

{

capacity = cap;

}

//-----COPY CONSTRUCTOR-----\\

/\*\*Copy constructor that copies the node that calls it

@pre Accepts a node to be copied

@param Node to be copied

@post Copies the node that called the constructor\*/

template<class T>

Node<T>::Node(const Node& newCopy)

//: keys(newCopy.keys), capacity(newCopy.capacity), parentPtr(newCopy.parentPtr)

{}

//-----GET KEY AT-----\\

/\*\*Returns the key at its position

@pre Position must be less than keys.size()

@param position: position of key that is being called

@post Returns the position of the key being searched\*/

template<class T>

int Node<T>::getKeyAt(int position)

{

return keys.at(position);

}

//-----SEARCH KEYS-----\\

/\*\*Search Keys

@param key: key that will be searched for

@post Return the position of the key in the vector of keys\*/

template<class T>

int Node<T>::searchKey(int key)

{

for(int i = 0; i < keys.size(); i++)

{

if(key < keys.at(i))

{

return i;

}

}

return keys.size();

}

//-----ADD NEW KEY-----\\

/\*\*Add a key

@pre Node must have room for the new key. size() <= capacity

@param newKey: key that will be entered

@post Return true or false on if the key was successfully added\*/

template<class T>

int Node<T>::addKey(int newKey)

{

for(int i = 0; i < keys.size(); i++)

{

if(newKey < keys.at(i))

{

keys.insert(keys.begin() + i, newKey);

return i;

}

else if(newKey == keys.at(i))

return -1;

}

//We add the newKey to the end, since it was larger than all existing.

//We checked to make sure there is room to add a key, otherwise

//push\_back would be overextending our keys <vector>.

//With the precondition met, a push\_back() is OK.

int position = keys.size();

keys.push\_back(newKey);

return position;

}

//-----GET PARENT-----\\

/\*\*Get Parent of Node

@post Return the parent of the node, otherwise will return NULL\*/

template<class T>

Node<T>\* Node<T>::getParent()

{

return parentPtr;

}

//-----SET PARENT-----\\

/\*\*Set Parent to a Node

@param newParentPtr: pointer to be set as parent to a node

@post Sets the parent to the node to newParentPtr\*/

template<class T>

void Node<T>::setParent(Node<T>\* newParentPtr)

{

parentPtr = newParentPtr;

}

/\*\*Get the size of the nodes child

@post Return the size the child node\*/

template<class T>

int Node<T>::getChildSize()

{

return 0;

}

//-----GET SIZE-----\\

/\*\*Get the size of the node

@post Return the number of items in the node\*/

template<class T>

int Node<T>::getSize()

{

return keys.size();

}

//-----CONTAINS-----\\

/\*\*Checks to see if a key is contained in the node

@param key: key that will be searched for

@post Return true or false depending if the key was found\*/

template<class T>

bool Node<T>::contains(int key)

{

for(int i = 0; i < keys.size(); i++)

{

if(key < keys.at(i))

{ //Exit early because we are checking a sorted list

//If the previous comparison was greater than, and now it is less than,

//it clearly is not in the list and would be between these two comparisons.

//As in, key > 5 but then key < 7 on the next loop, so it does not exist.

return false;

}

else if(key == keys.at(i))

{ //We found a match

return true;

}

}

//No matches were found

return false;

}

//-----SPLIT-----\\

/\*\*Split Node

@param newNodePtr: pointer to node that will be split, provided by the B+ Tree class

@post Creates a new node and places half the keys into the new node\*/

template<class T>

void Node<T>::split(Node<T>\* &newNodePtr)

{

int value = getSize();

for(int i = (keys.size()/2); i < keys.size(); i++)

{

newNodePtr->addKey(getKeyAt(i));

}

//Clear out the keys that were just moved from the existing node

keys.erase(keys.begin() + (keys.size()/2), keys.end());

//Check parent for an open pointer in children <vector>.

//If none are available, a split will be required.

}

//-----MERGE-----\\

/\*\*Merge two nodes together

@param otherNodePtr: Pointer of the node the be merged with

@post Merges the two nodes together, placing all keys into the orinigal node\*/

template<class T>

void Node<T>::mergeNodes(Node<T>\* &otherNodePtr)

{

for(int i = 0; i < otherNodePtr->getSize(); i++)

{

//Since the keys will be in order and otherNode is to the right,

//we can simply use push\_back() to insert.

keys.push\_back(otherNodePtr->getKeyAt(i));

}

//otherNode.getParent() and set it to NULL, since we are deleting its child.

//Check parent to see if we should be merging it with a sibling, after

//deleting one of its children.

delete otherNodePtr;

otherNodePtr = NULL;

}

//-----DESTRUCTOR-----\\

//

template<class T>

Node<T>::~Node()

{

//Destructor

}

**Leaf\_node.h (Component B)**/\*\*

@file LeafNode.h

@brief LeafNode class

@author Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

\*/

#ifndef \_LEAF\_NODE

#define \_LEAF\_NODE

#include <vector>

#include "Node.h"

using namespace std;

template<class T>

class LeafNode : public Node<T>

{

public:

LeafNode();

LeafNode(int cap);

//Copy Constructor

LeafNode(const Node<T>& newCopy);

//ADD KEY

int addKey(int newKey);

//GET CHILD

Node<T>\* getChild(int key);

//SET CHILD

void setChild(Node<T>\* newChild, int position);

//SPLIT

void split(Node<T>\* &newNodePtr);

//MERGE

void mergeNodes(Node<T>\* &otherNodePtr);

virtual ~LeafNode();

//These functions are implemented in Node.h

int getSize();

Node<T>\* getParent();

void setParent(Node<T>\* newParentPtr);

int getKeyAt(int position);

bool contains(int key);

};

#include "LeafNode.cpp"

#endif // \_Leaf\_NODE

**Leaf\_node.cpp (Component B)**/\*\*

@file LeafNode.cpp

@brief Implementation of LeafNode class

@author Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

\*/

#include "LeafNode.h"

#include <vector>

#include <iostream>

using namespace std;

//-----CONSTRUCTOR-----\\

/\*\*Default Constructor for LeafNode

@post Creates a leaf node\*/

template<class T>

LeafNode<T>::LeafNode()

{

//Node<T>::Node();

//capacity = 6; //6 is default

//Default Constructor

}

//-----CONSTRUCTOR WITH CAPACITY-----\\

/\*\*Default Constructor for LeafNode

@param cap: Capicity of the node

@post Creates a leaf node with a set capicity\*/

template<class T>

LeafNode<T>::LeafNode(int cap)

{

Node<T>::Node(cap);

}

//-----COPY CONSTRUCTOR AS NODE-----\\

/\*\*Copy constructor that copies the node that calls it

@pre Accepts a node to be copied

@param Node to be copied

@post Copies the node that called the constructor\*/

template<class T>

LeafNode<T>::LeafNode(const Node<T>& newCopy)

{}

//-----ADD NEW KEY-----\\

/\*\*Add a key

@pre Node must have room for the new key. size() <= capacity

@param newKey: key that will be entered

@post Return true or false on if the key was successfully added into the vector of keys\*/

template<class T>

int LeafNode<T>::addKey(int newKey)

{

//return true;

return Node<T>::addKey(newKey);

}

//-----GET CHILD-----\\

/\*\*Returns the child where the key located

@param key: key to be searched for

@post Returns the node where the key is located\*/

template<class T>

Node<T>\* LeafNode<T>::getChild(int key)

{

return this;

}

/\*\*Sets the child of a node

@pre Called onto a interior node

@param newChild: The child to be set

@param position: Position of the node

@post Set the child to the node which is being called by\*/

template<class T>

void LeafNode<T>::setChild(Node<T>\* newChild, int position)

{

cout << "Don't call me!" << endl;

}

//-----SPLIT-----\\

/\*\*Split node

@param newNodePtr: Pointer to the new node that is being created

@post Splits the node by placing half of the key into a new node\*/

template<class T>

void LeafNode<T>::split(Node<T>\* &newNodePtr)

{

Node<T>::split(newNodePtr);

}

//-----MERGE-----\\

/\*\*Merge two nodes together

@param otherNodePtr: Pointer of the node the be merged with

@post Merges the two nodes together, placing all keys into the orinigal node\*/

template<class T>

void LeafNode<T>::mergeNodes(Node<T>\* &otherNodePtr)

{

Node<T>::mergeNodes(otherNodePtr);

}

//-----DESTRUCTOR-----\\

//

template<class T>

LeafNode<T>::~LeafNode()

{

//Destructor

}

/\*--These functions are implemented in Node.cpp

//

int getSize();

Node<T>\* getParent();

void setParent(Node<T>\* newParentPtr);

int getKeyAt(int position);

bool contains(int key);

\*/

**Interior\_node.h (Component B)**/\*\*

@file InteriorNode.h

@brief InteriorNode class

@author Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

\*/

#ifndef \_INTERIOR\_NODE

#define \_INTERIOR\_NODE

#include <vector>

#include "Node.h"

using namespace std;

template<class T>

class InteriorNode : public Node<T>

{

private:

//vector<int> keys;

vector<Node<T>\* > children;

public:

InteriorNode();

InteriorNode(int cap);

//Copy Constructor

InteriorNode(const Node<T>& newCopy);

//GET CHILD SIZE

int getChildSize();

//GET CHILD

Node<T>\* getChild(int key);

//SET CHILD

void setChild(Node<T>\* newChild, int position);

//REMOVE CHILD

void removeChild(int position);

//SEARCH KEY

int searchKey(int key);

//ADD KEY

int addKey(int newKey);

//REMOVE KEY

void removeKey(int position);

//SPLIT

void split(Node<T>\* &newNodePtr);

//MERGE

void mergeNodes(Node<T>\* &otherNodePtr);

//DESTRUCTOR

virtual ~InteriorNode();

//These functions are implemented in Node.h

int getSize();

Node<T>\* getParent();

void setParent(Node<T>\* newParentPtr);

int getKeyAt(int position);

bool contains(int key);

};

#include "InteriorNode.cpp"

#endif // \_INTERIOR\_NODE

**Interior\_node.cpp (Component B)**/\*\*

@file InteriorNode.cpp

@brief Implementation of InteriorNode class

@author Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

\*/

#include "InteriorNode.h"

#include <vector>

#include <iostream>

using namespace std;

//-----CONSTRUCTOR-----\\

/\*\*Default Constructor for InteriorNode

@post Creates a interiorNode\*/

template<class T>

InteriorNode<T>::InteriorNode()

{

Node<T>::capacity = 6;

//Node<T>::Node();

//capacity = 6; //6 is default

//Default Constructor

}

//-----CONSTRUCTOR WITH CAPACITY-----\\

/\*\*Default Constructor for Node with capacity

@pre Accepts a capacity for a node, depending on what type of node is being created

@param cap: capacity of the size in the node

@post Creates a base Node for either a leaf node or a interior node with a capacity\*/

template<class T>

InteriorNode<T>::InteriorNode(int cap)

{

Node<T>::capacity = cap;

}

//-----COPY CONSTRUCTOR-----\\

/\*\*Copy constructor that copies the node that calls it

@pre Accepts a node to be copied

@param Node to be copied

@post Copies the node that called the constructor\*/

template<class T>

InteriorNode<T>::InteriorNode(const Node<T>& newCopy)

{}

//-----GET CHILD SIZE-----\\

/\*\*Get size of the node

@post Returns the size of the node\*/

template<class T>

int InteriorNode<T>::getChildSize()

{

return children.size();

}

//-----GET CHILD-----\\

/\*\*Returns the child where the key located

@param key: key to be searched for

@post Returns the node where the key is located\*/

template<class T>

Node<T>\* InteriorNode<T>::getChild(int key)

{

return children.at(searchKey(key));

}

//-----SET CHILD-----\\

/\*\*Sets the child of a node

@pre Called onto a interior node

@param newChild: The child to be set

@param position: Position of the node

@post Set the child to the node which is being called by\*/

template<class T>

void InteriorNode<T>::setChild(Node<T>\* newChild, int position)

{

children.insert(children.begin() + position, newChild);

}

//-----REMOVE CHILD-----\\

//All removes of children must be accompanied by a removal of a key

template<class T>

void InteriorNode<T>::removeChild(int position)

{

children.erase(children.begin() + position);

}

//-----SEARCH KEY-----\\

/\*\*Searches for a key in a node

@pre Called from a interior node

@param key: The key to be searched for

@post Returns the position of the key\*/

template<class T>

int InteriorNode<T>::searchKey(int key)

{

for(int i = 0; i < Node<T>::keys.size(); i++)

{

if(key <= Node<T>::keys.at(i))

{

return i;

}

}

return Node<T>::keys.size();

}

//-----ADD NEW KEY-----\\

/\*\*Add a key

@pre Node must have room for the new key. size() <= capacity

@param newKey: key that will be entered

@post Return true or false on if the key was successfully added\*/

template<class T>

int InteriorNode<T>::addKey(int newKey)

{

//return true;

return Node<T>::addKey(newKey);

}

//-----REMOVE KEY-----\\

/\*\*Remove a key

@pre Key must be located in the node

@param position: Location of the key in the vector

@post Removes the key from the vector\*/

template<class T>

void InteriorNode<T>::removeKey(int position)

{

Node<T>::keys.erase(Node<T>::keys.begin() + position);

}

//-----SPLIT-----\\

/\*\*Split node

@param newNodePtr: Pointer to the new node that is being created

@post Splits the node by placing half of the key into a new node\*/

template<class T>

void InteriorNode<T>::split(Node<T>\* &newNodePtr)

{

Node<T>::split(newNodePtr);

}

//-----MERGE-----\\

/\*\*Merge two nodes together

@param otherNodePtr: Pointer of the node the be merged with

@post Merges the two nodes together, placing all keys into the orinigal node\*/

template<class T>

void InteriorNode<T>::mergeNodes(Node<T>\* &otherNodePtr)

{

Node<T>::mergeNodes(otherNodePtr);

}

//-----DESTRUCTOR-----\\

//

template<class T>

InteriorNode<T>::~InteriorNode()

{

//Destructor

}

/\*--These functions are implemented in Node.cpp

//

int getSize();

Node<T>\* getParent();

void setParent(Node<T>\* newParentPtr);

int getKeyAt(int position);

bool contains(int key);

\*/

**Test Data (Component A):**

**testRun.cpp***testRun.cpp* contains all test runs of the sorts. Each type of data (integers and strings) is tested in all cases, including an empty file, a sorted file, a partially sorted file, reverse-sorted (worst-case) and a large file with over 200 records. Test output is below the source code.

/\*\*

@file testRun.cpp

@brief Test program for replacement and tournamentSort

@author Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

\*/

#include <vector>

#include "HeapClass.cpp"

using namespace std;

ifstream inputFile; //input file stream

vector<int> test\_input\_ints; //vector for integers

vector<string> test\_input\_strings; //vector for strings

string input\_string; //input for strings

int input\_int; //input for integers

vector<string> test\_data\_files; //vector of data files to be tested

void processFile(string name, string choice, string outputName);

int main()

{

processFile("intsEmpty.txt", "integers", "Sorted\_intsEmpty.txt");

processFile("intsLarge.txt", "integers", "Sorted\_intsLarge.txt");

processFile("intsSorted.txt", "integers", "Sorted\_intsSorted.txt");

processFile("intsPartiallySorted.txt", "integers", "Sorted\_intsPartiallySorted.txt");

processFile("intsReverseSorted.txt", "integers", "Sorted\_intsReverseSorted.txt");

processFile("stringsEmpty.txt", "strings", "Sorted\_stringsEmpty.txt");

processFile("stringsLarge.txt", "strings", "Sorted\_stringsLarge.txt");

processFile("stringsSorted.txt", "strings", "Sorted\_stringsSorted.txt");

processFile("stringsPartiallySorted.txt", "strings", "Sorted\_stringsPartiallySorted.txt");

processFile("stringsReverseSorted.txt", "strings", "Sorted\_stringsReverseSorted.txt");

return 0;

}

void processFile(string name, string choice, string outputName)

{

inputFile.open(name.c\_str());

cout << endl << "Input File: " << name << endl;

if(choice == "integers")

{

while(inputFile >> input\_int)

{

test\_input\_ints.push\_back(input\_int);

}

replacementSort(test\_input\_ints, outputName);

}

else if(choice == "strings")

{

while(inputFile >> input\_string)

{

test\_input\_strings.push\_back(input\_string);

}

replacementSort(test\_input\_strings, outputName);

}

inputFile.clear();

inputFile.close();

}

**Test Output**

Input File: intsEmpty.txt //empty integers list

Amount of Memory for Runs: 10

Number of Runs: 0

Smallest Run: 0

Largest Run: 0

Arithmetic Mean: -nan

Output File: Sorted\_intsEmpty.txt

NO OUTPUT

Input File: intsLarge.txt //large amount of integers

Amount of Memory for Runs: 10

Number of Runs: 27

Smallest Run: 1

Largest Run: 26

Arithmetic Mean: 18.52

Output File: Sorted\_intsLarge.txt

1 1 1 1 1 1 2 2 2 2 3

3 3 3 3 4 4 4 5 5 5

6 6 6 6 6 7 7 7 7 7

7 8 8 8 8 8 9 9 9 9

9 9 9 9 9 9 10 10 10 11

11 11 11 11 12 12 12 12 12 12

13 13 13 13 13 14 14 14 15 15

15 15 15 15 15 15 16 16 16 17

17 17 17 17 17 18 18 18 18 18

18 19 19 19 19 20 20 20 20 20

20 20 21 21 21 21 21 22 22 22

23 23 23 23 23 23 23 24 24 24

24 25 25 25 26 26 26 26 26 27

27 27 27 27 27 27 28 28 28 28

28 28 29 29 29 30 30 30 30 30

30 30 30 30 31 31 31 32 32 32

32 32 32 32 33 33 33 33 33 33

33 33 34 34 34 35 35 35 36 36

36 36 36 37 37 37 38 38 38 38

38 39 39 39 39 40 40 40 40 40

40 40 41 41 41 41 41 41 41 41

41 42 42 42 42 42 42 42 42 43

43 43 44 44 44 44 45 45 45 45

45 45 45 45 46 46 47 47 47 47

47 48 48 48 48 48 49 49 49 50

50 50 50 50 50 51 51 52 52 52

52 53 53 53 53 53 53 54 54 54

54 54 54 54 54 54 54 55 55 55

56 56 56 56 56 56 56 57 57 57

57 58 58 58 59 59 59 59 60 60

60 60 60 60 60 60 60 61 61 61

61 62 62 62 62 62 62 62 63 63

63 63 64 64 64 64 64 65 65 65

65 66 66 66 66 67 67 67 67 68

68 68 69 69 69 69 69 69 70 70

71 71 71 72 72 72 72 72 73 73

73 73 74 74 74 74 75 75 75 75

75 75 75 76 76 76 77 77 77 77

78 78 78 78 79 79 80 80 80 80

80 80 80 81 81 81 81 81 82 82

83 83 83 83 83 83 83 84 84 84

84 85 85 85 85 85 85 86 86 86

86 87 87 87 87 87 87 88 88 89

89 89 89 89 89 89 90 90 91 91

91 91 91 91 91 91 91 91 91 91

91 91 92 92 92 92 92 92 92 92

93 93 93 93 93 93 93 93 94 94

94 94 94 95 95 95 96 96 96 97

97 97 97 98 98 98 98 98 99 99

99 99 99 100 100 100 100 100 100

Input File: intsSorted.txt //already sorted integers

Amount of Memory for Runs: 10

Number of Runs: 1

Smallest Run: 53

Largest Run: 53

Arithmetic Mean: 53.00

Output File: Sorted\_intsSorted.txt

1 2 3 4 5 6 7 8 9 10 11

12 13 14 15 16 17 18 19 20 21

22 23 24 25 26 27 28 29 30 31

32 33 34 35 36 37 38 39 40 41

42 43 44 45 46 47 48 49 50 51

52 53

Input File: intsPartiallySorted.txt //partially sorted integers

Amount of Memory for Runs: 10

Number of Runs: 2

Smallest Run: 15

Largest Run: 33

Arithmetic Mean: 24.00

Output File: Sorted\_intsPartiallySorted.txt

1 1 1 2 3 4 5 6 8 9 10

12 13 14 15 16 17 18 19 20 21

22 23 24 25 26 27 28 29 30 31

32 33 34 35 36 37 38 39 40 41

42 43 44 45 46 48 50

Input File: intsReverseSorted.txt //reversed order integers

Amount of Memory for Runs: 10

Number of Runs: 6

Smallest Run: 1

Largest Run: 10

Arithmetic Mean: 8.50

Output File: Sorted\_intsReverseSorted.txt

1 2 3 4 5 6 7 8 9 10 11

12 13 14 15 16 17 18 19 20 21

22 23 24 25 26 27 28 29 29 30

31 32 33 34 35 36 37 38 39 40

41 42 43 44 45 46 47 48 49 50

Input File: stringsEmpty.txt //empty file of strings

Amount of Memory for Runs: 10

Number of Runs: 0

Smallest Run: 0

Largest Run: 0

Arithmetic Mean: -nan

Output File: Sorted\_stringsEmpty.txt

NO OUTPUT

Input File: stringsLarge.txt //larger amount of string records

Amount of Memory for Runs: 10

Number of Runs: 22

Smallest Run: 1

Largest Run: 25

Arithmetic Mean: 18.18

Output File: Sorted\_stringsLarge.txt

Adam Adan Adolfo Adrian Ahern Albert Alecia Alexander Allen Althea Alvarado

Alyse Amaya Anderson Andre Andy Angulo Anika Annelle Argelia Arnold

Asa Augustina Ayres Ballard Barb Barnhart Barton Bass Batten Beam

Beck Bello Berry Bethany Beulah Blackmon Blalock Bobby Bonilla Boudreaux

Bowman Bradbury Bradford Bradford Bradley Brandi Brice Bridget Brill Brock

Brooke Brown Bruns Burgess Byrnes Cannon Carpenter Carroll Carson Carswell

Cass Castellanos Castillo Cesar Chapman Cheryl Christian Chung Clair Clara

Clemmons Close Colette Coley Collazo Collier Collins Cone Consuelo Cook

Cooper Cora Corey Corie Cornelius Cottrell Crane Crews Cristen Cristina

Cruz Crystal Cummings Curry Curtis Daisy Dallas Dan Daniel Daniels

Danille Dawne Delagarza Delois Demetria Derek Desimone Dickey Dirk Donald

Dorsey Duncan Dupuis Dusti Earlene Edgardo Edison Edward Elane Ellis

Emilio Eric Estep Evelin Evelyne Everett Evers Ewing Ezra Fatima

Faye Ferguson Fernande Fleming Flora Flowers Floyd Foreman Forte Foss

Fox Francis Francisco Frasier Freddie Frederick Freeman Fry Gabriela Gayle

Gearldine Geneva Genevieve Gordon Greene Griffith Haines Hansen Hardison Hawk

Hawkins Henderson Henry Herbert Hermina Herrera Hickman Hogan Holloman Houston

Howard Ira Ismael Jackson Jacquelin Jacquie Jake Jamey Jamie Janay

Jaqueline Jeannie Jeffery Jenkins Jennings Jeraldine Jessie Jill Jimmy Joe

Jolanda Jordan Juan Kallie Karisa Katrina Keen Kelly Kelly Kevin

Klein Kopp Kristen Laurence Laurice Leah Legg Lehman Len Leona

Leslie Lessie Lillie Lim Lionel Lissette Logan Lopes Lori Luis

Lula Luvenia Lynn Mabel Madeleine Magana Maier Makeda Mann Maple

Marceline Marcella Marcia Margareta Marjory Marta Martha Marvin Matilde Mauricio

Maximina Mcbride Mcdowell Mclaughlin Meehan Meggan Meghan Melendez Mendoza Mesa

Meza Micheal Mitchell Mona Moody Morrell Morrison Morton Munoz Murphy

Nagel Nancy Nation Neal Ness Neva Newman Norman Nunez Obdulia

Olive Oliveira Olivia Owen Padilla Pak Park Parris Patricia Patrina

Peachey Pemberton Pete Peters Peterson Peyton Philip Phylis Pittman Poirier

Poore Porter Potter Prater Queenie Ralph Ramsey Rauch Raynor Regalado

Reyes Rhonda Richard Richie Riley Robena Robinson Robyn Rochelle Rod

Rodgers Rogers Rose Ross Ruby Russ Russell Samuel Sanford Saunders

Saxton Schaffer Schultz Scully Sean Sergio Shaina Sharp Sheba Sheehan

Shelley Shellie Shelton Sheppard Sheri Sherri Sheryl Shin Sid Singleton

Slagle Sophie Spaulding Spearman Stacy Stacy Stephen Stone Strickland Stroud

Su Sullivan Summers Susannah Tejeda Teresa Terry Theodora Theodore Theresia

Thersa Thomas Tiffany Timika Tommie Tonya Townsend Tracey Travis Trimble

Tristan Ty Ulysses Vada Valdez Vargas Vernon Versie Wallace Waller

Warner Warren Weaver Wendi Wentworth Wesley Wilhelm Willene Willia Willie

Wong Wood Woodrow Woodruff Woods Yolanda Yoon Zenaida Zenia

Input File: stringsSorted.txt //already sorted string records

Amount of Memory for Runs: 10

Number of Runs: 1

Smallest Run: 26

Largest Run: 26

Arithmetic Mean: 26.00

Output File: Sorted\_stringsSorted.txt

allie bob cat dog elaine fred george harry ink jane kelp

larry man no ollie potty que randy steve tongue unary

view walter xi yellow zebra

Input File: stringsPartiallySorted.txt //partially sorted string records

Amount of Memory for Runs: 10

Number of Runs: 2

Smallest Run: 8

Largest Run: 18

Arithmetic Mean: 13.00

Output File: Sorted\_stringsPartiallySorted.txt

allie bob cat dog elaine fred george harry ink jane kelp

larry man no ollie potty que randy steve tongue unary

view walter xi yello zebra

Input File: stringsReverseSorted.txt //reversed order string records

Amount of Memory for Runs: 10

Number of Runs: 3

Smallest Run: 6

Largest Run: 10

Arithmetic Mean: 8.67

Output File: Sorted\_stringsReverseSorted.txt

allie bob cat dog elaine fred george harry ink jane kelp

larry man no ollie potty que randy steve tongue unary

view walter xi yello zebra

**Test (Component B):**

**Test.cpp**#include <vector>

#include <iostream>

#include "Node.h"

#include "InteriorNode.h"

#include "LeafNode.h"

using namespace std;

//-----PROTOTYPES-----\\

void printNode(Node<int>\* aNodePtr);

void printTree(Node<int>\* rootPtr);

void printLeaf(Node<int>\* aNodePtr);

bool treeContains(Node<int>\* rootPtr, int keyToFind);

Node<int>\* searchLeaf(Node<int>\* rootPtr, int keyToFind);

void addKeyValuePair(Node<int>\* nodePtr, int key, int value);

//-----GLOBAL VARIABLES-----\\

Node<int>\* rootPtr;

Node<int>\* tempPtr;

int blockSize = 2;

int main()

{

/\*

int input = 1;

int key = 1;

cout << "Create Parent 1:" << endl;

tempPtr = new InteriorNode<int>;

addKeyValuePair(tempPtr, key+2, input+2);

printLeaf(tempPtr);

rootPtr = tempPtr;

//cout << "Create Leaf 1:" << endl;

tempPtr = new LeafNode<int>;

addKeyValuePair(tempPtr, key, input);

addKeyValuePair(tempPtr, key+1, input+1);

addKeyValuePair(tempPtr, key+2, input+2);

addKeyValuePair(tempPtr, key+1, input+5);

//printLeaf(tempPtr);

cout << "Child Node Added: " << endl;

rootPtr->setChild(tempPtr, 0);

printLeaf(rootPtr->getChild(2));

//cout << "Create Leaf 2:" << endl;

tempPtr = new LeafNode<int>;

addKeyValuePair(tempPtr, key+3, input+3);

addKeyValuePair(tempPtr, key+4, input+4);

addKeyValuePair(tempPtr, key+5, input+5);

addKeyValuePair(tempPtr, key+3, input+6);

//printLeaf(tempPtr);

cout << "Child Node Added: " << endl;

rootPtr->setChild(tempPtr, 1);

printLeaf(rootPtr->getChild(4));

addKeyValuePair(rootPtr, 7, 7);

cout << "LeafNode: " << endl;

printNode(tempPtr);

cout << "RootNode: " << endl;

printNode(rootPtr);

addKeyValuePair(rootPtr, 0, 10);

cout << "LeafNode: " << endl;

printNode(tempPtr);

cout << "RootNode: " << endl;

printNode(rootPtr);

cout << "Create Leaf: " << endl;

tempPtr = new LeafNode<int>;

addKeyValuePair(tempPtr, 1, 1);

addKeyValuePair(tempPtr, 2, 2);

printLeaf(tempPtr);

cout << endl;

cout << "Create Leaf: " << endl;

tempPtr = new LeafNode<int>;

addKeyValuePair(tempPtr, 3, 3);

addKeyValuePair(tempPtr, 4, 4);

printLeaf(tempPtr);

cout << endl;

cout << "Create Leaf: " << endl;

tempPtr = new LeafNode<int>;

addKeyValuePair(tempPtr, 5, 5);

addKeyValuePair(tempPtr, 6, 6);

printLeaf(tempPtr);

cout << endl;

cout << "Create Leaf: " << endl;

tempPtr = new LeafNode<int>;

addKeyValuePair(tempPtr, 7, 7);

addKeyValuePair(tempPtr, 8, 8);

printLeaf(tempPtr);

cout << endl;

\*/

//-----TEST LEAF NODE-----\\

//

int value = 5;

Node<int>\* Int1Ptr = new InteriorNode<int>;

Node<int>\* Int2Ptr = new InteriorNode<int>(\*Int1Ptr);

Node<int>\* L1Ptr = new LeafNode<int>;

Node<int>\* L2Ptr = new LeafNode<int>(\*L1Ptr);

for(int i = 1; i < 7; i++)

{

L1Ptr->addKey(i);

}

for(int i = 1; i < 7; i++)

{

Int1Ptr->addKey(i);

}

cout << "L1Ptr: " << endl;

printNode(L1Ptr);

cout << "L2Ptr: " << endl;

printNode(L2Ptr);

cout << endl;

L1Ptr->split(L2Ptr);

cout << "L1Ptr: " << endl;

printNode(L1Ptr);

cout << "L2Ptr: " << endl;

printNode(L2Ptr);

cout << endl;

L1Ptr->mergeNodes(L2Ptr);

cout << "L1Ptr: " << endl;

printNode(L1Ptr);

//The node that L2Ptr points to will be deleted

//Attempting to print it will result in a crash

//Test Contains

//cout << L1Ptr->contains(0) << endl;

//cout << "L1Ptr Size: " << L1Ptr->getSize() << endl;

cout << "Int1Ptr Size: " << Int1Ptr->getSize() << endl;

cout << "Int1Ptr Size: " << endl;

printNode(Int1Ptr);

cout << "Child Pointer Location: " << Int1Ptr->searchKey(3) << endl;

Node<int>\* Child1Ptr = new LeafNode<int>(\*L1Ptr);

Node<int>\* Child2Ptr = new LeafNode<int>(\*L1Ptr);

Node<int>\* Child3Ptr = new LeafNode<int>(\*L1Ptr);

Node<int>\* Child4Ptr = new LeafNode<int>(\*L1Ptr);

Node<int>\* L3Ptr = new LeafNode<int>(\*L1Ptr);

InteriorNode<int>\* Int3Ptr = new InteriorNode<int>;

InteriorNode<int>\* Int4Ptr = new InteriorNode<int>(5);

Int3Ptr->addKey(3);

Int3Ptr->addKey(6);

Int3Ptr->addKey(9);

for(int i = 1; i < 4; i++)

{

Child1Ptr->addKey(i);

Child2Ptr->addKey(i+3);

Child3Ptr->addKey(i+6);

Child4Ptr->addKey(i+9);

Child1Ptr->addValue(i, i-1);

Child2Ptr->addValue(i+3, i-1);

Child3Ptr->addValue(i+6, i-1);

Child4Ptr->addValue(i+9, i-1);

L3Ptr->addKey(i+3);

cout << "Add Value: " << i+3 << " at " << i-1 << endl;

L3Ptr->addValue(i\*12, i-1);

}

cout << "Before Insert Child: " << Int3Ptr->getChildSize() << endl;

Int3Ptr->setChild(Child1Ptr, 0);

cout << "Insert Child: " << Int3Ptr->getChildSize() << endl;

Int3Ptr->setChild(Child2Ptr, 1);

cout << "Insert Child: " << Int3Ptr->getChildSize() << endl;

Int3Ptr->setChild(Child3Ptr, 2);

cout << "Insert Child: " << Int3Ptr->getChildSize() << endl;

Int3Ptr->setChild(Child4Ptr, 3);

cout << "Insert Child: " << Int3Ptr->getChildSize() << endl;

Int3Ptr->removeChild(1);

Int3Ptr->removeKey(1);

cout << "Remove Child: " << Int3Ptr->getChildSize() << endl;

Int3Ptr->addKey(L3Ptr->getKeyAt(L3Ptr->getSize()-1));

Int3Ptr->setChild(L3Ptr, 1);

cout << "Insert Leaf: " << Int3Ptr->getChildSize() << endl;

printNode(Int3Ptr);

printNode(Int3Ptr->getChild(3));

printNode(Int3Ptr->getChild(4));

printNode(Int3Ptr->getChild(8));

printNode(Int3Ptr->getChild(20));

//Int3Ptr->getChild(0)->addKey(5);

if(Int3Ptr->getChild(20) == nullptr)

cout << "YUP, IT'S NULL!" << endl;

cout << endl << endl;

//int testKey = 12;

//cout << "Position of key in child: " << searchLeaf(Int3Ptr,testKey)->searchKey(testKey) << endl;

//cout << "Leaf Node: " << endl;

//printLeaf(L3Ptr);

int testKey = 5;

if(treeContains(Int3Ptr, testKey))

{

cout << "TRUE" << endl;

cout << "The key corresponds to a value of ";

cout << searchLeaf(Int3Ptr, testKey)->findValue(testKey) << endl;

}

else

cout << "FALSE" << endl;

cout << endl;

cout << tempPtr->getKeyAt(0) << endl;

/\*

//-----TEST INTERIOR NODE-----\\

//

Node<int>\* Int1Ptr = new InteriorNode<int>;

Node<int>\* Int2Ptr = new InteriorNode<int>(\*Int1Ptr);

for(int i = 1; i < 6; i++)

{

Int1Ptr->addKey(i);

}

cout << "Int1Ptr: " << endl;

printNode(Int1Ptr);

cout << "Int2Ptr: " << endl;

printNode(Int2Ptr);

cout << endl;

Int1Ptr->split(Int2Ptr);

cout << "Int1Ptr: " << endl;

printNode(Int1Ptr);

cout << "Int2Ptr: " << endl;

printNode(Int2Ptr);

cout << endl;

Int1Ptr->mergeNodes(Int2Ptr);

cout << "Int1Ptr: " << endl;

printNode(Int1Ptr);

//The node that L2Ptr points to will be deleted

//Attempting to print it will result in a crash

\*/

return 0;

}

/\*\*Add key and value to a node

@post Creates a base Node for either a leaf node or a interior node\*/

void addKeyValuePair(Node<int>\* nodePtr, int key, int value)

{

if(nodePtr->getSize() == 0)

{

nodePtr->addValue(value, nodePtr->addKey(key));

}

else if(!treeContains(nodePtr, key))

{

Node<int>\* tmpPtr = searchLeaf(rootPtr, key);

tmpPtr->addValue(value, tmpPtr->addKey(key));

}

}

/\*\*Print Node

@post prints the contents of the node to the screen\*/

void printNode(Node<int>\* aNodePtr)

{

for(int i = 0; i < aNodePtr->getSize(); i++)

{

cout << aNodePtr->getKeyAt(i) << " ";

}

cout << endl;

}

/\*\*Print leaf node

@post prints the contents of the leaf node to the screen\*/

void printLeaf(Node<int>\* aNodePtr)

{

printNode(aNodePtr);

for(int i = 0; i < aNodePtr->getSize(); i++)

{

cout << aNodePtr->findValue(aNodePtr->getKeyAt(i)) << " ";

}

cout << endl;

}

/\*\*Print tree

@post prints the contents of the tree to the screen\*/

void printTree(Node<int>\* rootPtr)

{

for(int i = 0; i < rootPtr->getSize(); i++)

{

printNode(rootPtr->getChild((i+1)\*3));

}

}

/\*\*Checks to see if a node contains the key searched for

@post Returns true or false depending if the key was succesfully found\*/

bool treeContains(Node<int>\* rootPtr, int keyToFind)

{

tempPtr = searchLeaf(rootPtr, keyToFind);

return tempPtr->contains(keyToFind);

}

/\*\*Searchs leaf for the key

@post returns the leaf node where the key is located\*/

Node<int>\* searchLeaf(Node<int>\* rootPtr, int keyToFind)

{

Node<int>\* tempPtr = rootPtr->getChild(keyToFind);

printLeaf(tempPtr);

if(tempPtr->getChildSize() == 0)

return tempPtr;

return searchLeaf(tempPtr, keyToFind);

}

**Output (Explanation during presentation)**L1Ptr:  
1 2 3 4 5 6  
L2Ptr:  
  
L1Ptr:  
1 2 3  
L2Ptr:  
4 5 6  
  
L1Ptr:  
1 2 3 4 5 6  
Int1Ptr Size: 6  
Int1Ptr Size:  
1 2 3 4 5 6  
Child Pointer Location: 2  
Add Value: 4 at 0  
Add Value: 5 at 1  
Add Value: 6 at 2  
Before Insert Child: 0  
Insert Child: 1  
Insert Child: 2  
Insert Child: 3  
Insert Child: 4  
Remove Child: 3  
Insert Leaf: 4  
3 6 9  
1 2 3  
4 5 6  
7 8 9  
Child Position1  
10 11 12  
Child Position1  
  
  
Does Tree contain key?  
4 5 6  
12 24 36  
TRUE  
The key corresponds to a value of 4 5 6  
12 24 36  
24  
  
4

**Individual Response Paragraphs**

Jared Kariniemi

I spent around 10 hours on this project altogether. As a group we spent many hours working out what was required conceptually before attempting to code anything, so we did plan well. The project was tested within the parameters of the accepted inputs quite extensively. Numbers and integers were tested, empty lists were tested, and fully and partially sorted lists were tested among other things. Prior to this project, I was not very good at using vectors, but this project used them extensively, and I am now relatively comfortable with vectors. I was, of course, familiar with selection sort, but replacement selection was completely new to me. Now I have a good understanding of how replacement selection works. As far as studying, I realized how valuable it is to work with others on programming assignments. Often when one is stuck on something, it is relatively simple and another person might notice the mistake right away. I would have also had a hard time understanding the conceptual parts of this assignment without this group, so in this case, studying in a group was essential.

Jason Pederson

For this project around 20 hours was spent on the program. A large majority of the time was spent on understanding the B+ tree so that we could properly implement it make a functioning tree. A lot of planning was done to make sure that our ideas and planned approach was sound before we tried to code anything. At the same time, we had a try a few approaches out before we could really focus on the best approach to programming this project. We tested the program for all the aspects that we were able to complete in the allotted time. As far as the language goes, I did not really learn anything about the language. The majority of learning was more about how these data structures work and implementation of such a ADT. I had a pretty firm grasp on C++ as a language and only maybe found a few tricks on how to implement a portion of the project more efficiently. I found out that I study best by just trying things out and taking a very hands-on approach. I do not do very well with reading things or just looking at things. So for me, the best way to learn about concepts in programming is to just try program something and figure things out as I’m working on them. Its not always the most efficient method, but it’s the method where I actually learn the most.

Chris Scholl

I spent about 10 hours on this program, most of which were spent reading about the B+ tree for component B, as well as contributing ideas for implementation (i.e. inheritance for B+ tree nodes) I also spent some time working on code for part A which made it so arbitrary data could be read in from a text file. As a group, we spent about 2 and a half hours working on the design for the B+ tree implementation, so I believe the project was well thought out before going to the computer. I didn’t learn anything completely new about the language. As for studying, I learned that having a plan before beginning to study is better than blindly going in studying.

Kelvin Schutz  
I spent 30 hours on the program, most of it spent on getting familiar with different B+ Tree implementation algorithms and familiarizing myself with various cases of each operation of the B+ Tree. Another large portion of time was spent refamiliarizing myself with class relationships and designing the base Node class to be implemented differently by the uniquenesses of Leaf and Interior Node classes. Naturally, the rest was spent on attempting to create proper encapsulation and developing a hierarchy that seemed logical to the average developer.

We spent the first few days simply confirming our understanding of the B+ Tree and what user level interactions would be taking place. From there we started laying out the ‘knowns’ of our classes and what gaps we would have to fill in through design choices. Due to the need for refreshing of past C++ material, some proof-of-concept testing was needed to see if our plans could be realized in code. Particularly, getting an abstract base class from which to build Leaf and Interior Node provided the greatest challenge.

Throughout development of each node class, we worked on the easiest methods first and tested their functionality using their known boundaries of operation. We tried to build incrementally at each step and work in a consistent methodology of using pointers rather than working with the objects themselves. Since we did not have enough time to work our code all the way to the top of the hierarchy, we were not able to completely implement and test fully functional merge and split operations, although they exist in basic function. We did find that by thoroughly testing up from the ground level, complex functions became progressively easier to implement since we had put in the time to consider all possible cases for functions that were highly relied upon.

Templates were a great refresher in this project. The largest struggle was properly utilizing inheritance with the numerous ways it could be designed into our implementation. 301 provided a grounds for being able to see the difference between all the kinds of class relationships, but this was the first time we were able to pick and choose a method of inheritance that fit the needs of our design. Overall, it was rewarding to implement inheritance into our program and see how it made things much easier down the development road.

Ultimately, a timed itinerary would have really helped in gauging where in the development timeline we were during each phase of development. For this project in particular, and the small amount of time we had, making enough time for both design and development was difficult, since the number of design considerations vastly outnumbered the amount of hours we had available. At a point, we had to stop designing and taking the first solution that could solve our design problem. With that, I would have spent more time dividing of components of the project and giving a deadline that we could spend on each item, before having to go with the best option at the moment.

Brandon Theisen

I spent a good 25+ hours or so, on this project. Most of it was doing the designing. As a group, we tried to think out each of the cases that would appear while searching, inserting, and removing values and keys into the tree. Our best work was doing the design, which took a lot longer than we anticipated. We tested each function that we implemented. While many of the programming requirements were not met, the building blocks are there and working. Each node is working fully, which allows us to insert keys into the nodes, splitting and merging are also working. The nodes are also able to set and get a child if it exists, also setting and getting the child of an interior node. We have it so the parents and their children are doubly linked so it is easier to traverse the tree. The biggest thing that I took is how important the design is. Something this complicated, needs to well thought out. If you just start coding, you will run into many problems, which will allow you to code this type of project. In 301 we touched on polymorphism and derived classes. But as we started to do a base node class, and created classes from that base, like what we did with the interior node and leaf node.