ECS 60 Programming Assignment #2, 50 points (my time 3 hours!!!) Fall 2017

Due Wednesday, October 25th, 11:59pm.

Files to handin to cs60 p2 are: BTree.cpp, InternalNode.cpp, InernalNode.h, LeafNode.cpp, LeafNode.h, authors.csv.

Files: BTreeDriver.cpp, BTreeDriverDebug.cpp, BTree.cpp, BTree.h, BTreeNode.cpp, BTreeNode.h, InternalNode.cpp, InernalNode.h, LeafNode.cpp, LeafNode.h, Makefile (uses BTreeDriverDebug.cpp to produce BTree), and Makefile2 (uses BTreeDriver.cpp to produce BTree2), QueueAr.cpp QueueAr.h, vector.h, vector.cpp, dsexceptions.h.

Your program will read a file that contains a series of integers that it will insert into a BTree, and then print to the screen a breadth first traversal of the BTree. You will find all files, including my executables, in ~ssdavis/60/p2. Your task will be to provide the implementation code for the BTree, InternalNode, and LeafNode classes. Since the insert methods for the nodes will be quite long, you should add new methods so that no method is more than 30 lines long. We will not grade on style, but the TAs and I will be of little help with poor quality code. You may only alter those files that will be handed in.

The program will take three command line parameters: 1) filename; 2) M, the number of children that internal nodes have; and 3) L, the number of integers each leaf node holds. The input file will not include more than 1000 integers. Since only integers will be stored in the BTree class, I did not implement it as a template. When a node splits, the new node created will always contain at least as many entries as the remaining old node, and will contain the larger values. The output must match that shown in the sample.

There is one change in the data structure from the book's; the internal nodes should keep track of the minimum value for every child. You will find this easier than not having the minimum for the first child.

How did I write this code so fast? I used procedural abstraction. I thought about the tasks, their subtasks, and the subtasks of those subtasks. (You should write them down.) If I could state a subtask, then I planned on writing a separate function to do just that subtask. I found that the tasks of the InternalNode class matched that of the LeafNode class, so I postponed as much work on the InternalNode class as long as possible so that I could learn from my LeafNode mistakes. I worked top down with stubs for the subtasks, and got that iteration to compile without warnings! This avoids making the same mistake repeatedly—the compiler is a very good teacher. When I could do even minimal testing, I did it. What I learned from debugging kept me from duplicating those mistakes in later code, and reminded me of issues that I had forgot about. Sometimes when I addressed a new issue in a method I discovered that I had to add a lot of code. If there is a whole lot of new code, the issue deserved its own method with an appropriate name. It is a simple matter of cutting and pasting, and keeps each method short, and understandable. By the time I was done my LeafNode class had nine methods, and no method has more than 30 lines. Once the LeafNode class worked perfectly, I copied the LeafNode class methods as InternalNode class methods, and did a search and replace from values to keys! Though there are child sibling and parent issues for the InternalNode class, the basic LeafNode code provided a great starting point. In the end my InernalNode class had 13 methods. In comparison to the first time I wrote this with long insert functions a few years ago, the debugging was much faster this time because of the small methods

Here is a good progression of parameters for testing using BTreeDebugDriver.cpp. You should compare your output to that of mine.

BTree1.dat 3 3	To check if LeafNode insertion works at all!
BTree2.dat 3 3	To check if ordering LeafNode ordering works.
BTree5.dat 3 5	To check proper ordering in a single LeafNode.
BTree5.dat 3 3	To check split of LeafNode with odd leafSize, and then check creation of Internal Node.
BTree5.dat 3 4	To check split LeafNode with even leafSize.
BTree5.dat 3 2	To check passing to left leaf sibling, ordering in InternalNode, and Internal key maintenance.
BTree12.dat 8 2	To check passing to right leaf sibling; and correct LeafNode leftSibling, rightSibling, and parent maintenance. Once this works, copy the LeafNode methods to InternalNode.
BTree5.dat 2 2	To check split of InternalNode class with even internalSize.
BTree12.dat 3 2	To check split of InternalNode class with odd internalSize, and correct InternalNode leftSibling, and rightSibling maintenance.
BTree12.dat 2 2	To check passing to left internal node, and parent maintenance during InternalNode splitting.
BTree25.dat 3 2	Final check #1. (I found a bug during this run)
BTree25.dat 4 3	Final check #2. (No bugs found yet.)

This is the most complex, time consuming program I ever assign. START EARLY!!

Here are a couple of other "tricks" that I DO use. I keep my functions alphabetized within each class, except that the constructors and destructors are listed first. I write the name of the function, after its closing "}". Both of these make searching for functions quicker. Besides the standard "}" comments of ECS 40, I provide comments whenever something is the least bit tricky. For example, what is the value of pos after the for loop? Is it the position for insertion, one before, or one after? Handy to know, a ten second comment when I write the for loop, and a pain to remember an hour later. Sometimes I describe the state of the object when it enters a function. It is effortless to write these comments when I am writing the code initially, and incredibly useful when debugging. Ten seconds now, can save minutes (or possibly hours for you) later on! I try to avoid using "i" in loops when there is a better name, such as pos. I may write the loop with i, and then do a quick search and replace with pos, or some other longer, more descriptive name.

I found I sometimes had to cast to the proper class type to deal with siblings, e.g., ((LeafNode*) rightSibling)->…. To avoid a ton of seg faults, test for NULL pointers before dealing with a parent or sibling. You may use the buildbags.cpp program in the p2 directory to construct files if you wish.

```
class BTree
{
  BTreeNode *root;
  int internalSize;
  int leafSize;
public:
  BTree(int ISize, int LSize);
  void insert(int value);
  void print();
}; // BTree class
```

```
class BTreeNode
{
protected:
  int count;
  int leafSize;
  InternalNode *parent;
  BTreeNode *leftSibling;
  BTreeNode *rightSibling;
public:
  BTreeNode(int LSize, InternalNode *p, BTreeNode *left,
    BTreeNode *right);
  BTreeNode* getLeftSibling();
  virtual int getMinimum()const = 0;
  BTreeNode* getRightSibling();
  virtual BTreeNode* insert(int value) = 0;
  virtual void print(Queue <BTreeNode*> &queue) = 0;
  void setLeftSibling(BTreeNode *left);
  void setParent(InternalNode *p);
  void setRightSibling(BTreeNode *right);
}; //BTreeNode class
```

```
class InternalNode:public BTreeNode
{
  int internalSize;
  BTreeNode **children;
  int *keys;
public:
  InternalNode(int ISize, int LSize, InternalNode *p,
    BTreeNode *left, BTreeNode *right);
  int getMinimum()const;
  InternalNode* insert(int value); // returns pointer to new InternalNode
    // if it splits else NULL
  void insert(BTreeNode *oldRoot, BTreeNode *node2); // if root splits use this
  void insert(BTreeNode *newNode); // from a sibling
  void print(Queue <BTreeNode*> &queue);
}; // InternalNode class
```

```
class LeafNode:public BTreeNode
{
  int *values;
public:
  LeafNode(int LSize, InternalNode *p, BTreeNode *left,
    BTreeNode *right);
  int getMinimum() const;
  LeafNode* insert(int value); // returns pointer to new Leaf if splits else NULL
  void print(Queue <BTreeNode*> &queue);
}; //LeafNode class
```

```
$ BTree BTree12.dat 3 2
Inserting 3.
Leaf: 3

Inserting 4.
Leaf: 3 4

Inserting 8.
Internal: 3 4
Leaf: 3
Leaf: 4 8

Inserting 1.
Internal: 1 4
Leaf: 1 3
Leaf: 4 8

Inserting 10.
Internal: 1 4 8
Leaf: 1 3
Leaf: 4
Leaf: 8 10

Inserting 2.
Internal: 1 3 8
Leaf: 1 2
Leaf: 3 4
Leaf: 8 10

Inserting 6.
Internal: 1 4
Internal: 1 3
Internal: 4 8
Leaf: 1 2
Leaf: 3
Leaf: 4 6
Leaf: 8 10

Inserting 9.
Internal: 1 4
Internal: 1 3
Internal: 4 8 9
Leaf: 1 2
Leaf: 3
Leaf: 4 6
Leaf: 8
Leaf: 9 10

Inserting 11.
Internal: 1 4
Internal: 1 3
Internal: 4 8 10
Leaf: 1 2
Leaf: 3
Leaf: 4 6
Leaf: 8 9
Leaf: 10 11

Inserting 12.
Internal: 1 8
Internal: 1 3 4
Internal: 8 10 11
Leaf: 1 2
Leaf: 3
Leaf: 4 6
Leaf: 8 9
Leaf: 10
Leaf: 11 12

Inserting 5.
Internal: 1 8
Internal: 1 3 5
Internal: 8 10 11
Leaf: 1 2
Leaf: 3 4
Leaf: 5 6
Leaf: 8 9
Leaf: 10
Leaf: 11 12

Inserting 7.
Internal: 1 5 8
Internal: 1 3
Internal: 5 6
Internal: 8 10 11
Leaf: 1 2
Leaf: 3 4
Leaf: 5
Leaf: 6 7
Leaf: 8 9
Leaf: 10
Leaf: 11 12

Internal: 1 5 8
Internal: 1 3
Internal: 5 6
Internal: 8 10 11
Leaf: 1 2
Leaf: 3 4
Leaf: 5
Leaf: 6 7
Leaf: 8 9
Leaf: 10
Leaf: 11 12
[ssdavis@lect1 p2]$


$ BTree2 BTree12.dat 3 2
Internal: 1 5 8
Internal: 1 3
Internal: 5 6
Internal: 8 10 11
Leaf: 1 2
Leaf: 3 4
Leaf: 5
Leaf: 6 7
Leaf: 8 9
Leaf: 10
Leaf: 11 12
[ssdavis@lect1 p2]$


$ BTree BTree25.dat 4 3
Inserting 24.
Leaf: 24

Inserting 53.
Leaf: 24 53

Inserting 10.
Leaf: 10 24 53

Inserting 67.
Internal: 10 53
Leaf: 10 24
Leaf: 53 67

Inserting 54.
Internal: 10 53
Leaf: 10 24
Leaf: 53 54 67

Inserting 27.
Internal: 10 53
Leaf: 10 24 27
Leaf: 53 54 67

Inserting 69.
Internal: 10 53 67
Leaf: 10 24 27
Leaf: 53 54
Leaf: 67 69

Inserting 30.
Internal: 10 30 67
Leaf: 10 24 27
Leaf: 30 53 54
Leaf: 67 69

Inserting 56.
Internal: 10 30 56
Leaf: 10 24 27
Leaf: 30 53 54
Leaf: 56 67 69

Inserting 80.
Internal: 10 30 56 69
Leaf: 10 24 27
Leaf: 30 53 54
Leaf: 56 67
Leaf: 69 80

Inserting 81.
Internal: 10 30 56 69
Leaf: 10 24 27
Leaf: 30 53 54
Leaf: 56 67
Leaf: 69 80 81

Inserting 37.
Internal: 10 30 54 69
Leaf: 10 24 27
Leaf: 30 37 53
Leaf: 54 56 67
Leaf: 69 80 81
```

```
Inserting 12.
Internal: 10 30
Internal: 10 24
Internal: 30 54 69
Leaf: 10 12
Leaf: 24 27
Leaf: 30 37 53
Leaf: 54 56 67
Leaf: 69 80 81

Inserting 8.
Internal: 8 30
Internal: 8 24
Internal: 30 54 69
Leaf: 8 10 12
Leaf: 24 27
Leaf: 30 37 53
Leaf: 54 56 67
Leaf: 69 80 81

Inserting 22.
Internal: 8 30
Internal: 8 22
Internal: 30 54 69
Leaf: 8 10 12
Leaf: 22 24 27
Leaf: 30 37 53
Leaf: 54 56 67
Leaf: 69 80 81

Inserting 47.
Internal: 8 30
Internal: 8 22
Internal: 30 47 54 69
Leaf: 8 10 12
Leaf: 22 24 27
Leaf: 30 37
Leaf: 47 53
Leaf: 54 56 67
Leaf: 69 80 81

Inserting 57.
Internal: 8 30
Internal: 8 22
Internal: 30 47 56 69
Leaf: 8 10 12
Leaf: 22 24 27
Leaf: 30 37
Leaf: 47 53 54
Leaf: 56 57 67
Leaf: 69 80 81

Inserting 40.
Internal: 8 30
Internal: 8 22
Internal: 30 47 56 69
Leaf: 8 10 12
Leaf: 22 24 27
Leaf: 30 37 40
Leaf: 47 53 54
Leaf: 56 57 67
Leaf: 69 80 81

Inserting 18.
Internal: 8 30
Internal: 8 12 22
Internal: 30 47 56 69
Leaf: 8 10
Leaf: 12 18
Leaf: 22 24 27
Leaf: 30 37 40
Leaf: 47 53 54
Leaf: 56 57 67
Leaf: 69 80 81

Inserting 44.
Internal: 8 40
Internal: 8 12 22 30
Internal: 40 47 56 69
Leaf: 8 10
Leaf: 12 18
Leaf: 22 24 27
Leaf: 30 37
Leaf: 40 44
Leaf: 47 53 54
Leaf: 56 57 67
Leaf: 69 80 81

Inserting 65.
Internal: 8 40 56
Internal: 8 12 22 30
Internal: 40 47
Internal: 56 65 69
Leaf: 8 10
Leaf: 12 18
Leaf: 22 24 27
Leaf: 30 37
Leaf: 40 44
Leaf: 47 53 54
Leaf: 56 57
Leaf: 65 67
Leaf: 69 80 81

Inserting 35.
Internal: 8 40 56
Internal: 8 12 22 30
Internal: 40 47
Internal: 56 65 69
Leaf: 8 10
Leaf: 12 18
Leaf: 22 24 27
Leaf: 30 35 37
Leaf: 40 44
Leaf: 47 53 54
Leaf: 56 57
Leaf: 65 67
Leaf: 69 80 81

Inserting 13.
Internal: 8 40 56
Internal: 8 12 22 30
Internal: 40 47
Internal: 56 65 69
Leaf: 8 10
Leaf: 12 13 18
Leaf: 22 24 27
Leaf: 30 35 37
Leaf: 40 44
Leaf: 47 53 54
Leaf: 56 57
Leaf: 65 67
Leaf: 69 80 81

Inserting 1.
Internal: 1 40 56
Internal: 1 12 22 30
Internal: 40 47
Internal: 56 65 69
Leaf: 1 8 10
Leaf: 12 13 18
Leaf: 22 24 27
Leaf: 30 35 37
Leaf: 40 44
Leaf: 47 53 54
Leaf: 56 57
Leaf: 65 67
Leaf: 69 80 81

Inserting 9.
Internal: 1 30 56
Internal: 1 9 12 22
Internal: 30 40 47
Internal: 56 65 69
Leaf: 1 8
Leaf: 9 10
Leaf: 12 13 18
Leaf: 22 24 27
Leaf: 30 35 37
Leaf: 40 44
Leaf: 47 53 54
Leaf: 56 57
Leaf: 65 67
Leaf: 69 80 81

Internal: 1 30 56
Internal: 1 9 12 22
Internal: 30 40 47
Internal: 56 65 69
Leaf: 1 8
Leaf: 9 10
Leaf: 12 13 18
Leaf: 22 24 27
Leaf: 30 35 37
Leaf: 40 44
Leaf: 47 53 54
Leaf: 56 57
Leaf: 65 67
Leaf: 69 80 81
[ssdavis@lect1 p2]$
```