**Project 2: Insertion Sort (Stacks) (C++)**

**CSC 323-32 Rafael Carmilema**

**Due Date: (10/20/2015)**

**Algorithm steps**:

Step 1: - open the input file;

- read and count the number of date item in the input file

- close the file

step 2: dynamically allocate the 1-D array of the size count+1

step 3: call buildHeap (see algorithm below)

step 4: call deleteHeap (see algorithm below)

buildHeap algorithm:

step 1: open the input file for the second time

step 2: data <-- get a data item from the input file

step 3: call insertOneDataItem

(inside of the insertOneDataItem, make sure the heap is not full, and call bubbleUp

step 4: call printHeap (prnit only the first 10 items of the heap to the output file)

// in the real life, this is only for your debugging purpose

step 5: repeat step 1 - 4 until the input file is empty

step 6: close the input file

deleteRoot algorithm:

step 1: open the output file

step 2: call deleteRoot

(inside of deleteRoot, make sure the heap is not empty,

print the root to the output file and call bubbleDown)

step 3: call printHeap (prnit only the first 10 items of the heap to the output file)

// in the real life, this is only for your debugging purpose

step 4: repeat step 1 and 3 untile the heap is empty

step 5: close the output file

#include<iostream>

#include<fstream>

#include<string>

using namespace std;

void insertOneDataItem(int a[],int x,int size);

void print(int a[],int x,std::ofstream &outputFile);

void bubbleUp(int a[]);

void bubbleDown(int a[]);

void buildHeap(std::ifstream &InputFile,std::ofstream &outputFile,int \*&a,int size);

void deleteRoot(std::ofstream &outputFile,int \*&a,int size);

void printDelete(int a[],int x,std::ofstream &outputFile);

bool isEmpty();

bool isFull(int a[],int size);

int count=0;

int i=1;

bool isEmpty(){

if(count==0)

return true;

return false;

}

bool isFull(int a[],int size){

if(a[0]==size-1)

return true;

return false;

}

void deleteRoot(std::ofstream &outputFile,int \*&a,int size){

int temp;

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

cout<<" "<<a[i]<<" ";

if(a[0]==0){

cout<<"heap is empty"<<endl;

return;

}

outputFile<<endl;

while(a[0]!=0){

//cout<<"delete "<<a[1]<<endl;

printDelete(a,a[1],outputFile);

temp=a[1];

a[1]=a[a[0]];

a[a[0]]=temp;

a[0]--;

bubbleDown(a);

}

cout<<endl;

for(int i=count;i>=1;i--)

cout<<" "<<a[i]<<" ";

outputFile<<"Final Heap"<<endl;

for(int i=count;i>=1;i--)

outputFile<<" "<<a[i]<<" ";

outputFile.close();

//for(int i=1;i<=count;i++)

// cout<<" "<<a[i]<<" ";

}

void buildHeap(std::ifstream &InputFile,std::ofstream &outputFile,int \*&a,int size)

{

//ifstream the\_file ( InputFile );

int x;

//int count=0;

outputFile<<"size "<<size<<endl;

outputFile<<"count "<<count<<endl;

while ( InputFile>>x){

++count;

insertOneDataItem(a,x,size);

print(a,x,outputFile);

}

outputFile<<"Final Heap"<<endl;

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

outputFile<<" "<<a[i]<<" ";

InputFile.close();

}

void insertOneDataItem(int a[],int x,int size){

if(isFull(a,size)){

cout<<"heap is full"<<endl;

return;

}

a[0]=count;

a[i]=x;

i++;

bubbleUp(a);

}

void bubbleUp(int a[]){

int child;

child=count;

int parent;

parent=child/2;

//left=parent\*2;

//right=parent\*2+1;

while(parent>=1&&a[parent]>=a[child]){

if(a[parent]>a[child]){

int temp;

temp=a[child];

a[child]=a[parent];

a[parent]=temp;

child=parent;

parent=parent/2;

}

}

}

void bubbleDown(int a[]){

cout<<endl;

//for(int i=1;i<=count;i++)

//cout<<" "<<a[i]<<" ";

int left;

int right;

int temp;

int min;

int root=1;

left=root\*2;

right=root\*2+1;

while(right<a[0]&&left<a[0]){

if(a[left]<a[right]&&right<a[0]){

min=left;

}

else if(a[left]>a[right]&&right<a[0] ){

min=right;

}

if(a[min]>a[root]){

return;

}

else{

temp=a[min];

a[min]=a[root];

a[root]=temp;

root=min;

left=root\*2;

right=root\*2+1;

}

}

//for(int i=1;i<=count;i++)

// cout<<" "<<a[i]<<" ";

}

void printDelete(int a[],int x,std::ofstream &outputFile){

int num=0;

outputFile<<"count "<<a[0]<<" delete "<<x<<" ";

for(int i=1;i<=a[0];i++){

if(num<10){

outputFile<<" "<<a[i]<<" ";

num++;

}

}

outputFile<<endl;

}

void print(int a[],int x,std::ofstream &outputFile){

int num=0;

outputFile<<"count "<<count<<" insert "<<x<<" ";

for(int i=1;i<=count;i++){

if(num<10){

outputFile<<" "<<a[i]<<" ";

num++;

}

}

outputFile<<endl;

}

int main ( int argc, char \*argv[] )

{

if ( argc < 2 ) // argc should be 2 for correct execution

// We print argv[0] assuming it is the program name

cout<<"usage: "<< argv[0] <<" <filename>\n";

else {

// We assume argv[1] is a filename to open

ifstream the\_file ( argv[1] );

// Always check to see if file opening succeeded

if ( !the\_file.is\_open() )

cout<<"Could not open file\n";

else {

int x;

int count=0;

int\* heap;

// the\_file.get ( x ) returns false if the end of the file

// is reached or an error occurs

while ( the\_file>>x){

count++;

//cout<<x<<endl;

}

the\_file.close();

heap= new int[count+1];

ifstream the\_file ( argv[1] );

ofstream output(argv[2]);

buildHeap(the\_file,output,heap,count+1);

deleteRoot(output,heap,count+1);

return 0;

}

}

}

**OUTPUT:**

**size 14**

**count 0**

**count 1 insert 10 10**

**count 2 insert 11 10 11**

**count 3 insert 19 10 11 19**

**count 4 insert 73 10 11 19 73**

**count 5 insert 33 10 11 19 73 33**

**count 6 insert 9 9 11 10 73 33 19**

**count 7 insert 63 9 11 10 73 33 19 63**

**count 8 insert 81 9 11 10 73 33 19 63 81**

**count 9 insert 99 9 11 10 73 33 19 63 81 99**

**count 10 insert 8 8 9 10 73 11 19 63 81 99 33**

**count 11 insert 29 8 9 10 73 11 19 63 81 99 33**

**count 12 insert 23 8 9 10 73 11 19 63 81 99 33**

**count 13 insert 18 8 9 10 73 11 18 63 81 99 33**

**Final Heap**

**8 9 10 73 11 18 63 81 99 33 29 23 19**

**count 13 delete 8 8 9 10 73 11 18 63 81 99 33**

**count 12 delete 9 9 11 10 73 19 18 63 81 99 33**

**count 11 delete 10 10 11 18 73 19 23 63 81 99 33**

**count 10 delete 11 11 19 18 73 29 23 63 81 99 33**

**count 9 delete 18 18 19 23 73 29 33 63 81 99**

**count 8 delete 19 19 29 23 73 99 33 63 81**

**count 7 delete 23 23 29 81 73 99 33 63**

**count 6 delete 29 29 63 81 73 99 33**

**count 5 delete 33 33 63 81 73 99**

**count 4 delete 63 63 99 81 73**

**count 3 delete 73 73 99 81**

**count 2 delete 81 81 99**

**count 1 delete 99 99**

**Final Heap**

**8 9 10 11 18 19 23 29 33 63 73 81 99**