Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное  
учреждение высшего образования

Новосибирский государственный технический университет

Кафедра Автоматизированных систем управления

**ОТЧЁТ**

**по лабораторной работе № 4**

**«B ДЕРЕВЬЯ»**

**по дисциплине: «Технология программирования»**

Выполнили: Проверил:

Студенты гр. «АВТ-813», «АВТФ» Лукоянычева Ольга Викторовна

Букова Анна

Ланг Татьяна

Пустовских Дмитрий

Новосибирск 2020

**Цель работы:** Познакомиться с иерархическими (древовидными) структурами данных, получить представление об использовании таких структур для размещения реальных данных, приобрести навыки программирования процедур доступа к данным.

**Содержание**

1. Описание программы…………………………………………………..…………. 3
   1. Общие сведения…………………………………………………..……………3
   2. Функциональное назначение……………………………………..………….. 3
   3. Описание логической структуры…………………………………..…………3

Классы………………………………………………………………….………3

Методы………………………………………………………………….……...3

Листинг…………………………………………………………………….….. 4

* 1. Вызов и загрузка……………………………………………………………...10
  2. Выходные данные……………………………………………………….……10

1. **Описание программы**
   1. **Общие сведения**

Данная программа должна моделировать работу B-дерева в файлах: первая запись файла является описанием структуры файла. Каждая вершина В - дерева, называемая страницей, содержит не менее n и не более 2n элементов, где n порядок В-дерева. Исключением из этого правила является корневая страница, на которой может находиться произвольное количество элементов (от 1 до 2n).

Программа использует язык С++.

* 1. **Функциональное назначение**

Данная программа решает задачу хранения и обработки информации и для ускорения доступа к данным, посредством хранения данных в B-дереве, в котором логическая связь между записями определяется с помощью указателей.

Файл, с которым осуществляется работа, задается пользователем.

* 1. **Описание логической структуры**

**Классы**

Программа использует объектно-ориентированный стиль написания. В рамках стиля и с целью выполнения поставленной задачи определяются следующие классы:

* Btree (работает с битовым файлом, представляет из себя B-дерево)

**Методы**

Для выполнения поставленной задачи должны определяться следующие методы:

* процедура включения в В-дерево insert;
* процедура удаления из В-дерева delete.
* процедура сжатия файла pack.

**Листинг**

#pragma once

#include <fstream>

#include <iostream>

#pragma warning(disable:4996)

using namespace std;

enum Access { New\_Tree, Existing\_Tree };

struct BeginFile

{

int m\_size\_page;

int m\_coutn\_page;

int m\_order\_tree;

int m\_ptr\_root\_page;

int m\_ptr\_last\_del\_page;

BeginFile()

{

m\_size\_page = 0;

m\_coutn\_page = 0;

m\_order\_tree = 0;

m\_ptr\_root\_page = 0;

m\_ptr\_last\_del\_page = 0;

}

};

template <typename T>

struct Data

{

int ptr;

T data;

Data()

{

ptr = 0;

data = T();

}

};

template <typename T>

class Btree

{

public:

Btree(const char\* name\_file, Access status = Existing\_Tree,int order = 4);

void insert(T data);

void remove(T data);

void pack();

void show();

private:

void delPage(int ptrPage);

Data<T> tossing(int ptrDonor,int ptrRecipient, Data<T> data);

void moves(int ptrDonor,int ptrRecipient);

void movesAll(int ptrDonor,int ptrRecipient);

void \_FILE\_WC(Access status, const char\* name\_file);

int addDataPage(int prtPage, Data<T> data);

void \_Load\_data(Access status, int order);

int ptrPort(int left, int reight);

void addStart(int ptr1, int ptr2);

void replaceData(T data, T data2);

void movesDel(int add,int del);

int findAncestor(int data);

int countAdd(int ptrPage);

int adjacent(int ptrNow);

int newPage(int prt = 0);

void newFile(int order);

void deleteElem(T data);

int findPage(T data);

T removeBeg(int pos);

int find(T data);

void existingFile();

int replace(T data);

bool left(int ptr,int ptrOther);

int countDel();

void updata();

bool isDelete(int prt);

int backPtr(int pos);

int nextPtr(int pos);

fstream\* m\_file;

BeginFile Start;

int m\_start\_tree;

};

template<typename T>

inline Btree<T>::Btree(const char\* name\_file, Access status, int order)

{

this->m\_start\_tree = sizeof(Start);

\_FILE\_WC(status, name\_file);

\_Load\_data(status, order);

}

template<typename T>

inline void Btree<T>::insert(T data)

{

Data<T> res;

res.data = data;

if (Start.m\_ptr\_root\_page == 0)

{

Start.m\_ptr\_root\_page = newPage();

addDataPage(Start.m\_ptr\_root\_page, res);

updata();

return;

}

if (find(data) != -1)

{

return;

}

int pageInsert, pageAncestor;

pageInsert = this->findPage(res.data);

while (true)

{

if (this->countAdd(pageInsert) == Start.m\_size\_page)

{

pageAncestor = this->findAncestor(pageInsert);

int ptrNew = newPage();

res = tossing(pageInsert, ptrNew, res);

addStart(res.ptr, ptrNew);

res.ptr = ptrNew;

if (pageAncestor == 0)

{

int ptrNew2 = newPage(pageInsert);

Start.m\_ptr\_root\_page = ptrNew2;

addDataPage(ptrNew2, res);

break;

}

pageInsert = pageAncestor;

}

else

{

addDataPage(pageInsert, res);

break;

}

}

updata();

}

template<typename T>

inline void Btree<T>::remove(T data)

{

int posDel = replace(data), posAbj = 0;

int size;

if (posDel != 0)

{

T dataRe = removeBeg(posDel);

replaceData(data, dataRe);

}

else

{

posDel = find(data);

deleteElem(data);

}

while (true)

{

size = countAdd(posDel);

if (posDel == Start.m\_ptr\_root\_page)

{

if (countAdd(posDel) == 0)

{

delPage(posDel);

Start.m\_ptr\_root\_page = posAbj;

break;

}

else

break;

}

if (size < Start.m\_order\_tree)

{

posAbj = adjacent(posDel);

if (countAdd(posAbj) > Start.m\_order\_tree)

{

moves(posAbj, posDel);

}

else

{

if (!left(posAbj, posDel))

swap(posAbj, posDel);

movesAll(posDel, posAbj);

delPage(posDel);

}

}

int newtPotr = findAncestor(posAbj);

if (countAdd(newtPotr) >= Start.m\_order\_tree)

break;

posDel = newtPotr;

}

updata();

}

template<typename T>

inline void Btree<T>::pack()

{

int ptr, count, ptrNow, delNext;

Data<T> dat;

this->m\_file->seekg(0,ios::end);

int totolSize = this->m\_file->tellg();

int sizePage = sizeof(int) + sizeof(int) + Start.m\_size\_page \*sizeof(Data<T>);

int countDl = countDel();

int first = Start.m\_ptr\_last\_del\_page;

for (int i = 0; i < countDl; ++i)

{

int nowStat = totolSize - sizePage \* (countDl - i);

int back = backPtr(first);

if (!isDelete(nowStat))

{

int posRe = findAncestor(nowStat);

if (posRe == 0)

{

Start.m\_ptr\_root\_page = first;

}

else

{

this->m\_file->seekg(posRe, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

this->m\_file->read((char\*)&count, sizeof(count));

if (nowStat == ptr)

{

dat.ptr = first;

this->m\_file->seekg(posRe, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

count = 0;

}

for (int i = 0; i < count; ++i)

{

ptrNow = m\_file->tellg();

this->m\_file->read((char\*)&dat, sizeof(dat));

if (dat.ptr == nowStat)

{

dat.ptr = first;

this->m\_file->seekg(ptrNow, ios::beg);

this->m\_file->write((char\*)&dat, sizeof(dat));

break;

}

}

}

}

else

{

int back2 = backPtr(nowStat);

if ( nextPtr(first) == nowStat)

{

addStart(first, first);

}

addStart(first, back2);

}

if(first != nowStat)

movesDel(nowStat, first);

if (i == 0)

{

Start.m\_ptr\_last\_del\_page = nowStat;

}

else

{

addStart(nowStat, back);

}

first = nextPtr(nowStat);

}

updata();

}

template<typename T>

inline T Btree<T>::removeBeg(int pos)

{

int prtInsert, ptr, count, ptrDel;

Data<T> dat, Del;

this->m\_file->seekg(pos, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

this->m\_file->read((char\*)&count, sizeof(count));

prtInsert = this->m\_file->tellg();

this->m\_file->read((char\*)&Del, sizeof(Del));

ptrDel = this->m\_file->tellg();

for (int i = 0; i < count-1; ++i)

{

this->m\_file->seekg(ptrDel, ios::beg);

this->m\_file->read((char\*)&dat, sizeof(dat));

ptrDel = this->m\_file->tellg();;

this->m\_file->seekg(prtInsert, ios::beg);

this->m\_file->write((char\*)&dat, sizeof(dat));

prtInsert = this->m\_file->tellg();;

}

count--;

this->m\_file->seekg(pos, ios::beg);

this->m\_file->write((char\*)&ptr, sizeof(ptr));

this->m\_file->write((char\*)&count, sizeof(count));

return Del.data;

}

template<typename T>

inline void Btree<T>::replaceData(T data, T data2)

{

int ptrPage, prtInsert, ptr, count,ptrNow;

Data<T> dat;

ptrPage = Start.m\_ptr\_root\_page;

while (true)

{

this->m\_file->seekg(ptrPage, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

this->m\_file->read((char\*)&count, sizeof(count));

prtInsert = ptr;

for (int i = 0; i < count; ++i)

{

ptrNow = this->m\_file->tellg();

this->m\_file->read((char\*)&dat, sizeof(dat));

if (data == dat.data)

{

dat.data = data2;

this->m\_file->seekg(ptrNow, ios::beg);

this->m\_file->write((char\*)&dat, sizeof(dat));

return;

}

if (data <= dat.data)

break;

prtInsert = dat.ptr;

}

ptrPage = prtInsert;

}

}

template<typename T>

inline int Btree<T>::replace(T data)

{

int ptrPage, ptr, count;

int pos = find(data);

Data<T> dat;

if (pos == -1)

{

throw new std::exception("Error, no data");

}

this->m\_file->seekg(pos, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

this->m\_file->read((char\*)&count, sizeof(count));

for (int i = 0; i < count; ++i)

{

this->m\_file->read((char\*)&dat, sizeof(dat));

ptrPage = dat.ptr;

if (data == dat.data)

break;

}

while (ptrPage != 0)

{

this->m\_file->seekg(ptrPage, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

if (ptr == 0)

break;

ptrPage = ptr;

}

return ptrPage;

}

template<typename T>

inline bool Btree<T>::left(int ptr,int ptrOther)

{

int prePtr = findAncestor(ptr), pre1, pre2, count;

Data<T> dat;

this->m\_file->seekg(prePtr, ios::beg);

this->m\_file->read((char\*)&pre1, sizeof(pre1));

this->m\_file->read((char\*)&count, sizeof(count));

for (int i = 0; i < count; ++i)

{

this->m\_file->read((char\*)&dat, sizeof(dat));

pre2 = dat.ptr;

if (ptr == pre2 && ptrOther == pre1)

return false ;

if (ptr == pre1 && ptrOther == pre2)

return true;

pre1 = pre2;

}

}

template<typename T>

inline int Btree<T>::countDel()

{

int count = 0;

int first = Start.m\_ptr\_last\_del\_page;

while (first != 0)

{

this->m\_file->seekg(first, ios::beg);

this->m\_file->read((char\*)&first,sizeof(first));

count++;

}

return count;

}

template<typename T>

inline int Btree<T>::find(T data)

{

int ptrPage, prtInsert, ptr, count;

Data<T> dat;

ptrPage = Start.m\_ptr\_root\_page;

while (true)

{

this->m\_file->seekg(ptrPage, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

this->m\_file->read((char\*)&count, sizeof(count));

prtInsert = ptr;

for (int i = 0; i < count; ++i)

{

this->m\_file->read((char\*)&dat, sizeof(dat));

if (data == dat.data)

return ptrPage;

if (data <= dat.data)

break;

prtInsert = dat.ptr;

}

if (prtInsert == 0)

{

return -1;

}

ptrPage = prtInsert;

}

}

template<typename T>

inline int Btree<T>::adjacent(int ptrNow)

{

int prePtr = findAncestor(ptrNow),pre1, pre2, count;

Data<T> dat;

this->m\_file->seekg(prePtr, ios::beg);

this->m\_file->read((char\*)&pre1, sizeof(pre1));

this->m\_file->read((char\*)&count, sizeof(count));

for (int i = 0; i < count; ++i)

{

this->m\_file->read((char\*)&dat, sizeof(dat));

pre2 = dat.ptr;

if (ptrNow == pre2)

return pre1;

if (ptrNow == pre1)

return pre2;

pre1 = pre2;

}

}

template<typename T>

inline int Btree<T>::newPage(int prt)

{

int ptr = prt,count = 0,ptrPage;

Data<T> dat;

if (this->Start.m\_ptr\_last\_del\_page == 0)

{

this->Start.m\_coutn\_page++;

this->m\_file->seekg(0, ios::end);

}

else

{

this->m\_file->seekg(this->Start.m\_ptr\_last\_del\_page, ios::beg);

this->m\_file->read((char\*)&ptrPage, sizeof(ptrPage));

this->m\_file->seekg(this->Start.m\_ptr\_last\_del\_page, ios::beg);

Start.m\_ptr\_last\_del\_page = ptrPage;

}

ptrPage = this->m\_file->tellg();

this->m\_file->write((char\*)&ptr,sizeof(ptr));

this->m\_file->write((char\*)&count, sizeof(count));

for (int i = 0; i < Start.m\_size\_page; ++i)

{

this->m\_file->write((char\*)&dat, sizeof(dat));

}

this->m\_file->seekg(0, ios::beg);

updata();

return ptrPage;

}

template<typename T>

inline void Btree<T>::show()

{

int ptr, sizepage, count, temp, ptrNow;

BeginFile Fil;

Data<T> dat;

this->m\_file->seekg(0, ios::beg);

m\_file->read((char\*)&Fil, sizeof(Fil));

sizepage = sizeof(int) + sizeof(int) + sizeof(dat) \* Fil.m\_size\_page;

cout << Fil.m\_order\_tree << " ";

cout << Fil.m\_coutn\_page << " ";

cout << Fil.m\_ptr\_last\_del\_page << " ";

cout << Fil.m\_ptr\_root\_page << " ";

cout << Fil.m\_size\_page << " ";

cout << endl;

temp = 0;

ptrNow = this->m\_file->tellg();

while (this->Start.m\_coutn\_page > temp)

{

cout << ptrNow << " :: ";

this->m\_file->seekg(ptrNow,ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

this->m\_file->read((char\*)&count, sizeof(count));

cout << ptr << " ";

cout << count << " : ";

for (int i = 0; i < count; ++i)

{

this->m\_file->read((char\*)&dat, sizeof(dat));

cout << "d="<<dat.data << " ";

cout << "p="<<dat.ptr << " /";

}

temp++;

ptrNow += sizepage;

cout << endl;

}

cout << endl;

}

template<typename T>

inline void Btree<T>::\_Load\_data(Access status, int order)

{

switch (status)

{

case New\_Tree:

{

newFile(order);

break;

}

case Existing\_Tree:

{

existingFile();

break;

}

}

}

template<typename T>

inline int Btree<T>::ptrPort(int left, int reight)

{

int prePtr = findAncestor(left), pre1, pre2, count,ptrElem;

Data<T> dat;

this->m\_file->seekg(prePtr, ios::beg);

this->m\_file->read((char\*)&pre1, sizeof(pre1));

this->m\_file->read((char\*)&count, sizeof(count));

for (int i = 0; i < count; ++i)

{

ptrElem = m\_file->tellg();

this->m\_file->read((char\*)&dat, sizeof(dat));

pre2 = dat.ptr;

if (left == pre2 && reight == pre1 || left == pre1 && reight == pre2)

return ptrElem;

pre1 = pre2;

}

}

template<typename T>

inline void Btree<T>::existingFile()

{

m\_file->read((char\*)&Start, sizeof(Start));

}

template<typename T>

inline int Btree<T>::findPage(T data)

{

int ptrPage, prtInsert,ptr, count;

Data<T> dat;

ptrPage = Start.m\_ptr\_root\_page;

while (true)

{

this->m\_file->seekg(ptrPage,ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

this->m\_file->read((char\*)&count, sizeof(count));

prtInsert = ptr;

for (int i = 0; i < count; ++i)

{

this->m\_file->read((char\*)&dat, sizeof(dat));

if (data < dat.data)

{

break;

}

prtInsert = dat.ptr;

}

if (prtInsert == 0)

{

break;

}

ptrPage = prtInsert;

}

return ptrPage;

}

template<typename T>

inline void Btree<T>::delPage(int ptrPage)

{

this->m\_file->seekg(ptrPage,ios::beg);

this->m\_file->write((char\*)&Start.m\_ptr\_last\_del\_page, sizeof(Start.m\_ptr\_last\_del\_page));

Start.m\_ptr\_last\_del\_page = ptrPage;

updata();

}

template<typename T>

inline Data<T> Btree<T>::tossing(int ptrDonor, int ptrRecipient, Data<T> data)

{

int ptr1, count1, ptr2, count2, prtInsert, prtInsert2;

Data<T> dat, dataNew,average;

dataNew = data;

this->m\_file->seekg(ptrDonor, ios::beg);

this->m\_file->read((char\*)&ptr1, sizeof(ptr1));

this->m\_file->read((char\*)&count1, sizeof(count1));

prtInsert = m\_file->tellg();

this->m\_file->seekg(ptrRecipient, ios::beg);

this->m\_file->read((char\*)&ptr2, sizeof(ptr2));

this->m\_file->read((char\*)&count2, sizeof(count2));

prtInsert2 = m\_file->tellg();

for (int i = 0; i < count1/2 + 1; ++i)

{

this->m\_file->seekg(prtInsert, ios::beg);

this->m\_file->read((char\*)&dat, sizeof(dat));

if (dataNew.data < dat.data)

{

this->m\_file->seekg(prtInsert, ios::beg);

this->m\_file->write((char\*)&dataNew, sizeof(dataNew));

average = dataNew;

dataNew = dat;

}

else

{

average = dat;

}

prtInsert = m\_file->tellg();

}

for (int i = 0; i < count1 / 2 - 1; ++i)

{

this->m\_file->seekg(prtInsert, ios::beg);

this->m\_file->read((char\*)&dat, sizeof(dat));

prtInsert = m\_file->tellg();

if (dataNew.data < dat.data)

{

this->m\_file->seekg(prtInsert2, ios::beg);

this->m\_file->write((char\*)&dataNew, sizeof(dataNew));

dataNew = dat;

}

else

{

this->m\_file->seekg(prtInsert2, ios::beg);

this->m\_file->write((char\*)&dat, sizeof(dat));

}

prtInsert2 = m\_file->tellg();

}

this->m\_file->seekg(prtInsert2, ios::beg);

this->m\_file->write((char\*)&dataNew, sizeof(dataNew));

count1 /= 2;

this->m\_file->seekg(ptrDonor, ios::beg);

this->m\_file->write((char\*)&ptr1, sizeof(ptr1));

this->m\_file->write((char\*)&count1, sizeof(count1));

this->m\_file->seekg(ptrRecipient, ios::beg);

this->m\_file->write((char\*)&ptr2, sizeof(ptr2));

this->m\_file->write((char\*)&count1, sizeof(count1));

return average;

}

template<typename T>

inline void Btree<T>::moves(int ptrDonor, int ptrRecipient)

{

int ptr,count,posNext;

int count1 = countAdd(ptrDonor);

int count2 = countAdd(ptrRecipient);

int ptrPor = ptrPort(ptrDonor, ptrRecipient);

int move = (count1 + count2 + 1) / 2 - count2;

Data<T> dat, dat2;

for (int i = 0; i < move; ++i)

{

m\_file->seekg(ptrPor,ios::beg);

this->m\_file->read((char\*)&dat, sizeof(dat));

m\_file->seekg(ptrRecipient, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

dat.ptr = ptr;

addDataPage(ptrRecipient, dat);

count2++;

count1--;

posNext = ptrDonor + sizeof(int) + sizeof(int) + count1 \* sizeof(dat);

m\_file->seekg(posNext, ios::beg);

this->m\_file->read((char\*)&dat2, sizeof(dat2));

m\_file->seekg(ptrPor, ios::beg);

this->m\_file->read((char\*)&dat, sizeof(dat));

dat.data = dat2.data;

m\_file->seekg(ptrPor, ios::beg);

this->m\_file->write((char\*)&dat, sizeof(dat));

m\_file->seekg(ptrRecipient, ios::beg);

this->m\_file->write((char\*)&dat2.ptr, sizeof(dat2.ptr));

}

m\_file->seekg(ptrDonor+ sizeof(int), ios::beg);

this->m\_file->write((char\*)&count1, sizeof(count2));

m\_file->seekg(ptrRecipient+sizeof(int), ios::beg);

this->m\_file->write((char\*)&count2, sizeof(count2));

}

template<typename T>

inline void Btree<T>::movesAll(int ptrDonor, int ptrRecipient)

{

int ptr, count, posNext;

int count1 = countAdd(ptrDonor);

int count2 = countAdd(ptrRecipient);

int ptrPor = ptrPort(ptrDonor, ptrRecipient);

int move = count1;

Data<T> dat, dat2;

m\_file->seekg(ptrPor, ios::beg);

this->m\_file->read((char\*)&dat, sizeof(dat));

m\_file->seekg(ptrDonor, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

dat.ptr = ptr;

for (int i = 0; i < move; ++i)

{

count1--;

posNext = ptrDonor + sizeof(int) + sizeof(int) + count1 \* sizeof(dat);

m\_file->seekg(posNext, ios::beg);

this->m\_file->read((char\*)&dat2, sizeof(dat2));

addDataPage(ptrRecipient, dat2);

count2++;

}

addDataPage(ptrRecipient, dat);

count2++;

deleteElem(dat.data);

m\_file->seekg(ptrDonor + sizeof(int), ios::beg);

this->m\_file->write((char\*)&count1, sizeof(count2));

m\_file->seekg(ptrRecipient + sizeof(int), ios::beg);

this->m\_file->write((char\*)&count2, sizeof(count2));

}

template<typename T>

inline int Btree<T>::findAncestor(int data)

{

int ptrPage, prtInsert, ptr, count, ptrAncestor;

Data<T> dat,datF;

ptrPage = Start.m\_ptr\_root\_page;

ptrAncestor = 0;

this->m\_file->seekg(data, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

this->m\_file->read((char\*)&count, sizeof(count));

this->m\_file->read((char\*)&datF, sizeof(datF));

while (true)

{

this->m\_file->seekg(ptrPage, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

this->m\_file->read((char\*)&count, sizeof(count));

prtInsert = ptr;

for (int i = 0; i < count; ++i)

{

this->m\_file->read((char\*)&dat, sizeof(dat));

if(datF.data == dat.data)

return ptrAncestor;

if (datF.data <= dat.data)

break;

prtInsert = dat.ptr;

}

ptrAncestor = ptrPage;

ptrPage = prtInsert;

}

}

template<typename T>

inline void Btree<T>::newFile(int order)

{

Start.m\_size\_page = order\*2;

Start.m\_order\_tree = order;

m\_file->write((char\*)&Start, sizeof(Start));

}

template<typename T>

inline int Btree<T>::addDataPage(int prtPage, Data<T> data)

{

int ptr, count,prtInsert;

Data<T> dat,dataNew;

dataNew = data;

this->m\_file->seekg(prtPage, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

this->m\_file->read((char\*)&count, sizeof(count));

prtInsert = m\_file->tellg();

for (int i = 0; i < count; ++i)

{

this->m\_file->seekg(prtInsert, ios::beg);

this->m\_file->read((char\*)&dat, sizeof(dat));

if (dataNew.data < dat.data)

{

this->m\_file->seekg(prtInsert, ios::beg);

this->m\_file->write((char\*)&dataNew, sizeof(dataNew));

dataNew = dat;

}

prtInsert = m\_file->tellg();

}

count++;

this->m\_file->seekg(prtInsert, ios::beg);

this->m\_file->write((char\*)&dataNew, sizeof(dataNew));

this->m\_file->seekg(prtPage, ios::beg);

this->m\_file->write((char\*)&ptr, sizeof(ptr));

this->m\_file->write((char\*)&count, sizeof(count));

return 1;

}

template<typename T>

inline int Btree<T>::countAdd(int ptrPage)

{

int ptr, count;

this->m\_file->seekg(ptrPage, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

this->m\_file->read((char\*)&count, sizeof(count));

this->m\_file->seekg(0, ios::beg);

return count;

}

template<typename T>

inline void Btree<T>::updata()

{

m\_file->seekp(0, ios::beg);

m\_file->write((char\*)&Start, sizeof(Start));

}

template<typename T>

inline bool Btree<T>::isDelete(int prt)

{

int first = Start.m\_ptr\_last\_del\_page;

while (first != 0)

{

if (prt == first)

return true;

this->m\_file->seekg(first, ios::beg);

this->m\_file->read((char\*)&first, sizeof(first));

}

return false;

}

template<typename T>

inline int Btree<T>::backPtr(int pos)

{

int first = Start.m\_ptr\_last\_del\_page;

if (pos == first)

{

return 0;

}

while (first != 0)

{

if (pos == nextPtr(first))

return first;

first = nextPtr(first);

}

}

template<typename T>

inline int Btree<T>::nextPtr(int pos)

{

this->m\_file->seekg(pos, ios::beg);

this->m\_file->read((char\*)&pos, sizeof(pos));

return pos;

}

template<typename T>

inline void Btree<T>::movesDel(int add, int del)

{

int ptr1, count1, posNext1;

int ptr2, count2, posNext2;

count1 = countAdd(add);

count2 = countAdd(del);

Data<T> dat, dat2;

m\_file->seekg(add, ios::beg);

this->m\_file->read((char\*)&ptr1, sizeof(ptr1));

this->m\_file->read((char\*)&count1, sizeof(count1));

posNext1 = this->m\_file->tellg();

m\_file->seekg(del, ios::beg);

this->m\_file->read((char\*)&ptr2, sizeof(ptr2));

this->m\_file->read((char\*)&count2, sizeof(count2));

posNext2 = this->m\_file->tellg();

for (int i = 0; i < count1; ++i)

{

m\_file->seekg(posNext1, ios::beg);

this->m\_file->read((char\*)&dat2, sizeof(dat2));

posNext1 = this->m\_file->tellg();

m\_file->seekg(posNext2, ios::beg);

this->m\_file->write((char\*)&dat2, sizeof(dat2));

posNext2 = this->m\_file->tellg();

}

m\_file->seekg(del, ios::beg);

this->m\_file->write((char\*)&ptr1, sizeof(ptr1));

this->m\_file->write((char\*)&count1, sizeof(count1));

m\_file->seekg(add, ios::beg);

this->m\_file->write((char\*)&ptr2, sizeof(ptr2));

this->m\_file->write((char\*)&count2, sizeof(count2));

}

template<typename T>

inline void Btree<T>::deleteElem(T data)

{

int prtInsert, ptr, count, ptrDel ,i;

int ptrD = find(data);

Data<T> dat,dat2;

this->m\_file->seekg(ptrD, ios::beg);

this->m\_file->read((char\*)&ptr, sizeof(ptr));

this->m\_file->read((char\*)&count, sizeof(count));

for ( i = 0; i < count - 1; ++i)

{

prtInsert = this->m\_file->tellg();

this->m\_file->read((char\*)&dat, sizeof(dat));

ptrDel = this->m\_file->tellg();

if (dat.data >= data)

{

this->m\_file->read((char\*)&dat2, sizeof(dat2));

this->m\_file->seekg(prtInsert,ios::beg);

this->m\_file->write((char\*)&dat2, sizeof(dat2));

this->m\_file->seekg(ptrDel,ios::beg);

}

}

count--;

this->m\_file->seekg(ptrD, ios::beg);

this->m\_file->write((char\*)&ptr, sizeof(ptr));

this->m\_file->write((char\*)&count, sizeof(count));

}

template<typename T>

inline void Btree<T>::addStart(int ptr1, int ptr2)

{

this->m\_file->seekg(ptr2, ios::beg);

this->m\_file->write((char\*)&ptr1, sizeof(ptr1));

}

template<typename T>

inline void Btree<T>::\_FILE\_WC(Access status, const char\* name\_file)

{

if (status == New\_Tree)

{

std::ofstream oFile(name\_file);

oFile.close();

}

m\_file = new fstream(name\_file, ios::binary | ios::in | ios::out);

m\_file->seekp(0, ios::beg);

if (!m\_file->is\_open())

{

throw new std::exception("Error, file cann't be opened");

}

}}

* 1. **Вызов и загрузка**

Работа программы начинается с исполнения метода «main».

Вызов к исполнению программы происходит из интегрированной среды разработки.

* 1. **Выходные данные**

Программа не возвращает значений, после завершения работы формируется бинарный файл с результатами работы – записями заданных чисел.