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 № 2**

**«Организация списковой структуры»**

**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 организацией данных в виде списка, получить представление о совместном хранении данных и описаний, приобрести навык 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 односвязного списка, организованного в файлах: первая запись файла является описанием структуры файла.

Программа использует язык С++.

* 1. **Функциональное назначение**

Данная программа решает задачу хранения и обработки информации посредством списковой структуры, в которых 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:

* Flist (работает с битовым файлом, представляет из себя односвязный список)

**Методы**

Для выполнения поставленной задачи должны определяться следующие методы:

* процедура включения записи в список insert;
* процедура удаления записи из списка delete.
* процедура поиска записи find;
* процедура сжатия файла pack.

**Листинг**

#pragma once

#include <iostream>

#include <fstream>

using namespace std;

enum **Access** {New\_List,Existing\_List};

template <class T>

class Flist

{

public:

**Flist**(const char \* name\_file, Access status = New\_List);

void **remove**(T data);

void **insert**(T data);

void **pack**();

void **show**();

~**Flist**();

private:

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

int **backPtr**(int pos,bool status = false);

bool **change**(int pos,int ptrDel);

void **\_Load\_data**(Access status);

void **checkTit**(int pos1, int pos2);

void **swap**(int pos1,int pos2);

void **skipItem**(int pos);

void **deletion**(int pos);

int **countActiv**();

int **countDel**();

void **sortDel**();

void **sortActiv**();

int **nextPtr**(int pos);

void **existingFile**();

int **find**(T data);

void **newFile**();

void **updata**();

fstream\* m\_file;

int m\_count\_app;

int m\_start\_list;

int m\_ptr\_last\_app;

int m\_ptr\_first\_app;

int m\_ptr\_last\_del\_app;

};

template<class T>

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

{

this->m\_start\_list = sizeof(m\_count\_app) + sizeof(m\_ptr\_first\_app) + sizeof(m\_ptr\_last\_del\_app)+ sizeof(m\_ptr\_last\_app);

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

\_Load\_data(status);

}

template<class T>

inline void Flist<T>::**pack**()

{

sortDel();

sortActiv();

}

template<class T>

inline void Flist<T>::**remove**(T data)

{

int pos = find(data), ptrDel;

if (pos == -1)

{

throw new out\_of\_range("Not, Data");

}

if (pos == this->m\_ptr\_last\_app && pos != this->m\_ptr\_first\_app)

ptrDel = backPtr(pos);

else

ptrDel = nextPtr(pos);

if(pos != this->m\_ptr\_first\_app)

skipItem(pos);

deletion(pos);

change(pos,ptrDel);

updata();

}

template<class T>

inline void Flist<T>::**insert**(T data)

{

bool deleteBit = 0;

int ptr = 0;

if (m\_ptr\_last\_del\_app == 0)

{

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

m\_count\_app++;

}

else

{

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

m\_ptr\_last\_del\_app = this->nextPtr(m\_ptr\_last\_del\_app);

}

int newEnd = m\_file->tellg();

if (this->m\_ptr\_first\_app == 0)

{

this->m\_ptr\_first\_app = newEnd;

}

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

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

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

if (m\_ptr\_last\_app != 0)

{

this->m\_file->seekg(m\_ptr\_last\_app + sizeof(deleteBit), ios::beg);

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

}

m\_ptr\_last\_app = newEnd;

updata();

}

template<class T>

inline void Flist<T>::**show**()

{

int temp;

bool bo;

int temp2 = 1;

T temp3;

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

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

cout << temp<< " ";

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

cout << temp << " ";

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

cout << temp << " ";

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

cout << temp << " ";

cout << endl;

temp = 0;

while (this->m\_count\_app > temp)

{

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

cout << bo << " ";

this->m\_file->read((char\*)&temp2, sizeof(temp2));

cout << temp2 << " ";

this->m\_file->read((char\*)&temp3, sizeof(temp3));

cout << temp3 << " ";

cout << endl;

temp++;

}

cout << endl;

}

template<class T>

inline Flist<T>::~**Flist**()

{

m\_file->close();

delete m\_file;

}

template<class T>

inline int Flist<T>::**find**(T data)

{

T dat;

bool flag;

int ptrOld = 0, ptrNew = 0;

this->m\_file->seekg(this->m\_ptr\_first\_app, ios::beg);

while (true)

{

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

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

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

if (dat == data)

{

if (ptrOld == 0)

{

ptrOld = this->m\_ptr\_first\_app;

}

return ptrOld;

}

if (ptrNew == 0)

{

return -1;

}

ptrOld = ptrNew;

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

}

}

template<class T>

inline bool Flist<T>::**change**(int pos,int ptrDel)

{

bool t = false;

if (pos == this->m\_ptr\_first\_app)

{

this->m\_ptr\_first\_app = ptrDel;

t = true;

}

if (pos == this->m\_ptr\_last\_app)

{

this->m\_ptr\_last\_app = ptrDel;

t = true;

}

return t;

}

template<class T>

inline void Flist<T>::**updata**()

{

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

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

m\_file->write((char\*)&m\_ptr\_first\_app, sizeof(m\_ptr\_first\_app));

m\_file->write((char\*)&m\_ptr\_last\_app, sizeof(m\_ptr\_last\_app));

m\_file->write((char\*)&m\_ptr\_last\_del\_app, sizeof(m\_ptr\_last\_del\_app));

}

template<class T>

inline int Flist<T>::**nextPtr**(int pos)

{

int nowPos = m\_file->tellg(), ptrNext;

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

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

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

return ptrNext;

}

template<class T>

inline int Flist<T>::**backPtr**(int pos,bool status)

{

int start, nowPos = m\_file->tellg(), ptrOld;

if (pos == this->m\_ptr\_first\_app || pos == this->m\_ptr\_last\_del\_app)

{

return -1;

}

if (status)

start = this->m\_ptr\_last\_del\_app;

else

start = this->m\_ptr\_first\_app;

ptrOld = start;

this->m\_file->seekg(start + sizeof(bool), ios::beg);

while (true)

{

int ptrNew;

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

if (ptrNew == pos)

{

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

return ptrOld;

}

ptrOld = ptrNew;

this->m\_file->seekg(ptrOld + sizeof(bool), ios::beg);

}

}

template<class T>

inline void Flist<T>::**skipItem**(int pos)

{

int ptrDel = nextPtr(pos);

int ptrNew = backPtr(pos);

this->m\_file->seekg(ptrNew + sizeof(bool), ios::beg);

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

}

template<class T>

inline void Flist<T>::**deletion**(int pos)

{

bool del = true;

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

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

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

m\_ptr\_last\_del\_app = pos;

}

template<class T>

inline int Flist<T>::**countActiv**()

{

return this->m\_count\_app - countDel();

}

template<class T>

inline int Flist<T>::**countDel**()

{

int count = 0;

int firstDel = this->m\_ptr\_last\_del\_app;

while (firstDel != 0)

{

firstDel = nextPtr(firstDel);

count++;

}

return count;

}

template<class T>

inline void Flist<T>::**sortDel**()

{

int sizeData = sizeof(bool) + sizeof(int) + sizeof(T),

countDel = this->countDel(),

preOnDel = this->m\_ptr\_last\_del\_app;

int TotalSizeFile = this->m\_start\_list + sizeData \* this->m\_count\_app;

while (countDel != 0)

{

int next = nextPtr(preOnDel);

int posSwap = TotalSizeFile - ((countDel)\*sizeData);

if (posSwap != preOnDel)

swap(posSwap, preOnDel);

if (next != posSwap)

preOnDel = next;

countDel--;

}

}

template<class T>

inline void Flist<T>::**sortActiv**()

{

int sizeData = sizeof(bool) + sizeof(int) + sizeof(T),

countApp = this->countActiv(),

preOnApp = this->m\_ptr\_first\_app,

now = this->m\_start\_list;

while (countApp != 0)

{

int next = nextPtr(preOnApp);

if (now != preOnApp)

swap(preOnApp, now);

if (next != now)

preOnApp = next;

countApp--;

now += sizeData;

}

}

template<class T>

inline void Flist<T>::**swap**(int pos1, int pos2)

{

int backPos1, backPos2;

int ptr1, ptr2;

bool flag1, flag2;

T Data1, Data2;

this->m\_file->seekg(pos1, ios::beg);

this->m\_file->read((char\*)&flag1, sizeof(flag1));

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

this->m\_file->read((char\*)&Data1, sizeof(Data1));

this->m\_file->seekg(pos2, ios::beg);

this->m\_file->read((char\*)&flag2, sizeof(flag2));

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

this->m\_file->read((char\*)&Data2, sizeof(Data2));

backPos1 = backPtr(pos1, flag1);

backPos2 = backPtr(pos2, flag2);

if (backPos2 == pos1)

backPos2 = pos2;

if (backPos1 == pos2)

backPos1 = pos1;

this->m\_file->seekg(pos2, ios::beg);

this->m\_file->write((char\*)&flag1, sizeof(flag1));

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

this->m\_file->write((char\*)&Data1, sizeof(Data1));

this->m\_file->seekg(pos1, ios::beg);

this->m\_file->write((char\*)&flag2, sizeof(flag2));

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

this->m\_file->write((char\*)&Data2, sizeof(Data2));

if(backPos1 != -1)

{

this->m\_file->seekg(backPos1 + sizeof(bool), ios::beg);

this->m\_file->write((char\*)&pos2, sizeof(ptr2));

}

if (backPos2 != -1)

{

this->m\_file->seekg(backPos2 + sizeof(bool), ios::beg);

this->m\_file->write((char\*)&pos1, sizeof(pos1));

}

checkTit(pos1,pos2);

updata();

}

template<class T>

inline void Flist<T>::**\_Load\_data**(Access status)

{

switch (status)

{

case New\_List:

{

newFile();

break;

}

case Existing\_List:

{

existingFile();

break;

}

}

}

template<class T>

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

{

switch (status)

{

case New\_List:

{

std::ofstream oFile(name\_file);

oFile.close();

break;

}

default:break;

}

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");

}

}

template<class T>

inline void Flist<T>::**newFile**()

{

m\_count\_app = 0;

m\_ptr\_first\_app = m\_start\_list;

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

m\_ptr\_last\_app = 0;

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

m\_file->write((char\*)&m\_ptr\_first\_app, sizeof(m\_ptr\_first\_app));

m\_file->write((char\*)&m\_ptr\_last\_app, sizeof(m\_ptr\_last\_app));

m\_file->write((char\*)&m\_ptr\_last\_del\_app, sizeof(m\_ptr\_last\_del\_app));

}

template<class T>

inline void Flist<T>::**existingFile**()

{

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

m\_file->read((char\*)&m\_ptr\_first\_app, sizeof(m\_ptr\_first\_app));

m\_file->read((char\*)&m\_ptr\_last\_app, sizeof(m\_ptr\_last\_app));

m\_file->read((char\*)&m\_ptr\_last\_del\_app, sizeof(m\_ptr\_last\_del\_app));

}

template<class T>

inline void Flist<T>::**checkTit**(int pos1, int pos2)

{

if (pos1 == this->m\_ptr\_first\_app)

this->m\_ptr\_first\_app = pos2;

else if (pos2 == this->m\_ptr\_first\_app)

this->m\_ptr\_first\_app = pos1;

if (pos1 == this->m\_ptr\_last\_app)

this->m\_ptr\_last\_app = pos2;

else if (pos2 == this->m\_ptr\_last\_app)

this->m\_ptr\_last\_app = pos1;

if (pos1 == this->m\_ptr\_last\_del\_app)

this->m\_ptr\_last\_del\_app = pos2;

else if (pos2 == this->m\_ptr\_last\_del\_app)

this->m\_ptr\_last\_del\_app = pos1;

}

* 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.