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 № 3**

**«Организация многосвязных 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 многосвязными структурами данных, получить представление об использовании таких структур для описания 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 многосвязного списка, организованного в файлах: первая запись файла является описанием структуры файла. структуры предлагается в виде двух основных списков списка изделий и списка-спецификации. Каждая запись списка изделий содержит название изделия и может порождать новый список - спецификацию данного изделия. Записи списка-спецификации содержат указатели на узлы и детали, входящие в спецификацию, и кратность вхождения.

Программа использует язык С++.

* 1. **Функциональное назначение**

Данная программа решает задачу хранения и обработки информации посредством многосвязных списковых структур, 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:

* multiListF (работает с битовым файлом, представляет из себя многосвязный список)

**Методы**

Для выполнения поставленной задачи должны определяться следующие методы:

* процедура включения изделия в список insert;
* процедура удаления изделия из списка delete.
* процедура вывода спецификации для заданного изделия print;
* процедура поиска изделия find;
* процедура установления связи между списком изделий и списком спецификаций link;
* процедура сжатия файла pack.

**Листинг**

#pragma once

#include <iostream>

#include <fstream>

#pragma warning(disable:4996)

using namespace std;

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

template <class T>

class multiListF

{

public:

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

void **remove**(T data);

void **insert**(T data);

void **pack**();

void **show**();

void **link**(T data1,T data2, int multi);

void **print**(T data);

void **showConf**();

~**multiListF**();

private:

template <class T>

struct **Data**

{

bool flag;

int ptr;

T data;

int prtConf;

**Data**(T data = T())

{

flag = false;

ptr = 0;

this->data = data;

prtConf = 0;

}

};

struct **DataConf**

{

bool flag;

int ptr;

int ptrProduct;

int multi;

**DataConf**()

{

flag = false;

ptr = 0;

ptrProduct = 0;

multi = 0;

}

};

struct **BeginFile**

{

int m\_count\_app;

int m\_ptr\_last\_app;

int m\_ptr\_first\_app;

int m\_ptr\_last\_del\_app;

char name\_conf[16];

**BeginFile**()

{

m\_count\_app = 0;

m\_ptr\_last\_app = 0;

m\_ptr\_first\_app = 0;

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

}

};

struct **BeginConfig**

{

int m\_count\_app;

int m\_ptr\_first\_app;

int m\_ptr\_last\_del\_app;

**BeginConfig**()

{

m\_count\_app = 0;

m\_ptr\_first\_app = 0;

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

}

};

int **find**(T data);

int **\_link**(int start,int ptr,int multi);

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

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

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

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

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

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

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

int **firstOccurrence**(int prt);

int **nextPtrConf**(int pos);

void **skipItem**(int pos);

int **endList**(int start);

void **deletion**(int pos);

int **nextPtr**(int pos);

void **existingFile**();

int **countActiv**();

void **sortActiv**();

int **countDel**();

void **sortDel**();

void **newFile**();

void **updata**();

void **updataConf**();

fstream\* m\_file;

fstream\* m\_file\_config;

int m\_start\_list;

BeginFile Start;

BeginConfig ConfStart;

};

template<class T>

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

{

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

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

\_Load\_data(status);

}

template<class T>

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

{

sortDel();

sortActiv();

}

template<class T>

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

{

int pos = find(data), ptrDel;

Data<T> dat;

if (pos == -1)

{

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

}

int posConf = firstOccurrence(pos);

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

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

if (posConf != 0|| dat.prtConf != 0)

{

throw new std::logic\_error("Error, there is a pointer in the configuration");

}

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

ptrDel = backPtr(pos);

else

ptrDel = nextPtr(pos);

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

skipItem(pos);

deletion(pos);

change(pos,ptrDel);

updata();

}

template<class T>

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

{

Data<T> dat(data);

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

{

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

Start.m\_count\_app++;

}

else

{

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

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

}

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

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

{

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

}

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

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

{

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

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

dat.ptr = newEnd;

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

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

}

Start.m\_ptr\_last\_app = newEnd;

updata();

}

template<class T>

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

{

int temp;

Data<T> dat;

BeginFile Fil;

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

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

cout << Fil.m\_count\_app << " ";

cout << Fil.m\_ptr\_last\_app << " ";

cout << Fil.m\_ptr\_first\_app << " ";

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

cout << Fil.name\_conf << " ";

cout << endl;

temp = 0;

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

{

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

cout << dat.flag << " ";

cout << dat.ptr << " ";

cout << dat.data << " ";

cout << dat.prtConf << " ";

cout << endl;

temp++;

}

cout << endl;

}

template<class T>

inline void multiListF<T>::**link**(T data1, T data2, int multi)

{

Data<T> dat;

int pos = find(data1);

int posConf = find(data2);

if (pos == -1 || posConf == -1)

{

throw new std::exception("Error access");

}

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

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

dat.prtConf = \_link(dat.prtConf, posConf, multi);

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

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

}

template<class T>

inline void multiListF<T>::**print**(T data)

{

Data<T> dat;

DataConf datCon;

int pos = find(data);

int start;

if (pos == -1 )

{

throw new std::exception("Error access");

}

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

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

cout << "\nProduct " << dat.data << "\nConfiguration : \n";

start = dat.prtConf;

while (start != 0)

{

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

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

this->m\_file->seekg(datCon.ptrProduct, ios::beg);

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

cout << "Product " << dat.data << " Multiplicity = " << datCon.multi << "\n";

start = nextPtrConf(start);

}

}

template<class T>

inline void multiListF<T>::**showConf**()

{

int temp;

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

DataConf dat;

BeginConfig Fil;

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

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

int nowPos1 = m\_file\_config->tellg();

int nowPos2 = m\_file\_config->tellp();

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

cout << Fil.m\_count\_app << " ";

cout << Fil.m\_ptr\_first\_app << " ";

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

cout << endl;

temp = 0;

while (Fil.m\_count\_app > temp)

{

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

cout << dat.flag << " ";

cout << dat.ptr << " ";

cout << dat.ptrProduct << " ";

cout << dat.multi << " ";

cout << endl;

temp++;

}

cout << endl;

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

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

}

template<class T>

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

{

m\_file->close();

delete m\_file;

m\_file\_config->close();

delete m\_file\_config;

}

template<class T>

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

{

Data<T> dat;

int ptrOld = 0;

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

while (true)

{

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

if (dat.data == data)

{

if (ptrOld == 0)

{

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

}

return ptrOld;

}

if (dat.ptr == 0)

{

return -1;

}

ptrOld = dat.ptr;

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

}

}

template<class T>

inline int multiListF<T>::**\_link**(int start,int ptr, int multi)

{

DataConf dat;

dat.ptrProduct = ptr;

dat.multi = multi;

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

{

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

ConfStart.m\_count\_app++;

}

else

{

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

ConfStart.m\_ptr\_last\_del\_app = this->nextPtrConf(ConfStart.m\_ptr\_last\_del\_app);

}

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

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

if (start != 0)

{

int end = endList(start);

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

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

dat.ptr = newEnd;

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

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

}

updataConf();

if (start == 0)

{

return newEnd;

}

return start;

}

template<class T>

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

{

bool t = false;

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

{

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

t = true;

}

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

{

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

t = true;

}

return t;

}

template<class T>

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

{

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

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

}

template<class T>

inline void multiListF<T>::**updataConf**()

{

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

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

}

template<class T>

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

{

Data<T> dat;

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

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

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

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

return dat.ptr;

}

template<class T>

inline int multiListF<T>::**nextPtrConf**(int pos)

{

DataConf dat;

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

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

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

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

return dat.ptr;

}

template<class T>

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

{

Data<T> dat;

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

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

{

return -1;

}

if (status)

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

else

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

ptrOld = start;

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

while (true)

{

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

if (dat.ptr == pos)

{

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

return ptrOld;

}

ptrOld = dat.ptr;

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

}

}

template<class T>

inline void multiListF<T>::**changeConfig**(int pos1, int pos2)

{

int temp;

int start;

DataConf dat;

BeginConfig Fil;

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

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

start = Fil.m\_ptr\_first\_app;

temp = 0;

while (Fil.m\_count\_app > temp)

{

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

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

if (dat.ptrProduct == pos1)

{

dat.ptrProduct = pos2;

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

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

}

else

{

if (dat.ptrProduct == pos2)

{

dat.ptrProduct = pos1;

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

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

}

}

start += sizeof(DataConf);

temp++;

}

}

template<class T>

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

{

Data<T> dat;

int ptrDel = nextPtr(pos);

int ptrNew = backPtr(pos);

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

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

dat.ptr = ptrDel;

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

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

}

template<class T>

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

{

Data<T> dat;

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

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

dat.ptr = Start.m\_ptr\_last\_del\_app;

dat.flag = true;

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

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

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

}

template<class T>

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

{

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

}

template<class T>

inline int multiListF<T>::**endList**(int start)

{

DataConf dat;

int ptrEnd = start;

while(true)

{

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

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

if (dat.ptr == 0)

{

break;

}

ptrEnd = dat.ptr;

}

return ptrEnd;

}

template<class T>

inline int multiListF<T>::**firstOccurrence**(int prt)

{

int temp;

DataConf dat;

BeginConfig Fil;

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

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

temp = 0;

while (Fil.m\_count\_app > temp)

{

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

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

if (dat.ptrProduct == prt)

return nowPos;

temp++;

}

return 0;

}

template<class T>

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

{

int count = 0;

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

while (firstDel != 0)

{

firstDel = nextPtr(firstDel);

count++;

}

return count;

}

template<class T>

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

{

int sizeData = sizeof(Data<T>),

countDel = this->countDel(),

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

int TotalSizeFile = this->m\_start\_list + sizeData \* this->Start.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 multiListF<T>::**sortActiv**()

{

int sizeData = sizeof(Data<T>),

countApp = this->countActiv(),

preOnApp = this->Start.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 multiListF<T>::**swap**(int pos1, int pos2)

{

Data<T> dat1, dat2, dat;

int backPos1, backPos2;

int ptr1, ptr2;

bool flag1, flag2;

T Data1, Data2;

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

this->m\_file->read((char\*)&dat1, sizeof(dat1));

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

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

backPos1 = backPtr(pos1, dat1.flag);

backPos2 = backPtr(pos2, dat2.flag);

if (backPos2 == pos1)

backPos2 = pos2;

if (backPos1 == pos2)

backPos1 = pos1;

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

this->m\_file->write((char\*)&dat1, sizeof(dat1));

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

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

if(backPos1 != -1)

{

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

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

dat.ptr = pos2;

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

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

}

if (backPos2 != -1)

{

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

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

dat.ptr = pos1;

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

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

}

changeConfig(pos1, pos2);

checkTit(pos1,pos2);

updata();

}

template<class T>

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

{

switch (status)

{

case New\_List:

{

newFile();

break;

}

case Existing\_List:

{

existingFile();

break;

}

}

}

template<class T>

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

{

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

strcpy(Start.name\_conf, "C:\\config.ion");

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

std::ofstream oFile(Start.name\_conf);

oFile.close();

this->m\_file\_config = new fstream(Start.name\_conf, ios::binary | ios::in | ios::out);

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

{

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

}

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

ConfStart.m\_ptr\_first\_app = sizeof(ConfStart);

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

}

template<class T>

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

{

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

this->m\_file\_config = new fstream(Start.name\_conf, ios::binary | ios::in | ios::out);

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

{

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

}

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

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

}

template<class T>

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

{

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

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

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

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

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

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

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

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

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

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

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

this->Start.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.