1. Общая постановка задачи

Необходимо разработать класс «Блюдо», содержащий информацию: название, количество калорий(CAL), тип кухни (страна изобретения), время приготовления. Необходимо создать функцию, которая будет сравнивать объекты по калорийности, сортировку массива выполнять методом простого выбора, создать список типов кухней и количества представленных блюд. Вывести по убыванию количества блюд.

1. Требования

1. Данные считываются из файла и записываются в файл;

2. Строковые данные записываются в собственный класс LinkNode;

3. Для ошибок должен быть прописан отдельный класс;

4. Сортировать узлы простым способом и найти максимальные и минимальные значения;

5. Данные могут быть введены и удалены вручную;

6. Для разрабатываемого класса необходимо выполнить:

1) Описание полей класса.

2) Конструктор без параметров, конструктор с параметрами

3) Конструктор копирования, оператор присваивания, деструктор.

4) Перегрузку бинарных операторов

5) Перегрузку операторов отношения.

6) Перегрузку операторов инкремента и декремента.

7) Перегрузку операторов << и >> через дружественные функции.

7. Программа должна быть написана в соответствие со стилем программирования C++

3. Спецификация

1. Файл должен быть открыт
2. В противном случае выводим ошибку, содержащую текст: «Can't open the file!» и попробуйте снова открыть
3. Все взаимодействия с пользователем должны иметь обратную связь, т.е. когда пользователь делает выбор, выводится соответствующее приглашение
4. Все данные должны иметь правильный тип

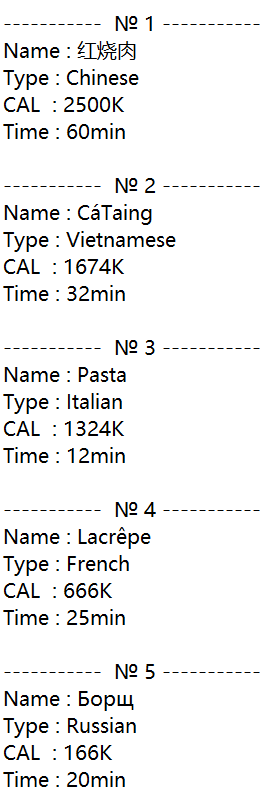
* название, тип кухни, время приготовления должны содержать элементы типа *char*; количество калорий(CAL),
* количество калорий(CAL) должны иметь тип данных *unsigned int*;
* часть чисел времени приготовления должны больше чем 0

1. Могут переставить или выводит узлы связного списка по выбору пользователя

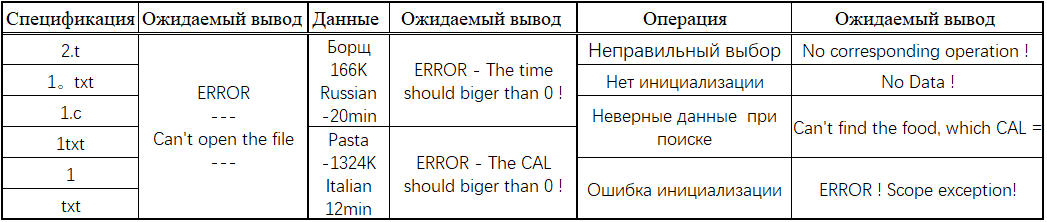
* Inserting data from a file
* Inserting data manually
* Print data
* Choice food by CAL
* Bubble sort
* Delete the data from LinkNodeList
* Clear ALL the data from LinkNodeList
* Destroy the data Clear the data from LinkNodeList
* Output to file

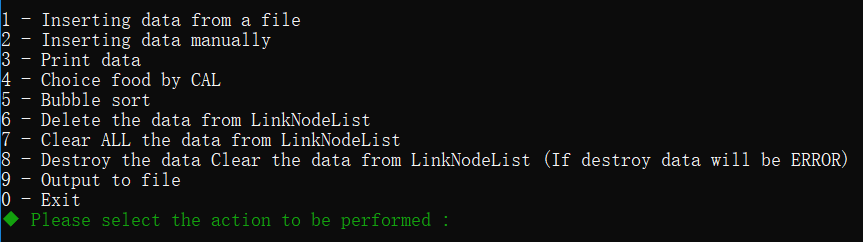
1. Тест план

для тестового примера используется файл с название 1.txt. Эта таблица представлена для правильного ввода.

**Ввода: Вывод:**

Эта таблица представлена для некорректного ввода данных:





1. Программа

**foodHeader.h**

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<string>

#include<fstream>

#include<conio.h>

#include<process.h>

#include<Windows.h>

using namespace std;

#ifndef H\_FOOD

#define H\_FOOD

/\*--------------------------------------------------------------------

Вариант 19

Разработать класс «Блюдо», содержащий информацию :

▪ название, - m\_name

▪ количество калорий, m\_CAL

▪ тип кухни(страна изобретения), m\_type

▪ время приготовления - m\_time

--------------------------------------------------------------------\*/

class MyString

{

public:

MyString(const char\* s);

MyString();

char\* s\_string;

};

class Food

{

Food& operator++();

Food operator++(int);

Food& operator--();

Food operator--(int);

public:

Food();

explicit Food(string name, unsigned int CAL, string typeFood, string timeCook);

Food(const Food& food);

int \_insignificance = 0.0; //псевдокод

// Food name

void setName(string name) const;

string getName();

// CAL

void setCAL(unsigned int CAL) const;

unsigned int getCAL();

// Type of food

void setType(string typeFood) const;

string getType();

// Cooking time

void setTimeCook(string timeCook) const;

string getTimeCook();

~Food();

public:

mutable string m\_NameFood; // 10 ~ 20 内存对齐

mutable string m\_Type; // 5 ~ 20

double \_insignificance2 = 0; // 8 - Псевдокод

mutable string m\_TimeCook; // 3 ~ 6

mutable unsigned int m\_CAL; // 4

};

class smartPointer

{

public:

smartPointer(Food\* food);

//重载->

Food\* operator->();

//重载 \*

Food& operator\*();

~smartPointer();

private:

Food\* m\_food;

};

// 定义节点数据类型 Связвный список

struct LinkNode

{

friend ostream& operator<<(ostream& cout, LinkNode\* pCurrent);

friend ofstream& operator<<(ofstream& ofs, LinkNode\* pCurrent);// Перегрузку

friend ifstream& operator>>(ifstream& ifs, LinkNode\* newNode);

LinkNode();

// 数据域

Food\* data;

// 指针域

LinkNode\* next;

};

struct LinkNodeList

{

LinkNode\* header;

LinkNodeList();

~LinkNodeList();

// 初始化链表 Инициализировать связанный список

LinkNode\* Init\_LinkListFood(string fileName);

// вставить new\_CAL перед old\_CAL

void InSetByCAL\_LinkListFood(LinkNode\* header);

// 遍历 пересечение

void ForEach\_LinkListFood(LinkNode\* header);

// 销毁 уничтожить

void Destroy\_LinkListFood(LinkNode\* header);

// 删除值为cal的节点 Удалить одно Node

void RemByCAL\_LinkListFood(LinkNode\* header);

// 清空 Удалить все

void Clear\_LinkListFood(LinkNode\* header);

// 按热值选择 Выберите по теплотворной способности

void ChoiceCAL\_LinkListFood(LinkNode\* header);

// 冒泡排序 Пузырьковая сортировка

void BubbleSort\_LinkListFood(LinkNode\* header);

// 输出到文件 вывод в файле

void OutPutInFile\_LinkNodeFood(LinkNode\* header);

};

class MyOutOfRangeException :public exception

{

public:

MyOutOfRangeException(string errorInfo);

virtual ~MyOutOfRangeException();

virtual const char\* what() const;

string m\_ErrorInfo;

};

void UserChoice();

#endif // DEBUG

**100 - 面向对象OOP作业.cpp**

#include"foodHeader.h"

int NODE\_SUM = 0;

inline void \_ERROUT(int err\_)

{

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_RED);

if (err\_ == 0){cerr << "！ERROR！ --- Can't open the file ---" << endl << endl;}

else if (err\_ == 1) {cerr << "ERROR! --- Don't have data ---" << endl << endl;}

else if (err\_ == 2) {cerr << "Please import the data at first !" << endl << endl;}

else{ cerr << "ERROR" << endl << endl;}

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE);

}

MyString::MyString()

{

this->s\_string = (char\*)malloc(sizeof(char) \* 100);

}

MyString::MyString(const char\* s)

{

strcpy(s\_string, s);

}

void InfoTe()

{

MyString\* str = new MyString;

cin >> str->s\_string;

}

// ===================================

Food::Food() : m\_NameFood("None"), m\_CAL(0), m\_Type("None"), m\_TimeCook("None")

{

if (m\_CAL < 0)

{

throw MyOutOfRangeException(string("ERROR ! Scope exception!"));

}

}

Food::Food(string name, unsigned int CAL, string typeFood, string timeCook) : m\_NameFood(name), m\_CAL(CAL), m\_Type(typeFood), m\_TimeCook(timeCook)

{}

Food::Food(const Food& food)

{

//food.getCAL()

//this->setCAL() = food.getCAL();

//m\_NameFood = food.getName();

//strcpy(m\_Name, p.m\_Name);

}

Food::~Food()

{}

//===============================

LinkNode::LinkNode()

{

data = new Food;

}

//===============================

LinkNodeList::LinkNodeList()

{

header = nullptr;

}

LinkNodeList::~LinkNodeList()

{}

//===============================

MyOutOfRangeException::MyOutOfRangeException(string errorInfo)

{

this->m\_ErrorInfo = errorInfo;

}

MyOutOfRangeException::~MyOutOfRangeException()

{}

const char\* MyOutOfRangeException::what() const

{

return this->m\_ErrorInfo.c\_str();

}

//=======================================================================

// Food:: name

void Food::setName(string name) const

{

this->m\_NameFood = name;

// strcpy(m\_NameFood, name);

}

string Food::getName()

{

return this->m\_NameFood;

}

// CAL

void Food::setCAL(unsigned int CAL) const

{

this->m\_CAL = CAL;

}

unsigned int Food::getCAL()

{

return this->m\_CAL;

}

// Type of food

void Food::setType(string typeFood) const

{

this->m\_Type = typeFood;

}

string Food::getType()

{

return this->m\_Type;

}

// Cooking time

void Food::setTimeCook(string timeCook) const

{

this->m\_TimeCook = timeCook;

}

string Food::getTimeCook()

{

return this->m\_TimeCook;

}

//前置++重载

Food& Food::operator++()

{

this->\_insignificance++;

return \*this;

}

//后置++ 重载

Food Food::operator++(int)

{

Food tmp = \*this;

\_insignificance++;

return tmp;

}

//前置-- 重载

Food& Food::operator--()

{

this->\_insignificance--;

return \*this;

}

//后置-- 重载

Food Food::operator--(int)

{

Food tmp = \*this;

\_insignificance--;

return tmp;

}

// Перегрузку "<<"

ostream& operator<<(ostream& cout, LinkNode\* pCurrent)

{

// 输出当前节点的数据域

cout << "Name : " << pCurrent->data->getName() << endl;

cout << "Type : " << pCurrent->data->getType() << endl;

cout << "CAL : " << pCurrent->data->getCAL() << "K" << endl;

cout << "Time : " << pCurrent->data->getTimeCook() << endl << endl;

return cout; // 注意，返回一个 cout

}

// Перегрузку "<<"

ofstream& operator<<(ofstream& ofs, LinkNode\* pCurrent)

{

// 输出当前节点的数据域

ofs << "Name : " << pCurrent->data->getName() << endl;

ofs << "Type : " << pCurrent->data->getType() << endl;

ofs << "CAL : " << pCurrent->data->getCAL() << "K" << endl;

ofs << "Time : " << pCurrent->data->getTimeCook() << endl << endl;

return ofs; // 注意，返回一个 ofs

}

// Перегрузку ">>"

ifstream& operator>>(ifstream& ifs, LinkNode\* newNode)

{

string nameFood;

unsigned int CAL;

string typeFood;

string timeCook;

string temp;

ifs >> nameFood;// cout << nameFood << endl;

newNode->data->setName(nameFood);

ifs >> temp; CAL = stoi(temp); //cout << CAL << endl;

if (stoi(temp) < 0)

{

newNode->data->setTimeCook("ERROR - The CAL should biger than 0 !");

}

newNode->data->setCAL(CAL);

ifs >> typeFood; //cout << typeFood << endl;

newNode->data->setType(typeFood);

ifs >> timeCook;//cout << timeCook << endl;

newNode->data->setTimeCook(timeCook);

newNode->next = NULL;

if (stoi(timeCook) < 0)

{

newNode->data->setTimeCook("ERROR - The time should biger than 0 !");

}

ifs >> temp;

return ifs; // 注意，返回一个 ifs

}

//===============================

smartPointer::smartPointer(Food\* food)

{

this->m\_food = food;

}

//重载->

Food\* smartPointer::operator->()

{

return this->m\_food;

}

//重载 \*

Food& smartPointer::operator\*()

{

return \*this->m\_food;

}

smartPointer::~smartPointer()

{

cout << "The smart pointer has been used." << endl;

if (this->m\_food != NULL)

{

delete this->m\_food;

this->m\_food = NULL;

}

}

//===============================

// 初始化

LinkNode\* LinkNodeList::Init\_LinkListFood(string fileName)

{

// 读数据

string temp;

ifstream ifs;

NODE\_SUM = 0;

ifs.open(fileName, ios::in);

// 是否打开成功

if (!ifs.is\_open()) {

\_ERROUT(0);

return NULL;

}

ifs >> temp;

// 创建头节点

LinkNode\* header = new LinkNode;

header->data = NULL;

header->next = NULL;

// 尾部(临时辅助)指针

LinkNode\* pRearCurrent = header;

//string nameFood;

//unsigned int CAL;

//string typeFood;

//string timeCook;

while (!ifs.eof())

{

// 先创建 新节点

LinkNode\* newNode = new LinkNode;

++NODE\_SUM;

//ifs >> nameFood;// cout << nameFood << endl;

//newNode->data->setName(nameFood);

//ifs >> temp; CAL = stoi(temp); //cout << CAL << endl;

//newNode->data->setCAL(CAL);

//ifs >> typeFood; //cout << typeFood << endl;

//newNode->data->setType(typeFood);

//ifs >> timeCook;//cout << timeCook << endl;

//newNode->data->setTimeCook(timeCook);

//newNode->next = NULL;

//ifs >> temp;

ifs >> newNode; // Перегрузку ">>"

// 新节点 插入到链表中

pRearCurrent->next = newNode;

// 更新尾部(临时辅助)指针指向

pRearCurrent = newNode; // newNode 相当于 temp

}

ifs.close();

cout << endl << "Successful data import ! Number of nodes : " << NODE\_SUM << endl << endl;

return header;

}

// 遍历

void LinkNodeList::ForEach\_LinkListFood(LinkNode\* header)

{

if (NULL == header)

{

\_ERROUT(1);

return;

}

// 辅助指针变量

LinkNode\* pCurrent = header->next;

while (pCurrent != NULL)

{

// 输出当前节点的数据域

//cout << "Name : " << pCurrent->data->getName() << endl;

//cout << "Type : " << pCurrent->data->getType() << endl;

//cout << "CAL : " << pCurrent->data->getCAL() << "K" << endl;

//cout << "Time : " << pCurrent->data->getTimeCook() << endl << endl;

cout << pCurrent; // 重载 - Перегрузку <<

pCurrent = pCurrent->next;

}

}

// 冒泡排序（根据 CAL ）Пузырьковая сортировка (из CAL)

void LinkNodeList::BubbleSort\_LinkListFood(LinkNode\* header)

{

if (NULL == header)

{

\_ERROUT(1);

return;

}

int choiceSort;

LinkNode\* pCurFront = header->next;

LinkNode\* pCurBehind = pCurFront->next;

LinkNode\* tempDate = new LinkNode;

cout << "1 - Min -> Max\n2 - Max -> Min\n";

cout << "◆ Please select the action to be performed : "; cin >> choiceSort;

for (int i = 0; i < NODE\_SUM; ++i)

{

for (int j = 0; j < NODE\_SUM - i ; ++j)

{

if (choiceSort == 1)

{

if (pCurFront->data->getCAL() > pCurBehind->data->getCAL())

{

// cout << pCurFront->data->getName() << endl;

// cout << pCurBehind->data->getName() << endl;

tempDate->data = pCurFront->data;

pCurFront->data = pCurBehind->data;

pCurBehind->data = tempDate->data;

}

}

else if (choiceSort == 2)

{

if (pCurFront->data->getCAL() < pCurBehind->data->getCAL())

{

tempDate->data = pCurFront->data;

pCurFront->data = pCurBehind->data;

pCurBehind->data = tempDate->data;

}

}

else

{

cerr << "No corresponding operation ! " << endl;

}

pCurFront/\*->next\*/ = pCurBehind; // 加 ->next 会导致节点丢失！

pCurBehind = pCurBehind->next;

if (pCurBehind == NULL)

{

pCurFront = header->next;

pCurBehind = pCurFront->next;

//break; // 导致本层循环失效！

}

}

}

// Max and Min:

LinkNode\* pCurrent = header->next;

if (choiceSort == 1) {cout << endl << "------------ Min CAL ------------" << endl;}

else{ cout << "------------ Max CAL ------------" << endl;}

cout << pCurrent;

while (pCurrent != NULL)

{

if (pCurrent->next == NULL)

{

if (choiceSort == 1) {cout << "------------ Max CAL ------------" << endl;}

else{ cout << endl << "------------ Min CAL ------------" << endl;}

cout << pCurrent << "---------------------------------" << endl;

}

pCurrent = pCurrent->next;

}

}

// 按热值选择

void LinkNodeList::ChoiceCAL\_LinkListFood(LinkNode\* header)

{

unsigned int minCAL, maxCAL, check = 0;

cout << "The minnimun of CAL : "; cin >> minCAL;

cout << "The maximun of CAL : "; cin >> maxCAL;

if (NULL == header)

{

\_ERROUT(1);

return;

}

// 辅助指针变量

LinkNode\* pCurrent = header->next;

while (pCurrent != NULL)

{

if ((pCurrent->data->getCAL() >= minCAL) && (pCurrent->data->getCAL() <= maxCAL))

{

check++;

// 输出当前节点的数据域

//cout << "Name : " << pCurrent->data->getName() << endl;

//cout << "Type : " << pCurrent->data->getType() << endl;

//cout << "CAL : " << pCurrent->data->getCAL() << "K" << endl;

//cout << "Time : " << pCurrent->data->getTimeCook() << endl << endl;

cout << endl;

cout << pCurrent; // Перегрузку "<<"

}

pCurrent = pCurrent->next;

}

if (check == NULL)

{

cout << "No Data !" << endl << endl;

}

}

// 输出到文件 вывод в файле

void LinkNodeList::OutPutInFile\_LinkNodeFood(LinkNode\* header)

{

ofstream ofs;

int num = 0;

ofs.open("./Output.txt", ios::out);

// 是否打开成功

if (!ofs.is\_open()) {

\_ERROUT(0);

return; // 修改：在此处抛出异常！=====================

}

if (NULL == header)

{

\_ERROUT(1);

return;

}

// 辅助指针变量

LinkNode\* pCurrent = header->next;

ofs << "Number of food : " << NODE\_SUM << endl << endl;

while (pCurrent != NULL)

{

// 输出当前节点的数据域

ofs << "----------- № " << ++num << " -----------"<< endl;

//ofs << "Name : " << pCurrent->data->getName() << endl;

//ofs << "Type : " << pCurrent->data->getType() << endl;

//ofs << "CAL : " << pCurrent->data->getCAL() << "K" << endl;

//ofs << "Time : " << pCurrent->data->getTimeCook() << endl << endl;

ofs << pCurrent;

pCurrent = pCurrent->next;

}

ofs.close();

cout << "Has been output to the file - \"Output.txt\"" << endl << endl;

}

// 在 值为 old\_CAL 后面插入一个新的数据 new\_CAL

// Вставить новые данные new\_CAL после значения old\_CAL

void LinkNodeList::InSetByCAL\_LinkListFood(LinkNode\* header)

{

if (header == nullptr)

{

\_ERROUT(2);

return;

}

unsigned int old\_CAL;

MyString\* str = new MyString;

cout << " - If there have corresponding CAL, the new data will INSERT before this bar\n - If not, insert to the END \n 0 - Exit\n\n";

cout << "Please enter which CAL to insert before : "; cin >> str->s\_string;

old\_CAL = \*str->s\_string;

if (cin.fill() == 1) // 标志位是否损坏

{

cin.clear();

cout << endl <<"Have quit" << endl << endl;

return;

}

if (\*str->s\_string == '0')

{

cout << endl << "Have quit" << endl << endl;

return;

}

// 定义两个临时指针变量，一个指向前一个，一个指向后一个

LinkNode\* pPrev = header;

LinkNode\* pCurrent = header->next;

++NODE\_SUM;

while (pCurrent != NULL)

{

if (pCurrent->data->getCAL() == old\_CAL)

{

break;

}

pPrev = pCurrent;

pCurrent = pCurrent->next;

}

// 创建 new\_val 的新节点

LinkNode\* newNode = new LinkNode;

string temp;

unsigned int new\_CAL;

cout << "Name : "; cin >> temp;

newNode->data->setName(temp);

cout << "CAL : "; cin >> new\_CAL;

newNode->data->setCAL(new\_CAL);

cout << "Type : "; cin >> temp;

newNode->data->setType(temp);

cout << "Time : "; cin >> temp;

newNode->data->setTimeCook(temp);

newNode->next = pCurrent;

pPrev->next = newNode;

cout << "Data inserted successfully !" << endl << endl;

}

// 删除值为cal的节点

// Удалить одно Node

void LinkNodeList::RemByCAL\_LinkListFood(LinkNode\* header)

{

if (header == nullptr)

{

\_ERROUT(2);

return;

}

unsigned int del\_CAL;

cout << " - Press CAL to delete data \n\n";

cout << "Please enter which CAL you want to delete : "; cin >> del\_CAL;

// 定义两个临时指针变量，一个指向前，一个指向后

LinkNode\* pRear = new LinkNode;

LinkNode\* pCurrent = new LinkNode;

pRear = header;

pCurrent = pRear->next;

while (pCurrent != NULL)

{

if (header == NULL)

{

break;

}

if (pCurrent->data->getCAL() == del\_CAL)

{

break;

}

else

{

pRear = pCurrent;

pCurrent = pCurrent->next;

}

}

if (pCurrent == NULL)

{

cout << "Can't find the food, which CAL = " << del\_CAL << endl << endl;

return;

}

else

{

pRear->next = pCurrent->next;

delete pCurrent;

--NODE\_SUM;

cout << "Data deleted successfully !" << endl << endl;

}

return;

}

// 清空 Удалить все

void LinkNodeList::Clear\_LinkListFood(LinkNode\* header)

{

if (NULL == header)

{

\_ERROUT(1);

return;

}

LinkNode\* pCurrent = header->next;

// 先要缓存当前节点的下一节点

LinkNode\* pNext;

while (pCurrent != NULL)

{

pNext = pCurrent->next;

delete pCurrent;

pCurrent = pNext;

}

cout << "ALL data have cleaned" << endl;

header->next = NULL; // 勿忘！！

}

// 销毁 Destroy

void LinkNodeList::Destroy\_LinkListFood(LinkNode\* header)

{

unsigned int choice01;

if (NULL == header)

{

\_ERROUT(1);

return;

}

else

{

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_RED);

cout << " ！ ！ ！Destroy nades may cause the program to CRASH ！ ！ ！\n\n-------Sure to continue ?-------\n\n 0 - Cancel \n 1 - Continue\n";

for (int i = 0; i < 6; ++i) { system("color 4C"); system("color 0C"); }

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE);

cout << "◆ Please select the action to be performed >>> ";

cin >> choice01;

if (choice01 == 0)

{

cout << "Destroy have quit" << endl << endl;

return;

}

else if(choice01 == 1)

{

int temp = 0;

// 创建临时指针

LinkNode\* pCurrent = header;

LinkNode\* pNext;

while (pCurrent != NULL)

{

pNext = pCurrent->next;

if (temp != 0)

{

cout << "Destroy :" << pCurrent->data->getName() << endl;

}

else

{

cout << "Destroy : header" << endl;

}

cout << "IP distroy：" << pCurrent->data << endl << endl;;

delete pCurrent;

pCurrent = pNext;

++temp;

}

}

else

{

cout << "No corresponding operation ! " << endl << endl;

return;

}

}

}

void UserChoice()

{

LinkNodeList myList;

int numChoice;

string fileName = "./1.txt";

enum kEnumFood

{

exit, init,manualInsertionByCAL, printForEach, choiceByCAL, bubbleSort, delByCAL, clear, destroy, outPutInFile

};

while (true)

{

cout << "--------------------------------------------------" << endl << endl;

cout << "1 - Inserting data from a file\n2 - Inserting data manually\n3 - Print data\n4 - Choice food by CAL\n5 - Bubble sort\n6 - Delete the data from LinkNodeList\n7 - Clear ALL the data from LinkNodeList\n8 - Destroy the data Clear the data from LinkNodeList (If destroy data will be ERROR)\n9 - Output to file\n0 - Exit\n";

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_GREEN);

cout << "◆ Please select the action to be performed >>> ";

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE);

cin >> numChoice;

system("cls");

cout << endl;

switch (numChoice)

{

case exit:

return;

case init:

cout << "◆ Please enter the required file name : "; cin >> fileName;

myList.header = myList.Init\_LinkListFood(fileName);

break;

case manualInsertionByCAL:

myList.InSetByCAL\_LinkListFood(myList.header);

break;

case printForEach:

myList.ForEach\_LinkListFood(myList.header);

break;

case choiceByCAL:

myList.ChoiceCAL\_LinkListFood(myList.header);

break;

case bubbleSort:

myList.BubbleSort\_LinkListFood(myList.header);

myList.ForEach\_LinkListFood(myList.header);

break;

case delByCAL:

myList.RemByCAL\_LinkListFood(myList.header);

break;

case clear:

myList.Clear\_LinkListFood(myList.header);

break;

case destroy:

myList.Destroy\_LinkListFood(myList.header);

break;

case outPutInFile:

myList.OutPutInFile\_LinkNodeFood(myList.header);

break;

default:

cout << "No corresponding operation ! " << endl;

break;

}

}

}

int main()

{

// InfoTe();

UserChoice();

}

1. Итоги

Таким образом, мы разработали класс «Блюдо», выполнив основные требования: разработали функцию, которая определяет максимальный и минимальный CAL , выполнили сортировку методом простого выбора по CAL, через шаблонную функцию, и вывели блюдо в сортировкой пузырьком. Также построили свой класс для типа стринг, построенного на динамическом массиве типа char. Написал нескольких функций, чтобы инициализировать, объездить, уничтожить, убирать связный список. И написал функции, они могут вставить, удалить, выбрать данные в связанном списоке.