**Файлы**

Файл содержащий мази:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| МАЗИ | название | цена | производитель | масса | концентрация | способ нанесения |
| данные | вазилин | 2.99 | LuciktTheCompany | 80 г | 20 | Втирание |

Файл содержащий растворы:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| РАСТВОРЫ | название | цена | производитель | Объём | концентрация | способ применения |
| данные | Медицинский спирт | 8 | Новае Жыцця | 500 мл | 96 | Внутренний |

Файл содержащий таблетки:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ТАБЛЕТКИ | название | цена | производитель | Масса | количество в упаковке |
| данные | анальгин | 4.45 | йцукен | 500 мг | 20 |

Файл содержащий порошки:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ПОРОШКИ | название | цена | производитель | Масса | основная жидкость |
| данные | терафлю | 89 | Терафлю компани | 50г | Горячая вода |

Файл содержащий аэрозоли:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| АЭРОЗОЛИ | название | цена | производитель | объём | время активности |
| данные | ксилин | 3,00 | здоровое дыхание | 20мл | 5ч |

**Файлы**

Файл содержащий мази:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| МАЗИ | название | цена | производитель | масса | концентрация | способ нанесения |
| данные | вазилин | 2.99 | LuciktTheCompany | 80 г | 20 | Втирание |

Файл содержащий растворы:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| РАСТВОРЫ | название | цена | производитель | Объём | концентрация | способ применения |
| данные | Медицинский спирт | 8 | Новае Жыцця | 500 мл | 96 | Внутренний |

Файл содержащий витамины:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ВИТАМИНЫ | название | цена | производитель | Масса | количество в упаковке | тип витамина |
| данные | зщшоргпав | апр.45 | йцукен | 500 мг | 20 | Б12 |

Файл содержащий порошки:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ПОРОШКИ | название | цена | производитель | Масса | основная жидкость |
| данные | терафлю | 89 | Терафлю компани | 50г | Горячая вода |

Файл содержащий аэрозоли:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| АЭРОЗОЛИ | название | цена | производитель | объём | время активности |
| данные | ксилин | 3,00 | здоровое дыхание | 20мл | 5ч |

Файл содержащий таблетки общего назначения:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ОБЩЕГО НАЗНАЧЕНИЯ | название | цена | производитель | Масса | количество в упаковке | НАЗНАЧЕНИЕ |
| данные | анальгин | апр.45 | йцукен | 500 мг | 20 | от головной боли |

#pragma once

#include <iostream>

#include <exception>

#include <cassert>

#include <vector>

#include <initializer\_list>

using namespace std;

template <class SC>

class List // шаблон класса списков

{

struct Element // структура представляющая единичный элемент списка

{

SC data; // переменная необходимая для хранения данных в элементе списка

Element\* next; // указатель на следующий элемент списка

Element\* prev; // указатель на предыдущий элемент списка

};

private: // данные доступные для класса

Element\* Head; // указатель первый добавленный элемент

Element\* Curr; // указатель на текущий элемент

public: // данные и методы доступные для пользователя класса

int length; // количество элементов в списке

List(); // конструктор

List(SC x); // конструктор с параметром

List(const List<SC>& C); // конструктор копий

~List(); // деструктор

void init(); // текущий элемент ссылается на первый

void push(SC data); // добавляем новый элемент в список

SC pop(); // извлекаем текущий элемент

void delFromData(SC adr); // находим и удаляем элемент содержащий переданные данные

void loop(); // вывести весь список в консоль

void clear(); // очистить весь список

void next(); // перейти к следующему элементу

void prev(); // перейти к предыдущему элементу

int isNoEmpty(); // проверка состояния списка (если список не пуст)

SC lp(); // получить значение текущего элемента списка не удаляя его

void operator +(SC v); // переопределяем оператор + для добавления нового элемента

void sort(); // сортировка списка

void past(List<SC>& a); // вставить в список все элементы другого списка

};

template <class SC>

List<SC>::List() // реализация конструктора

{

Head = NULL; // обнуляем указатель на первый элемент

Curr = NULL; // обнуляем указатель на текущий элемент

length = 0; // обнуляем количество элементов в списке

}

template <class SC>

List<SC>::List(SC x) // реализация конструктора с параметром

{

Head = NULL; // обнуляем указатель на первый элемент

Curr = NULL; // обнуляем указатель на текущий элемент

length = 0; // обнуляем количество элементов в списке

push(x); // добавляем в список элемент с данными, переданными в конструктор

}

template <class SC>

List<SC>::List(const List<SC>& C) // реализация конструктора копий

{

this->Head = NULL; // обнуляем указатель на первый элемент

this->Curr = NULL; // обнуляем указалель на текущий элемент

this->length = 0; // обнуляем количество элементов в списке

Element\* p = C.Curr; // сохраняем указатель на текущий элемент списка

// отправленного параметром в конструктор

for (int i = 0; i < C.length; i++) // заводим цикл на количество элементов списка-параметра

{

push(p->data); // добавляем в список текущий элемент списка-параметра

p = p->next; // переходим к следующему элементу списка-параметра

}

}

template <class SC>

List<SC>::~List() // реализация деструктора

{

clear(); // очистить список

}

template <class SC>

void List<SC>::init() // приведения списка к состоянию текущий элемент - первый не удаленный

{

Curr = Head; // устанавливаем текущий элемент на первый не удаленный

}

template <class SC> void List<SC>::delFromData(SC adr) // реализация удаления элемента списка с поиском по содержимому

{

if (isNoEmpty()) // если список не пуст

{

Element\* tmp = Curr; // сохранить указатель на текущую позицию в списке

for (int i = 0; i < length; i++) // завести цикл на количество элементов

{

if (Curr->data == adr) // если содержимое текущего элемента - искомое

{

if (Curr == tmp) // если искомое значение находится в текущем состоянии до поиска

tmp = tmp->next; // установить временный указатель на следующий элемент списка

pop(); // извлечь значение

}

next(); // перейти к следующему элементу

}

Curr = tmp; // вернуть указатель текущего элемента на прежнее состояние

}

}

template <class SC> void List<SC>::next() // реализация перехода к следующему элементу

{

if (isNoEmpty()) // если список не пуст

Curr = Curr->next; // установить текущую позицию на следующую

}

template <class SC> void List<SC>::prev() // реализация перехода к предыдущему элементу

{

if (!isNoEmpty()) // если список не пуст

Curr = Curr->prev; // установить текущую позицию на предыдущую

}

template <class SC> int List<SC>::isNoEmpty() // реализация проверки списка на наличие элементов

{

if (Curr == NULL) // если текущая позиция имеет нулевое значение

return 0; // вернуть "ложь"

else // иначе

return 1; // вернуть "истина"

}

template <class SC> SC List<SC>::lp() // реализация получения значения текущего элемента

{

if (isNoEmpty()) // если список не пуст

return Curr->data; // вернуть данные текущего элемента

else // иначе

return 0; // вернуть "ложь"

}

template <class SC> void List<SC>::push(SC data) // реализация добавления элемента

{

Element\* inserted; // создать новый указатель на элемент

inserted = new Element; // выделить память под элемент

inserted->data = data; // установить входной параметр в поле данных элемента

if (!isNoEmpty()) // если список пуст

{

Head = inserted; // установить указатель первого элемента на новый элемент

Curr = inserted; // установить указатель текущего элемента на новый элемент

Curr->next = inserted; // установить указатель следующего элемента на новый элемент

Curr->prev = inserted; // установить указатель предыдущего элемента на новый элемент

}

else // если список не пуст

{

inserted->next = Curr->next; // перенаправляем указатель следующего элемента в добавляемом

inserted->next->prev = inserted; // перенаправляем указатель следующего элемента на добавляемый

Curr->next = inserted; // перенаправляем следующий указатель на добавляемый

inserted->prev = Curr; // перенаправляем предыдущий добавляемого на текущий

}

length++; // увеличиваем количество элементов в списке

Curr = inserted; // устанавливаем текущий указатель на добавленный

}

template <class SC> SC List<SC>::pop() // реализация извлечения текущего элемента

{

SC tag; // переменная под возвращаемое значение

if (!isNoEmpty()) return 0; // если список пуст вернуть "ложь"

Element\* temp = Curr; // сохраняем указатель на текущий элемент

tag = temp->data; // присваиваем переменной значение данных текущего элемента

if (length == 1) // если элемент единственный в списке

{

Head = NULL; // обнулить значение первого элемента

Curr = NULL; // обнулить значение текущего элемента

}

else // если элемент не единственный

{

Curr->next->prev = Curr->prev; // связываем следующий и предыдущий

Curr->prev->next = Curr->next; // связываем предыдущий и следующий

Curr = Curr->next; // перенаправляем текущий элемент на следующий

}

if (temp == Head) // если удаляемый элемент - первый добавленный

Head = Head->next; // перенаправить первый на следующий

length--; // уменьшить количество элементов

delete temp; // удалить предыдущий текущий элемент

return tag; // вернуть данные удаленного элемента

}

template <class SC> void List<SC>::loop() // реализация вывода всех элементов списка

{

if (isNoEmpty()) // если список не пуст

{

Element\* tempCar = Head; // сохраняем указатель на текущий элемент

for (int i = 0; i < length; i++) // заводим цикл на количество элементов списка

{

cout << tempCar->data << " "; // выводим данные текущего элемента

tempCar = tempCar->next; // переходим к следующему элементу

}

cout << "\n"; // вывод терминального символа

}

else // если список пуст

cout << "List is empty\n"; // вывести предупреждение

}

template <class SC> void List<SC>::clear() // реализация очистки всего списка

{

for (int i = 0; i < length;) // заводим цикл на количество элементов списка

pop(); // извлекаем текущий элемент

}

template <class SC> void List<SC>::operator + (SC v) // переопределение оператора суммы

{

push(v); // добавляем параметр в список

}

template <class SC> void List<SC>::sort() // реализация сортировки списка

{

SC Temp; // временная переменная под данные

int i, j; // счетчики цикла

for (j = 0; j < length - 1; j++) // заводим цикл на количество элементов минус 1

{

init(); // устанавливаем текущую позицию на первый элемент

for (i = 0; i < length - 1; i++) // заводим вложенный цикл на количество элементов минус 1

{

if (Curr->data > Curr->next->data) // если данные текущего элемента больше данных следующего

{

Temp = pop(); // извлечь текущий

next(); // перейти к следующему

push(Temp); // вставить данные

}

else // если данные текущего элемента меньше данных следующего

next(); // перейти к следующему

}

i = 0; // обнуляем счетчик

}

}

template <class SC> void List<SC>::past(List<SC>& a) // вставка одного списка в другой

{

init(); // устанавливаем текущий элемент на первый

a.init(); // устанавливаем текущий элемент на первый в списке - параметре

int i; // циклическая переменная

for (i = 0; i < a.length;) // заводим цикл на количество элементов списка-параметра

{

if (a.Curr->data < Curr->data) // если данные параметра меньше данных текущего элемента списка

{

prev(); // перейти к предыдущему в основном списке

push(a.Curr->data); // вставить в список данные текущего элемента списка-параметра

a.next(); // перейти к следующему элементу списка-параметра

next(); // перейти к следующему элементу основного списка

i++; // увеличить итерацию

}

else // если данные параметра не меньше данных текущего элемента списка

next(); // перейти к следующему элементу списка

}

}