1. Постановка задачи

2. Стандартная библиотека языка программирования C++

3. Описание иерархии объектов и методов объектов

4. Описание алгоритма основной программы

5. Тексты модуля, программы

6. Результат тестирования на различных данных (2-3 варианта)

7. Список использованных источников

1. Постановка задачи

Задача «Студенты». Необходимо разработать структуру данных для хранения и обработки информации о студентах учебного заведения, используя объектно-ориентированный подход. Разработать методы работы с информацией.

1. Стандартная библиотека языка программирования C++

При разработке программ на языке C++ весьма ускоряет процесс и облегчает работу использование многолетних наработок, объединённых в одну большую библиотеку языка C++, называемую Стандартная библиотека языка программирования C++. В языке программирования C++ термин Стандартная Библиотека означает коллекцию классов и функций, написанных на базовом языке. Стандартная Библиотека поддерживает несколько основных контейнеров, функций для работы с этими контейнерами, объектов-функции, основных типов строк и потоков (включая интерактивный и файловый ввод-вывод), поддержку некоторых языковых особенностей, и часто используемые функции для выполнения таких задач, как, например, нахождение квадратного корня числа. Стандартная Библиотека языка C++ также включает в себя спецификации стандарта ISO C90 стандартной библиотеки языка Си. Функциональные особенности Стандартной Библиотеки объявляются внутри пространства имен std.

Стандартная библиотека шаблонов (STL) — подмножество стандартной библиотеки C++ и содержит контейнеры, алгоритмы, итераторы, объекты-функции и т. д. Заголовочные файлы стандартной библиотеки C++ не имеют расширения «.h».

Для управления наборами объектов в стандартной библиотеке C++ определены контейнеры. Контейнер представляет коллекцию объектов определенного типа. Последовательный же контейнер (sequential container) позволяет контролировать порядок, в котором элементы располагаются в коллекции, и управлять доступом к этим элементам.

Типы последовательных контейнеров:

* **vector**: массив переменного размера.

Поддерживает произвольный доступ к любому элементу в контейнере.

Обеспечивает добавление и удаление элементов из любого места контейнера.

* **deque**: двусторонняя очередь.

Поддерживает произвольный доступ к любому элементу в контейнере.

Обеспечивает удаление и добавление элементов в начале и в конце контейнера.

* **list**: двухсвязный список

Поддерживает только последовательный двухнаправленный доступ к элементам.

Обеспечивает удаление и добавление элементов в начале и в конце контейнера.

* **forward\_list**: односвязный список.

Поддерживает только однонаправленный последовательный доступ к элементам.

Обеспечивает удаление и добавление элементов в начале и в конце контейнера.

* **array**: массив фиксированного размера.

Поддерживает произвольный доступ к любому элементу в контейнере.

Добавлять или удалять элементы из контейнера нельзя.

* **string**: представляет контейнер, аналогичный вектору, который состоит из символов, то есть строку

Таким образом, стандартная библиотека C++ по умолчанию содержит ряд контейнеров, которые представляют определенные структуры данных. Все они имеют как некоторые общие, так и специфические возможности. За исключением класса array все они поддерживают добавление и удаление элементов. Основное различие между ними состоит в том, как они обеспечивают добавление и удаление элементов, а также доступ к элементам в контейнере. И в зависимости от ситуации и потребностей можно использовать тот или иной тип контейнеров.

Для использования определенного контейнера в программу необходимо добавить соответствующий заголовочный файл, который, как правило, называется по имени класса контейнера.

Кроме того, при определении переменной контейнера необходимо установить ограничение на тип элементов, которые будет содержать контейнер. Тип элементов передается в угловых скобках. Например, определение переменной класса vector, которая хранит числовые данные типа int:

std::vector<int> v1;

Для дальнейшей работы из стандартной библиотеки весьма полезны конструкции типа vector, string, потоковый ввод-вывод.

Класс string позволяет работать со строками, предоставляя большое количество функций, реализованных как непосредственно методы класса, таки дополнительно для него. В частности, есть функции перевода строки в число stoi, stof – аналоги atoi, atof, но принимающие в качестве аргумента переменную типа string.

int n;

float d;

string s = "123.34";

d = stof(s);

s = "1234";

n = stoi(s);

Метод find\_first\_of(string s, std::string::size\_type index) позволяет найти место вхождения строки s с указанной позиции index. Если второй параметр не указывается, то поиск ведётся с начала исходной строки.

Контейнер vector std::vector<T> v1;

В качестве T может быть любой тип данных, в том числе и сам вектор ещё раз, таким образом можно делать многомерные массивы переменной длины.

1. Разработка классов данных для решения поставленной задачи
   1. Класс MyDate

Создан для работы с датами, позволяет хранить дату как набор из трёх целых чисел.

class MyDate

{

public:

MyDate(int day, int month, int year);

MyDate();

~MyDate();

int Year; // Год

int Month; // Месяц

int Day; // День

bool Grate(MyDate d2);

bool Less(MyDate d2);

bool Equal(MyDate d2);

};

Класс MyDate помимо конструктора по умолчанию имеет конструктор с тремя целыми числами – день, месяц, год. Также есть три публичных метода, позволяющие по параметру типа MyDate выяснить, является ли переданная дата больше, меньше или равной значению объекта.

* 1. Класс Student

Базовый класс, содержащий основную информацию об одном студенте.

using namespace std;

class Student

{

public:

Student();

~Student();

MyDate birthDate; // Дата рождения

string code; // Персональнй код

string depart; // Факультет

string group; // Группа

string surname; // Фамилия

string name; // Имя

string patronymic; // Отчество

int kind; // Вид обучения. 0 - обычное, 1 платное

double pay; // Сумма платежа

void GetData(string str);

string ToCSV();

};

* 1. Класс Students

Для работы со всем списком студентов создан класс Students, который содержит список студентов и позволяет формировать различные запросы по ним.

class Students

{

public:

Students();

~Students();

bool ReadFromFile(const string FName);

bool WriteToFile(const string FName);

int GetYoung(MyDate date);

bool AddNew(Student student);

private:

vector<Student> students;

};

Реализованы функции чтения данных из файла и записи в файл в специальном формате. Формат записи текстовый, разделитель полей точка с запятой(формат csv).

Реализованы метод поиска студентов, родившихся после указанной даты, добавление студента в общий список.

* 1. Схема взаимосвязи классов

Класс Students содержит поле, являющееся массивом(вектором) из элементов класса Student.

Student содержит поле, являющееся объектом типа MyDate.

1. Основная программа

В последнее время при разработке программного обеспечения методами объектно-ориентированного программирования выработался подход, когда все необходимые действия совершаются в классе, а основная программа только вызывает эти методы с необходимыми начальными данными и получает результаты. Если со временем возникает задача нахождения какой-либо информации, осуществления расчётов, то более целесообразно реализовать это в классе, чем каждый раз писать эти функции в разных местах большого проекта.

В связи с этим основная программа данной работы состоит из небольшого количества операторов, демонстрирующих работу методов базового класса. В программе StudentsTest.cpp реализованы следующие действия с классом Students

- создание объекта класса

- вызов метода чтения данных из файла

- запрос ввода года

- по заданному году найдены данные об общем количестве студентов, родившихся в этом году и позже

- создан объект класса Student

- по строке данных осуществлено заполнение данными объекта класса Student

- в объект класса Students в список студентов добавлен Student с помощью метода AddNew.

- запись информации в файл.

1. Тексты модуля, программы
   1. Модуль MyDate

Файл MyDate.h

#pragma once

class MyDate

{

public:

MyDate(int day, int month, int year);

MyDate();

~MyDate();

int Year; // Год

int Month; // Месяц

int Day; // День

bool Grate(MyDate d2);

bool Less(MyDate d2);

bool Equal(MyDate d2);

};

Файл MyDate.cpp

#include "stdafx.h"

#include "MyDate.h"

MyDate::MyDate(int day, int month, int year)

{

Day = day;

Month = month;

Year = year;

}

// Парамтр d2 больше, чем значение

bool MyDate::Grate(MyDate d2)

{

bool ret = (Year < d2.Year) || (Year == d2.Year) && (Month < d2.Month) || (Year == d2.Year) && (Month == d2.Month) && (Day < d2.Day);

return ret;

}

// Парамтр d2 меньше, чем значение

bool MyDate::Less(MyDate d2)

{

bool ret = !Grate(d2) && !Equal(d2);

return ret;

}

// Парамтр d2 равен значению

bool MyDate::Equal(MyDate d2)

{

bool ret = (Year == d2.Year) && (Month == d2.Month) && (Day == d2.Day);

return ret;

}

MyDate::MyDate()

{

}

MyDate::~MyDate()

{

}

* 1. Модуль Student

Файл Student.h

#pragma once

#include <string>

#include "MyDate.h"

using namespace std;

class Student

{

public:

Student();

~Student();

MyDate birthDate; // Дата рождения

string code; // Персональнй код

string depart; // Факультет

string group; // Группа

string surname; // Фамилия

string name; // Имя

string patronymic; // Отчество

int kind; // Вид обучения. 0 - обычное, 1 платное

double pay; // Сумма платежа

void GetData(string str);

string ToCSV();

};

Файл Student.cpp

#include "stdafx.h"

#include <vector>

#include "Student.h"

Student::Student()

{

kind = 0;

}

Student::~Student()

{

}

// Чтение данных по одному студенту из строки с разделителем ";"

void Student::GetData(string str)

{

vector<string> vec;

std::string::size\_type found0 = -1;

std::string::size\_type found1 = str.find\_first\_of(";");

while (found1 != std::string::npos)

{

string ss = str.substr(found0+1, found1 - found0-1);

vec.push\_back(ss);

found0 = found1;

found1 = str.find\_first\_of(";", found0 + 1);

}

code = vec[0];

surname = vec[1];

name = vec[2];

patronymic = vec[3];

depart = vec[5];

group = vec[6];

string dateB = vec[4];

found1 = dateB.find\_first\_of(".", 0);

birthDate.Day = stoi(dateB.substr(0, found1));

found0 = found1;

found1 = dateB.find\_first\_of(".", found1+1);

birthDate.Month = stoi(dateB.substr(found0+1, found1 - found0-1));

found0 = found1;

found1 = dateB.find\_first\_of(".", found1 + 1);

birthDate.Year = stoi(dateB.substr(found0+1, found1 - found0-1));

kind = stoi(vec[7]);

pay = stof(vec[8]);

}

// Запись данных по одному студенту в строку с разделителем ";"

string Student::ToCSV()

{

char \*str = new char[256];

string s = "";

s += code + ";";

s += surname + ";";

s += name + ";";

s += patronymic + ";";

sprintf\_s(str, 256, "%02d.%02d.%4d", birthDate.Day, birthDate.Month, birthDate.Year);

s = s + str + ";";

s += depart + ";";

s += group + ";";

sprintf\_s(str, 256, "%d", kind);

s = s + str + ";";

sprintf\_s(str, 256, "%f", pay);

s = s + str + ";";

return s;

}

* 1. Модуль Students

Файл Students.h

#pragma once

#include <vector>

#include <fstream>

#include <iostream>

#include "Student.h"

class Students

{

public:

Students();

~Students();

bool ReadFromFile(const string FName);

bool WriteToFile(const string FName);

int GetYoung(MyDate date);

bool AddNew(Student student);

private:

vector<Student> students;

};

Файл Students.cpp

#include "stdafx.h"

#include "Students.h"

using namespace std;

Students::Students()

{

}

Students::~Students()

{

}

// Нахождение количества студентов, у которых параметр date меньше, чем birthDate

int Students::GetYoung(MyDate date)

{

int num = students.size();

int count = 0;

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

{

if (students[i].birthDate.Less(date))

{

count++;

}

}

return count;

}

// Чтение данных из файла в формате с разделителем ";"

bool Students::ReadFromFile(const string FName)

{

ifstream FRead(FName);

bool ret = true;

int bufsize = 256;

char \*buff = new char[bufsize]; // буфер промежуточного хранения считываемого из файла текста

Student student;

//int num = 0;

try

{

if (!FRead.is\_open()) // если файл не открыт

{

return false;

}

else

{

while (FRead.getline(buff, bufsize))

{

// FRead >> buff;

string s = buff;

student.GetData(s);

students.push\_back(student);

}

FRead.close(); // закрываем файл

}

}

catch (...)

{

ret = false;

}

delete buff;

}

// Запись данных в файл в формате с разделителем ";"

bool Students::WriteToFile(const string FName)

{

ofstream FWrite(FName);

bool ret = true;

int bufsize = 256;

char \*buff = new char[bufsize]; // буфер промежуточного хранения считываемого из файла текста

Student student;

//int num = 0;

try

{

if (!FWrite.is\_open()) // если файл не открыт

{

return false;

}

else

{

int num = students.size();

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

FWrite << students[i].ToCSV() << endl;

FWrite.close(); // закрываем файл

}

}

catch (...)

{

ret = false;

}

return ret;

}

// Добавление студента в список

bool Students::AddNew(Student student)

{

bool ret = true;

try

{

students.push\_back(student);

}

catch (...)

{

ret = false;

}

return ret;

}

* 1. Модуль StudentsTest.cpp

Файл StudentsTest.cpp

// StudentsTest.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <conio.h>

#include "Student.h"

#include "Students.h"

int main()

{

setlocale(LC\_ALL, "rus");

Students sts;

sts.ReadFromFile("s-list.txt");

cout << "Введите год ";

int year;

cin >> year;

MyDate \*d = new MyDate(1, 1, year);

int col = sts.GetYoung(\*d);

cout << "Количество студентов, родившихся после 1.1." << year << " " << col << endl;

delete d;

Student st;

string ss = "023578;Бицадзе;Автандил;Иванович; 11.03.1923;первый факультет;2012-бис;0;0;";

st.GetData(ss);

if (sts.AddNew(st))

{

cout << "В список студентов добавлен " << st.surname << " " << st.name << " " << st.patronymic << endl;

}

else

{

cout << "В список студентов не удалось добавить " << st.surname << " " << st.name << " " << st.patronymic << endl;

}

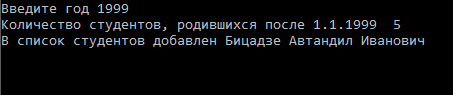
sts.WriteToFile("s-list.out");

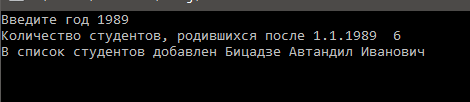
\_getch();

return 0;

}

1. Результат тестирования на различных данных





1. Список использованных источников
2. <http://www.cplusplus.com/>
3. <https://metanit.com/cpp/>
4. К. Джамса. Учимся программировать на языке C++: Пер. с англ. - Москва: Мир, 1997. 320с.