Тилебалиев Самат. гр. 931921

Часть 1.  
  
1) Постановка задачи.

Необходимо создать шаблонный класс, реализующий стек, поддерживающий следующие операции:

1. Помещение объекта в стек;

2. Извлечение объекта из стека;

3. Получение размерности стека.

В случае попытки вызова операции извлечение объекта из стека при условии, что стек пуст, должно генерироваться исключение класса EStackEmpty (наследник класса EStackException). Данный класс должен содержать публичный метод char\* what(), возвращающий диагностическое сообщение.

2) Предполагаемое решение.

Стек - это контейнер, представляющий собой список элементов, организованных по принципу LIFO(last in - first out) или "Последний пришел - первым вышел". То есть новые данные, добавленные в этот контейнер, будут обработаны и удалены в первую очередь.

Необходимо решить, как представлять сами данные внутри стека.

Один из неудачных способов реализации - организовать хранение данных в виде динамического массива.

Если количество элементов в стеке будет превышать начальному запросу выделения памяти, то необходимо заново выделять память и копировать все элементы контейнера. Такая реализация может иметь линейную сложность при помещении элемента в конец контейнера. Ещё хотелось бы отметить, что удаление последнего элемента (без перераспределения памяти) в контейнере лишь приводит к затиранию этого элемента, но память под этот элемент будет занята. Минусы в конечном счёте приводят к не эффективному использованию памяти, к скорости работы контейнера.

Лучшее решение - односвязный список. Это последовательность узлов, в котором каждый узел содержит необходимые данные: значение и указатель на предыдущий узел. Для работы с таким списком необходимо знать последний созданный узел. А для стека это самое то, ведь как было сказано ранее, все операции над таким контейнером начинаются "с конца". Операции удаления и добавления имеют постоянную сложность, т.к. выделяется фрагмент памяти под каждый один элемент, тогда как в массиве элементы хранятся в непрерывном одном блоке.

Итак, в первую очередь, класс Стек должен быть шаблонным, чтобы иметь возможность хранить данные любого типа. Должен быть реализован вспомогательный класс, отвечающий за узлы. Сам стек должен содержать указатель на последний узел для обработки данных и все необходимые методы для эффективного использования контейнера.

Я определил вспомогательный класс (название Node) внутри самого Стека (Stack) в private секции в целях обеспечения локальности (в терминах области видимости) и инкапсуляции. Также в этой же секции имеются член-данные - указатель на последний узел и целочисленный тип(size\_), хранящий количество элементов в контейнере.

Реализовал необходимый интерфейс взаимодействия со стеком для клиентов:

1. Возможность помещать данные в стек - функция push, которая принимает значение (lvalue, rvalue) и помещает его в конец контейнера. Гарантирует константность для lvalue-значений. Инкрементирует (size\_).

2. Извлечение объекта из стека - функция top, которая возвращает последнее значение в контейнере. У клиента есть возможность модифицировать эти данные.

3. Получение размерности стека - функция size, возвращает член-данные класса (size\_). Гарантирует его константность.

4. Удаление последнего элемента из стека - функция pop. Удаляет элемент и декрементирует (size\_).

5. Проверка на пустоту стека - функция empty, проверяет валидность указателя на последний узел.

Клиенту предоставляется возможность копировать, присваивать друг другу стек.

Также дополнительно были реализованы для стека два exception-объекты: EStackEmpty и его базовый класс EStackException. Для вывода сообщения об ошибке используется виртуальный метод what(), который возвращает const char\*.

3) Коды программ.

github.com/tmpsam22

stack.h exception.h  
  
4. Инструкция пользователя

В корневой папке в ветке stack\_test имеется скрипт start\_build.sh, который собирает проект и запускает исполняемый файл с названием only\_stack в целях демонстрации работы стека и его тестирования. Структура данного скрипта:

#!/bin/bash

if [ ! -d build ]

then

mkdir build

fi

cd build && \

cmake .. && \

cmake --build . && \

./only\_stack

Если скрипт не запускается, то необходимо выполнить следующие шаги самостоятельно:

1. mkdir build  
2. cd build

3. cmake ..

4. cmake --build .

5. ./only\_stack # запуск исполняемого файла

Входных и выходных параметров программа не ожидает.  
Программа при запуске вызывает тестовую функцию, которая выводит в консоль сообщения из пункта 5.

5. Тестирование

1. Вызов функции top для получения последнего элемента из стека.

std::cout << "----------------------------" << std::endl;

std::cout << "EXCEPTION TEST" << std::endl;

try

{

Stack<int> widget\_empty;

widget\_empty.top(); // ожидается исключение EStackException

}

catch(custom::exceptions::EStackException& e)

{

// вывод : Runtime error: stack is empty. Called function top()

std::cerr << e.what() << std::endl;

}

**Вывод из консоли:**  
----------------------------

EXCEPTION TEST

Runtime error: stack is empty. Called function top()

Ожидаемый результат.

2. Вызов функции push для добавления элемента в стек

std::cout << "----------------------------" << std::endl;

std::cout << "PUSH TEST" << std::endl;

//Инициализация стека другим стеком.

Stack<int> widget\_first;

display(widget\_first, "widget\_first");

std::vector<int> values{ 1, 2, 4, 5};

for (auto& value : values)

{

widget\_first.push(value);

}

display(widget\_first, "widget\_first");

**Вывод из консоли:**  
----------------------------

PUSH TEST

widget\_first

widget\_first

5 4 2 1

Ожидаемый результат.

3. Копирование стека

std::cout << "----------------------------" << std::endl;

std::cout << "COPY CTOR TEST" << std::endl;

Stack<int> widget\_second = widget\_first; // глубокое копирование:

// все элементы копируются в новые динамические фрагменты

display(widget\_first, "widget first");

display(widget\_second, "widget second");

**Вывод из консоли:**

----------------------------

COPY CTOR TEST

widget first

5 4 2 1

widget second

5 4 2 1

Ожидаемое поведение.

4. Оператор присваивания.

std::cout << "----------------------------" << std::endl;

std::cout << "ASSIGNMENT OPERATOR TEST" << std::endl;

//Присваивание стека:

Stack<int> widget\_third{ 99 };

display(widget\_third, "widget\_third");

Stack<int> widget\_fourth{ 100 } ;

widget\_third = widget\_fourth; // значение последнего элемента равно теперь 100

display(widget\_third, "widget\_third");

display(widget\_fourth, "widget\_fourth");

**Вывод из консоли:**

----------------------------

ASSIGNMENT OPERATOR TEST

widget\_third

99

widget\_third

100

widget\_fourth

100

Ожидаемое поведение.

4. Конструктор с перемещением

std::cout << "----------------------------" << std::endl;

// конструктор перемещения

std::cout << "CTOR WITH MOVE SEMANTICS TEST" << std::endl;

Stack<int> widget\_fivth = std::move(widget\_first);

display(widget\_fivth, "widget\_fivth");

display(widget\_first, "widget\_first");

**Вывод из консоли:**

----------------------------

CTOR WITH MOVE SEMANTICS TEST

widget\_fivth

5 4 2 1

widget\_first

Ожидаемое поведение.

5. Оператор присваивания с перемещением

std::cout << "----------------------------" << std::endl;

// оператор присваивания с перемещением

std::cout << "ASIGNMENT OPERATOR WITH MOVE SEMATICS TEST" << std::endl;

Stack<int> widget\_sixth { 120 };

display(widget\_sixth, "widget\_sixth");

widget\_sixth = std::move(widget\_fivth);

display(widget\_sixth, "widget\_sixth");

display(widget\_fivth, "widget\_fivth");

**Вывод из консоли:**

----------------------------

ASIGNMENT OPERATOR WITH MOVE SEMATICS TEST

widget\_sixth

120

widget\_sixth

5 4 2 1

widget\_fivth

Ожидаемое поведение

6. Вывод размерности и стека и вызов функции isEmpty

// вывод размерности стеков

std::cout << "SIZES:\n";

std::cout << "widget\_sixth:\t" << widget\_sixth.size() << std::endl;

std::cout << "widgex\_fivth:\t" << widget\_fivth.size() << std::endl;

// проверяем на пустоту стека

std::cout << "is widget\_fivth EMPTY:\t" << std::boolalpha << widget\_fivth.empty() << std::endl;

**Вывод из консоли:**

SIZES:

widget\_sixth: 4

widgex\_fivth: 0

is widget\_fivth EMPTY: true

Ожидаемое поведение.

Удаление последнего элемента в стеке.

std::cout << "----------------------------" << std::endl;

std::cout << "TOP from widget\_sixth is:\t " << widget\_sixth.top() << std::endl;

widget\_sixth.pop();

display(widget\_sixth, "after POP values in widget\_sixth:");

std::cout << "SIZE of widget\_sixth: " << widget\_sixth.size() << std::endl;

**Вывод из консоли:**

----------------------------

TOP from widget\_sixth is: 5

after POP values in widget\_sixth:

4 2 1

SIZE of widget\_sixth: 3

Ожидаемое поведение.  
  
  
Часть 2.

1) Постановки задачи.

Реализовать класс PersonKeeper с методами readPersons и writePersons. Метод readPersons должен считывать информацию о людях из входного потока (файла), создавать на основе этой информации объекты класса Person, и помещать их в стек. Формат входного файла должен быть такой:

Фамилия Имя Отчество

В качестве разделителей могут выступать пробелы, табуляции, переводы строки.

Пример файла:

Иванов Василий Иванович

Сидоров Александр Михайлович

…

Метод readPersons должен возвращать стек.

Метод writePersons должен записывать в поток из стека (стек передается аргументом) информацию о людях в соответствии с вышеописанным форматом. Передаваемый методу writePersons стек не должен изменяться.

Класс PersonKeeper должен быть реализован в соответствии с шаблоном Singleton.  
  
2) Проектное решение

Рассмотрим паттерн Singleton. Мы хотим, чтобы клиент взаимодейстовал лишь с одним объектом PersonKeeper.

При создании экземпляра класса будут вызываться его конструкторы, поэтому нам необходимо запретить вызовы, объявив конструкторы закрытыми.

Нужно определить статический объект PersonKeeper, при этом в области видимости самого класса PersonKeeper[см. предыдущее рассуждение].

Проинициализировать такой объект можно двумя способами:

1) Инициализация PersonKeeper в глобальном пространстве [нелокальный статический объект].

Мы знаем, что порядок инициализации статических объектов, объявленных в глобальном пространстве имен, определенных в нескольких единицах трансляции не определен. То есть, если инициализация объекта просходит во одной единице трансляции, а используется, к примеру, во второй, то такой объект может оказаться неинициализированным при прямом обращении. Определить проинициализирован ли объект при заданном верном порядке является почти невыполнимой задачей. Поэтому, предлагаю второй способ.

2) Инициализация локального статического объекта PersonKeeper с помощью статической функции.

Такой подход будет гарантировать, что объект будет проинициализирован при первом вызове этой функции и при повторном использовании этой функции, она будет возвращать ранее инициализированный объект.

Методы readPerson и writePersons для большого функционала должны работать с объектами ifstream, fstream и ofstream.

Для удобства пользования PersonKeeper, интерфейс должен быть единственным, это возможно благодаря перегрузкам функции на C++.

Методы должны работать с существующими и валидными объектами fstream//ifstream//ofstream, либо на вход подается имя файла для чтения или новой записи с желаемыми флагами открытия.  
Очевидно, объект PersonKeeper не должен иметь конструктор копироваия и оператор присваивания.  
  
3) Коды программ.

github.com/tmpsam22/Stack

PersonKeeper:

include/person/person\_keeper.h

src/person/person\_keeper.cpp

Person:

include/person/person.h

src/person/person.cpp  
  
4) Инструкция пользования.

В корневой папке ветках develop, main имеется скрипт start\_build.sh, который собирает проект и запускает исполняемый файл lab\_with\_person\_keeper в целях демонстрации работы методов Person и PersonKeeper . Структура данного скрипта:

#!/bin/bash

if [ ! -d build ]

then

mkdir build

fi

cd build && \

cmake .. && \

cmake --build . && \

./lab\_with\_person\_keeper

Если скрипт не запускается, то необходимо выполнить следующие шаги самостоятельно:

1. mkdir build  
2. cd build

3. cmake ..

4. cmake --build .

5. ./lab\_with\_person\_keeper # запуск исполняемого файла

В корневой папке есть примеры данных: valid\_names.txt, в котором данные записаны в правильном виде и invalid\_symbol.txt, в котором имеется символ “:”, что приводит в вызову исключения при попытке извлечь данные из этого файла с помощью интерфейса PersonKeeper.

Входных параметров программа не ожидает.  
Программа при запуске вызывает тестовую функцию, которая выводит в консоль сообщения из пункта 5.

В ходе работы программы, в корневой папке появиться файл с именем default\_output.txt.

5) Тестирование

Сначала протестируем интерфейс класс PersonKeeper.

1. Считывание из файла ФИО, в котором имеется некорректный символ:

try {

// создается статический объект PersonKeeper

// извлекаем ФИО из файла

PersonKeeper::CreateInstance().readPersons(invalid\_file);

} catch( std::exception& e) {

std::cerr << e.what() << std::endl;

}

**Вывод из консоли:**  
  
--------------TEST--------------

EXPECT INVALID SYMBOL IN FILE

Got unexpected symbol in file

Ожидаемое поведение.

2.а Считывание корректных ФИО из файла, аргумент функции – название файла:

display\_test("READING valid data, arg is string");

// Считываем валидные данные (Ожидается ФИО) из файла:

auto stack = PersonKeeper::CreateInstance().readPersons(valid\_file);

display(stack);

**Вывод из консоли:**

--------------TEST--------------

READING valid data, arg is string

Maxim Elonov Maksonovich

Aleksandr Dmitriev Dmitrievich

Bear Russian Sibirovich  
  
Ожидаемое поведение.

2.б Считывание корректных ФИО из файла, аргумент функции – объект ifstream:

display\_test("READING valid data, arg is ifstream");

std::ifstream ifile {valid\_file};

stack = PersonKeeper::CreateInstance().readPersons(ifile);

display(stack);

**Вывод из консоли:**

--------------TEST--------------

READING valid data, arg is ifstream

Maxim Elonov Maksonovich

Aleksandr Dmitriev Dmitrievich

Bear Russian Sibirovich

Ожидаемое поведение.

2.в Считывание корректных ФИО из файла, аргумент функции – объект fstream:

display\_test("READING valid data, arg is fstream");

std::fstream file {valid\_file};

stack = PersonKeeper::CreateInstance().readPersons(file);

display(stack);

**Вывод из консоли:**

--------------TEST--------------

READING valid data, arg is fstream

Maxim Elonov Maksonovich

Aleksandr Dmitriev Dmitrievich

Bear Russian Sibirovich

Ожидаемое поведение.

3.а Запись ФИО в файл, аргумент функции – название файла

display\_test("WRITING valid data to file, arg is string");

Stack<Person> stack\_w

{

Person (

Person::first\_name{"A"},

Person::second\_name{"B"},

Person::patronymic{"C"}

)

};

PersonKeeper::CreateInstance().writePersons(stack\_w, out\_filename);

std::cout << "Checking result:\n";

display(

PersonKeeper::CreateInstance().readPersons(out\_filename)

);

**Вывод из консоли:**

--------------TEST--------------

WRITING valid data to file, arg is string

Checking result:

B A C

Ожидаемое поведение.

3.а Запись ФИО в файл, аргумент функции – объект fstream

display\_test("WRITING valid data to file, arg is fstream");

std::fstream file\_w {out\_filename};

PersonKeeper::CreateInstance().writePersons(stack\_w, file\_w);

std::cout << "Checking result:\n";

display(

PersonKeeper::CreateInstance().readPersons(out\_filename)

);

**Вывод из консоли:**

--------------TEST--------------

WRITING valid data to file, arg is fstream

Checking result:

B A C

Ожидаемое поведение.

3.б Запись ФИО в файл, аргумент функции – объект ofstream

display\_test("WRITING valid data to file, arg is ofstream");

std::ofstream ofile {out\_filename};

PersonKeeper::CreateInstance().writePersons(stack\_w, ofile);

std::cout << "Checking result:\n";

display(

PersonKeeper::CreateInstance().readPersons(out\_filename)

);

**Вывод из консоли:**

--------------TEST--------------

WRITING valid data to file, arg is ofstream

Checking result:

B A C

Ожидаемое поведение.

Теперь протестируем класс Person

1. Получить имя объекта Person

std::cout << "\nTesting class Person\n";

auto& person = stack\_w.top(); // получили объект Person для тестирования

display\_test("Getting first name from Person");

std::cout << "first name: "<< person.getFirstName() << std::endl;

**Вывод из консоли:**

--------------TEST--------------

Getting first name from Person

first name: A

Ожидаемое поведение.

2. Получить фамилию объекта Person

display\_test("Getting second name from Person");

std::cout << "second name: " << person.getSecondName() << std::endl;

**Вывод из консоли:**

--------------TEST--------------

Getting second name from Person

second name: B

Ожидаемое поведение.

3. Получить отчество объекта Person

display\_test("Getting patronymic from Person");

std::cout << "third name: "<< person.getPatronymic() << std::endl;

**Вывод из консоли:**

--------------TEST--------------

Getting patronymic from Person

third name: C

Ожидаемое поведение.

4. Установить имя для объекта Person

display\_test("Setting \"X\" first name to Person");

person.setFirstName("X");

std::cout << "now first name: "<< person.getFirstName() << std::endl;

**Вывод из консоли:**

--------------TEST--------------

Setting "X" first name to Person

now first name: X

Ожидаемое поведение.

5. Установить фамилию для объекта Person

display\_test("Setting \"Y\" second name to Person");

person.setSecondName("Y");

std::cout << "now second name: " << person.getSecondName() << std::endl;

**Вывод из консоли:**

--------------TEST--------------

Setting "Y" second name to Person

now second name: Y

Ожидаемое поведение.

6. Установить отчество для объекта Person

display\_test("Setting \"Z\" patronymic to Person");

person.setPatronymic("Z");

std::cout << "now third name: "<< person.getPatronymic() << std::endl;

**Вывод из консоли:**

--------------TEST--------------

Setting "Z" patronymic to Person

now third name: Z

Ожидаемое поведение.