НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Р.Е. АЛЕКСЕЕВА

ИНСТИТУТ РАДИОЭЛЕКТРОНИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра
«КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ В ПРОЕКТИРОВАНИИ И ПРОИЗВОДСТВЕ»

Курсовой проект

на тему

«Разработка программы «Шаблонный вектор переменной длины»

Выполнил
студент гр. 16-ИСТв-1
Сироткин Ф.А.
Проверил
Леснов И.В.

Нижний Новгород 2016

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Кафедра Компьютерные технологии в проектировании и производстве

УТВЕРЖДАЮ
Зав. кафедрой
_____ И.О. Фамилия
«___» _____ 20___ г.

ЗАДАНИЕ

на выполнение курсового проекта

по направлению подготовки (специальности)

09.03.02 Информационные системы и технологии

(код и наименование)

студенту Сироткину Ф.А. (Ф.И.О.) группы 16-ИСТв-1

1. Тема курсового проекта «Разработка программы «Шаблонный вектор переменной длины»

2. Срок сдачи студентом законченной работы 28 декабря 2016

3. Исходные данные к работе Разработать шаблонный класс Vect для представления динамических одномерных массивов (векторов).

Руководитель _____ И.О. Фамилия
(подпись)

Задание принял к исполнению _____
(дата)

Студент _____ И.О. Фамилия
(подпись)

Подп. и дата | Взам. инв. № | Инв. № дубл. | Подп. и дата | Инв. № подл

Ли	Изм.	№ докум.	Подп.	Дата	КП-НГТУ-16-ИСТв-1		
Разраб.		RePack by				Лит	Лист 2 / Листов 27
Пров.							
Т. контр.							
Н. контр.							
Утв.							

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»

(НГТУ)

АННОТАЦИЯ

к курсовому проекту

по направлению подготовки (специальности)
09.03.02 Информационные системы и технологии
(код и наименование)

студента Сироткина Ф.А. (Ф.И.О.) группы 16-ИСТв-1

по теме «Разработка программы «Шаблонный вектор переменной длины»

_____/_____
подпись студента /расшифровка подписи

«___» _____ 20____ г.

Содержание

Введение

Программирование – процесс создания компьютерных программ. В более широком смысле под программированием понимают весь спектр деятельности, связанный с созданием и поддержанием в рабочем состоянии программ – программного обеспечения.

Программирование основывается на использовании языков программирования, на которых записываются инструкции для компьютера. Современное приложение содержит множество таких инструкций, связанных между собой.

Главное требование, которому должно удовлетворять приложение, – работать в полном соответствии со спецификацией и адекватно реагировать на любые действия пользователя.

Большинство программ предназначено для обработки информации. В качестве данных могут выступать самые разные виды характеристик реального мира: текстовые документы, результаты научных экспериментов, информация о работниках и т.д. Что бы данные собой ни представляли, хранятся и обрабатываются они примерно одинаковыми способами. Термин «структура данных» говорит о том, как храниться информация в памяти компьютера, а «алгоритм» – как эта информация обрабатывается.

Классы в Си++ представляют собой прекрасный механизм для создания библиотеки структур данных. Этот механизм называется «Класс-контейнер» для хранения и обработки данных. При этом не имеет значения, какие именно данные хранятся, будь то базовые типы или же экземпляры классов. Они выделяют память, необходимую для хранения данных, и определяют механизм доступа к ним.

Работая с массивом во многих ситуациях бывает неудобно, медленно и неэффективно. Проблема также заключается в том, что их размеры нужно указывать на этапе компиляции, то есть в исходном коде. К сожалению, во время написания программы обычно не бывает известно, сколько элементов потребуется записать в массив. Поэтому, как правило, рассчитывают на худший вариант, и размер массива указывают «с запасом». В итоге при работе программы выясняется, что-либо в массиве остается очень много свободного места, либо все-таки не хватает ячеек, а это может привести к чему угодно, вплоть до зависания программы.

Для борьбы с такими трудностями был разработан данный курсовой проект, который описывает шаблонный расширяемый вектор переменной длины.

1. Разработка и анализ технического задания

1.2 Описание предметной области

Первое, с чего начинается создание структуры данных – это выбор его типа данных. Типы данных бывают стандартные (целочисленные, с плавающей точкой, логические, символьные) и определимые пользователем (объекты реального мира).

При создании такой структуры данных необходимо учитывать работу последовательного контейнера, инкапсулирующего массивы переменной длины.

Хранение контейнера должно обрабатывается автоматически, расширяясь и сужаясь по мере необходимости. Векторы переменной длины обычно занимают больше места, чем статические массивы, поскольку некоторое количество памяти выделяется про запас на обработку будущего роста. Таким образом, память для вектора переменной длины требуется выделять не при каждой вставке элемента, а только после исчерпания выделенной памяти под него.

1.3 Разработка технического задания

Разработать шаблонный класс Vect для представления динамических одномерных массивов (векторов). Класс должен обеспечивать хранение данных любого типа T, для которого предусмотрены конструктор по умолчанию, конструктор копирования и операция присваивания.

1.1.1 Назначение разработки

Программа представляет собой библиотеку, предназначенную для удобной работы с шаблонным одномерным массивом (вектором) переменной длины, где не нужно специально следить его размерностью.

1.1.2 Область применения

Работа со динамическими структурами данных, предназначенными для хранения элементов заданного типа в смежных блоках памяти, где работа с выделением и удалением данных из памяти должна осуществятся автоматически.

1.1.3 Функциональные требования к системе

Класс Vect должен содержать:

– конструктор по умолчанию, создающий вектор нулевого размера;

– конструктор с параметрами, создающий вектор заданного размера;

– перегруженную операцию индексирования, возвращающую ссылку на соответствующий элемент вектора;

– метод, добавляющий элемент в произвольную позицию вектора;

– метод, добавляющий элемент в конец вектора;

– метод, удаляющий элемент из конца вектора.

1.1.4 Количественные требования к системе

– в системе должен присутствовать как минимум один элемент класса-контейнера;

– максимальное значение размерности вектора ограничивается возможностями используемой операционной системы.

1.1.5 Требования к техническим средствам

Требования к техническим средствам определяются установленной операционной системой. Программа не является ресурсоемкой и не предъявляет никаких особых требований к техническим средствам.

1.1.6 Требования к безопасности и целостности информации

В системе предусмотрена генерация и обработка исключительных ситуаций для возможных ошибок, которые могут быть вызваны множеством различных обстоятельств, таких, как выход за пределы памяти, попытка инициализировать объект недопустимым значением или использовать индекс, выходящий за пределы вектора.

1.1.7 Требования к информационной и программной совместимости

Система является кроссплатформенным (межплатформенным) программным обеспечением, которое позволяет работать более чем на одной аппаратной платформе и/или операционной системе на уровне компиляции (примером является программное обеспечение, предназначенное для работы в операционных системах Linux и Windows одновременно).

1.1.8 Требования к графическому интерфейсу

Программа не предъявляет особых требований к интерфейсу пользователя.

1.2 Анализ технического задания

Одним из важных вопросов при разработке контейнерного класса является вопрос реализации хранения элементов контейнера, или, другими словами, выбор подходящей структуры данных (массив, список, бинарное дерево и т.п.).

В данном курсовом проекте показан выбор на динамическом массиве, размещаемом в памяти посредством операции new, поскольку это наиболее простое решение.

После размещения адрес первого элемента вектора будет запоминаться в поле T* first, а адрес элемента, следующего за последним, – в поле T* last.

С учетом того, что используемый в классе метод size() возвращает количество элементов в векторе, размещение контейнера в памяти поясняется на рисунке ниже:

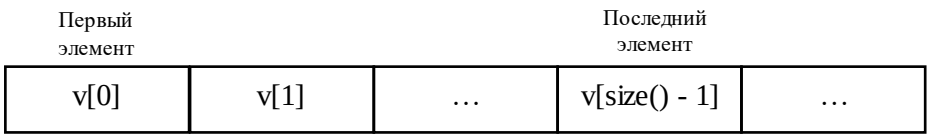

Рисунок 1 – Размещение вектора в памяти

В клиенте main() продемонстрировано использование контейнерного класса.

1.3 Выбор методов и средств решения технического задания

При решении задачи основном акцентом при разработке контейнерного класса Vect реализовать как можно более похожим – с точки зрения его интерфейса и, частично, реализации – на контейнерный класс std::vector из библиотеки STL.

2 Разработка программного обеспечения

2.1 Разработка алгоритмов

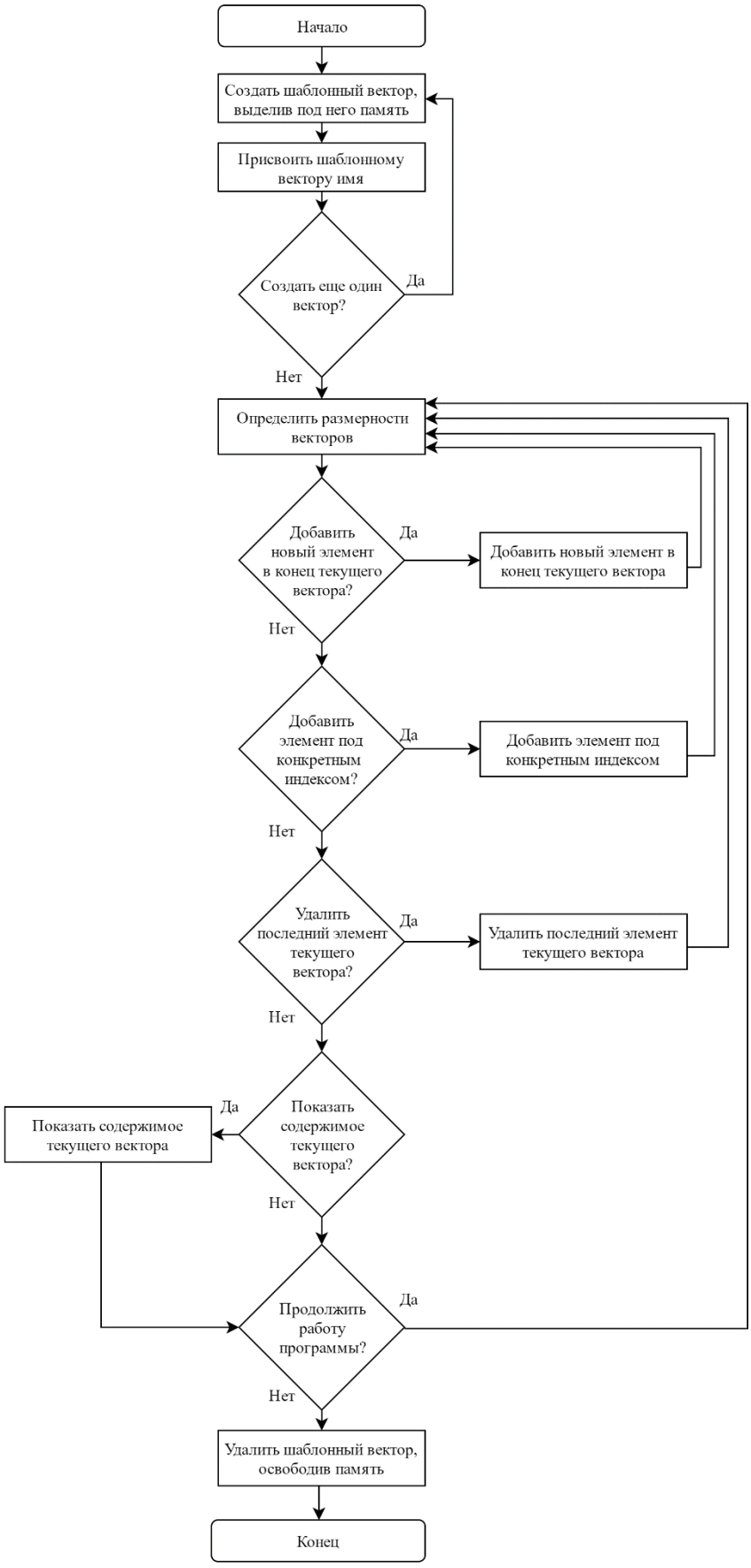

Рисунок 2 – Алгоритм работы программы

Алгоритм работы программы достаточно прост. Сначала создаются объекты шаблонных векторов переменной длины. Затем, при необходимости, над ними выполняются методы по добавлению новых элементов в конец вектора, добавлению элементов по выбранному конкретному индексу, удаление последних элементов из вектора и показ содержимого самого вектора. После чего происходит удаление векторов из памяти. Алгоритм работы системы предусматривает обработку исключительных ситуаций, например, обращение по индексу к несуществующему элементу, удаление последнего элемента вектора, который не содержит элементов.

2.1.1 Разработка классов

Модуль vectorDebug.hpp содержит иерархию классов исключений. В базовом классе VectorDebug определены два метода. Виртуальный метод ErrMsg() обеспечивает вывод по умолчанию сообщения об ошибке (в производных классах этот метод замещается конкретными методами). Метод Continue() определяет стратегию продолжения работы программы после обнаружения ошибки. Стратегия зависит от конфигурации программы. Информация о конфигурации кодируется наличием или отсутствием единственной директивы в начале файла #define DEBUG.

Если лексема DEBUG определена, то программа компилируется в отладочной конфигурации, если не определена – в выпускной конфигурации. Для метода Continue() в отладочной конфигурации выводят сообщение «Debug: program is being continued», после чего работа программы продолжается; В выпускной конфигурации повторно порождается (оператор throw) исключение, которое было первопричиной цепочки событий, завершившихся вызовом данного метода.

В производных классах VectRangeErr и VectPopErr метод ErrMsg() переопределяется с учетом вывода информации о конкретной ошибке. После вывода вызывается метод Continue базового класса.

Модуль vector.hpp содержит определение шаблонного класса Vect. Для управления выделением и освобождением ресурсов класс снабжен методами Allocate() и Destroy(). Они размещены в защищенной части класса, так как относятся к его реализации. Операция new T[n] (где n – количество элементов, которое мы хотим поместить в массив) не только выделяет память, но и инициализирует элементы путем вызова T() — конструктора по умолчанию для типа T. Поэтому в методе Destroy() сначала в цикле вызываются деструкторы ~T() всех элементов массива, а потом операцией delete освобождается память.

В методе Destroy() определен цикл for. Его логическое выражение записано в виде p != last, а не p < last. Подобный способ проверки обусловлен тем, что для произвольной структуры данных (например, для списка) соседние элементы не обязаны занимать подряд идущие ячейки памяти, а могут быть разбросаны в памяти.

Деструктор и конструктор копирования содержат вывод сообщений типа «Здесь был я!», дополненных печатью содержимого markName. Весь этот вывод осуществляется только в

отладочной конфигурации программы. Операция присваивания (конструктор копирования) реализована стандартным способом.

В операции доступа по индексу operator[]() служба внутренней охраны (оператор if) проверяет, не несет ли индекс с собой значение, лежащее вне пределов допустимого диапазона). Если индекс оказывается неблагонадежным, служба бьет тревогу, вызывая исключение типа VectRangeErr. Вызов происходит через анонимный экземпляр класса с передачей конструктору трех аргументов: минимальной и максимальной границы и текущего значения индекса.

В методе получения размера вектора size предусмотрена проверка на лояльность взаимоотношений между переменными first и last. В случае каких-либо ошибок или сбоев указанная проверка весьма полезна. При несоблюдении указанного условия генерируется исключение типа VectorDebug.

Метод insert() (вставка элемента со значением _x в произвольную позицию _P) работает следующим образом. Создается новый вектор размером на единицу больше существующего вектора; адреса размещения нового вектора сохраняются в локальных переменных new_first, new_last. Затем в новый вектор копируется первая часть существующего вектора, начиная с первого элемента и заканчивая элементом, предшествующим элементу с адресом _P.

Поскольку доступ к элементам осуществляется через смещение относительно указателя first, перед копированием требуется вычислить offset – смещение, соответствующее адресу _P. Затем элементу с адресом new_first + offset присваивается значение _x. После этого оставшаяся часть существующего вектора копируется в новый вектор, начиная с позиции new_first + offset + 1.

Затем освобождаются ресурсы, занятые существующим вектором (метод Destroy()), и назначить полям first и last новые значения.

Метод push_back() (вставка элемента со значением _x в конец вектора) получился очень простым благодаря использованию серверной функции insert(). Если вектор пуст, то выделяются ресурсы для хранения одного элемента и этому элементу присваивается значение _x. В противном случае вызывается insert() для вставки элемента в позицию, определенную с помощью end().

В методе pop_back() (удаление элемента из конца вектора) проверяется, не пуст ли вектор. В пустом векторе first = last = 0. Если это так, удалять нечего и, значит, клиент ошибся, затребовав такую операцию. Следовательно, надо ему об этом сообщить: генерируется исключение типа VectPopErr. В противном случае определяется указатель на последний элемент вектора, для него вызывается деструктор ~T() и значение last уменьшается на единицу (учитывая арифметику указателей).

Модуль main.cpp содержит определение глобальной шаблонной функции reverseVecor() и определение функции main. Функция reverseVecor() обеспечивает реверсивный вывод содержимого вектора в поток cout. Попутно она позволяет проверить передачу объектов конкретного класса Vect в качестве аргументов функции.

Функция main() предназначена для тестирования класса Vect.

Обработчики catch имеют параметр объект базового класса VectorDebug, но благодаря полиморфизму они будут перехватывать и исключения производных классов VectRangeErr и VectPopErr, так что в результате будет вызываться конкретный метод ErrMsg() для данного типа ошибки.

После вывода сообщения об ошибке метод ErrMsg() вызывает метод Continue(), работа которого зависит от конфигурации программы. В отладочной конфигурации Continue() сообщает о продолжении работы программы и возвращает управление клиенту. В выпускной конфигурации Continue() порождает исключение повторно.

2.1.2 Программная реализация

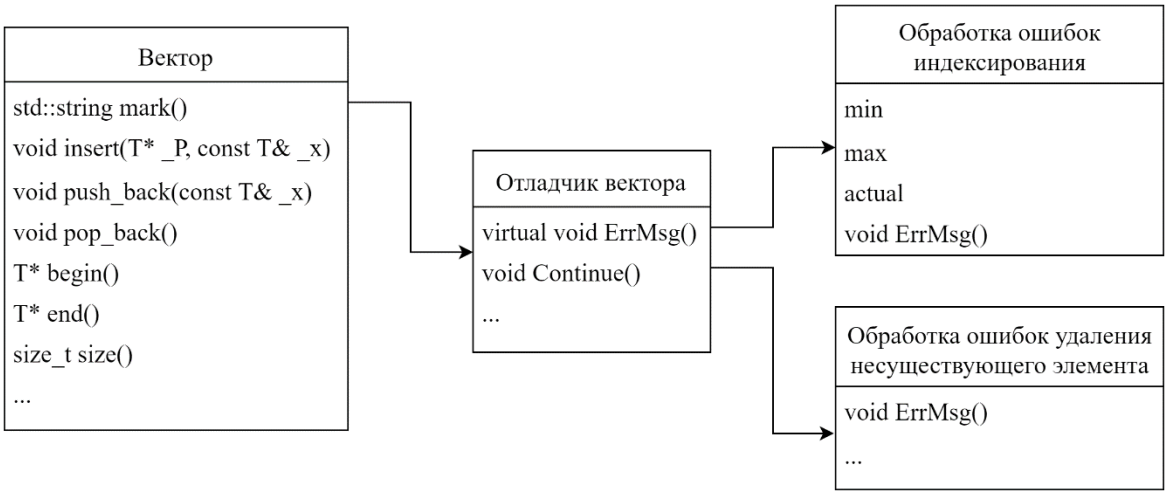

Рисунок 3 – Иерархия классов в программе

В состав интерфейса включены методы:

– void insert(T* _P, const T& _x) – вставка элемента в позицию _P;

– void push_back(const T& _x) – вставка элемента в конец вектора;

– void pop_back() – удаление элемента из конца вектора;

– T* begin() – получение указателя на первый элемент;

– T* end() – получение указателя на элемент, следующий за последним;

– size_t size() – получение размера вектора.

Эти методы имитируют интерфейс контейнерных класса std::vector.

К двум конструкторам, требующимся по заданию, добавим еще конструктор копирования, чтобы обеспечить возможность передачи объектов класса в качестве аргументов для любой внешней функции.

По некоторым параметрам наш класс Vect, однако, будет превосходить класс std::vector. Для целей отладки и обучения мы хотим визуализировать работу таких методов класса, как конструктор копирования и деструктор, путем вывода на терминал соответствующих сообщений. Чтобы эти сообщения были более информативны, мы снабдим каждый объект класса так называемым отладочным именем (поле markName), которое позволит распознавать источник сообщения.

Эти средства могут оказаться полезными при поиске неочевидных ошибок в программе.

Задание требует также предусмотреть генерацию и обработку исключений для возможных ошибочных ситуаций. Наиболее очевидными из таких ситуаций являются: ошибка индексирования, когда клиент пытается получить доступ к несуществующему элементу вектора, и ошибка удаления несуществующего элемента из конца вектора – когда вектор пуст.

Для обработки исключительных ситуаций создана иерархию классов во главе с базовым классом VectError. Производный класс VectRangeErr будет предназначен для обработки ошибок индексирования, а производный класс VectPopErr – для обработки ошибки удаления несуществующего элемента.

2.1.3 Правила именования переменных

В ходе работы над курсовым проектом были выработаны следующие соглашения по именованию переменных и написанию кода:

– все переменные в программе имеют осмысленное название.

– имена всех пользовательских классов начинаются с заглавной буквы

– имена объектов классов и переменных, связанных с пользовательскими классами, начинаются с буквы, совпадающей с начальной буквой в имени класса;

– текст программы снабжен комментариями, важные блоки функций отделены друг от друга с помощью комментариев;

– текст программы структурирован, вложенные операторы смещены вправо относительно заголовков.

2.1.4 Структура программы

В данной программе было разработано четыре класса.

Первый класс вектор (Vect) содержит данные и поведение шаблонного вектора переменной длины. В нем определены методы для конструирования объекта, вставки в него новых элементов, удаление из него элементов, показ содержимого, удаление векторов из памяти и другие вспомогательные методы.

Второй базовый класс отладки вектора (VectorDebug) содержит виртуальный константный метод о выводе ошибки, который переопределяется в производных классах, так, что конкретная реализация метода для вызова будет определяться во время исполнения программы. Его назначение – отладка программы при работе в шаблонными векторами переменной длины.

Третий и четвертый классы VectRangeErr и VectPopErr являются производными (дочерними) от класса вектор (Vect). В каждом из этих классов переопределены виртуальные методы базового класса (VectorDebug), которые сообщают о конкретной возможной исключительной ситуации.

2.1.5 Организация входных и выходных данных

Входными данными программы являются шаблонные векторы переменной длины в виде класса-контейнера.

Выходные данные – это измененное состояние для шаблонных векторов переменной длины.

3 Руководство пользователя

3.1 Инсталляция программы

Для установки программы достаточно скопировать заголовочные файлы (header-файлы) в любую доступную директорию компьютера с созданным проектом.

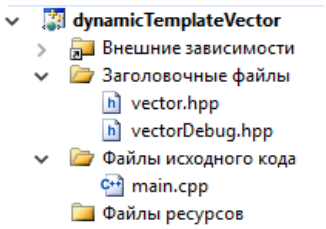

Рисунок 4 – Обозреватель решения в IDE Visual Studio

Пример подключения в клиенте main():

```
#include "vector.hpp"
```

Если по каким-то причинам заголовочный файл будет отсутствовать, то при компиляции модулей программы будет выведено сообщение об ошибке.

3.2 Описание интерфейса

Работа с приложением представляет собой так называемое «консольное приложение», где в роли интерфейсов выступают методы, реализующие собой поведение класса.

3.3 Работа с программой

Работа с шаблонным вектором переменной длины сопровождается правильными вызовами методов, описанных в классе Vect.

Результат работы программы представлен на рисунках ниже:

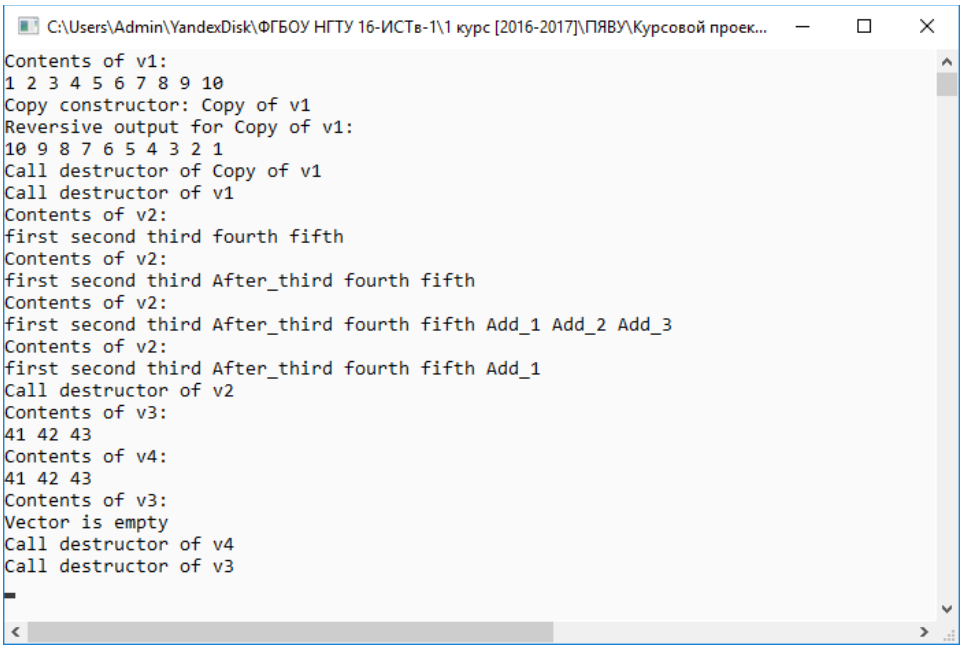

Рисунок 5 – Выполнение программы в отладочной конфигурации

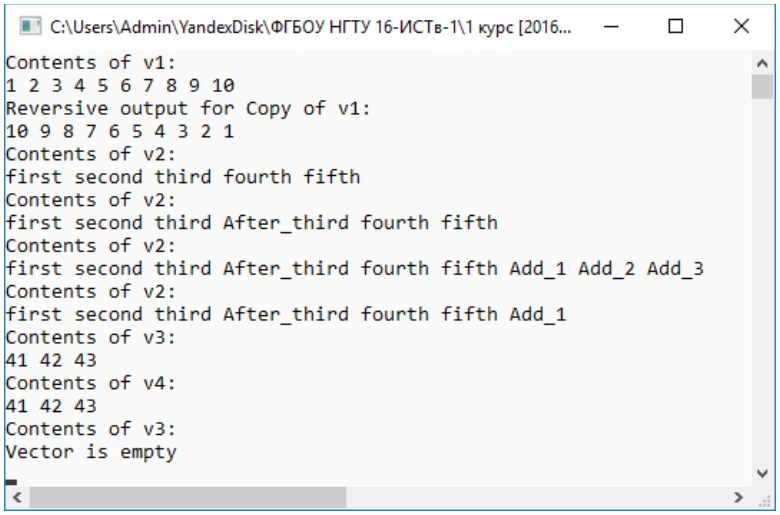

Рисунок 6 – Выполнение программы в выпускной конфигурации

Возможный вариант создания экземпляра класса вектора и вызов его методов:

```
/* Создание шаблонного вектора из целых чисел и 10-и элементов */
Vect <int> v1(10);

/* Присвоение имени вектору v1 */
v1.mark(string("v1"));

/*
 * Получение размера массива
 *
 * где size_t базовый беззнаковый целочисленный тип для хранения
 * максимального размера теоретически возможного объекта любого типа
 *
 */
size_t n = v1.size();

/* Заполнение массива целыми числами от 1 до 10 */
for (size_t i = 0; i < n; ++i) {
        v1[i] = i + 1;
}

/* Показать содержимое вектора v1 */
v1.show();
```

Заключение

В ходе курсового проектирования была разработана программа, реализующая работу шаблонного вектора переменной длины.

Программа позволяет хранить и обрабатывать данные вектора любого типа данных.

Реализованы конструкторы, позволяющие создать вектор нулевой длины или вектор заданного размера.

Описана перегрузка операции индексирования для получения ссылки на соответствующий элемент вектора.

Разработаны методы, позволяющие добавить элемент в конец вектора, добавить элемент в произвольную позицию вектора, удалить элемент из конца вектора.

Предусмотрена генерация и обработка исключений для возможных ошибочных ситуаций.

К достоинствам системы можно отнести простоту работы с программой и удобную отладку при работе с векторами.

Данный проект может дальше развиваться, можно добавлять новые методы, реализующие алгоритмы обработки векторов и, при необходимости, расширять класс для работы с векторами.

Задание, поставленное на курсовое проектирование, выполнено в полном объеме. Проект полностью соответствует техническому заданию.

Список используемой литературы

1 ГОСТ 34.602-89 Стандарты информационной технологии. Техническое задание на создание автоматизированной системы;

2 Крупник А.Б. – Самоучитель С++. – СПб.: Питер, 2005. - 252 с.: ил.;

3 Лафоре Р. – Объектно-ориентированное программирование в С++. Классика Computer Science. 4-е изд. – СПб.: Питер, 2016. – 928 с.: ил. – (Серия «Классика computer science»);

4 Макконнелл С. – Совершенный код. Мастер-класс / Пер. с англ. – М. : Издательство "Русская редакция", 2010. – 896 стр. : ил.;

5 РД 50-34.698-90 Автоматизированные системы. Требования к содержанию документов;

6 Стандарт предприятия СТП 1-У-НГТУ-2004. Общие требования к оформлению пояснительных записок дипломных и курсовых проектов. – Нижний Новгород.: НГТУ, 2004. – 22 с.;

7 Столяров А.В. – Введение в язык Си++: Учебное пособие. – 3-е изд. – М.:МАКС Пресс, 2012. – 128 с.: ил.;

8 Шилдт Г. – Полный справочник по С++, 4-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2016. – 800 с.: ил. – Парал. Тит. англ.;

9 Эккель Б. – Философия С++. Введение в стандартный С++. 2-е изд. – СП.: Питер, 2004. – 572 с.: ил.;

Листинг программы

```
/* * * * * * * * * * * *
 * * * vector.hpp  * * *
 * * * * * * * * * * * */
#ifndef _VECTOR_
#define _VECTOR_

        #include <iostream>
        #include <string>
        #include "vectorDebug.hpp"

        /* Шаблонный класс Vect */
        template <class T> class Vect {
                public:
                        Vect() : first(0), last(0) {}
                        Vect(size_t _n, const T& _v = T()) {
                                Allocate( _n );
                                for (size_t i = 0; i < _n; ++i)  {
                                        *(first + i) = _v;
                                }
                        }
                        Vect( const Vect& ); // конструктор копирования
                        Vect& operator =(const Vect&); // перегрузка операции присваивания

                        /* Деструктор */
                        ~Vect() {
                                #ifdef DEBUG
                                        cout << "Call destructor of " << markName << endl;
                                #endif
                        Destroy();
                        first = 0,
                    last = 0;
                }
```

```
/* Установка отладочного имени */
void mark(std::string&  name) {
        markName = name;
}

/* Получение отладочного имени */
std::string  mark() const {
        return markName;
}

/* Получение размера вектора */
size_t size() const;

/* Получение указателя на 1-й элемент */
T* begin() const {
        return first;
}

/* Получение указателя на n + 1 элемент */
T* end() const {
        return last;
}

/* Операция индексирования */
T& operator[](size_t i);

/* Вставка элемента в позицию _P */
void insert(T* _P, const T& _x);

/* Вставка элемента элемента в конец вектора */
void push_back(const T& _x);

/* Удаление элемента из конца вектора */
void pop_back();

/* Вывод содержимого вектора */
void show() const;
```

```
protected:
	/* Распределение вектора в памяти */
	void Allocate( size_t _n ) {
		first = new T [_n * sizeof(T)];
		last = first + _n;
	}

	/* Удаление вектора из памяти */
	void Destroy() {
		for ( T* p = first; p != last; ++p ) {
			p->~T();
		}
		delete [] first;
	}

	T* first; // указатель на 1-й элемент
	T* last;  // указатель на элемент, следующий за последним
	std::string markName;
};

/*
 * Конструктор копирования
 *
 */
template <class T> Vect<T>::Vect(const Vect& other) {
	size_t n = other.size();
	Allocate( n );
	for (size_t i = 0; i < n; ++i) {
		*(first + i) = *(other.first + i);
	}
	markName = string("Copy of ") + other.markName;
	#ifdef DEBUG
		cout << "Copy constructor: " << markName << endl;
	#endif
}
```

```
/*
 * Операция присваивания
 *
 */
template <class T> Vect<T>& Vect<T>::operator =(const Vect& other) {
	if (this == &other) {
		return *this;
	}
	Destroy();
	size_t n = other.size(); Allocate(n);
	for (size_t i = 0; i < n; ++i) {
		*(first + i) = *(other.first + i);
	}
	return *this;
}



/*
 * Получение размера вектора
 *
 */
template <class T> size_t Vect<T>::size() const {
	if (first > last) {
		throw VectorDebug();
	}
	return (0 == first ? 0 : last - first);
}

/*
 * Операция доступа по индексу
 *
 */
template <class T> T& Vect<T>::operator[](size_t i) {
	if (i < 0 || i > ( size() - 1 )) {
		throw VectRangeErr(0, last - first - 1, i);
	}
	return ( *( first + i ) );
}
```

```
/*
 * Вставка элемента со значением _x в позицию _P
 *
 */
template <class T> void Vect<T>::insert(T* _P, const T& _x) {
	size_t n = size() + 1; // новый размер
	T* new_first = new T [n * sizeof(T)];
	T* new_last = new_first + n;
	size_t offset = _P - first; // смещение
	for (size_t i = 0; i < offset; ++i) {
		*(new_first + i) = *(first + i);
   }
	*( new_first + offset ) = _x;
	for (size_t i = offset; i < n; ++i) {
		*(new_first + i + 1) = *(first + i);
	}
	Destroy();
	first = new_first;
	last = new_last;
}

/*
 * Вставка элемента в конец вектора
 *
 */
template <class T> void Vect<T>::push_back(const T& _x) {
	if (!size()) {
		Allocate(1);
		*first = _x;
	} else {
		insert( end(), _x );
	}
}
```

```
/*
 * Удаление элемента из конца  вектора
 *
 */
template <class T> void Vect<T>::pop_back() {
	if (last == first) {
		throw VectPopErr();
	}
	T* p = end() - 1;
	p->~T();
	last--;
}

/*
 * Вывод содержимого вектора
 *
 */
template <class T> void Vect<T>::show() const {
	cout << "Contents of " << markName << ":" << endl;
	size_t n = size();
	if (n != 0) {
		for (size_t i = 0; i < n; ++i) {
			cout << *(first + i) << " ";
		}
	} else {
		cout << "Vector is empty";
	}
	cout << endl;
}

#endif /* _VECTOR_ */
/* * * * * * * * * * * */
```

```
/* * * * * * * * * * * * * * *
 * * * vectorDebug.hpp * * *
 * * * * * * * * * * * * * * */
#ifndef _VECTOR_DEBUG_
#define _VECTOR_DEBUG_
        #include <iostream>
        #define DEBUG
        class VectorDebug {
                public:
                        VectorDebug() {}
                        virtual void ErrMsg() const {
                                std::cerr << "Error with Vect object.\n";
                        }
                        void Continue() const {
                                #ifdef DEBUG
                                        std::cerr << "Debug: program is being continued.\n";
                                #else
                                        throw;
                                #endif
                        }
                };

                class VectRangeErr : public  VectorDebug {
                public:
                        VectRangeErr(int _min, int  _max, int _actual) :
                                                min(_min),  max(_max), actual(_actual) {}
                        void ErrMsg() const {
                                std::cerr << "Error of index:  ";
                                std::cerr << "possible  range: " << min << " - " << max << ", ";
                                std::cerr << "actual index:  " << actual << std::endl;
                                Continue();
                        }

                private:
                        int min, max;
                        int actual;
                };
```

```
        class VectPopErr : public VectorDebug {
            public:
                void ErrMsg() const {
                    std::cerr << "Error of pop\n";
                    Continue();
                }
        };
#endif /* _VECTOR_DEBUG_ */
/* * * * * * * * * * * * */

/* * * * * * * * * * *
 * * * main.cpp  * * *
 * * * * * * * * * * */
#include <iostream>
#include <string>
#include <cstdlib>
#include "vector.hpp"

using namespace std;

template <class T> void reverseVecor( Vect <T> v ) {
   std::cout << "Reversive output for " << v.mark() << ":" << endl;
   size_t n = v.size();
   for (int i = n - 1; i >= 0; --i) {
            std::cout << v[i] << " ";
        }
   std::cout << endl;
}

int main() {

        try {
            /* Создание шаблонного вектора из целых чисел и 10-и элементов */
            Vect <int> v1(10);

            /* Присвоение имени вектору v1 */
            v1.mark(string("v1"));
```

```
        /*
         * Получение размера массива
         *
         * где size_t базовый беззнаковый целочисленный  тип для хранения
         * максимального размера теоретически возможного объекта любого типа
         *
         */
        size_t n = v1.size();

        /* Заполнение  массива целыми числами от 1 до 10 */
        for (size_t i = 0; i < n; ++i) {
                v1[i] = i + 1;
        }

        /* Показать содержимое вектора v1 */
        v1.show();

        /* Вызов пользовательской функции  для реверса вектора */
        reverseVecor(v1);
}
catch (VectorDebug& vre)  {
        vre.ErrMsg();
}

 try {
        /* Создание массива initStr из пяти элементов типа строка*/
        string initStr[5]  = { "first",  "second", "third",  "fourth",  "fifth"  };

        /* Создание шаблонного  вектора из 5-и элементов типа строка */
         Vect <string> v2(5);

        /* Присвоение имени вектору v2 */
        v2.mark(string("v2"));

        /* Получение размера массива */
        size_t n = v2.size();

        /* Заполнение  шаблонного  вектора v2 элементами массива initStr */
```

```
        for (size_t i = 0; i < n; ++i) {
                v2[i] = initStr[i];
        }

        /* Показ содержимого вектора v2 */
        v2.show();

        /* Вставка элемента под индексом 4 */
        v2.insert(v2.begin() + 3, "After_third");

        /* Показ содержимого вектора v2 */
        v2.show();

        /* Попытка вывести седьмой элемент приведет к порождению исключения */
        // cout << v2[6] << endl;

        /* Добавление трех новых элементов в вектор v2 */
        v2.push_back("Add_1"); v2.push_back("Add_2"); v2.push_back("Add_3");

        /* Показ содержимого вектора v2 */
        v2.show();


        /* Удаление двух элементов из конца вектора v2 */
         v2.pop_back(); v2.pop_back();

        /* Показ содержимого вектора v2 */
        v2.show();
 }
 catch (VectorDebug& vre) {
        vre.ErrMsg();
}

 try {
        /* Создание шаблонного вектора v3 из целочисленных элементов */
         Vect <int> v3;

        /* Присвоение имени вектору v3 */
```

```
        v3.mark(string("v3"));

        /* Добавление трех новых элементов в вектор v3 */
        v3.push_back(41); v3.push_back(42); v3.push_back(43);

        /* Показ содержимого вектора v3 */
        v3.show();

        /* Создание шаблонного вектора v4 из целочисленных элементов */
        Vect <int> v4;

        /* Присвоение имени вектору v4 */
        v4.mark(string("v4"));

        /* Создание шаблонного вектора v4 из целочисленных элементов */
        v4 = v3;

        /* Показ содержимого вектора v4 */
        v4.show();

        /*
         * Удаление большего числа элементов чем в нем есть
         * на самом деле приведет к ошибке
         */
        v3.pop_back(); v3.pop_back();
        v3.pop_back(); // v3.pop_back();

        /* Показ содержимого вектора v3 */
        v3.show();
    }
    catch (VectorDebug& vre)  {
        vre.ErrMsg();
    }

    system("pause >nul");
}
/* * * * * * * * * * */
```