

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ**
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Нижегородский государственный университет им. Н.И. Лобачевского»
Национальный исследовательский университет**

**Институт информационных технологий, математики и механики
Кафедра математического обеспечения и суперкомпьютерных технологий**

**ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ
«Реализация списка»**

Выполнил:
студент группы 361706-1
Резанцев Сергей Алексеевич
_____ Подпись

Научный руководитель:
ассистент каф. МОСТ ИИТММ
_____ Лебедев И. Г.

Нижний Новгород
2018 г.

Содержание	
Введение	3
Постановка задачи	4
Руководство пользователя	5
Руководство программиста	6
Описание структуры программы	6
Описание структур данных	6
Описание алгоритмов	7
Оценка сложности некоторых алгоритмов	8
Заключение	10
Литература	11

1. Введение

Целью данной лабораторной работы является реализовать список, и для создания этой программы понадобится создать классы с шаблонами и различными функциями.

Очень часто, при разработке приложений, оперирующих с большим количеством входных данных, возникает вопрос об их хранении во время выполнения программы. Приводить все из них не имеет смысла, остановлюсь лишь на массивах. Несомненно, данный тип решает вопрос хранения данных, однако, очевидно, что он не лишен недостатков. Главным из них, несомненно, является его фиксированный размер. Это свойство не поддается изменению даже у динамически созданных массивов, что довольно часто заставляет программистов, использующих исключительно их, выделять память "с запасом". Ну а во-первых, даже "запас" ограничен, и никто не может дать гарантии, что и его будет достаточно, а во-вторых, наоборот, "запаса" может хватить настолько, что немалая часть отведенной программе памяти будет занята понапрасну. Данную проблему решает другой тип хранения данных, которому и посвящена эта статья - связанный список динамических переменных, или проще - динамический список. Компоненты добавляются и удаляются во время выполнения программы, и их количество зависит исключительно от размера доступной памяти. Однако, за это преимущество приходится расплачиваться недостатком - если в случае с массивом, мы в любой момент получаем доступ к любому компоненту, то в случае со списком, в один момент времени нам доступны максимум 3 компонента (это зависит от способа представления списка в программе). В большинстве случаев, это очень даже приемлемая цена ...

2. Постановка задачи

Реализовать класс TList и TNode для работы со списками. Поля классов должны быть закрыты.

В классах обязательно должны присутствовать методы:

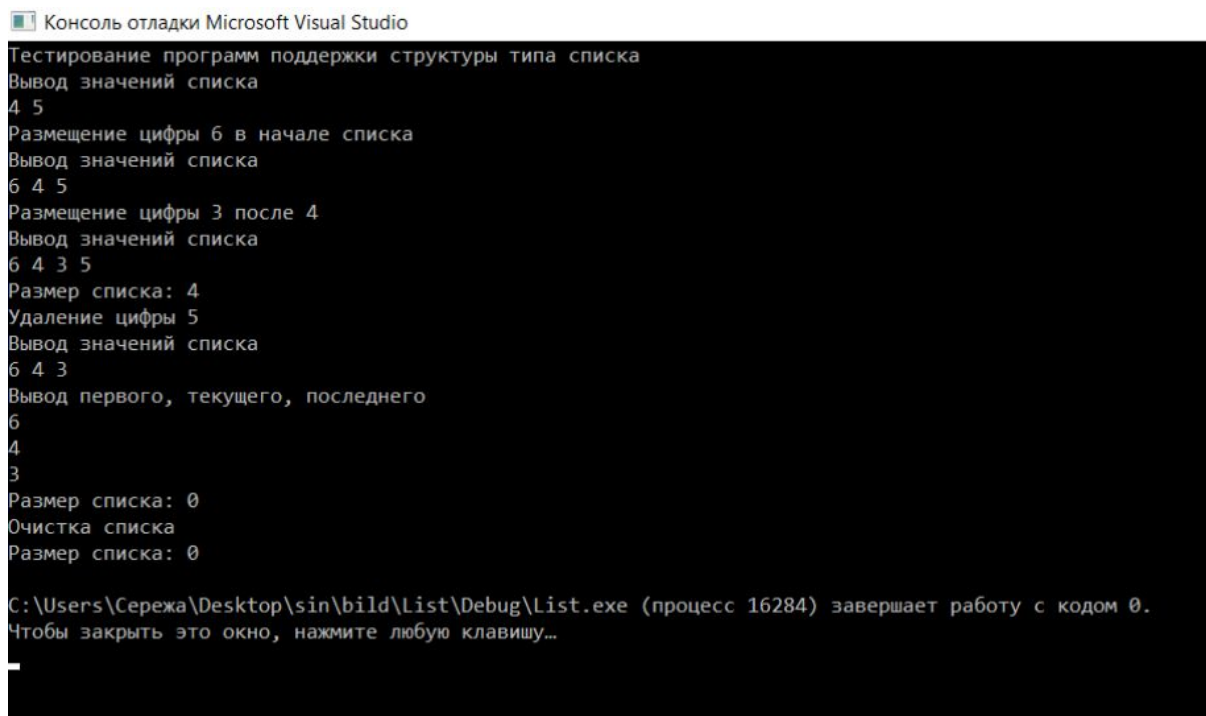
- 1)добавление элемента в начало, в конец и текущего
- 2)удаление элемента в начале, в конце и текущего

Должны быть реализованы конструкторы: копирования и инициализатор.

3. Руководство пользователя

Чтобы начать работу с программой запустите приложение List.

На экране появится следующее:



```
Консоль отладки Microsoft Visual Studio
Тестирование программ поддержки структуры типа списка
Вывод значений списка
4 5
Размещение цифры 6 в начале списка
Вывод значений списка
6 4 5
Размещение цифры 3 после 4
Вывод значений списка
6 4 3 5
Размер списка: 4
Удаление цифры 5
Вывод значений списка
6 4 3
Вывод первого, текущего, последнего
6
4
3
Размер списка: 0
Очистка списка
Размер списка: 0

C:\Users\Сергея\Desktop\sin\bild>List\Debug>List.exe (процесс 16284) завершает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу...
_
```

Затем программа завершится.

4. Руководство программиста

4.1. Описание структуры программы

В решении 2 проекта: главный проект и библиотека. В библиотеке 2 исполнительных файла и 2 заголовочных. В главном проекте только 1 исполнительный файл.

4.2. Описание структур данных

В программе описаны классы

TNode:

В нем 2 поля:

T data; - переменная для хранения данных

TNode<T>* next; - указатель на следующий узел списка

И реализованы следующие функции:

TNode(T d = 0, TNode<T>* n = NULL) - конструктор по умолчанию

TNode(const TNode<T> &A) - конструктор копирования

T GetData() - возвращает данные

TNode<T>* GetNext() - получить указатель на следующий узел

void SetData(T d) - задать данные

void SetNext(TNode<T>* n) - установить указатель на след. звено списка

TList:

В нем 1 поля:

TNode<T>* list; - указатель на начало списка

И реализованы следующие функции:

TList(); - конструктор по умолчанию

TList(TList<T> &list2); - конструктор копирования

void PutBegin(T A); - положить элемент в начало списка

`void PutAfter(T A, TNode<T>* node);` - положить элемент после какого-то элемента

`void PutEnd(T A);` - положить элемент в конец списка

`int GetSize();` - узнать размер списка

`TNode<T>* Search(T A);` - найти указатель на элемент в списке

`void Delete(T A);` - удалить элемент из списка

`void Clean();` - очистить список

`T GetBegin();` - взять элемент из начала списка

`T GetCurrent(TNode<T>* A);` - взять текущий элемент

`T GetEnd();` - взять элемент из конца списка

4.3. Описание алгоритмов

1)Добавление элемента списка в начало.

При добавлении элемента в начало списка мы создаем указатель на объект класса `TNode`. Затем выделяем память под объект этого класса и с помощью конструктора с параметрами для `TNode`, передав туда значение, которое необходимо положить в начало списка, и указатель на текущее начало, создаем очередное звено списка. Указатель на начало списка переопределяем на только что добавленный элемент.

2)Удаление элемента списка из начала.

Для удаления звена списка из начала выполняем проверку на пустоту списка. Если список пуст, то бросаем исключение. Иначе создаем указатель `*A` на объект класса `TNode`, которому присваиваем значение текущего начала списка. Создаем временную переменную `temp`, в которую записываем значение, хранящееся в первом элементе списка. Начало списка устанавливаем на следующий за удаляемым элемент. Удаляем указатель `*A` для того, чтобы очистить память, занимаемую бывшим первым элементом.

3)Добавление звена списка в конец.

При добавлении звена списка в конец проверяем, есть ли элементы в списке. Если есть, то создаем указатель *В на объект класса TNode, в него записываем значение начала списка. В цикле ищем текущий последний элемент. Как только конец списка будет найден, выделяем память под новое звено списка и с помощью конструктора по умолчанию TNode создаем его. Устанавливаем для текущего последнего элемента указатель на следующий – только что созданный.

В том случае, если в списке не было элементов, то указателю на начало списка присваиваем значение, указывающее на звено, созданное с помощью конструктора TNode.

4) Удаление звена списка из конца.

Для удаления звена списка из конца выполняем проверку на пустоту списка. Если список пуст, то бросаем исключение. Иначе необходимо проверить: в списке больше одного элемента или ровно один. Для этого смотрим на следующий за первым элемент. Если указатель на него равен нулю, то мы возвращаем только данные из первого элемента списка, начало списка обнуляем.

Если элементов больше одного, то создаем указатель *А на объект класса TNode. Ищем в цикле предпоследнее звено списка. Создаем еще один указатель *В на объект класса TNode. В него записываем указатель на последнее звено списка. Получаем данные из этого звена. Удаляем указатель *В и тем самым освобождаем память, занимаемую бывшим последним элементом. Для *А, устанавливаем в качестве следующего за ним 0, т.к. он теперь стал последним.

4.4. Оценка сложности некоторых алгоритмов

Операционная система - Windows 10

Характеристики компьютера:

Intel Core i5

8 GB DDR3 L Memory

128 GB SSD + 1000 GB HDD

Кол-во элементов в списке	Т работы PutBegin()	Т работы PutEnd
10000	0	0

100000	0	0.001
1000000	0	0.26

Как видно из таблицы метод PutBegin(), работает значительно быстрее метода PutEnd(). Это происходит за счет того, что при добавлении элемента в начало списка, к памяти мы обращаемся всего 1 раз, за $O(1)$, а для добавления элемента в конец списка, мы должны пройти по всем n элементам списка, тем самым мы n раз обратимся к памяти, и сложность алгоритма равна $O(n)$, что существенно замедляет работу.

Делаем вывод, при работе со списками в приоритете стоит использовать метод PutBegin().

5. Заключение

В ходе выполнения лабораторной работы была разработана библиотека, реализующая шаблонный класс списка. Она позволяет при работе со списком выполнять базовые операции извлечения/добавление элементов списка.

Проведены эксперименты и установлен наиболее оптимальный метод работы со списками.

Предоставлено описание примера работы со списком в разделе «Руководство пользователя».

Также разработаны и доведены до успешного выполнения тесты, проверяющие корректность методов классов TList и TNode.

6. Литература

1. Васильев А.Н. Самоучитель С++ с примерами и задачами. -СПб.: Наука и Техника, 2016. -480с.
2. Т. А. Павловская С/С++ Программирование на языке высокого уровня. - СПб.:Питер, 2011. - 461 с.
- 3.Крапенко С. Н. и др. Методы объектно-ориентированного программирования. <http://e-learning.unn.ru/course/view.php?id=251>.
- 4.Страуструп. Б. Курс «Язык программирование С++ для профессионалов» <http://www.intuit.ru/studies/courses/98/98/info>
- 5.Гергель В.П. Методические материалы по курсу “Методы программирования 2”:
[<http://www.itmm.unn.ru/files/2018/10/Primer-1.1.-Struktury-hraneniya-mnozhestva.pdf>], 2015.