Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

О Т Ч Е Т (В А Р И А Н Т 1 2)

по лабораторной работе № 1 0

Название: **Создание контейнеров**

Дисциплина: Объектно-ориентированное программирование

Студент ИУ6-23Б 27.05.2023 В.К. Залыгин
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель А.М.Минитаева
(Подпись, дата) (И.О. Фамилия)

Москва, 2023

Цель работы

Изучить основные принципы работы с контейнерами и способы их построения. Разработать собственный контейнер на основе предложенной структуры данных, используя механизм наследования.

Задание

Моделировать стек, в качестве элементов которого могут использоваться целые числа и слова. Операции: добавление элемента, удаление элемента, печать элементов стека. Создать класс-потомок, который содержит процедуру определения элемента, имеющего максимальную длину при печати. Тестировать полученную модель.

Разработать собственную иерархию классов, готовые контейнеры Qt не использовать. Пользовательский интерфейс для работы с моделью реализовать на Qt. В отчете представить диаграмму классов и обосновать выбранную структуру представления данных.

Проект программы

Существует несколько способов реализовать механизм хранения нескольких типов данных в контейнере. В рамках данной лабораторной работы было решено придерживаться модели, при которой существует некоторый базовый класс элемента стека, а возможность хранения нескольких типов данных реализуется через создание наследуемых классов-обёрток элементов над требуемыми типами.

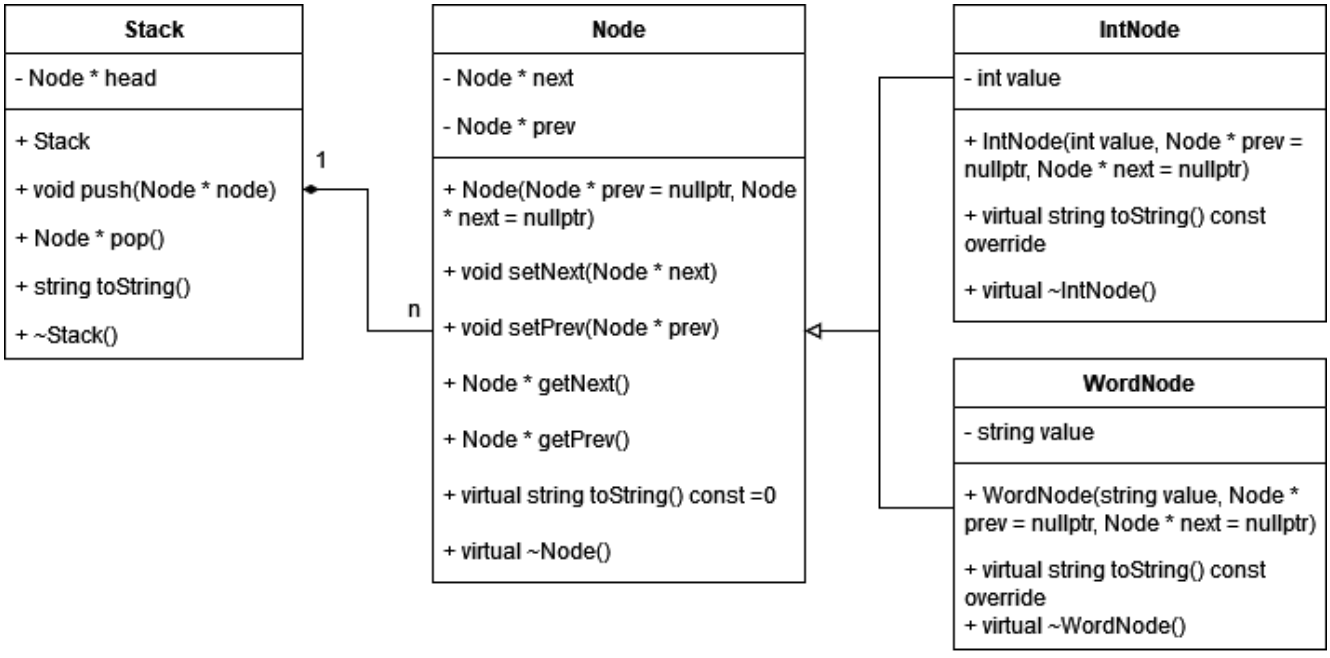
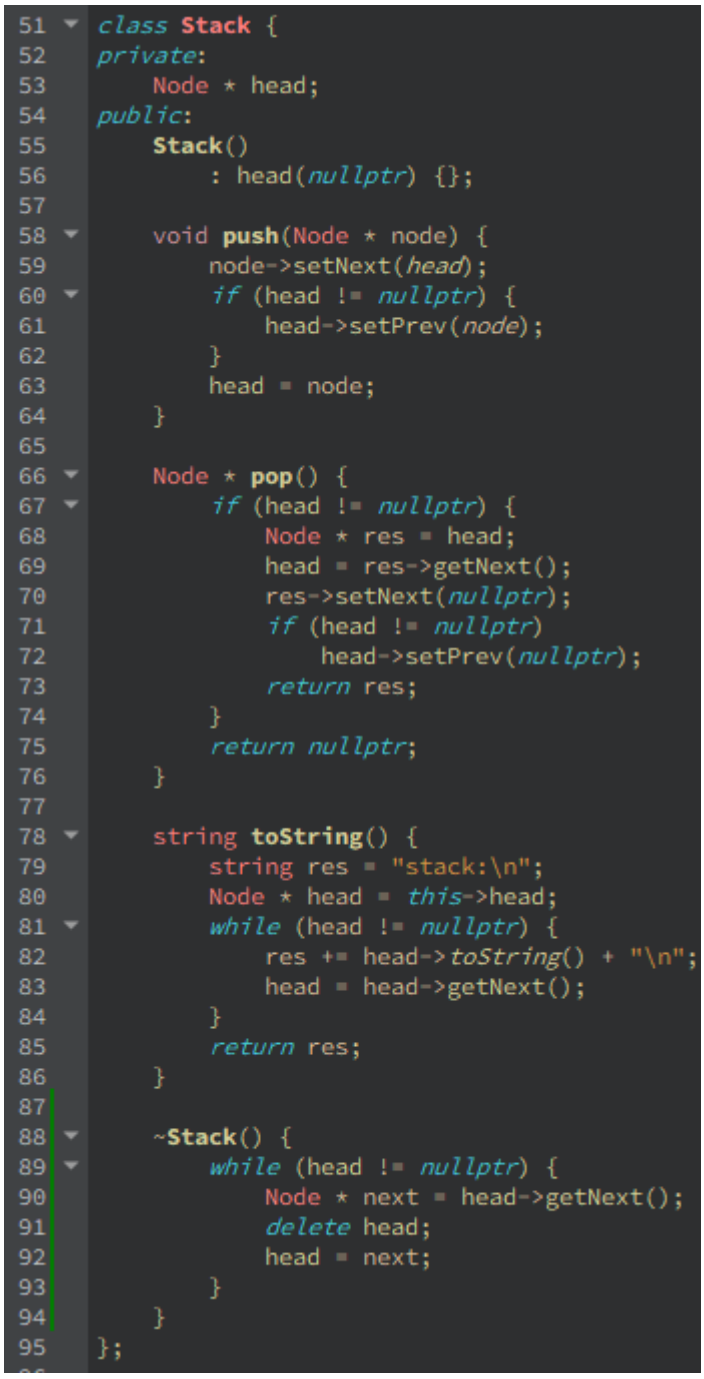


Рисунок 1 - диаграмма классов

Текст программы

```cpp
#include <QCoreApplication>
#include <string>
#include <iostream>

using std::string;

class Node {
private:
    Node * next;
    Node * prev;
public:
    Node(Node * prev = nullptr, Node * next = nullptr)
        : next(next), prev(prev) {};

    void setNext(Node * next) { this->next = next; }
    void setPrev(Node * prev) { this->prev = prev; }

    Node * getNext() { return next; }
    Node * getPrev() { return prev; }

    virtual string toString() const =0;
    virtual ~Node() {};
};

class IntNode : public Node{
private:
    int value;
public:
    IntNode(int value, Node * prev = nullptr, Node * next = nullptr)
        : value(value), Node(prev, next) {};

    virtual string toString() const override {
        return std::to_string(value);
    }
    virtual ~IntNode() {};
};

class WordNode : public Node {
private:
    string value;
public:
    WordNode(string value, Node * prev = nullptr, Node * next = nullptr)
        : value(value), Node(prev, next) {};

    virtual string toString() const override {
        return value;
    }
    virtual ~WordNode() {};
};
```

Рисунок 2 - код программы



```
class Stack {
private:
    Node * head;
public:
    Stack()
        : head(nullptr) {};

    void push(Node * node) {
        node->setNext(head);
        if (head != nullptr) {
            head->setPrev(node);
        }
        head = node;
    }

    Node * pop() {
        if (head != nullptr) {
            Node * res = head;
            head = res->getNext();
            res->setNext(nullptr);
            if (head != nullptr)
                head->setPrev(nullptr);
            return res;
        }
        return nullptr;
    }

    string toString() {
        string res = "stack:\n";
        Node * head = this->head;
        while (head != nullptr) {
            res += head->toString() + "\n";
            head = head->getNext();
        }
        return res;
    }

    ~Stack() {
        while (head != nullptr) {
            Node * next = head->getNext();
            delete head;
            head = next;
        }
    }
};
```

Рисунок 3 - код программы

```cpp
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Stack stack;

    std::cout << "avaliable actions:\n\t1. push new value to stack\n\t2. pop value from stack\n\t3. print stack\n\t4. exit\n";
    while (true) {
        std::cout << "choose action:? ";
        int a = 0;
        std::cin >> a;

        int num;
        string word;
        Node * popped;

        switch (a) {
        case 1:
            std::cout << "a number (1) or a word (2):? ";
            std::cin >> a;
            switch (a) {
            case 1:
                std::cin >> num;
                stack.push(new IntNode(num));
                break;
            case 2:
                std::cin >> word;
                stack.push(new WordNode(word));
                break;
            default:
                std::cout << "unknown action. try again\n";
                break;
            }
            break;
        case 2:
            popped = stack.pop();
            std::cout << "popped element: " << popped->toString() << "\n";
            delete popped;
            break;
        case 3:
            std::cout << stack.toString();
            break;
        case 4:
            return 0;
        default:
            std::cout << "unknown action. try again\n";
            break;
        }
    }

    return a.exec();
}

```

Рисунок 4 - код программы

Тестовые данные

```
avaliable actions:
        1. push new value to stack
        2. pop value from stack
        3. print stack
        4. exit
choose action:? 1
a number (1) or a word (2):? 1
123
choose action:? 3
stack:
123
choose action:? 1
a number (1) or a word (2):? 2
hello!
choose action:? 3
stack:
hello!
123
choose action:? 2
popped element: hello!
choose action:? 3
stack:
123
choose action:?
```

Рисунок 5 - пример работы программы

Вывод

Были изучены основные принципы работы с контейнерами и способы их построения. Разработан собственный контейнер на основе предложенной структуры данных с помощью механизма наследования.