**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Кафедра безопасности информационных систем**

**ОТЧЁТ**

по практической работе № 6 на тему:   
**«Разработка программ моделирования работы стека и очереди.»**

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы ИСТ-312, Серафимович Г.П.

«14» Октября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/Г.П. Серафимович /

Принял: к.ф.-м.н., доцент, И.А. Моисеев

«15» Октября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/ И.А. Моисеев /

**Основная часть**

**Цель работы:**

Целью этой работы является исследование и практическая реализация основных структур данных: стека и очереди, а также понимание их внутреннего функционирования на примерах. Работа направлена на укрепление навыков создания и использования линейных списков, а также на моделирование реальных процессов, таких как обработка данных в стеке и симуляция работы процессорного конвейера с использованием очереди.

**Теоретическая часть:**  
 Стек — это структура данных, которая работает по принципу LIFO (Last In, First Out), что означает, что последним помещённый элемент извлекается первым. Основные операции для стека включают:

1. push: добавление элемента в стек.
2. pop: удаление верхнего элемента стека.
3. peek: просмотр верхнего элемента стека без его удаления.

Использование стека актуально для задач, где требуется "отложенное выполнение", например, при работе с обратным порядком данных (например, когда нужно выводить элементы в обратной последовательности).  
Очередь — это структура данных, работающая по принципу FIFO (First In, First Out), что означает, что первым добавленный элемент извлекается первым. Основные операции для очереди включают:

1. enqueue: добавление элемента в очередь.
2. dequeue: удаление первого элемента из очереди.

Очереди широко используются в моделировании различных процессов, например, в операционных системах при организации выполнения команд с разделением на стадии (конвейерная обработка).

**Практическая часть:**

**Задача 1.**

**Реализация стека для обратного вывода слов из предложения**

**Необходимо создать стек и поместить в него слова из следующего предложения: «Разработать алгоритм и программу умножения двух целочисленных матриц». Каждое слово следует добавлять в стек в обратном порядке (начиная с конца предложения). Затем из стека нужно последовательно вывести слова в прямом порядке (по одному за раз).**

**План выполнения задачи:**

**• Поместить слова предложения в стек в обратном порядке.**

**• Вывести слова, сохраняя их последовательность.**

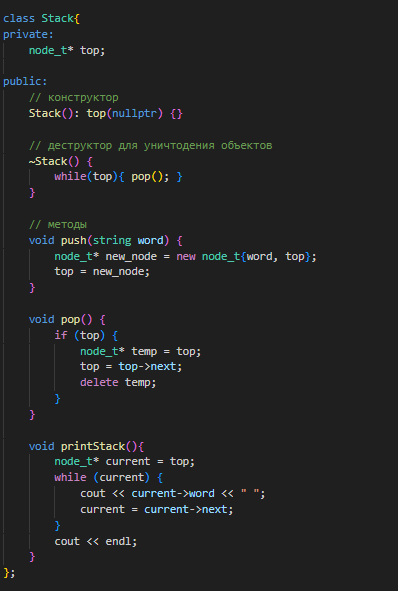


Рисунок 1.1 stack

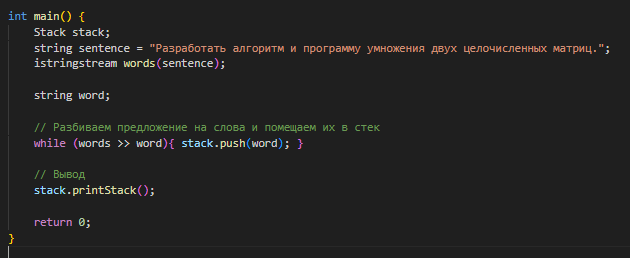


Рисунок 1.2 main

**Результат выполнения задачи:**



Рисунок 1.3 Результат выполнения задачи №1.

**Задача 2.** Моделирование работы процессора с конвейерной архитектурой с использованием очереди.

Необходимо создать модель процессора, который имеет конвейерную структуру, состоящую из пяти этапов обработки команд (C1–C5). Каждая команда должна пройти через все пять этапов: загрузка, декодирование, получение операндов, выполнение и запись результата.

Следует реализовать очередь, которая будет моделировать последовательную обработку 10 команд в конвейере.

План выполнения задачи:

• Создать модель конвейера на основе очереди, где каждая команда последовательно проходит через 5 этапов обработки.

• Пропустить через конвейер 10 команд.

• После завершения обработки каждой команды выводить её на экран.

Результат выполнения задачи:

**Задача 2**

#include <iostream>

#include <queue>

#include <vector>

#include <string>

using namespace std;

int main() {

const int numStages = 5;

const int numCommands = 10;

vector<queue<string>> pipeline(numStages);

vector<string> commands;

// Инициализация команд

for (int i = 1; i <= numCommands; ++i) {

commands.push\_back("Команда " + to\_string(i));

}

int cycle = 1;

int cmdIndex = 0;

while (cmdIndex < numCommands || !pipeline[numStages - 1].empty()) {

if (cmdIndex < numCommands) {

pipeline[0].push(commands[cmdIndex]);

cmdIndex++;

}

for (int stage = numStages - 1; stage > 0; --stage) {

if (!pipeline[stage - 1].empty()) {

pipeline[stage].push(pipeline[stage - 1].front());

pipeline[stage - 1].pop();

}

}

cout << "Цикл " << cycle << ": ";

for (int stage = 0; stage < numStages; ++stage) {

if (!pipeline[stage].empty()) {

cout << "С" << (stage + 1) << " [" << pipeline[stage].front() << "] ";

} else {

cout << "С" << (stage + 1) << "[ пусто] ";

}

}

cout << endl;

if (!pipeline[numStages - 1].empty()) {

cout << pipeline[numStages - 1].front() << " завершена." << endl;

pipeline[numStages - 1].pop();

}

cycle++;

}

return 0;

}

Рисунок 2.1main

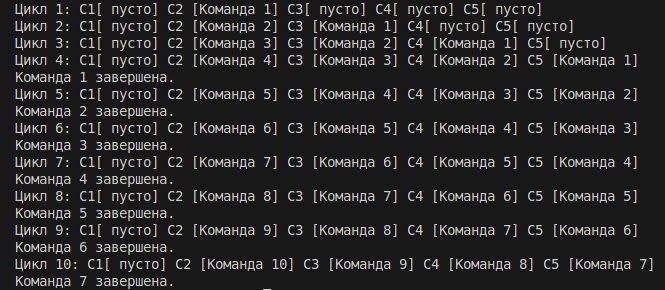


Рисунок 2.2 Результат для выполненя задачи №2.

**Выводы:**

Работа со стеком продемонстрировала, как эффективно использовать эту структуру данных для обработки информации в обратном порядке. Мы создали стек, который позволил помещать слова предложения в обратной последовательности, а затем выводить их в правильном порядке. Это иллюстрирует принцип LIFO (последний пришел — первый вышел), который часто применяется в алгоритмах, связанных с парсингом, отменой действий и рекурсивными вызовами.

**Приложение**

Задача №1.

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

typedef struct Node{

string word;

Node\* next;

}node\_t;

class Stack{

private:

node\_t\* top;

public:

// конструктор

Stack(): top(nullptr) {}

// деструктор для уничтодения объектов

~Stack() {

while(top){ pop(); }

}

// методы

void push(string word) {

node\_t\* new\_node = new node\_t{word, top};

top = new\_node;

}

void pop() {

if (top) {

node\_t\* temp = top;

top = top->next;

delete temp;

}

}

void printStack(){

node\_t\* current = top;

while (current) {

cout << current->word << " ";

current = current->next;

}

cout << endl;

}

};

int main() {

Stack stack;

string sentence = "Разработать алгоритм и программу умножения двух целочисленных матриц.";

istringstream words(sentence);

string word;

// Разбиваем предложение на слова и помещаем их в стек

while (words >> word){ stack.push(word); }

// Вывод

stack.printStack();

return 0;

}

Задача №2.

#include <iostream>

#include <queue>

#include <vector>

#include <string>

using namespace std;

int main() {

const int numStages = 5;

const int numCommands = 10;

vector<queue<string>> pipeline(numStages);

vector<string> commands;

// Инициализация команд

for (int i = 1; i <= numCommands; ++i) {

commands.push\_back("Команда " + to\_string(i));

}

int cycle = 1;

int cmdIndex = 0;

while (cmdIndex < numCommands || !pipeline[numStages - 1].empty()) {

if (cmdIndex < numCommands) {

pipeline[0].push(commands[cmdIndex]);

cmdIndex++;

}

for (int stage = numStages - 1; stage > 0; --stage) {

if (!pipeline[stage - 1].empty()) {

pipeline[stage].push(pipeline[stage - 1].front());

pipeline[stage - 1].pop();

}

}

cout << "Цикл " << cycle << ": ";

for (int stage = 0; stage < numStages; ++stage) {

if (!pipeline[stage].empty()) {

cout << "С" << (stage + 1) << " [" << pipeline[stage].front() << "] ";

} else {

cout << "С" << (stage + 1) << "[ пусто] ";

}

}

cout << endl;

if (!pipeline[numStages - 1].empty()) {

cout << pipeline[numStages - 1].front() << " завершена." << endl;

pipeline[numStages - 1].pop();

}

cycle++;

}

return 0;

}