Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №11.3**

Дисциплина: основы алгоритмизации и программирования

Тема: “Стек"

**Вар.20**

Выполнил работу

студент группы ИВТ-20-2Б

Кузнецов Н.Д

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2020

**Цель задачи**

1) Получить практические навыки работы с однонаправленными списками;

2) Получить практические навыки работы с двунаправленными списками;

3) Получить практические навыки работы со стеками;

4) Получить практические навыки работы со стеками.  
Для достижения поставленной цели необходимо решить следующие задачи:

* Разработать структуру однонаправленного списка.
* Организовать ввод-вывод данных.
* Реализовать функцию удаления элементов из списка.
* Разработать программу.

**Постановка задачи**

Реализовать стек. Тип информационного поля int. Удалить из стека последний элемент с четным информационным полем.

**Анализ задачи**

1. Для решения задачи используются типы данных **bool**, **int**, класс **vector**.
   1. Для считывания и запоминания элементов стека используется **vector** типа int..
   2. Поле data, списка так же типа int.

…  
 struct List

{

int data;

List\* next\_block;

List\* prev\_block;

};  
 …

* 1. Для корректного ввода длины стека используется функция GetSize, которая не принимает значений и возвращает тип данных **int**.

int size;

while (!(cin >> size) || (cin.peek() != '\n') || (size < 1))

{

cin.clear();

while (cin.get() != '\n')

{

cout << "Введите корректное число!\t";

}

}

return size;

* + 1. Для корректного ввода значения поля data реализована функция GetInt, аналогично функции GetSize (см.п.1.3).

int GetInt()

{

int num;

while (!(cin >> num) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n')

{

cout << "Введите корректное число!\t";

}

}

return num;

}

1. Для хранения данных, вводимых пользователем, реализован стек через двунаправленный список (см.п.1.2).
   1. Для заполнения стека реализована функция Create\_list, которая в качестве параметра принимает длину стека, возвращает указатель на первый элемент стека.

List\* first\_element = nullptr;

List\* current\_element;

if (size == 1)

{

cout << "Введите 0-й элемент стека:\t";

current\_element = new List;

current\_element->data = GetInt();

first\_element = current\_element;

current\_element->next\_block = nullptr;

current\_element->prev\_block = nullptr;

}

else

{

cout << "Введите 0-й элемент стека:\t";

current\_element = new List;

current\_element->data = GetInt();

first\_element = current\_element;

current\_element->next\_block = nullptr;

current\_element->prev\_block = nullptr;

for (int i = 1; i < size; i++)

{

cout << "Введите " << i << "-й элемент стека:\t";

List\* new\_element = new List;

current\_element->next\_block = new\_element;

new\_element->prev\_block = current\_element;

current\_element = new\_element;

current\_element->data = GetInt();

current\_element->next\_block = nullptr;

}

}

return first\_element;

}

* 1. Для удаления элемента из стека описана функция Pop, которая принимает указатель на первый элемент и возвращает указатель на 1 элемент в стеке.

List\* Pop(List\* first\_element)

{

List\* tmp = first\_element;

while (tmp->next\_block != nullptr)

{

tmp = tmp->next\_block;

}

if (tmp->prev\_block != nullptr)

{

List\* predtmp = tmp->prev\_block;

delete tmp;

predtmp->next\_block = nullptr;

}

else

{

delete tmp;

tmp = nullptr;

first\_element = nullptr;

}

return first\_element;

}

* 1. Для добавления элемента в стек реализована функция Push\_list, которая принимает указатель на первый элемент и значение, которое нужно добавить (value). Функция возвращает указатель на первый элемент.

List\* Push\_list(List\* first\_element, int value)

{

List\* current\_element = first\_element;

if (first\_element == nullptr)

{

first\_element = new List;

first\_element->data = value;

first\_element->next\_block = nullptr;

first\_element->prev\_block = nullptr;

}

else

{

{

while (current\_element->next\_block != nullptr)

current\_element = current\_element->next\_block;

}

List\* new\_element = new List;

new\_element->data = value;

new\_element->next\_block = nullptr;

new\_element->prev\_block = current\_element;

current\_element->next\_block = new\_element;

}

return first\_element;

}

* 1. Для вывода элементов стека используется **vector**, как временное хранилище данных. Циклом while элементы поочередно выводятся на консоль, удаляются из стека и добавляются в vector, аналогично происходит заполнение стека обратно.

void Print\_list(List\*& first\_element)

{

vector <int> tmp;

List\* current\_element = first\_element;

// проход до последнего элемента стека

int i = 0;

if (first\_element == nullptr)

{

cout << "Список пуст!" << endl;

return;

}

while (current\_element->next\_block != nullptr)

{

current\_element = current\_element->next\_block;

i++;

}

while (i >= 0)

{

cout << i << "-й элемент стека:\t" << current\_element->data << endl;

i--;

List\* temp = current\_element;

current\_element = current\_element->prev\_block;

tmp.push\_back(temp->data);

first\_element = Pop(first\_element);

}

int j = tmp.size() - 1;

while (j >= 0)

{

first\_element = Push\_list(first\_element, tmp[j]);

j--;

}

}

* 1. Для удаления последнего элемента стека с четным полем разработана функция Delete\_element, которая принимает ссылку на указатель первого элемента стека (низка стека). Функция возвращает false в случае, если элемент в стеке с таким полем не найден, иначе функция возвращает true. Аналогично п.2.4 все данные из стека переносятся в vector (**tmp**). C помощью цикла for из tmp удаляется последний элемент стека с четным значением. Аналогично п.2.4 элементы из vector’a переносятся обратно в стек.

bool Delete\_element(List\*& first\_element)

{

vector <int> tmp;

bool check = false;

List\* current\_element = first\_element;

//проход до вершины стека

int i = 0;

while (current\_element->next\_block != nullptr)

{

current\_element = current\_element->next\_block;

i++;

}

while (i >= 0)

{

List\* temp = current\_element;

current\_element = current\_element->prev\_block;

tmp.push\_back(temp->data);

first\_element = Pop(first\_element);

i--;

}

for (int j = 0; j < tmp.size(); j++)

{

if (tmp[j] % 2 == 0)

{

tmp.erase(tmp.begin() + j);

check = true;

break;

}

}

for (int k = tmp.size() - 1; k >= 0; k--)

{

first\_element = Push\_list(first\_element, tmp[k]);

}

return check;

}

1. Для хранения данных реализован стек по принципу работы двунаправленного списка (см.п.1.2).
2. Структура стека имеет 3 поля, поле информации (int), 2 указателя: на следующий и на прошлый элемент стека (см.п.1.2).
3. Для ввода-вывода используются операторы cin, cout, для этого подключена соответствующая библиотека (см.п.2.4).
4. В функции **main** переменной **size** присваивается результат выполнения функции **GetSize**, далее создается указатель на первый элемент стека и ему присваивается результат выполнения функции **Create\_list**, в которую передается переменная **size**. Далее стек выводится пользователю на консоль. Оператором if проверяется результат выполнения функции **Delete\_element**. В случае истины пользователю выводится в консоль измененный стек, в противном же случае, пользователю выводится сообщение о том, что элемента с четным значением поля в стеке нет.

int main()

{

setlocale(LC\_ALL, "ru");

cout << "Введите размер стека:\t";

int size = GetSize();

List\* stack = Create\_list(size);

cout << "============================\n";

Print\_list(stack);

cout << "============================\n";

if (Delete\_element(stack))

{

Print\_list(stack);

}

else

{

cout << "Четных элементов в стеке нет!\n";

Print\_list(stack);

cout << "============================\n";

}

}

**Код через struct**

#include <iostream>

#include <vector>

using namespace std;

struct List

{

int data;

List\* next\_block;

List\* prev\_block;

};

int GetSize()

{

int size;

while (!(cin >> size) || (cin.peek() != '\n') || (size < 1))

{

cin.clear();

while (cin.get() != '\n')

{

cout << "Введите корректное число!\t";

}

}

return size;

}

int GetInt()

{

int num;

while (!(cin >> num) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n')

{

cout << "Введите корректное число!\t";

}

}

return num;

}

List\* Create\_list(int size)

{

List\* first\_element = nullptr;

List\* current\_element;

if (size == 1)

{

cout << "Введите 0-й элемент стека:\t";

current\_element = new List;

current\_element->data = GetInt();

first\_element = current\_element;

current\_element->next\_block = nullptr;

current\_element->prev\_block = nullptr;

}

else

{

cout << "Введите 0-й элемент стека:\t";

current\_element = new List;

current\_element->data = GetInt();

first\_element = current\_element;

current\_element->next\_block = nullptr;

current\_element->prev\_block = nullptr;

for (int i = 1; i < size; i++)

{

cout << "Введите " << i << "-й элемент стека:\t";

List\* new\_element = new List;

current\_element->next\_block = new\_element;

new\_element->prev\_block = current\_element;

current\_element = new\_element;

current\_element->data = GetInt();

current\_element->next\_block = nullptr;

}

}

return first\_element;

}

List\* Pop(List\* first\_element)

{

List\* tmp = first\_element;

while (tmp->next\_block != nullptr)

{

tmp = tmp->next\_block;

}

if (tmp->prev\_block != nullptr)

{

List\* predtmp = tmp->prev\_block;

delete tmp;

predtmp->next\_block = nullptr;

}

else

{

delete tmp;

tmp = nullptr;

first\_element = nullptr;

}

return first\_element;

}

List\* Push\_list(List\* first\_element, int value)

{

List\* current\_element = first\_element;

if (first\_element == nullptr)

{

first\_element = new List;

first\_element->data = value;

first\_element->next\_block = nullptr;

first\_element->prev\_block = nullptr;

}

else

{

{

while (current\_element->next\_block != nullptr)

current\_element = current\_element->next\_block;

}

List\* new\_element = new List;

new\_element->data = value;

new\_element->next\_block = nullptr;

new\_element->prev\_block = current\_element;

current\_element->next\_block = new\_element;

}

return first\_element;

}

void Print\_list(List\*& first\_element)

{

vector <int> tmp;

List\* current\_element = first\_element;

// проход до последнего элемента стека

int i = 0;

if (first\_element == nullptr)

{

cout << "Список пуст!" << endl;

return;

}

while (current\_element->next\_block != nullptr)

{

current\_element = current\_element->next\_block;

i++;

}

while (i >= 0)

{

cout << i << "-й элемент стека:\t" << current\_element->data << endl;

i--;

List\* temp = current\_element;

current\_element = current\_element->prev\_block;

tmp.push\_back(temp->data);

first\_element = Pop(first\_element);

}

int j = tmp.size() - 1;

while (j >= 0)

{

first\_element = Push\_list(first\_element, tmp[j]);

j--;

}

}

bool Delete\_element(List\*& first\_element)

{

vector <int> tmp;

bool check = false;

List\* current\_element = first\_element;

//проход до вершины стека

int i = 0;

while (current\_element->next\_block != nullptr)

{

current\_element = current\_element->next\_block;

i++;

}

while (i >= 0)

{

List\* temp = current\_element;

current\_element = current\_element->prev\_block;

tmp.push\_back(temp->data);

first\_element = Pop(first\_element);

i--;

}

for (int j = 0; j < tmp.size(); j++)

{

if (tmp[j] % 2 == 0)

{

tmp.erase(tmp.begin() + j);

check = true;

break;

}

}

for (int k = tmp.size() - 1; k >= 0; k--)

{

first\_element = Push\_list(first\_element, tmp[k]);

}

return check;

}

int main()

{

setlocale(LC\_ALL, "ru");

cout << "Введите размер стека:\t";

int size = GetSize();

List\* stack = Create\_list(size);

cout << "============================\n";

Print\_list(stack);

cout << "============================\n";

if (Delete\_element(stack))

{

Print\_list(stack);

}

else

{

cout << "Четных элементов в стеке нет!\n";

Print\_list(stack);

cout << "============================\n";

}

}

**Код через STL**

#include <iostream>

#include <vector>

using namespace std;

struct List

{

int data;

List\* next\_block;

List\* prev\_block;

};

int GetSize()

{

int size;

while (!(cin >> size) || (cin.peek() != '\n') || (size < 1))

{

cin.clear();

while (cin.get() != '\n')

{

cout << "Введите корректное число!\t";

}

}

return size;

}

int GetInt()

{

int num;

while (!(cin >> num) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n')

{

cout << "Введите корректное число!\t";

}

}

return num;

}

List\* Create\_list(int size)

{

List\* first\_element = nullptr;

List\* current\_element;

if (size == 1)

{

cout << "Введите 0-й элемент стека:\t";

current\_element = new List;

current\_element->data = GetInt();

first\_element = current\_element;

current\_element->next\_block = nullptr;

current\_element->prev\_block = nullptr;

}

else

{

cout << "Введите 0-й элемент стека:\t";

current\_element = new List;

current\_element->data = GetInt();

first\_element = current\_element;

current\_element->next\_block = nullptr;

current\_element->prev\_block = nullptr;

for (int i = 1; i < size; i++)

{

cout << "Введите " << i << "-й элемент стека:\t";

List\* new\_element = new List;

current\_element->next\_block= new\_element;

new\_element->prev\_block = current\_element;

current\_element = new\_element;

current\_element->data = GetInt();

current\_element->next\_block = nullptr;

}

}

return first\_element;

}

List\* Pop(List\*& first\_element)

{

List\* tmp = first\_element;

while (tmp->next\_block != nullptr)

{

tmp = tmp->next\_block;

}

List\* current\_element = tmp;

tmp = tmp->prev\_block;

delete current\_element;

tmp->next\_block = nullptr;

return first\_element;

}

List\* Push\_list(List\* first\_element, int value)

{

List\* current\_element = first\_element;

if (first\_element == nullptr)

{

first\_element = new List;

first\_element->data = value;

first\_element->next\_block = nullptr;

first\_element->prev\_block = nullptr;

}

else

{

{

while (current\_element->next\_block != nullptr)

current\_element = current\_element->next\_block;

}

List\* new\_element = new List;

new\_element->data = value;

new\_element->next\_block = nullptr;

new\_element->prev\_block = current\_element;

current\_element->next\_block = new\_element;

}

return first\_element;

}

List\* Print\_list(List\*& first\_element)

{

vector <int> tmp;

int i = 0;

List\* current\_element = first\_element;

// проход до последнего элемента стека

while (current\_element->next\_block != nullptr)

{

current\_element = current\_element->next\_block;

i++;

}

while (current\_element != nullptr)

{

cout << i << "-й элемент стека:\t" << current\_element->data << endl;

i--;

tmp.push\_back(current\_element->data);

first\_element = Pop(first\_element);

}

int j = tmp.size();

while (j >= 0)

{

first\_element = Push\_list(first\_element, tmp[j]);

j--;

}

return first\_element;

}

bool Delete\_element(List\*& first\_element)

{

vector <int> tmp;

bool check = false;

List\* current\_element = first\_element;

//проход до вершины стека

while (current\_element->next\_block != nullptr)

{

current\_element = current\_element->next\_block;

}

while (current\_element != nullptr)

{

tmp.push\_back(current\_element->data);

first\_element = Pop(first\_element);

}

int j = tmp.size() - 1;

for (j; j >= 0; j--)

{

if (tmp[j] % 2 == 0)

{

tmp.erase(tmp.begin() + j);

check = true;

break;

}

}

for (int i = tmp.size(); i >= 0; i--)

{

first\_element = Push\_list(first\_element, tmp[i]);

}

return check;

}

int main()

{

setlocale(LC\_ALL, "ru");

cout << "Введите размер стека:\t";

int size = GetSize();

List\* stack = Create\_list(size);

stack = Print\_list(stack);

cout << "============================\n";

if (Delete\_element(stack))

{

cout << "============================\n";

Print\_list(stack);

}

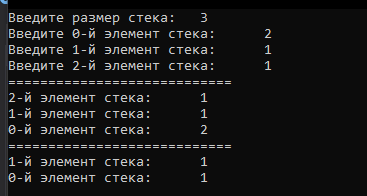
else

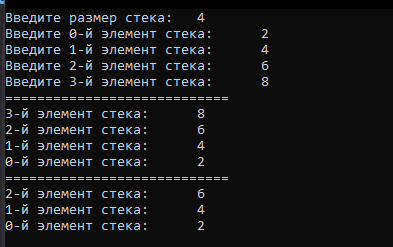
{

cout << "Четных элементов в стеке нет!\n";

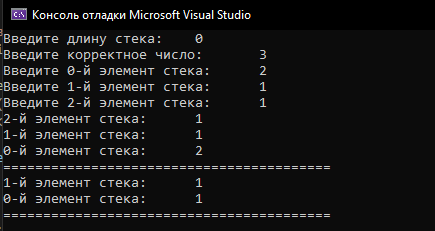
}}

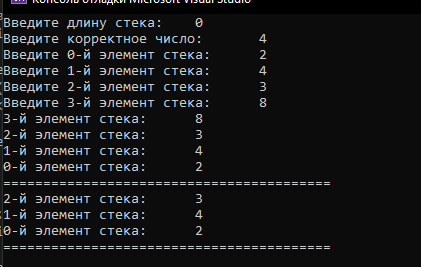
**Работа кода struct**





**Работа кода STL**





**Блок-схема**

