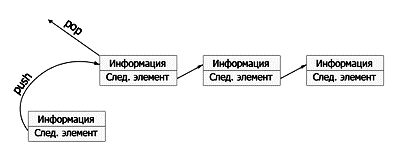
Стек

Стек – это **структура данных**, представляющая собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»). Часто приводят аналогию с магазином в огнестрельном оружии (стрельба начнётся с патрона, заряженного последним).

Стек выглядит примерно так:

[](http://mksite.me/wp-content/uploads/2014/08/400px-Stack_preview.png)

Например, запишем поразрядно очень **большое число в стек**(например, 9^100).

Начнём с того, что нам нужно **создать нашу структуру**:

struct number {

int x; // Здесь будет 1 цифра

number \*adr; // Здесь адрес следующего элем.

};

Затем создадим два указателя типа number.

number \*top = NULL, \*p;

Теперь допустим, что мы это число считываем из файла по одной цифре. (file – указатель на файл).

do {

p = new number; // Создаём новый элемент списка

p->adr = top; // Привязываем к предыдущему элементу

file >> p->x; // Считываем цифру из файла

top = p; // Смещаем вершину стека

} while(!file.eof()); // Повторяем цикл пока есть еще что-то в файле

Таким образом **создают динамическую структуру данных и добавляют данные в стек**. Как Вы можете видеть здесь нет никаких ограничений по размеру, как например есть в массиве. Если исходные данные записывались так: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Тогда список будет выглядеть следующим образом: 9, 8, 7, 6, 5, 4, 3, 2, 1, 0.

Реализуем **вывод всех элементов из списка**.

if (!top){ //условие при котором стек пуст  
puts ("Stack Pyst!");  
}   
while ( top!= NULL){  
 cout << " " << top->value << '\t' << top << '\n';<<  
 top = top -> adress;  
}

Теперь попробуем вставить элемент в середину нашего стека. Например, между цифрой 7 и 6 мы вставим цифру 15.

p = top; // Чтобы не потерять вершину стека, будем использовать дополнительный указатель

while(p->x != 7 && p->adr != NULL) { // Пока мы не дошли до цифры 7, или если её нет, то до конца списка

p = p->adr; // Смещаемся вниз

}

number \*new\_el = new number; // Создаём новый элемент списка

new\_el->x = 15; // Сохраняем в него число 15

new\_el->adr = p->adr; // Привязываем новый элемент с элементов с цифрой 8

p->adr = new\_el; // Привязываем элемент с цифрой 7 к новому элементу

Теперь список будет выглядеть следующим образом: 9, 8, 7, 15, 6, 5, 4, 3, 2, 1, 0.

Реализуем удаление произвольного элемента. Допустим список у нас как в предыдущем примере. И нам нужно удалить наше вставленное число 15.

while(p->x != 15 && p->adr != NULL) { // Пока мы не дошли до числа 15, или если его нет, то до конца списка

p = p->adr; // Смещаемся вниз

}

number \*p2 = p->adr; // Сохраняем адрес элемента в новый указатель

p->adr = p->adr->adr; // Привязываем адрес к следующему элементу после удаляемого

delete p2; // Удаляем элемент

Полный текст модуля ([ссылка](http://cpp.sh/9a3gv))

#include "stdafx.h"

#include <iostream>

#include <stdio.h>

#include <fstream>

using namespace std;

struct number

{

int value; // Элемент стека

number \*adr; // Адрес следующего элемента

};

int main()

{

number \*top = NULL, \*p, \*tail;

ifstream file("c:\\temp\\file.txt");

if (!file.is\_open())

perror("File opening error");

else

{

// Добавление элементов в стэк

do

{

p = new number; // Создаём новый элемент списка

p->adr = top; // Привязываем к предыдущему элементу

file >> p->value; // Считываем цифру из файла

cout << "Read value: " << p->value << "\tAddress: " << p << '\n';

top = p; // Смещаем вершину стека

} while (!file.eof()); // Повторяем цикл пока есть еще что-то в файле

file.close();

tail = top; // сохраняем адрес начала списка

// Печать стека

top = tail; // переходим в конец списка

if (!top) // Условие при котором стек пуст

{

puts("Stack is empty!");

}

cout << "\nValue\t Address\n";

while (top != NULL)

{

cout << " " << top->value << '\t' << top << '\n';

top = top->adr;

}

cout << endl;

top = tail; // переходим в конец списка

// Перейти к 5 по счёту элементу стека

// вставить элемент посередине нашего списка

p = top; // Чтобы не потерять вершину стека, будем использовать дополнительный указатель

while (p->value != 7 && p->adr != NULL) { // Пока мы не дошли до цифры 7, и если её нет, то до конца списка

p = p->adr; // Смещаемся вниз

}

number \*new\_el = new number; // Создаём новый элемент списка

new\_el->value = 15; // Сохраняем в него число 15

new\_el->adr = p->adr; // Привязываем новый элемент с элементов с цифрой 8

p->adr = new\_el; // Привязываем элемент с цифрой 7 к новому элементу

// Печать стека

top = tail; // переходим в конец списка

if (!top) // Условие при котором стек пуст

{

puts("Stack is empty!");

}

cout << "\nValue\t Address\n";

while (top != NULL)

{

cout << " " << top->value << '\t' << top << '\n';

top = top->adr;

}

cout << endl;

top = tail; // переходим в конец списка

// удаление элемента из стека

//p = start;

while (p->adr != NULL) // Пока не дошли до конца списка

{

if (p->adr->value != 15) // Пока не дошли до числа 15

p = p->adr; // Смещаемся вниз

else

break;

}

number \*p2 = p->adr; // Сохраняем адрес элемента в новый указатель

p->adr = p->adr->adr; // Привязываем адрес к следующему элементу после удаляемого

delete p2; // Удаляем элемент

// Печать стека

top = tail; // переходим в конец списка

if (!top) // Условие при котором стек пуст

{

puts("Stack is empty!");

}

cout << "\nValue\t Address\n";

while (top != NULL)

{

cout << " " << top->value << '\t' << top << '\n';

top = top->adr;

}

cout << endl;

top = tail; // переходим в конец списка

}

return 0;

}

Очередь

Очередь – это структура данных, представляющая собой список элементов, организованных по принципу FIFO (англ. first in — first out, «первым пришёл — первым вышел»). Очередь мы можем сравнить с массивом. Отличие только в возможностях.

Мы создадим список, который мы заполним очень большим числом, а потом выведем его на экран.

Итак, создаём структуру:

struct number

{

int x;

number \*adr;

}

Отличие очереди от стека в том, что нам нужно выделить память в начале, под первый элемент.

number \*top, \*p, \*first; // Создаем указатели

top = new number; // Выделяем память под первый элемент

top->adr = NULL; // Поскольку он и последний, то следующий элемент NULL

file>>top->x; // Считываем данные из файла

first = top; // Сохраняем адрес первого элемента

do

{

p = new number; // Выделяем память под новый элемент

p->adr = NULL; // Поскольку он теперь последний, то следующий элемент NULL

file>>p->x; // Считываем данные из файла

top->adr = p; // Присоединяем новый элемент к предыдущему

top = p; // Смещаем наш следующий указатель

} while(!file.eof()); // Повторяем цикл пока есть данные в файле

Таким образом заполняется список данными из файла.

Реализуем вывод списка на экран.

top = first; // Берем адрес вершины очереди

do

{

cout<<top->x; // Выводим данные

top = top->adr; // Смещаемся на второй элемент

} while(top != NULL); // Выполняем цикл пока не дошли до конца очереди

Реализуем добавление элемента в середину списка.

top = first;

while(top != 7 && top != NULL)

{

top = top->adr;

}

p = new number;

p->x = 16;

p->adr = top->adr;

top->adr = p;