МИНОБРНАУКИ РОССИИ

ФГБОУ ВО Череповецкий государственный университет

Институт информационных технологий

Кафедра: МПО ЭВМ

Дисциплина: Структуры и алгоритмы обработки данных

ЛАБОРАТОРНАЯ РАБОТА № 1

Линейные списки

Выполнила: Пикалова А.С.,

студент гр. 1ПИб-01-21оп.

Проверил: Пышницкий К.М.

Череповец, 2023 г.

Выполнить задания, используя для пpедставления очеpедей и стеков:

а) массивы; б) динамические списки.

Требования к программам:

1. Количество элементов исходных линейных списков заранее не определено и задается случайным образом. При дальнейшей обработке считается, что количество элементов списка не известно, т.е. обработка производится, пока не достигнут конец списка.

2. Программа должна сформировать исходные линейные списки, вывести их содержимое на экран (при этом данные из списков не должны быть потеряны), произвести обработку и вывести содержимое итогового списка на экран.

Даны стек и очередь целых чисел от 0 до 1000. Из общих элементов сформировать новый стек.

**Теория:**

**Стек** (англ. stack — стопка; читается стэк) — абстрактный тип данных, представляющий собой список элементов, организованных по принципу **LIFO** (англ. last in — first out, «последним пришёл — первым вышел»).

Стек состоит из ячеек, которые представлены в виде структуры, содержащей какие-либо данные и указатель типа данной структуры на следующий элемент. Блок вида **«Данные/\*next»** и есть наша ячейка. \*next, как мы видим, указывает на следующий элемент, другими словами, указатель \*next хранит адрес следующей ячейки. Указатель \*TOP указывает на вершину стек, то есть хранит её адрес.

**Организация данных** с помощью стека эффективна, когда нужно реализовать:

* обмен данными между методами приложения с помощью параметров;
* синтаксический анализ разнообразных выражений.

В программировании стек можно **реализовывать** разными способами, например:

* в виде статического массива;
* в виде динамического массива;
* в виде односвязного списка;
* в виде двусвязного списка.

Над стеком и его элементами можно выполнять следующие операции:

* добавление элемента в стек (push);
* вытягивание (удаление) элемента из стека (pop);
* просмотр элемента в вершине стека без его вытягивания из стека;
* проверка, пустой ли стек;
* определение количества элементов в стеке;
* отображение (просмотр) всего стека.

Если стек пуст, то значение указателя является ссылка на NULL. Перед началом заполнения стека его нужно обнулить

**Очередь** – это динамическая структура данных, которая состоит из набора элементов, которые размещены последовательно друг за другом. При этом добавление элементов осуществляется с одной стороны, а удаление (вытягивание) – с другой стороны.

Очередь работает по принципу **FIFO** (First In — First Out), то есть «первым пришел – первым вышел» и иногда очередь называют циклической памятью или списком типа FIFO.

Элементы, находящиеся в середине очереди, напрямую недоступны для обработки. Элементы могут поступать в очередь и обрабатываться в ней неравномерно, поэтому длина ее будет изменяться. В частности, возможен случай, когда очередь окажется пустой. Признаком того может служить равенство I = J. В процессе формирования или добавления элементов значение индекса J увеличивается, поэтому возможна ситуация, когда J становиться больше N. Если при этом I=1, то имеет место переполнения очереди.

|  |
| --- |
| Чтобы отличить пустую очередь от переполненной договорились, если за хвостом следует голова – очередь пустая, а если между хвостом и головой остается 1 элемент – очередь переполнена, т.е. в массив из N элементов можно поместить N-1 элементов. |

Очередь, как динамическая структура, определяется **двумя указателями**: на начало, откуда удаляются элементы очереди, и на конец – куда добавляются элементы очереди.



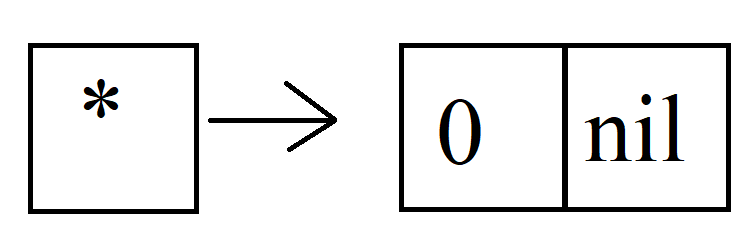
Различают следующие **виды очередей**:

* простая очередь;
* кольцевая очередь. В такой очереди элемент, который выходит с начала очереди, будет помещен в ее конец;
* очередь с приоритетами. В такой очереди элементы размещаются по их приоритетам (весовым коэффициентам). Первым из очереди выходит элемент с наивысшим приоритетом.

(Любой вид очереди может иметь ограничение по размеру)

В программе очередь можно реализовывать в виде:

* статического массива с ограничением на размер в очереди;
* динамического массива;
* односвязного списка;
* двусвязного списка.



Если пустой очередью считать очередь без единого звена, то процедура принимает вид:

Алгоритм:

1. Очередь массивом:
   1. Структура Queue2, которая содержит в себе 3 целочисленные переменные, где:  
      1)head - поле, где хранится позиция первого элемента.  
      2)tail - поле, где хранится позиция последнего элемента.  
      3)size - размер массива.  
      А также содержит указатель на целочисленную переменную data, которая указывает на первый элемент массива
   2. Функция InitQueue2, которая принимает значения ссылки на переменную q типа Queue2 и целочисленную переменную capacity(вместимость). Функция нужна для установки размера очереди.
   3. Функция next2, которая принимает значения ссылки на переменную q типа Queue2 и целочисленную переменную n(значения текущего элемента). Функция возвращает целочисленное значение следующего элемента.
   4. Функция empty2, которая принимает значения ссылки на переменную q типа Queue2. Функция проверяет полученную очередь на пустоту, возвращая булевое значение.
   5. Функция add2 которая принимает значения ссылки на переменную q типа Queue2 и целочисленную переменную value. Функция добавляет новый элемент в очередь значением value
   6. Функция del2, которая принимает значения ссылки на переменную q типа Queue2. Функция выводит голову очереди.
   7. Функция printqueue2, которая принимает значения ссылки на переменную q типа Queue2. Функция находит в очереди элемент
2. Стек массивом:  
   1. Структура Stack2:  
   Содержит в себе целочисленную переменную top, который является индексом верхнего элемента стека и указатель на целочисленную переменную data, который является указателем на элемент массива.   
   2. Функция InitStack2 принимает ссылку на переменную st типа Stack2 и целочисленную переменную capacity.   
   3. Функция push2 принимает ссылку на переменную st типа Stack2 и целочисленную переменную value. Функция добавляет элемент в стек.  
   4. Функция pop2 принимает ссылку на переменную st типа Stack2. Функция удаляет элемент из стека.  
   5. Функция nullStack2 принимает ссылку на переменную st типа Stack2. Функция обнуляет стек.  
   6. Функция empty2 принимает ссылку на переменную st типа Stack2. Функция проверяет стек на пустоту.  
   7. Функция printstack2 принимает ссылку на переменную st типа Stack2. Функция выводит стек.
3. Очередь динамическим списком (Класс Queue):   
   Класс содержит в себе структуру Queue1, которая содержит в себе целочисленную переменную data, в которой хранится значения узла и указатель на переменную next типа Queue1, в которой хранится указатель на следующий элемент массива. Также класс содержит два указателя на переменные head и tail типа Queue1, которые являются указателями на начало и конец списка.  
   1. Функция empty, которая проверяет очередь на пустоту.  
   2. Функция add, которая принимает целочисленное значение value. Функция добавляет элемент в очередь со значением value.  
   3. Функция del, удаляет элемент очереди.  
   4. Функция next, переходит к следующему элементу очереди.  
   5. Функция nullQueue обнуляет очередь.  
   6. Функция printqueue выводит очередь.  
   7. Функция search принимает целочисленное значение n1 и находит в очереди элементы с данным значением.
4. Стек динамическим списком:  
   1. Структура Stack содержит в себе целочисленную переменную data, в которой хранится значения узла и указатель на переменную next типа Stack, в которой хранится указатель на следующий элемент массива.  
   2. Функция Init Stack принимает ссылку на переменную top типа Stack.  
   3. Функция push принимает ссылку на переменную top типа Stack и целочисленную переменную value. Функция добавляет элементы в стек.  
   4. Функция pop принимает ссылку на переменную top типа Stack. Функция удаляет элемент из стека.  
   5. Функция empty принимает ссылку на переменную top типа Stack. Функция проверяет стек на пустоту.  
   6. Функция nullStack принимает ссылку на переменную top типа Stack. Функция обнуляет стек.  
   7. Функция printstack принимает ссылку на переменную top типа Stack. Функция выводит стек с удалением из него элементов.  
   8. Функция search принимает ссылку на переменную top типа Stack и целочисленную переменную n. Функция находит в стеке узел со значением n, и если находит, останавливает поиск, если нет, добавляет элемент в стек.
5. Функция main:
   1. Реализация массивом  
      1) Объявляются переменные s1 и s2 типа Stack2, переменная q2 типа Queue2.  
      2) Создаются и инициализируются целочисленные переменные size\_s1 и size\_q2 случайными значениями.  
      3) Если размеры массивов не равны нулю:  
       1. Устанавливаются размеры для стеков s1 и s2, а также для очереди q2.  
       2. Очередь q2 и стек s1 заполняются случайными числами.  
       3.Вывод s1 и q2.  
       4. Программа полностью проходит по стеку s1, постепенно доставая по элементу из стека и сравнивая полученное значение со значениями очереди q2. При нахождении такого же элемента в q2, программа записывает значение в стек s2.  
       5. Вывод s2.
   2. Реализация динамическим списком  
      1) Объявляются указатели на переменные st1 и st2 типа Stack, переменная qu2 типа Queue.  
      2) Создаются и инициализируются целочисленные переменные size\_st1 и size\_qu2 случайными значениями.  
      3) Если размеры массивов не равны нулю:  
       1. Очередь qu2 и стек st1 заполняются случайными числами.  
       2. Вывод st1 и qu2.  
       3. Программа проходит по st1, постепенно доставая по элементу из стека и сравнивая полученное значение со значениями очереди qu2. Если в очереди находится такой же элемент, он записывается в переменную n2, после чего заносится в стек st2, если стек st2 еще не содержит такого значения.  
      4) Вывод st2.

Текст программы:

#include <locale.h>

#include <iostream>

#include <string>

using namespace std;

//////////////////////////////////////////////массивом

struct Stack2 {

int top; int\* data; //top - индекс верхнего элемента стека, data - указатель на элемент массива

};

void InitStack2(Stack2& st, int capacity) {

st.data = new int[capacity];

st.top = -1;

}

void push2(Stack2& st, int value) {//добавление в стек

st.data[++st.top] = value;

}

int pop2(Stack2& st) {//удаление из стека

return st.data[st.top--];

}

void nullStack2(Stack2& st) {//обнуление стека

st.top = -1;

}

bool empty2(Stack2& st) {//проверка на пустоту

return st.top == -1;

}

void printstack2(Stack2& st) {//вывод стека с опустошением

if (!empty2(st)) {

int i = st.top;

while (i >= 0) {

cout << st.data[i--] << " ";

} cout << endl;

}

else cout << " пусто" << endl;

}

struct Queue2 {

int head, tail, size; // head - поле, где хранится позиция первого элеменета, tail - последнего, size - размер массива

int\* data; // указатель на первый элемент массива

};

void nullQueue2(Queue2& q) {

q.head = 0;

q.tail = q.size - 1;

}

void InitQueue2(Queue2& q, int capacity) {// установка размера очереди

q.size = capacity + 1;

q.data = new int[q.size];

nullQueue2(q);

}

int next2(Queue2& q, int n) { // функция, возвращающая значение следующего элементу

return (n + 1) % q.size;

}

bool empty2(Queue2& q) {// функция, проверяющая очередь на пустоту

return next2(q, q.tail) == q.head;

}

void add2(Queue2& q, int value) {// добавление в очередь элементов

if (next2(q, next2(q, q.tail)) == q.head)// проверка на переполнение

{

cout << "Очередь переполнена" << endl;

}

else {

q.tail = next2(q, q.tail);

q.data[q.tail] = value;

}

}

int del2(Queue2& q) { //вывод головы очереди

if (empty2(q)) { cout << "Очередь пуста" << endl; return 0; }

else {

int d = q.data[q.head];

q.head = next2(q, q.head);

return d;

}

}

void printqueue2(Queue2& q) { // вывод очереди

if (empty2(q)) { cout << "Очередь пуста" << endl; }

else {

int i = q.head;

while (i != q.tail + 1) {

cout << q.data[i] << " ";

i = next2(q, i);

} cout << endl;

}

}

//////////////////////////////////////////////////динамическим списком

//создаем стек динамическим списком

struct Stack {

int data; Stack\* next;// data - поле, где храниться данные, next - указатель на следущий элемент списка

};

void InitStack(Stack\*& top) {

top = NULL;

}

void push(Stack\*& top, int value) {//добавление элемента в стек

Stack\* tmp = new Stack;

tmp->next = top;

top = tmp;

top->data = value;

}

int pop(Stack\*& top) {//удаление элемента из стека

Stack\* tmp = top;

int d = top->data;

top = top->next;

return d;

}

bool empty(Stack\*& top) {// проверка стека на пустоту

return top == NULL;

}

void nullStack(Stack\*& top) {//обнуление стека

Stack\* tmp;

while (!empty(top)) {

tmp = top;

top = top->next;

delete(tmp);

}

}

void printstack(Stack\*& top) {// вывод стека с очищением стека

Stack\* tmp = top;

if (tmp == NULL) cout << "Отсутствуют." << endl;

while (!tmp == NULL) {

int d = tmp->data;

tmp = tmp->next;

cout << d << " ";

} delete (tmp);

}

void search(Stack\*& top, int n) {//нахождение одинаковых элементов

Stack\* tmp = top;

do {

if (n == pop(tmp))

break;

if (empty(tmp))

push(top, n);

} while (!empty(tmp));

}

//создаем очередь динамическим списком

class Queue {

private:

struct Queue1 {

int data;// поле, где храниться данные

Queue1\* next;//указатель на следущий элемент списка

};

Queue1\* head, \* tail;// указатель на начало списка и конец

public:

Queue() { head = NULL; tail = NULL; }

bool empty() {

return head == NULL;

}

void add(int value) {//добавление элементов в очередь

if (empty()) {

head = new Queue1;

head->data = value;

head->next = NULL;

tail = head;

}// в создании первого узла, первый элемент и хвост и голова.

else {

tail->next = new Queue1;

tail = tail->next;

tail->data = value;

tail->next = NULL;

}

}

int del() {//удаление элемента очереди

if (empty()) {

cout << "Очередь пуста" << endl;

return 0;

}

else {

int d = head->data;

Queue1\* tmp = head;

head = head->next;

delete(tmp);

return d;

}

}

int next() {//переход к следующему элементу очереди

if (empty()) {

cout << "Очередь пуста" << endl;

return 0;

}

else {

int d = head->data;

Queue1\* tmp = head;

head = head->next;

return d;

}

}

void nullQueue() {//обнуление очереди

Queue1\* tmp;

while (!empty()) {

tmp = head;

head = head->next;

delete(tmp);

}

}

void printqueue() {//вывод очереди

Queue1\* tmp;

tmp = head;

while (!tmp == NULL) {

int d = tmp->data;

tmp = tmp->next;

cout << d << " ";

} delete(tmp);

}

int search(int n1) {//нахождение одинаковых элементов

Queue1\* tmp = head;

while (!empty()) {

int n2 = next();

if (n1 == n2) {

head = tmp;

return n2;

}

}

head = tmp;

return -1;

}

};

int main() {

setlocale(LC\_ALL, "rus");

srand(time(0));

cout << endl;

//Реализация массивом

Queue2 q2;// объявление объектов структуры Queue

Stack2 s1, s2;// объявление объектов структуры Stack2

int size\_s1 = rand() % 10 + 1, size\_q2 = rand() % 10 + 1;// размеры структур

cout << "Размер стека : " << size\_s1 << "\n" << "Размер очереди: " << size\_q2 << endl;

if (size\_s1 != 0 && size\_q2 != 0)

{

InitStack2(s1, size\_s1);//с помощью функции установил размер стека s1

InitQueue2(q2, size\_q2);//с помощью функции установил размер очереди q2

InitStack2(s2, size\_s1);//с помощью функции установил размер стака s1

for (int i = 0; i < size\_s1; i++) {// добавление элементов в стек

int j = rand() % 10;

push2(s1, j);

}

for (int i = 0; i < size\_q2; i++) {// добавление элементов в очередь

int j = rand() % 10;

add2(q2, j);

}

cout << "Стек: ";

if (!empty2(s1)) printstack2(s1);//вывод стека s1

else cout << "Пусто" << endl;

cout << "Очередь: ";

if (!empty2(q2)) printqueue2(q2);//вывод очереди q2

else cout << "Пусто" << endl;

cout << "Общие элементы: ";

while (!empty2(s1)) {

int n1 = pop2(s1); // в n1 записываем стек

while (!empty2(q2)) {

int n2 = del2(q2);// в n2 записываем очередь

if (n1 == n2) { //если элемент стека равен эл. очереди

if (empty2(s2)) {

push2(s2, n2);

}

else { // если 2 стек не пуст, проверяет, нет ли такого элемента

for (int i = s2.top; i >= 0; i--) {

if (n2 == s2.data[i])

break;

if (i == 0)

push2(s2, n2); // вставляет

}

}

break;

}

}

q2.head = 0;

}

if (!empty2(s2)) printstack2(s2);// выводит новый стек

else cout << "Отсутствуют." << endl;

}

cout << endl << endl;

//динамический список

Queue qu2;

Stack\* st1, \* st2;

InitStack(st2);

InitStack(st1);

int size\_st1 = rand() % 10 + 1, size\_qu2 = rand() % 10 + 1;// определение размера структур

if (size\_st1 != 0 && size\_qu2 != 0) {

cout << "Размер стека: " << size\_st1 << "\n" << "Размер очереди: " << size\_qu2 << endl;

for (int i = 0; i < size\_st1; i++) {

int j = rand() % 10;

push(st1, j); // заполняется стек

}

for (int i = 0; i < size\_qu2; i++) {

int j = rand() % 10;

qu2.add(j);// заполняется очередь

}

cout << "Стек: ";

if (!empty(st1)) { printstack(st1); cout << endl; } //выводит динамический стек

else cout << "Пусто" << endl;

cout << "Очередь: ";

if (!qu2.empty()) { qu2.printqueue(); cout << endl; }// выводит динамическую очередь

else cout << "Пусто" << endl;

cout << "Общие элементы: ";

while (!empty(st1))// пока стек не пуст

{

int n2 = qu2.search(pop(st1));//ищем похожий элемент

if (n2 >= 0) {

if (empty(st2))

push(st2, n2);// вставляем

else

search(st2, n2);//продолжаем искать

}

}

}

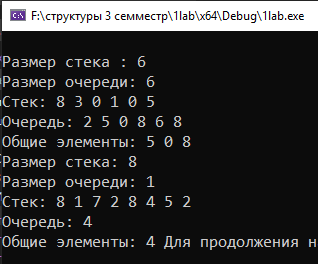
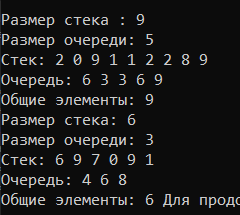
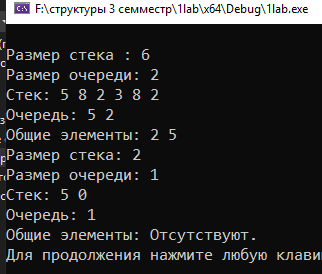
printstack(st2);// выводим новый стек

system("pause");

return 0;

}

Тестирование:

Вывод: на основе своего варианта были построены очередь и стек, на основе массива и динамического списка, также из общих элементов строиться новый стек из общих элементов.