**Лабоpатоpная pабота №1**

**Линейные списки**

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.

13 вариант

13. Даны два стека целых чисел. Сфоpмиpовать очередь из элементов пеpвого стека, кpатных максимуму втоpого.

а) массивы;

Алгоритм:

Создается структура Queue1, в ней содержатся указатель на нулевой элемент массива элементов очереди, четыре элемента типа int – указатели на начало и конец очереди, количество элементов очереди.

Функция add добавляет элемент в очередь, обновляя указатель на конец очереди, добавляя элемент в массив и увеличивая количество элементов в очереди.

Функция ShowQueue если очередь пуста выводит соответствующее сообщение. Если очередь не пуста создается временный указатель на первый элемент очереди и с его помощью через next проходим по всем элементам очереди, на каждом элементе выводится значение поля data.

Удаление элементов из очереди происходит путем сдвига указателя начала очереди и уменьшения количества элементов функцией del.

Создается структура Stack1, в ней содержатся указатель типа int, массив, который будет использоваться для хранения данных в стеке, переменная int top, которая указывает на вершину (последний добавленный элемент) стека.

Функция InitStack инициализирует стек, выделяя место для массива и устанавливая вершину стека

Функция push добавляет элемент в стек, увеличивая вершину стека и помещая значение в массив.

Функция pop удаляет элемент из стека, возвращая значение и уменьшая вершину стека.

NullStack Эта функция "стирает" стек, устанавливая вершину стека (переменную top) в значение -1. Это действие позволяет "очистить" стек, установив его в начальное состояние без каких-либо элементов.

Функция empty проверяет, пуст ли стек, возвращая значение true в случае, если вершина стека (top) имеет значение -1, что означает, что стек пуст. Если стек не пуст, функция возвращает false.

В функции main случайным образом генерируются значения для размера и элементов стеков. Добавление элементов происходит путем вызова push для стеков и add для очереди. Далее идет обращение поочередно ко всем элементам стека, значение элементов выводятся и ищется максимальный. После выводится максимальный элемент. Соответственно выводятся элементы первого стека ищутся и считаются кратные максимуму второго. Инициализируем очередь размером по количеству кратных элементов. Еще раз проходим по первому стеку на этот раз записываем кратные в очередь. Выводим очередь поочередно обращаясь к каждому элементу.

#include <iostream>

#include <windows.h>

#include<time.h>

using namespace std;

struct Queue1 {

int\* elements; // массив для хранения элементов

int head, tail; // указатели на начало и конец очереди

int size; // количество элементов очереди

};

void InitQueue(Queue1& q, int capacity) {

q.head = -1;

q.size = capacity + 1;

q.tail = -1; // указатель конца очереди

q.elements = new int[capacity + 1]; // выделение памяти для массива

}

bool empty(Queue1& q)

{

return(q.head == -1 && q.tail == -1);

}

void add(Queue1& q, int item) {

if ((((q.tail + 1) % q.size) + 1) % q.size == q.head) { // проверяем, не заполнена ли очередь

cout << "Queue is full"; // выводим сообщение, если очередь переполнена

return; // выходим из функции

}

if (q.head == -1)

q.head = 0;

q.tail = (q.tail + 1) % q.size;// обновляем указатель на конец очереди

q.elements[q.tail] = item; // добавляем элемент в очередь

}

int del(Queue1& q)

{

if (empty(q)) {

cout << "Очередь пустая!\n";

return 0;

}

else {

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

if (q.head == q.tail)

q.head = q.tail = -1;

else

q.head = (q.head + 1) % q.size;

return d;

}

}

//STACK

struct Stack1 { /\*создание струкруты стека\*/

int top;

int\* data;

};

void InitStack(Stack1& st, int capacity) { /\*инициализирует стек и создает место\*/

st.data = new int[capacity];

st.top = -1;

}

void push(Stack1& st, int value) { /\*добавляет элемент\*/

st.data[++st.top] = value; //++i увеличивает i на 1, потом возвращает

}

int pop(Stack1& st) { /\*удаляет по одному\*/

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

}

void nullStack(Stack1& st) { /\*стирает стек\*/

st.top = -1;

}

bool empty(Stack1& st) { /\*проверяет наличие элементов стека\*/

return st.top == -1;

}

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(0));

Stack1 st1, st2;

Queue1 que;

int w = rand()%10, e = rand()%10;

InitStack(st1, w);

InitStack(st2, e);

for (int i = 0; i < w; i++) {//создание стека 1

push(st1, rand()%10);

}

for (int i = 0; i < e; i++) {//создание стека 2

push(st2, rand()%10);

}

int j = 0;

cout << "st2: ";//выводим стек 2, ищем максимум

if (!empty(st2)) {

j = st2.data[st2.top];

int i = st2.top;

while (i != -1) {

int k = st2.data[i--];

if (j < k) j = k;

cout << k << ' ';

}

}

cout << endl << "st2 max: " << j << endl;

int c = 0;

cout << "st1: ";//выводим стек 1, считаем количество элементов кратных максимуму

if (!empty(st1)) {

int i = st1.top;

while (i != -1) {

int k = st1.data[i--];

cout << k << " ";

if (j != 0 && k % j == 0) {

c++;

}

}

}

InitQueue(que, c);

if (!empty(st1)) { //вносим элементы в очередь

int i = st1.top;

while (i != -1) {

int k = st1.data[i--];

if (j != 0 && k % j == 0) {

add(que, k);

}

}

}

cout << endl << "Queue: ";

if (!empty(que)) { //выводим очередь

int p = que.head;

while (p != (que.tail + 1) % que.size)

{

cout << \*que.elements << " ";

p = (p + 1) % que.size;

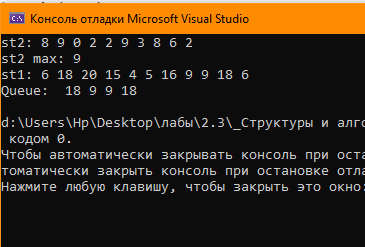
}

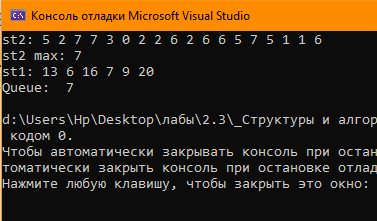
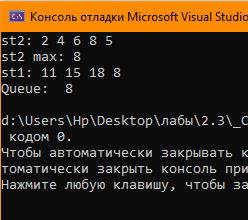
}

cout << endl;

return 0;

}



б) динамические списки.

Алгоритм:

InitStack эта функция инициализирует стек, устанавливая указатель top в NULL.

Функция push добавляет элемент в стек. Создается новый узел, в который помещается значение, затем этот узел становится вершиной (top) стека.

Функция pop извлекает элемент из стека. Значение из вершины (top) копируется в переменную d, затем освобождается память, выделенная для вершины, и вершина перемещается на следующий элемент. Значение d возвращается как результат.

Функция empty проверяет пуст ли стек, возвращая true если top равно NULL, иначе возвращается false.

Функция nullStack "стирает" стек, удаляя все его элементы. Она продолжает удалять элементы, пока стек не будет пустым, используя функцию empty для проверки.

Класс Queue. Сначала идет описания структуры Node, которая представляет узел в связанном списке. Каждый узел содержит два поля: data: Поле, которое содержит данные узла, next: Указатель на следующий узел в связанном списке. После идут два указателя на начало и конец очереди head и tail соответственно.

Метод empty проверяет, пуста ли очередь, возвращая true, если head равен NULL.

Метод add добавляет элемент в очередь. Если очередь пуста, создается новый узел и он становится её началом и концом. В противном случае новый узел добавляется в конец очереди.

Метод ShowQueue если очередь не пуста создается временный указатель tmp с помощью которого проходим по всем элементам очереди и выводим значение поля data.

Метод del удаляет элемент из начала очереди и возвращает его значение. Если очередь пуста, выводится сообщение "Queue is empty" и возвращается 0.

Далее идет функция main, где программа создает два стека (st1 и st2) и один объект класса очереди que. Стеки заполняются случайными числами. Далее идет обращение поочередно ко всем элементам стека, значение элементов выводятся и ищется максимальный. После выводится максимальный элемент. Соответственно выводятся элементы первого стека ищутся и считаются кратные максимуму второго. Еще раз проходим по первому стеку на этот раз записываем кратные в очередь. Выводим очередь поочередно обращаясь к каждому элементу.

#include <iostream>

#include <windows.h>

#include<time.h>

using namespace std;

// Определение структуры узла для стека

struct Node { /\*динамич. список\*/

int data, top;

Node\* next;

};

// Функция инициализации стека

void InitStack(Node\*& top) {

top = NULL;

}

// Функция добавления элемента в стек

void push(Node\*& top, int value) {

Node\* tmp = new Node;

tmp->next = top;

top = tmp;

top->data = value;

}

// Функция извлечения элемента из стека

int pop(Node\*& top) {

Node\* tmp = top;

int d = top->data;

top = top->next;

delete(tmp);

return d;

}

// Функция проверки наличия элементов в стеке

bool empty(Node\*& top) {

return top == NULL;

}

// Функция очистки стека

void nullStack(Node\*& top) {

Node\* tmp;

while (!empty(top)) {

tmp = top;

top = top->next;

delete(tmp);

}

}

// Определение класса очереди

class Queue {

struct Node

{

int data;

Node\* next;

};

Node\* head, \* tail;

public:

Queue() {

head = 0;

tail = 0;

}

// Метод проверки наличия элементов в очереди

bool empty() {

return head == NULL;

}

// Метод добавления элемента в очередь

void add(int value) {

Node\* tmp = new Node();

tmp->data = value;

tmp->next = NULL;

if (head == NULL)

{

head = tmp;

tail = tmp;

}

else

{

tail->next = tmp;

tail = tmp;

}

}

void ShowQueue()//выводим очередь

{

if (empty())

cout << "Очередь пустая!\n";

else

{

Node\* tmp = head;

while (tmp != NULL)

{

cout << tmp->data << " ";

tmp = tmp->next;

}

}

cout << endl;

}

// Метод удаления элемента из очереди

int del()

{

if (empty())

cout << "Очередь пустая!\n";

else {

int d = head->data;

if (head == tail)

{

delete(head);

head = tail = NULL;

}

else

{

Node\* tmp = head;

head = head->next;

delete(tmp);

}

return d;

}

}

};

// Точка входа в программу

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(0));

Node\* st1, \* st2; // Создание указателей на вершину стеков

Queue que; // Создание объекта класса Queue для очереди

InitStack(st1); // Инициализация стека 1

InitStack(st2); // Инициализация стека 2

int w = rand() % (20 - 4 + 1) + 4, e = rand() % (20 - 4 + 1) + 4;

for (int i = 0; i < w; i++) {

push(st1, rand() % (20 - 4 + 1) + 4); // Добавление случайного значения в стек 1

}

for (int i = 0; i < e; i++) {

push(st2, rand() % 10); // Добавление случайного значения в стек 2

}

Node\* tmp1 = st2;

int j = 0;

cout << "Динамические списки" << endl << "st2: ";

if (!empty(st2)) {

j = tmp1->data;

Node\* tmp = st2;

while (tmp != NULL)

{

int k = tmp->data;

if (j < k) j = k;

cout << k << ' ';

tmp = tmp->next;

}

}

cout << endl << "st2 max: " << j << endl;

int c = 0;

cout << "st1: ";

if (!empty(st1)) {

Node\* tmp = st1;

while (tmp != NULL)

{

int k = tmp->data;

cout << k << " ";

tmp = tmp->next;

if (j != 0 && k % j == 0) {

que.add(k);

c++;

}

}

}

cout << endl << "Queue: ";

if (!que.empty())

{

que.ShowQueue();

}

cout << endl;

return 0;

}

