Лабораторная работа №1

Линейные списки

ЗАДАНИЕ

Вариант 6

1. Даны две очеpеди целых чисел. Сфоpмиpовать стек из элементов пеpвой очеpеди, кpатных максимуму втоpой.

Выполнить задания, используя для пpедставления очеpедей и стеков:

а) массивы.

б) динамические списки.

ХОД РАБОТЫ

Стек – структура данных, которая работает по принципу последним пришёл, первым ушёл (LIFO – last in, first out).

Очередь – структура данных, которая работает по принципу первым пришёл, первым ушёл (FIFO – first in, first out).

а) выполнить задания, используя для пpедставления очеpедей и стеков: массивы.

Результат работы программы представлен в соответствии с рисунком 1.

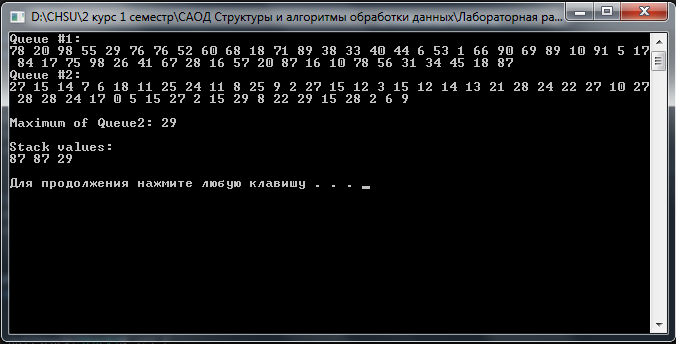


Рисунок 1 – результаты работы программы.

Текст программы:

#include <iostream>

#include <time.h>

using namespace std;

// Описание стека

struct Stack {

int top;

int\* data;

int capacity;

};

void InitStack(Stack& st, int capacity) {

st.data = new int[capacity];

st.top = -1;

st.capacity = capacity;

}

void push(Stack& st, int value) {

if (st.top + 1 < st.capacity) {

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

}

}

int pop(Stack& st) {

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

}

int print(Stack& st) {

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

}

void nullStack(Stack& st) {

st.top = -1;

}

bool empty(Stack& st) {

return st.top == -1;

}

void printStack(Stack& st) {

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

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

}

}

// Описание очереди массивом

struct Queue {

int head, tail, size;

int\* data;

};

void nullQueue(Queue& q) {

q.head = 0;

q.tail = q.size - 1;

}

void InitQueue(Queue& q, int capacity) {

q.size = capacity + 1;

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

nullQueue(q);

}

int next(Queue& q, int n) {

return (n + 1) % q.size;

}

bool empty(Queue& q) {

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

}

void add(Queue& q, int value) {

if (next(q, next(q, q.tail)) == q.head)

cout << "Queue overflow" << endl;

else {

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

q.data[q.tail] = value;

}

}

int del(Queue& q) {

if (empty(q)) {

cout << "Queue is empty" << endl;

return 0;

}

else {

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

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

return d;

}

}

void printQueue(Queue& q) {

int i = q.head;

while (i != q.tail) {

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

i = next(q, i);

}

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

}

void main() {

srand(time(0));

// Создание очереди 1

Queue q1;

int k = rand() % 50 + 1;

InitQueue(q1, k);

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

int n = rand() % 100;

add(q1, n);

}

// Создание очереди 2

Queue q2;

k = rand() % 50 + 1;

InitQueue(q2, k);

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

int n = rand() % 30;

add(q2, n);

}

// Вывод очередей

cout << "Queue #1:" << endl;

printQueue(q1);

cout << endl;

cout << "Queue #2:" << endl;

printQueue(q2);

cout << endl;

// Максимум Queue2

int maxQ = 0;

while (!empty(q2)) {

int x;

x = del(q2);

if (x > maxQ)

maxQ = x;

}

cout << "\nMaximum of Queue2: " << maxQ << endl;

// Создание стека из 1 очереди, кратных максимуму второй очереди

Stack st;

InitStack(st, q1.size);

while (!q1.empty()) {

int x = q1.del();

if (maxQ == 0){

push(stack, x);

}

else if (x % maxQ == 0)

push(stack, x);

}

cout << endl;

// Чтение стека

cout << "Stack values: " << endl;

printStack(st);

cout << endl;

system("pause");

}

Алгоритм программы:

Инициализация первой очереди q1 заданной длины k (случайно сгенерированное число от 1 до 50).

Инициализация второй очереди q1 заданной длины k (новое случайно сгенерированное число от 1 до 50).

Структура Queue1:

head - индекс головы очереди (первый элемент)

tail - индекс хвоста очереди (последний элемент)

size - размер очереди

data - массив для хранения элементов очереди

initQueue - инициализация очереди с указанной вместимостью

nullQueue(Queue1& q): сброс очереди

next(Queue1& q, int n): вычисление следующего индекса в очереди

empty(Queue1& q): проверка, пуста ли очередь

add(Queue1& q, int value): добавление элемента в очередь

del(Queue1& q): извлечение элемента из очереди

printQueue(Queue1& q): вывод содержимого очереди

Заполнение первой очереди q1 случайно сгенерированным числами от 0 до 99.

Заполнение второй очереди q2 случайно сгенерированными числами от 0 до 99.

Вывод содержимого обеих очередей на экран.

Создание пустого стека stack. Содержит следующие поля:

top: индекс верхнего элемента стека

data: массив для хранения элементов стека

capacity: максимальный размер стека

initStack(Stack1& st, int capacity): инициализация стека с указанной вместимостью

push(Stack1& st, int value): добавление элемента в стек

pop(Stack1& st): извлечение элемента из стека

nullStack(Stack1& st): сброс стека

empty(Stack1& st): проверка, пуст ли стек

printStack(Stack1& st): вывод содержимого стека

Инициализация и заполнение двух очередей.

Создание стека из ПЕРВОЙ очереди, кратных максимуму второй очереди:

Создание пустого стека.

Последовательная обработка элементов второй очереди: поиск и извлечение максимального элемента из второй очереди.

Последовательная обработка элементов второй очереди: если элемент первой очереди кратен максимуму второй очереди, добавление его в стек.

Вывод содержимого стека.

б) Выполнить задания, используя для пpедставления очеpедей и стеков: динамические списки.

Результат работы программы представлен в соответствии с рисунком 1.

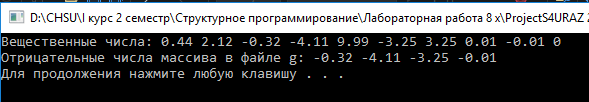


Рисунок 1 – результаты работы программы.

Алгоритм программы:

1. Определение структуры Node для стека, которая содержит поле data (целочисленное значение узла) и поле next (указатель на следующий узел в стеке).

Инициализация стека функцией InitStack, устанавливая указатель top на NULL.

Функция push добавляет новое значение в стек. Создаётся новый узел со значением value. Этот стек становится новым верхним узлом стека.

Функция empty проверяет пуст ли стек.

Функция del удаляет верхний элемент из стека, возвращает его значение и обновляет указатель top.

Функция nullStack освобождает память, занятую стеком, удаляя все узлы.

1. Создание класса Queue для очереди.

Содержит структуру Node для узлов очереди, указатели head, tail и top, чтобы отслеживать начало и конец очереди. Конструктор класса инициализирует указатели в null.

1. В функции main:

Инициализируется генерация случайных чисел.

Создаётся две очереди и заполняются случайным количеством элементов от 1 до 55 со случайными значениями от 0 до 99.

Выводятся элементы первой и второй очереди.

Поиск максимума во второй очереди:

Инициализируется целочисленная переменная maxQ со значением 0. В цикле проверяются значения второй очереди на максимальное значение.

Инициализируется стек. Из первой очереди берутся элементы и проверяется условие, кратно ли это значение элемента из первой очереди найденному максимальному числу из второй очереди. Если кратно, то добавляется в стек.

Выводится содержимое стека на экран.

Текст программы:

#include <iostream>

#include <time.h>

using namespace std;

// Описание стека

struct Node {

int data;

Node\* next;

};

void InitStack(Node\*& top) {

top = NULL;

}

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

Node\* tmp = new Node;

tmp->next = top;

top = tmp;

top->data = value;

}

bool empty(Node\*& top) {

return top == NULL;

}

int del(Node\*& top) {

Node\* tmp = top;

int d = top->data;

top = top->next;

delete(tmp);

return d;

}

void nullStack(Node\*& top) {

Node\* tmp;

while (!empty(top)) {

tmp = top;

top = top->next;

delete(tmp);

}

}

// Описание очереди

class Queue {

private:

struct Node {

int data;

Node\* next;

};

Node\* head, \* tail, \* top;

public:

Queue() {

head = NULL;

tail = NULL;

top = NULL;

}

bool empty() {

return head == NULL;

}

void add(int value) {

if (empty()) {

head = new Node;

head->data = value;

head->next = NULL;

tail = head;

top = head;

}

else {

tail->next = new Node;

tail = tail->next;

tail->data = value;

tail->next = NULL;

}

}

int del() {

if (empty()) {

cout << "Queue is empty" << endl;

return 0;

}

else {

int d = head->data;

Node\* tmp = head;

head = head->next;

delete(tmp);

return d;

}

}

void nullQueue() {

Node\* tmp;

while (!empty()) {

tmp = head;

head = head->next;

delete(tmp);

}

}

void printQueue() {

if (empty()) {

cout << "Queue is empty" << endl;

}

else {

while (!empty()) {

int d = head->data;

head = head->next;

cout << d << " ";

}

cout << endl;

}

head = top;

}

};

void main() {

srand(time(0));

// Создание очереди 1 и 2

Queue q1, q2;

int k = rand() % 50 + 1;

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

int n = rand() % 100;

q1.add(n);

}

k = rand() % 50 + 1;

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

int n = rand() % 30;

q2.add(n);

}

// Вывод очередей

cout << "Queue #1:" << endl;

q1.printQueue();

cout << "Queue #2:" << endl;

q2.printQueue();

// Максимум Queue2

int maxQ = 0;

if (!q2.empty())

maxQ = q2.del();

while (!q2.empty()) {

int x;

x = q2.del();

if (x > maxQ)

maxQ = x;

}

cout << "\nMaximum of Queue2: " << maxQ << endl;

// Создание стека из ПЕРВОЙ очереди, кратных максимуму второй очереди

Node\* stack;

InitStack(stack);

while (!q1.empty()) {

int x = q1.del();

if (maxQ == 0){

push(stack, x);

}

else if (x % maxQ == 0)

push(stack, x);

}

cout << endl;

// Чтение стека

cout << "Stack values: ";

if (!empty(stack))

while (!empty(stack))

cout << del(stack) << " ";

else cout << "Stack is empty" << endl;

cout << endl;

system("pause");

}

Вывод:

В ходе выполнения лабораторной работы мы научились использовать линейные списки и закрепили полученные навыки.