Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»  
(ВлГУ)

Институт прикладной математики, физики и информатики

Кафедра физики и прикладной математики

ЛАБОРАТОРНАЯ РАБОТА №7  
по дисциплине  
«Основы программирования»  
на тему:  
« Линейные динамические структуры»

Выполнил:  
ст. гр. ПМИ-123

Рушев А.М.  
  
  
Принял:  
ст. преподаватель   
каф. ФиПМ  
Шишкина М.В.

Владимир,2024

**Цель работы**

Получение навыков организации динамических структур данных и дальнейшей работы с ними на языке программирования С++.

**Постановка задачи**

Реализовать создание линейных динамических структур (по принципу стека и очереди) для хранения целочисленных данных. Для этого написать функции добавления и удаления элементов в стек, функции добавления и удаления элементов из очереди, в функциях предусмотреть проверку на пустоту списка. При этом, в функциях добавления элемента, если список пуст, необходимо создать первый элемент. Если список пуст и при этом вызвана функция извлечения элемента, выдать соответствующее сообщение. Создать и проинициализировать целочисленный массив. В цикле перебора элементов, добавить элементы в стек и очередь. Каждый элемент должен быть добавлен в обе структуры. После того как, все элементы добавлены, вывести элементы на экран из стека и из очереди.

**Теоретическая часть**

Существует несколько разновидностей динамических структур: список, дерево.

Прежде чем переходить к описанию структур, следует запомнить несколько простых определений:

* Потомок — элемент структуры, идущий после текущего. В зависимости от вида динамической структуры у элемента может быть более одного потомка.
* Предок — элемент структуры, идущий до текущего.
* Головной элемент (Head) — первый элемент списка.
* Хвостовой элемент (Tail) — последний элемент списка.

Структура данных Список

Список — это линейная динамическая структура данных, у каждого элемента может быть только один предок и только один потомок. По сути своей это очень похоже на обыкновенный массив, с той лишь разницей, что размер его не имеет ограничений. Списки также подразделяются на несколько типов.

* Односвязный список — элемент имеет указатель только на своего потомка.



Рисунок 1 – Односвязный список

* Двусвязный список — элемент имеет указатели и на потомка, и на родителя.

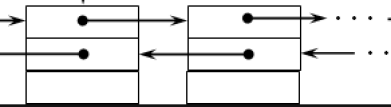


Рисунок 2 – Двусвязный список

* Замкнутый (кольцевой, циклический) список — головной и хвостовой элементы которого указывают друг на друга.

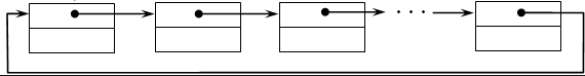


Рисунок 3 – Замкнутый список

На базе простого однонаправленного списка могут быть построены такие структуры данных, как очередь (queue) и стек (stack).

**Очередь** есть ничто иное, как список, операции чтения и добавления элементов, в котором подвержены определенным правилам. При этом, при чтении элемента, он удаляется из очереди. Все операции проводятся по принципу «Первый пришел, первый вышел» (FIFO — first in, first out). Таким образом, для чтения в очереди доступна только голова, в то время как добавление проводится только в хвост.

**Стек** во многом похож на очередь, с той лишь разницей, что извлечение и добавление элементов в нем происходит по правилу «Последний пришел, первый вышел» (LIFO — last in, first out). Добавление и извлечение элементов проводится от головы. По принципу похоже на работу обоймы огнестрельного оружия.

**Практическая часть**

Листинг кода приложения:

#include <iostream>

using namespace std;

struct Node

{

int data;

Node\* next;

};

struct Q {

Node\* h;

Node\* t;

};

struct Stack

{

Node\* top;

};

void InputMassiv(int\* arr, int n) {

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

arr[i] = 1 + rand() % 50;

}

}

void PrintMassiv(int\* arr, int n) {

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

cout << arr[i] << "\t";

}

cout << endl;

}

void AddElem(Q &q, int value) {

Node\* temp = new Node;

temp->data = value;

temp->next = NULL;

if (q.h == NULL) {

q.h = temp;

q.t = temp;

}

else {

q.t->next = temp;

q.t = q.t->next;

}

}

void Print(Node\* h) {

Node\* temp = h;

while (temp != NULL) {

cout << temp->data << "\t";

temp = temp->next;

}

cout << endl;

}

void PrintStack(Node\* stack) {

Node\* temp = stack;

while (temp != NULL) {

cout << temp->data << "\t";

temp = temp->next;

}

cout << endl;

}

int Izvl(Q& q) {

Node\* temp = q.h;

int a = temp->data;

q.h = temp->next;

delete temp;

return a;

}

bool GetSosiska(Q& q) {

if (q.h != NULL) {

return true;

}

else {

return false;

}

}

void AddStack(Node\*&stack, int value) {

Node\* temp = new Node;

temp->data = value;

temp->next = NULL;

if (stack == NULL) {

stack = temp;

}

else{

temp->next = stack;

stack = temp;

}

}

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Лабораторная работа №7" << endl;

const int n = 5;

int arr[n];

InputMassiv(arr, n);

PrintMassiv(arr, n);

Q Q1 = {NULL, NULL};

Node\* stack = NULL;

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

AddElem(Q1, arr[i]);

}

Print(Q1.h);

cout << Izvl(Q1) << endl;

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

if (GetSosiska(Q1) != false) {

cout << Izvl(Q1) << endl;

}

else {

cout << "Сосисок нет" << endl;

}

}

cout << "Стэк: " << endl;

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

AddStack(stack, arr[i]);

}

PrintStack(stack);

system("pause");

return 0;

}

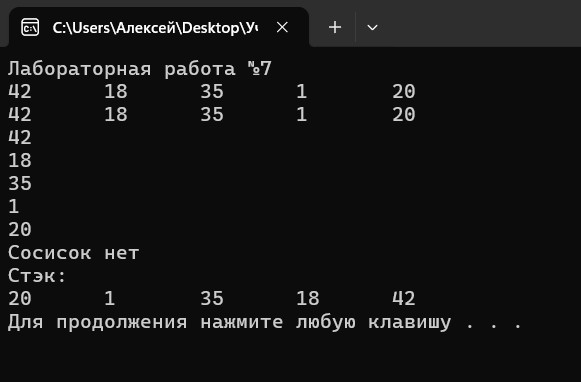
****

Рисунок 1 – Результат выполнения кода

**Вывод:** Линейные динамические структуры данных, такие как списки, стеки и очереди, имеют свои плюсы и минусы:

Плюсы:

1. Гибкость: линейные динамические структуры данных позволяют эффективно управлять данными, изменять их размер и структуру по мере необходимости.

2. Эффективность: операции вставки, удаления и обновления элементов в линейных динамических структурах данных могут быть выполнены сравнительно быстро, особенно в сравнении с массивами.

3. Эффективное использование памяти: линейные динамические структуры могут динамически расширяться или сжиматься по требованию, что позволяет более эффективно использовать память.

Минусы:

1. Производительность: в отличие от массивов, линейные динамические структуры требуют дополнительных вычислительных затрат при доступе к элементам и выполнении операций.

2. Затраты на управление памятью: динамическое выделение и освобождение памяти для элементов линейных структур может вызывать накладные расходы и увеличивать вероятность утечек памяти.

3. Сложности в управлении: линейные динамические структуры могут быть сложны в управлении и требуют аккуратного управления указателями или ссылками для избежания ошибок.

4. Изменение размера: изменение размера линейных динамических структур может привести к перераспределению памяти и затратам на копирование элементов, что может сказаться на производительности.