Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе**

Дисциплина: Основы алгоритмизации и программирования

Тема: Блоковый ввод-вывод

Выполнил работу

студент группы РИС-22-1б

Рыжков Н.С.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2023

**Цель и задачи**

Цель – написать программу на языке программирования C++, в которой будут использованы списки, стеки, очереди.

Для достижения поставленной цели необходимо решить следующие задачи:

* Изучить тему списков, стеков и очередей
* Выбрать наиболее подходящие инструменты языка для выполнения задачи
* Реализовать задачу на языке С++

**Анализ задачи**

Написать программу, в которой создаются динамические структуры и выполнить их обработку в соответствии со своим вариантом. Записи в линейном списке содержат ключевое поле типа \*char(строка символов). Сформировать двунаправленный список. Удалить элемент с заданным ключом. Добавить К элементов после элемента с заданным ключом.

Выделим шаги:

1. Создаем класс `Node` для представления узлов двусвязного списка и класс `DoublyLinkedList` для создания самого списка, а затем вызываем методы этого объекта для добавления и удаления элементов в списке.
2. В функции `main()` создаем объект списка путем вызова конструктора без аргументов. Затем три элемента ('a', 'b' и 'c') добавляем в список с помощью метода `addNode()`, после чего на экран выводим текущее содержимое списка с помощью метода `printList()`.
3. Затем из списка удаляем элемент со значением 'b' (методом `removeNode()`), после чего опять выводим содержимое обновленного списка. Наконец, к первому элементу ('a') добавляем два новых элемента - 'x' и 'y', с помощью метода `addNodesAfter()`, а затем на экран выводим полученное окончательное состояние списка.

**Листинг программы**

#include <iostream>

#include <stack>

using namespace std;

class Node

{

public:

char data;

Node \*next;

Node \*prev;

};

class DoublyLinkedList

{

private:

Node \*head;

Node \*tail;

public:

DoublyLinkedList()

{ head = NULL;

tail = NULL;

}

void addNode(char n)

{ Node \*temp = new Node();

temp->data = n;

temp->next = NULL;

if (head == NULL)

{ temp->prev = NULL;

head = temp;

tail = temp;

}

else

{

tail->next = temp;

temp->prev = tail;

tail = temp;

}

}

void addNode(char n, int pos)

{

Node \*newNode = new Node();

newNode->data = n;

if (pos == 1)

{

newNode->next = head;

head->prev = newNode;

head = newNode;

}

else

{ Node \*temp = head;

for (int i = 1; temp != NULL && i < pos - 1; ++i)

temp = temp->next;

if (temp != NULL)

{ newNode->next = temp->next;

newNode->next->prev = newNode;

}

else

{ newNode->next = NULL;

newNode->prev = tail;

tail->next = newNode;

tail = newNode;

}

newNode->prev = temp;

temp->next = newNode;

} }

void addNodesAfter(char target, const string &nodes)

{ Node \*temp = head;

while (temp != NULL)

{ if (temp->data == target)

{ for (char node : nodes)

{

Node \*newNode = new Node(); newNode->data = node;

newNode->next = temp->next; newNode->prev = temp;

if (temp->next != NULL)

temp->next->prev = newNode; else tail = newNode;

temp->next = newNode;

temp = newNode;

}

break; }

temp = temp->next;

} }

void removeNode(char n)

{ Node \*temp = head;

while (temp != NULL)

{ if (temp->data == n)

{ if (temp == head && temp == tail) { head = NULL;

tail = NULL;

}

else if (temp == head)

{ head = temp->next;

head->prev = NULL;

}

else if (temp == tail)

{ tail = temp->prev;

tail->next = NULL;

}

else

{ temp->prev->next = temp->next; temp->next->prev = temp->prev;

}

delete temp;

return; }

temp = temp->next;

} }

void printList()

{ Node \*temp = head;

while (temp != NULL)

{ cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

} }; int main()

{ DoublyLinkedList list;

list.addNode('a');

list.addNode('b');

list.addNode('c');

list.printList(); // выведет: a b c

list.removeNode('b');

list.printList(); // выведет: a c

list.addNode('d', 2);

list.printList(); // выведет: a d c

list.addNodesAfter('a', "xy");

list.printList(); // выведет: a x y d c

system("pause");

return 0;

}

**Результат работы программы**

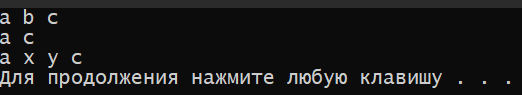


Рисунок 1 – Вывод программы

**Блок-схема программы**

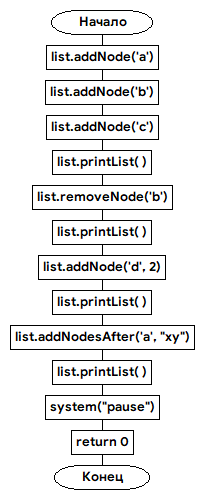


Рисунок 3 – Блок схема главной функции

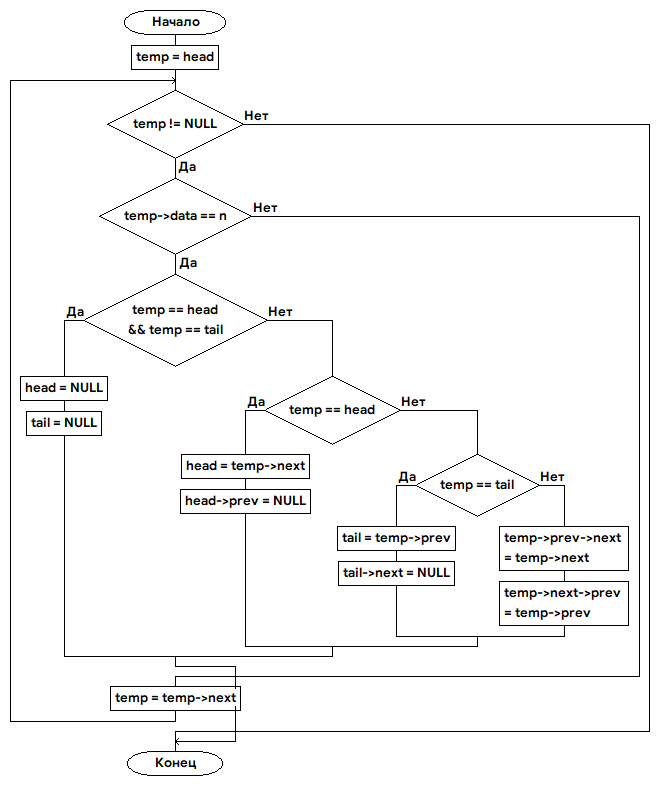


Рисунок 4 – Блок-схема функции ***void removeNode(char n)***

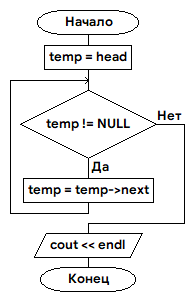


Рисунок 5 – Блок-схема функции ***void printList()***

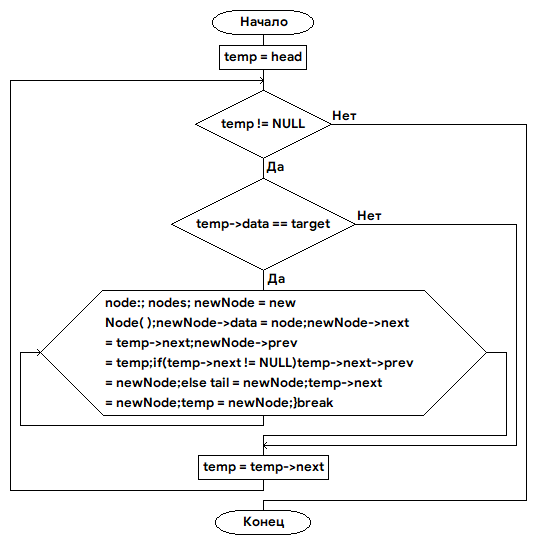


Рисунок 6 – Блок-схема функции ***void addNodesAfter(char target, const string &nodes***

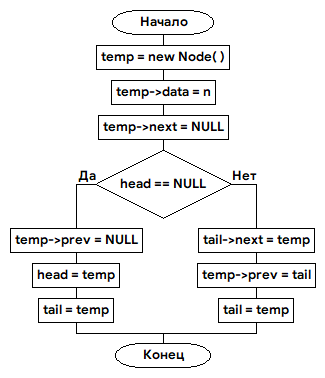


Рисунок 7 – Блок-схема функции ***void addNode(char n)***

Реализация односвязного списка на языке C++

Он начинается с определения класса Node, который представляет узел списка. Каждый узел состоит из поля data для хранения данных и указателя next для указания на следующий узел в списке.

Далее определяется класс firstLinkedList, который представляет сам список. Он содержит два поля: указатель на голову списка head и указатель на конец списка tail. В конструкторе оба поля инициализируются значением NULL.

Класс firstLinkedList имеет несколько методов.

* Метод addNode добавляет новый узел в конец списка.
* Метод insertMultipleAfter позволяет вставить несколько новых узлов после заданного элемента, передавая значения вектора values.
* Метод deleteNode удаляет первый найденный узел с указанным значением value.
* Метод printList выводит данные всех узлов списка.

В функции main создается объект list класса firstLinkedList и добавляются три узла с данными 'a', 'b' и 'c'. Затем вызываются методы insertMultipleAfter и deleteNode для вставки новых узлов и удаления узла с данными 'x'. После каждого изменения списка вызывается метод printList для вывода его содержимого на экран.

**Листинг программы**

#include <iostream>

#include <vector>

using namespace std;

class Node {

public:

char data;

Node \*next;

};

class firstLinkedList {

private:

Node \*head;

Node \*tail;

public:

firstLinkedList() {

head = NULL;

tail = NULL;

}

void addNode(char n) {

Node \*temp = new Node();

temp->data = n;

temp->next = NULL;

if (head == NULL) {

head = temp;

tail = temp;

} else {

tail->next = temp;

tail = temp;

}

}

void insertMultipleAfter(char after, const vector<char> &values) {

Node \*temp = head;

while (temp != NULL) {

if (temp->data == after) {

Node \*prev = temp;

for (char value : values) {

Node \*newNode = new Node();

newNode->data = value;

newNode->next = prev->next;

prev->next = newNode;

prev = newNode;

if (prev->next == NULL) {

tail = prev;

}

}

break;

}

temp = temp->next;

}

}

void deleteNode(char value) {

if (head == NULL) return;

if (head->data == value) {

Node \*temp = head;

head = head->next;

delete temp;

return;

}

Node \*prev = head;

Node \*current = head->next;

while (current != NULL) {

if (current->data == value) {

prev->next = current->next;

if (current->next == NULL) {

tail = prev;

}

delete current;

break;

}

prev = current;

current = current->next;

}

}

void printList() {

Node \*temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << endl;

}

};

int main() {

firstLinkedList list;

list.addNode('a');

list.addNode('b');

list.addNode('c');

list.printList(); // выведет: a b c

list.insertMultipleAfter('b', {'x', 'y', 'z'});

list.printList(); // выведет: a b x y z c

list.deleteNode('x');

list.printList(); // выведет: a b y z c

system("pause");

return 0;

}

**Результат работы программы**

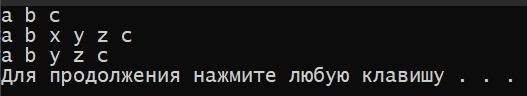


Рисунок 1 – Вывод программы

**Блок-схема программы**

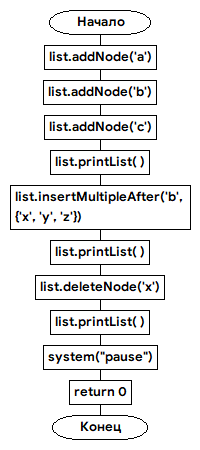


Рисунок 3 – Блок схема главной функции

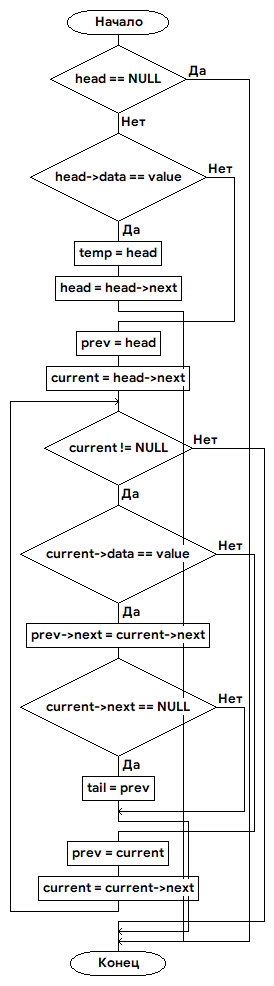


Рисунок 4 – Блок-схема функции ***void deleteNode(char value)***

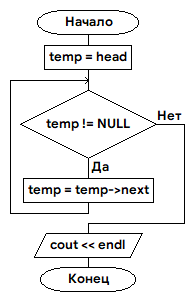


Рисунок 5 – Блок-схема функции ***void printList()***

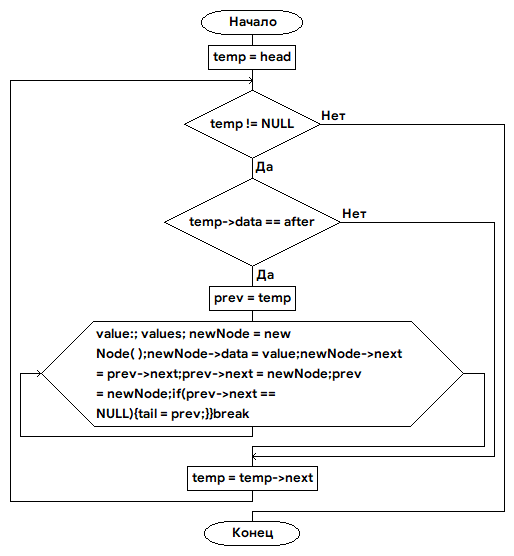


Рисунок 6 – Блок-схема функции ***void insertMultipleAfter(char after, const vector<char> &values)***

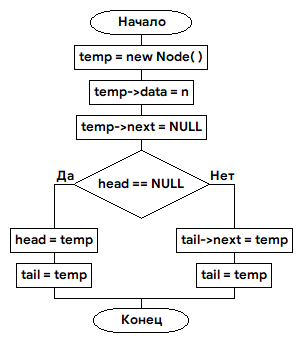


Рисунок 7 – Блок-схема функции ***void addNode(char n)***

Реализация очереди на языке C++

Очередь реализована на основе класса Queue, который имеет следующие методы:

* enqueue(const Data& data) - добавление элемента в конец очереди;
* dequeue() - удаление элемента из начала очереди;
* insertAt(int index, const Data& data) - вставка элемента на определенную позицию в очереди;
* removeAt(int index) - удаление элемента с определенной позиции в очереди;
* display() - вывод всех элементов очереди на экран.

В основном цикле программы происходит обработка выбранных пользователем действий:

* 1 - добавление элемента в очередь;
* 2 - удаление первого элемента из очереди;
* 3 - вставка элемента на определенную позицию в очереди;
* 4 - удаление элемента с определенной позиции в очереди;
* 5 - вывод всех элементов очереди на экран;
* 6 - завершение работы программы.

Пользователь выбирает действия с помощью ввода соответствующего номера. При выполнении действий, где требуется ввод индекса позиции или значения элемента, пользователь также должен ввести соответствующие данные.

**Листинг программы**

#include <iostream>

#include <vector>

struct Data {

int value;

};

class Queue {

public:

void enqueue(const Data& data) {

queue.push\_back(data);

}

void dequeue() {

if (!queue.empty()) {

queue.erase(queue.begin());

}

}

void insertAt(int index, const Data& data) {

if (index >= 0 && index <= queue.size()) {

queue.insert(queue.begin() + index, data);

}

}

void removeAt(int index) {

if (index >= 0 && index < queue.size()) {

queue.erase(queue.begin() + index);

}

}

void display() {

for (const auto& data : queue) {

std::cout << data.value << " ";

}

std::cout << std::endl;

}

private:

std::vector<Data> queue;

};

int main() {

Queue q;

int choice, index;

Data data;

while (true) {

std::cout << "1. Enqueue\n2. Dequeue\n3. Insert at index\n4. Remove at index\n5. Display\n6. Exit\n";

std::cin >> choice;

switch (choice) {

case 1:

std::cout << "Enter value: ";

std::cin >> data.value;

q.enqueue(data);

break;

case 2:

q.dequeue();

break;

case 3:

std::cout << "Enter index: ";

std::cin >> index;

std::cout << "Enter value: ";

std::cin >> data.value;

q.insertAt(index, data);

break;

case 4:

std::cout << "Enter index: ";

std::cin >> index;

q.removeAt(index);

break;

case 5:

q.display();

break;

case 6:

return 0;

default:

std::cout << "Invalid choice\n";

}

}

return 0;

}

**Блок-схема программы**

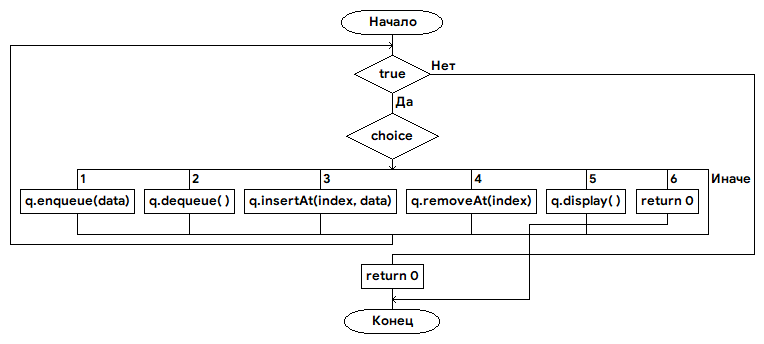


Рисунок 3 – Блок схема главной функции

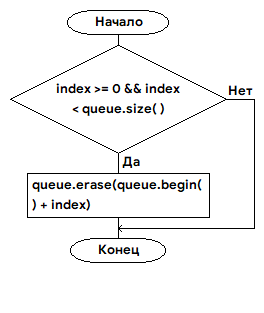


Рисунок 4 – Блок-схема функции

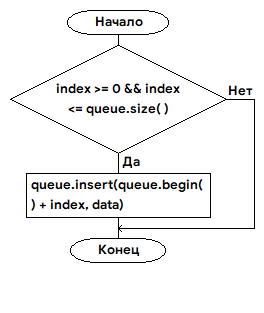


Рисунок 5 – Блок-схема функции

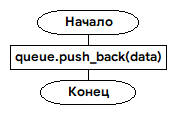


Рисунок 6 – Блок-схема функции

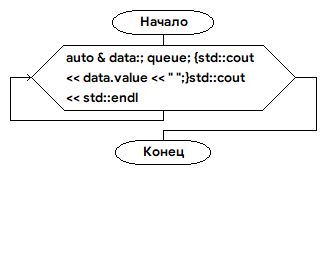


Рисунок 7 – Блок-схема функции

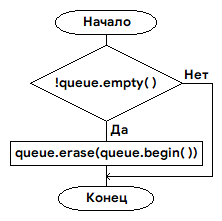


Рисунок 8 – Блок-схема функции

Реализация стека на языке C++

Данный код реализует стек для хранения информации о братьях. В рамках функции main() происходит инициализация стека, добавление элементов в стек (функцией push()), удаление элементов из стека (функцией pop()), вывод на экран содержимого стека (функцией print()).

Структура Brothers хранит имя, возраст и флаг женат/не женат для каждого члена семьи.

Функция push() помещает новый элемент на вершину стека, если это не приводит к выходу за пределы массива. Функция pop() удаляет последний добавленный элемент из вершины стека, если он есть.

Функции isEmpty() и size() используются для проверки состояния пустоты или размерности массива соответствующих логических полостью аргументного списка Stack stack ответную результатом типа bool строки - true or false.

clear () удаление всех элементов. Также определена функции top(), которая возвращает значение первого элемента сверху без его удаления из Stack stack.

Возможна модификация значений устанавливаемых перед объектами struct Brothers после выполнения pop(). Существует условие проверки признак marital status ("женат-не женат") каждой запись Before:«Not married: name = Valeriy , age is 9» After: «Married: name = Valeriy , age is 9» с исправлениями через назначение значение группе данных после изменений .

Последняя строка программы system("pause"); нужна только для того, чтобы консольное окно оставалось открытым до нажатия на любую клавишу пользователем.

**Листинг программы**

#include <iostream>

#include <string.h>

#define N 6

using namespace std;

struct Brothers

{

string name;

int age;

bool married;

};

struct Stack

{

Brothers array[N];

int arrayElementsCount = 0;

void push(Brothers Brothers)

{

if (size() >= N)

{

cout << "Stack is full" << endl;

}

else

{

array[arrayElementsCount] = Brothers;

arrayElementsCount++;

}

}

void pop()

{

if (isEmpty())

{

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

}

else

{

array[arrayElementsCount - 1] = {};

arrayElementsCount--;

}

}

bool isEmpty()

{

if (arrayElementsCount == 0)

return true;

else

return false;

}

int size()

{

return arrayElementsCount;

}

void clear()

{

for (int i = size() - 1; i >= 0; i--)

{

array[i] = {};

}

arrayElementsCount = 0;

}

Brothers top()

{

if (!isEmpty())

{

return array[arrayElementsCount - 1];

}

else

{

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

return {"Null element", 0, false};

}

}

};

void print(Stack stack)

{

Stack stackCopy = stack;

Brothers group;

for (int i = 0; i < stack.size(); i++)

{

group = stackCopy.top();

if (group.married)

{

cout << "Married: name = " << group.name << " , age is " << group.age << endl;

}

else

cout << "Not married: name = " << group.name << " , age is " << group.age << endl;

stackCopy.pop();

}

}

int main()

{

Stack stack;

Brothers group;

stack.push({"Oleg", 39, true});

stack.push({"Igor", 83, true});

stack.push({"Valeriy", 9, false});

stack.push({"Arkadiy", 14, false});

print(stack);

cout << endl;

group = stack.top();

stack.pop();

group.married = true;

stack.push(group);

print(stack);

cout << endl;

stack.push({"Vitaly", 93, false});

stack.push({"Stas", 452, true});

stack.push({"Maxim", 8, false});

print(stack);

cout << endl;

stack.pop();

print(stack);

cout << endl;

for (int i = 0; i < N; i++)

{

stack.pop();

}

print(stack);

system("pause");

return 0;

}

**Блок-схема программы**

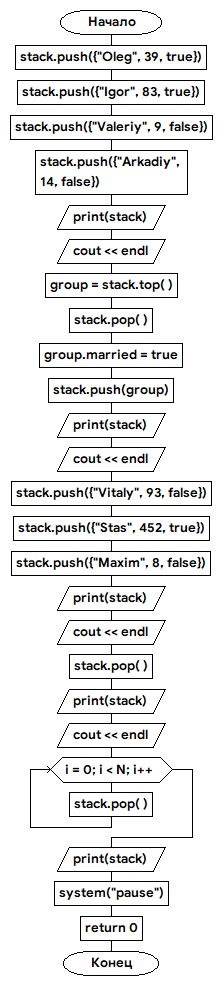


Рисунок 3 – Блок схема главной функции

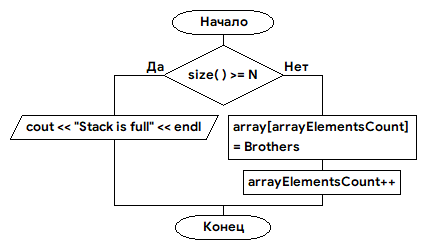


Рисунок 4 – Блок-схема функции

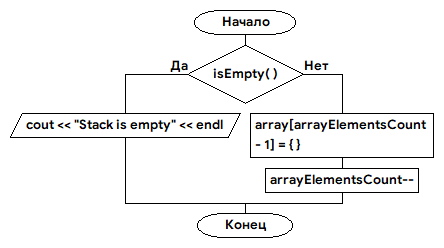


Рисунок 5 – Блок-схема функции

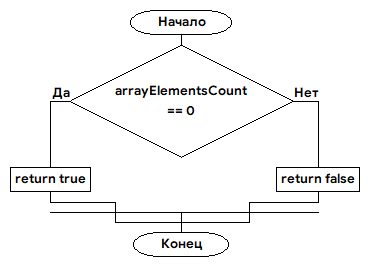


Рисунок 6 – Блок-схема функции

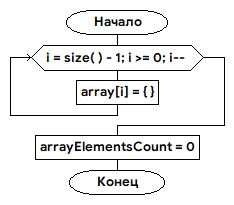


Рисунок 7 – Блок-схема функции

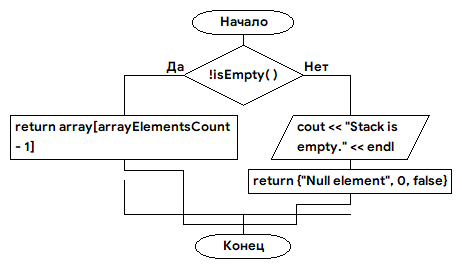


Рисунок 8 – Блок-схема функции