МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

« МИРЭА Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Вычислительной техники

УЧЕБНОЕ ЗАДАНИЕ

по дисциплине

« Объектно-ориентированное программирование»

Наименование задачи:

« Задача 9_1_2 »

С тудент группы	ИКБО-13-21	Дамарад Д.В.
Руководитель практики	Ассистент	Асадова Ю.С.
Работа представлена	«___»_____ 2022 г.	_____
		(подпись студента)
Оценка		_____
		(подпись руководителя)

Москва 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Постановка задачи

Перегрузка побитовых логических операции

Задан элемент, состоящий из ячейки памяти данных **объемом один байт** и шаблона активных битов (размер также равен 1 байту). Между данными из ячеек памяти двух элементов можно выполнить побитовые логические операции умножения и сложения. От каждого элемента в операциях участвуют только те биты данных, которые соответствуют шаблону активных битов элемента.

Работа с элементами выполняется следующим образом. Первоначально создаём элементы, определяем для них содержимое ячейки памяти и значение шаблона в шестнадцатеричной системе счисления. Далее описываем логические выражения, включающие эти элементы.

Написать программу, которая моделирует работу с элементами.

В основной программе реализовать алгоритм:

1. Ввод количества элементов n.
2. В цикле для каждого элемента вводится исходное значение ячейки памяти и значение шаблона активных битов. Далее создается объект, в конструктор которого передаются значения памяти и шаблона. Каждому объекту присваивается свой номер от 1 до n.
3. В цикле, последовательно и построчно, вводится «номер первого объекта»«символ логической операции & или |»«номер второго объекта»
4. После каждого нового ввода логического выражения выполняется логическая операция, результат записывается в ячейку памяти первого элемента (объекта).
5. Цикл завершается в тот момент, когда на ввод больше нет данных.
6. Выводится результат последней операции в шестнадцатеричном формате.

Количество элементов больше или равно 2.
Использовать перегрузку логических побитовых операций, реализовав в составе описания класса.

Пояснения.
Значения в пояснении заданы в шестнадцатеричной системе счисления.
Значение логической единицы (1) в шаблоне задаёт активный бит значения из ячейки памяти. Если значение шаблона равно 15, то активными будут считаться 4-й, 2-й и 0-й биты значения из ячейки памяти.
В логической операции между двумя элементами участвуют только те активные биты ячеек памяти, позиции которых совпадают у обоих элементов (находятся на пересечении). Например, если значение шаблона одного элемента равно 0F, а другого 0C, то в логической операции участвуют только 3-й и 2-й биты обоих значений. Соответственно, при записи результата в первый элемент изменениям подвергаются только те биты, которые участвовали в операции.

Первый элемент e1: значение памяти 8F, значение шаблона 0F.
Второй элемент e2: значение памяти 02, значение шаблона 01.
Операция e1 & e2. Значение первого элемента равно 8E,

Первый элемент e1: значение памяти 8F, значение шаблона 0F.
Второй элемент e2: значение памяти 02, значение шаблона F0.
Операция e1 & e2. Значение первого элемента равно 8F,

Описание входных данных

Первая строка содержит значение количества элементов n: «Натуральное значение»

Далее n строк содержат «Шестнадцатеричное значение»«Шестнадцатеричное значение»

Начиная с n + 2 строки: «Натуральное значение»«Знак операции»«Натуральное значение»

Описание выходных данных

«Шестнадцатеричное значение»

Метод решения

Для решения данной задачи используются:

- Объекты стандартных потоков ввода и выводы cin и cout соответсвенно для ввода и вывода на экран.
- Объекты класса Memory ssb количестве заданном пользователем.
- Методы - операторы & b и | для объектов класса Triangle.
- Библиотека контейнеров vector - для хранения объектов.
- Функция pow из библиотеки cmath, функция reverse из algoritmh, функции byteToString и stringToByte - для заданных в постановке операций.

Класс Memory:

- Поля:
    - Поле, хранящее значение ячейки памяти:
        - Наименование - memory;
        - Тип - byte (синоним для unsigned char);
        - Модификатор доступа - private.
    - Поле, хранящее значение шаблона:
        - Наименование - temp;
        - Тип - byte (синоним для unsigned char);
        - Модификатор доступа - private.
    - Методы:
        - Метод Memory:
            - Функционал - параметризированный конструктор.
        - Метода Getmemoryvalue:

- Функционал - возращает значение закрытого поля memory.

Описание алгоритма

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

Функция: main

Функционал: Основной алгоритм программы

Параметры: Отсутсвуют

Возвращаемое значение: Целочисленное значение - код возврата

Алгоритм функции представлен в таблице 1.

Таблица 1. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление целочисленных переменных n,m1,m2 и i. Присвоение i=0. Объявление символьной переменной op	2	
2		Считывание значения n с клавиатуры	3	
3		Создание контейнера vec для объектов класса Memory	4	
4	i меньше n	Объявление целочисленных переменных memoryValue,tempValu	5	

		e		
			8	
5		Считывание в 16-ричной системе счисления переменных memoryValue,tempValue	6	
6		Вызов метода push_back от контейнера vec с параметром Memory с параметрами memoryValue,tempValue	7	
7		Увеличение i на 1	4	
8	Считывание значений m1,op,m2 с клавиатуры		9	
			10	
9	op равно '&'	Присваивание элементу контейнера vec с индексом m1-1 значения выражения: элемент контейнера vec с индексом m1-1 & элемент контейнера vec с индексом m2-1	8	
		Присваивание элементу контейнера vec с индексом m1-1 значения выражения: элемент контейнера vec с индексом m1-1 | элемент контейнера vec с индексом m2-1	8	
10	Переведенное в целочисленный тип значение метода Get_memoryValue от элемента	Вывод на экран "0"	11	

	контейнера vec с индексом m1-1 меньше 16			
			11	
11		Вызов от cout метода setf с параметром ios::uppercase; вывод в 16-ричной системе счисления переведенного в целочисленный тип значения метода Get_memoryValue от элемента контейнера vec с индексом m1-1	∅	

Конструктор класса: Memory

Модификатор доступа: public

Функционал: Параметризированный конструктор

Параметры: byte memory, byte temp - значения ячейки памяти и шаблона

Алгоритм конструктора представлен в таблице 2.

Таблица 2. Алгоритм конструктора класса Memory

№	Предикат	Действия	№ перехода	Комментарий
1		Присваивание указателю на поле класса memory значения параметра memory	2	
2		Присваивание указателю на поле класса temp значения параметра temp	∅	

Класс объекта: Memory

Модификатор доступа: public

Метод: Get_memoryValue

Функционал: Возврат значения закрытого поля memory

Параметры: Отсутсвуют

Возвращаемое значение: byte - значение закрытого поля memory

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода Get_memoryValue класса Memory

№	Предикат	Действия	№ перехода	Комментарий
1		Возврат значения поля memory	∅	

Функция: byteToString

Функционал: Перевод byte в строку двоичных цифр

Параметры: byte b - значения байта, которое необходимо перевести в string

Возвращаемое значение: string bin - переведенное в строку двоичных цифр значение байта b

Алгоритм функции представлен в таблице 4.

Таблица 4. Алгоритм функции byteToString

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление целочисленной переменной n и присваивание ей приведенного к целочисленному типу значения b	2	
2		Создание строки bin	3	
3	num -	Прибавить к bin значение	4	

	истина	метода to_string с параметром n%2		
			5	
4		Присваивание n значения n, разделенного на 2	3	
5		Вызов функции reverse с параметрами bin.begin() и bin.end()	6	
6	bin.length() не равно 8	Вызов метода insert от строки bin с параметрами: bin.begin(), '0'	6	
		Возврат bin	∅	

Функция: stringToByte

Функционал: Перевод строки двоичных чисел в byte

Параметры: string s - строка, которую необходимо перевести в byte

Возвращаемое значение: byte - значение строки s, переведенное в byte

Алгоритм функции представлен в таблице 5.

Таблица 5. Алгоритм функции stringToByte

№	Предикат	Действия	№ перехода	Комментарий
1		Инициализация целочисленной переменной i, присваивание i равно 7	2	
2		Инициализация byte result, присваивание ей нуля	3	
3	Символ ch в строке s		4	
			6	
4	ch минус '0' равно 1	Прибавить к result значение функции pow с параметрами	5	

		2, i		
			5	
5		Отнять 1 от i	6	
6		Вернуть result	∅	

Функция: operator &

Функционал: Выполнение побитового умножения

Параметры: Ссылки на объекты m1,m2 класса Memory

Возвращаемое значение: Объект класса Memory

Алгоритм функции представлен в таблице 6.

Таблица 6. Алгоритм функции operator &

№	Предикат	Действия	№ перехода	Комментарий
1		Инициализация целочисленной переменной i, присваивание ей нуля	2	
2		Объявление строки m1, присваивание ей значения функции byteToString с параметром m1.memory	3	
3		Объявление строки m2, присваивание ей значения функции byteToString с параметром m2.memory	4	
4		Объявление строки temp1, присваивание ей значения функции byteToString с параметром m1.temp	5	
5		Объявление строки temp2, присваивание ей значения функции byteToString с параметром m2.temp	6	
6		Объвление строки result, присваивание ей значения memory1	7	

7	i меньше 8		8	
		Возврат объекта класса Memory с параметрами: значение функции stringToByte с параметром result, m1.temp	∅	
8	temp1[i] равно '1' и temp2[i] равно '1'		9	
			7	
9	memory1[i] равно '1' и memory2[i] равно '1'	result[i] равно '1'	10	
		result[i] равно '0'	10	
10		Увеличение i на 1	7	

Функция: operator |

Функционал: Выполнение побитового логического сложения

Параметры: Ссылки на объекты m1,m2 класса Memory

Возвращаемое значение: Объект класса Memory

Алгоритм функции представлен в таблице 7.

Таблица 7. Алгоритм функции operator |

№	Предикат	Действия	№ перехода	Комментарий
1		Инициализация целочисленной переменной i, присваивание ей нуля	2	
2		Объявление строки m1, присваивание ей значения функции byteToString с параметром m1.memory	3	

3		Объявление строки m2, присваивание ей значения функции byteToString с параметром m2.memory	4	
4		Объявление строки temp1, присваивание ей значения функции byteToString с параметром m1.temp	5	
5		Объявление строки temp2, присваивание ей значения функции byteToString с параметром m2.temp	6	
6		Объвление строки result, присваивание ей значения memory1	7	
7	i меньше 8		8	
		Возврат объекта класса Memory с параметрами: значение функции stringToByte с параметром result, m1.temp	∅	
8	temp1[i] равно '1' и temp2[i] равно '1'		9	
			7	
9	memory1[i] равно '1' и memory2[i] равно '1'	result[i] равно '1'	10	
		result[i] равно '0'	10	
10		Увеличение i на 1	7	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

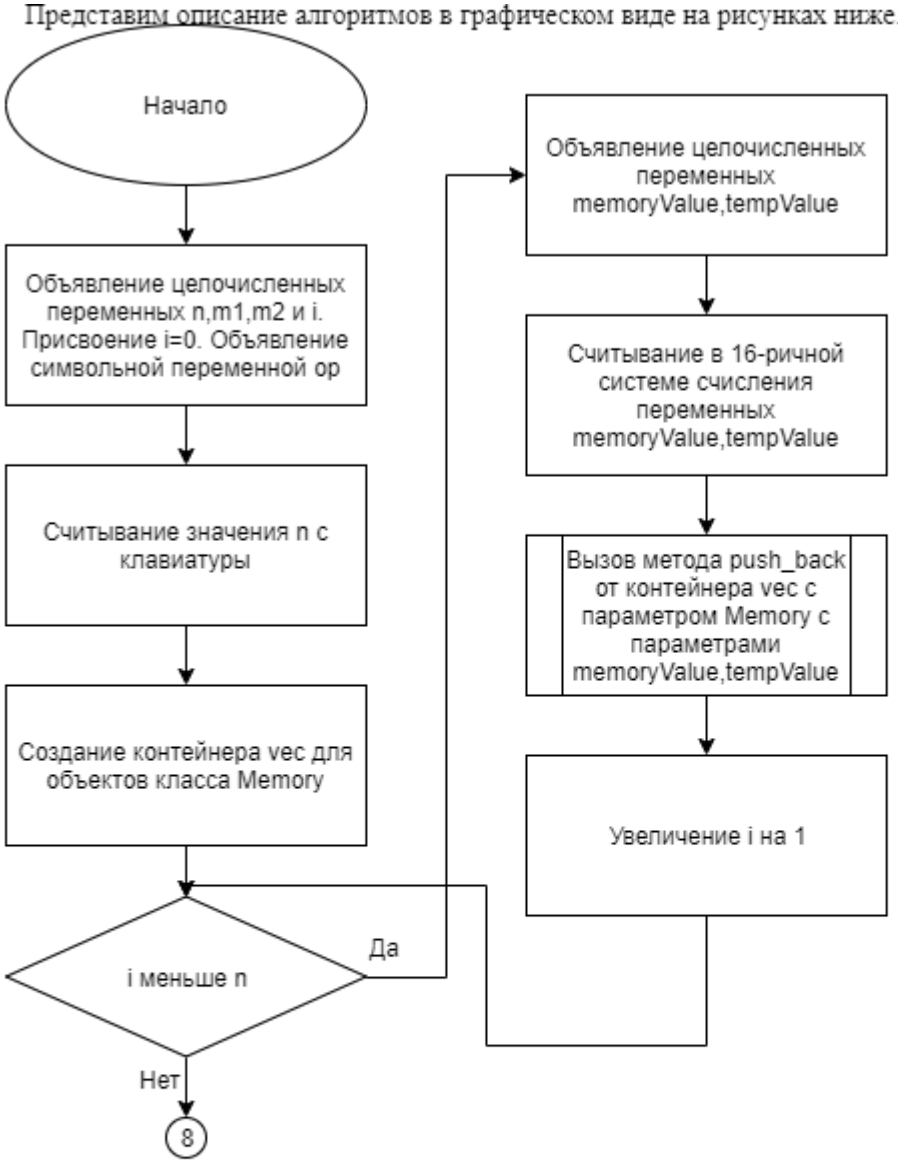

Рис. 1. Блок-схема алгоритма.

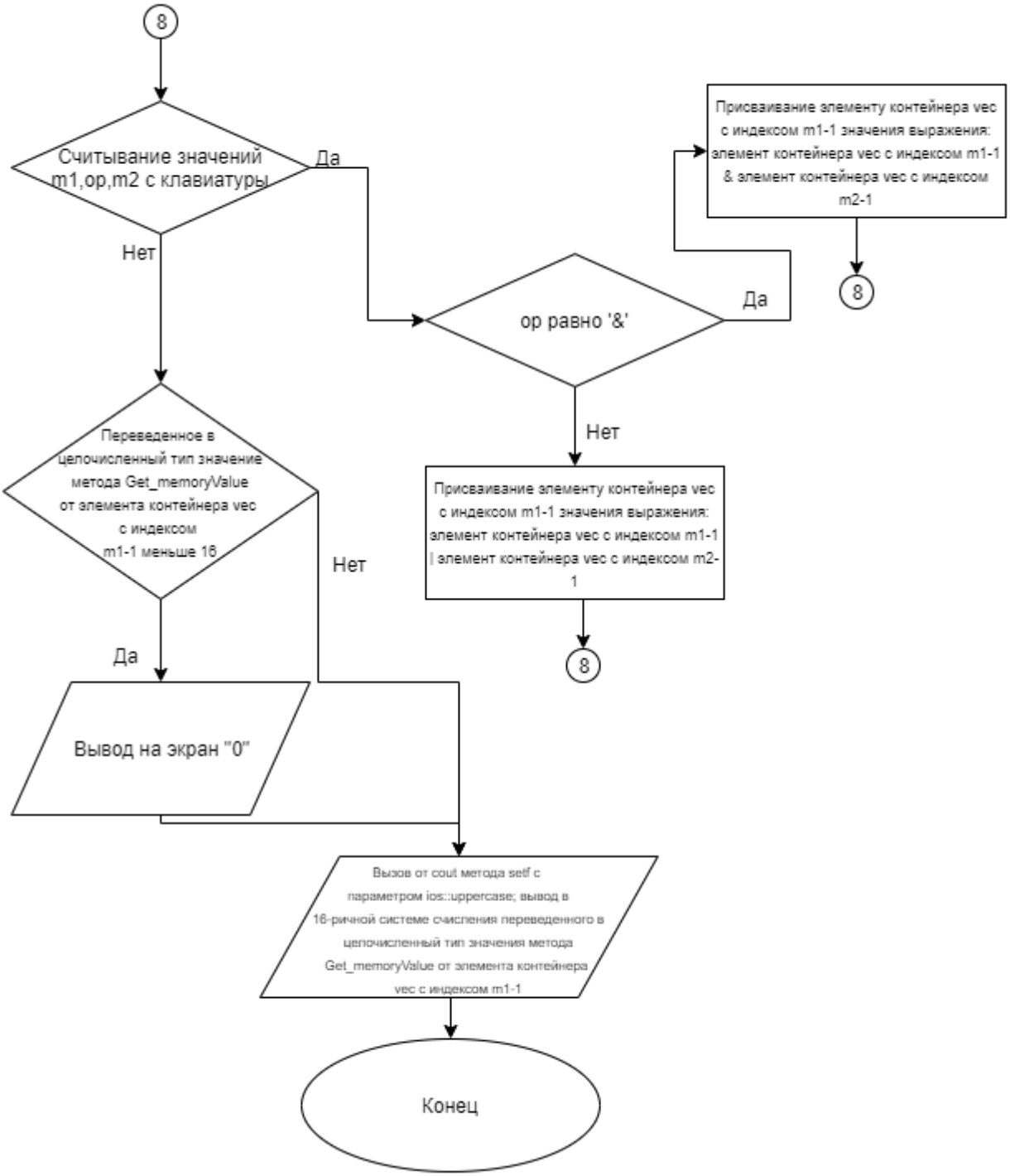

Рис. 2. Блок-схема алгоритма.

Рис. 3. Блок-схема алгоритма.

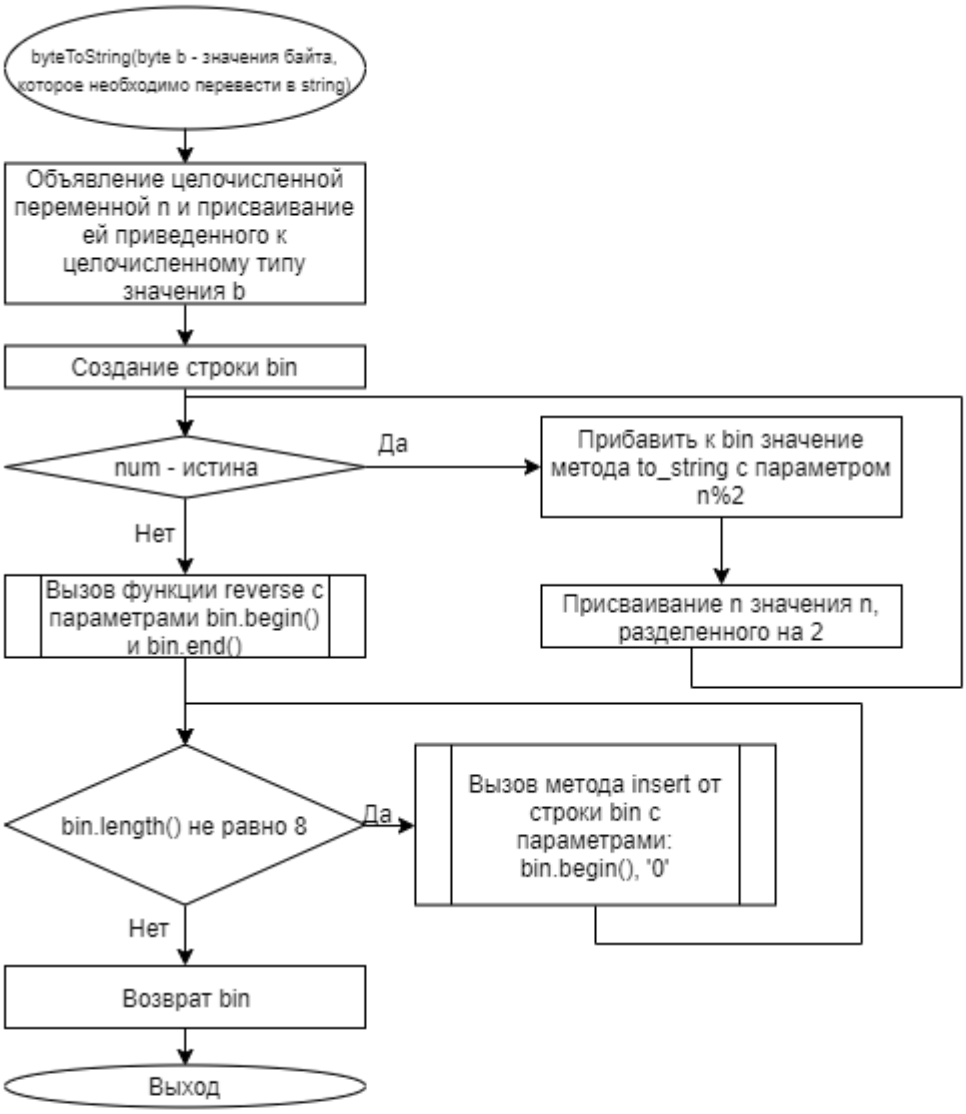

Рис. 4. Блок-схема алгоритма.

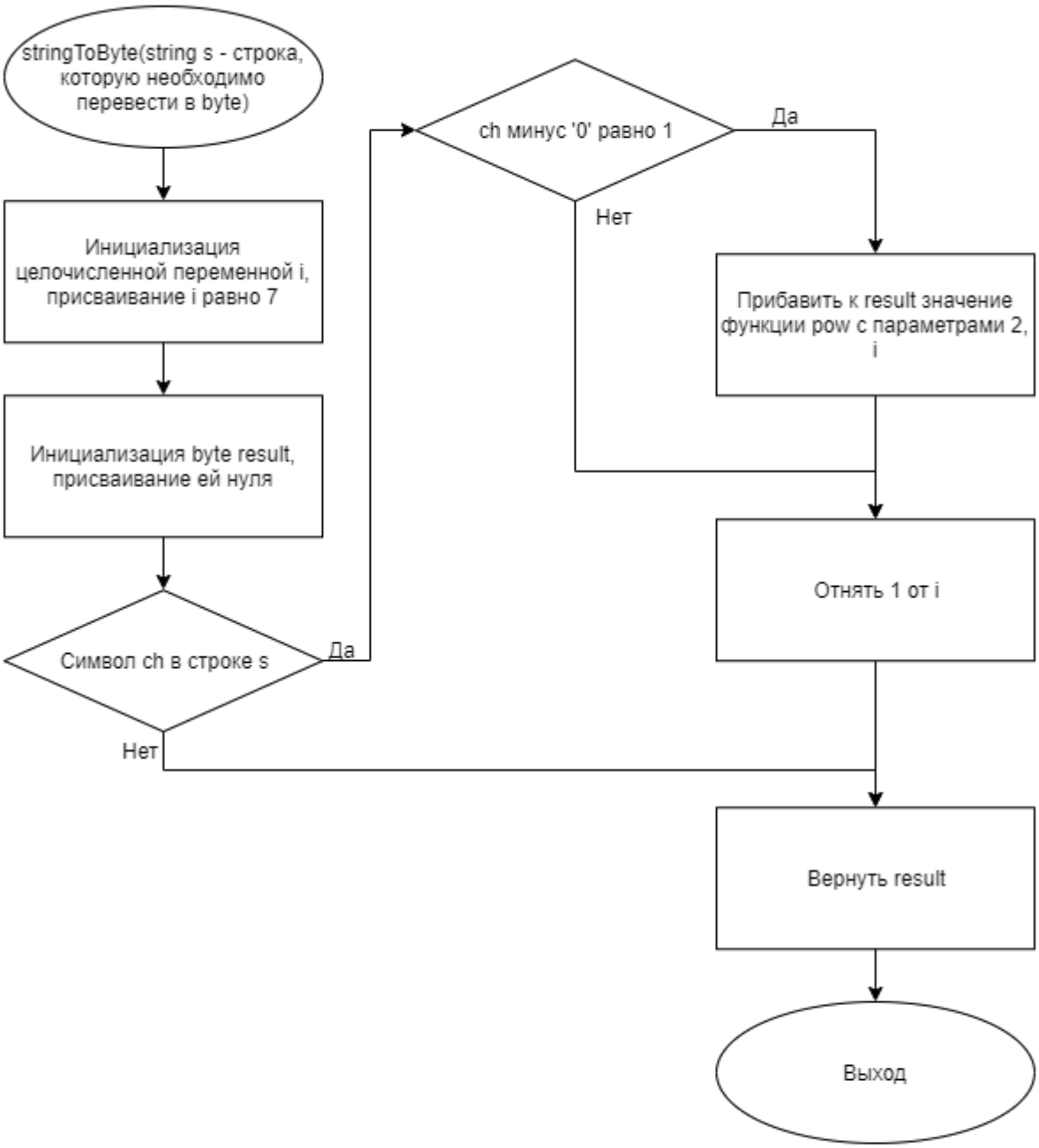

Рис. 5. Блок-схема алгоритма.

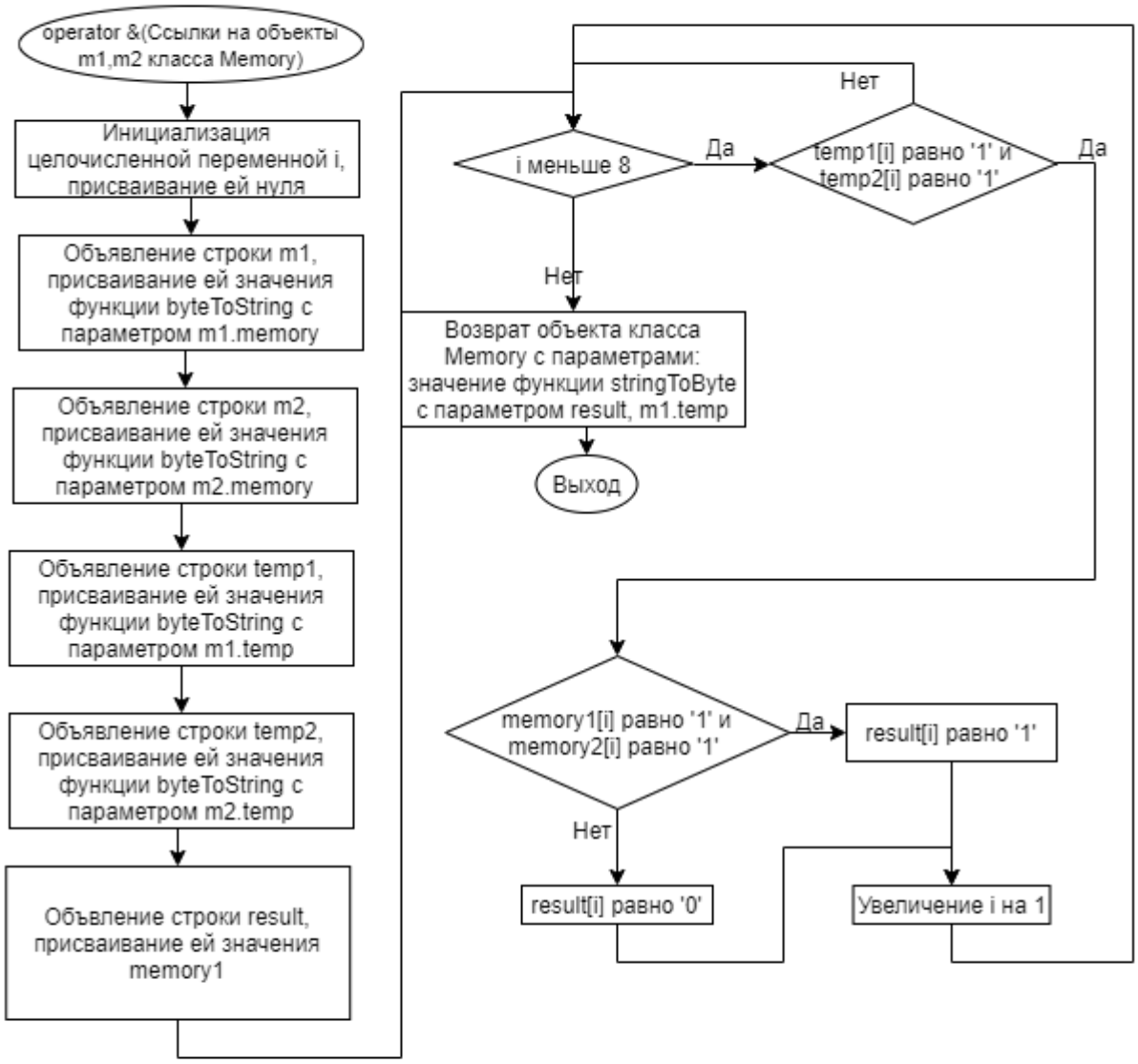

Рис. 6. Блок-схема алгоритма.

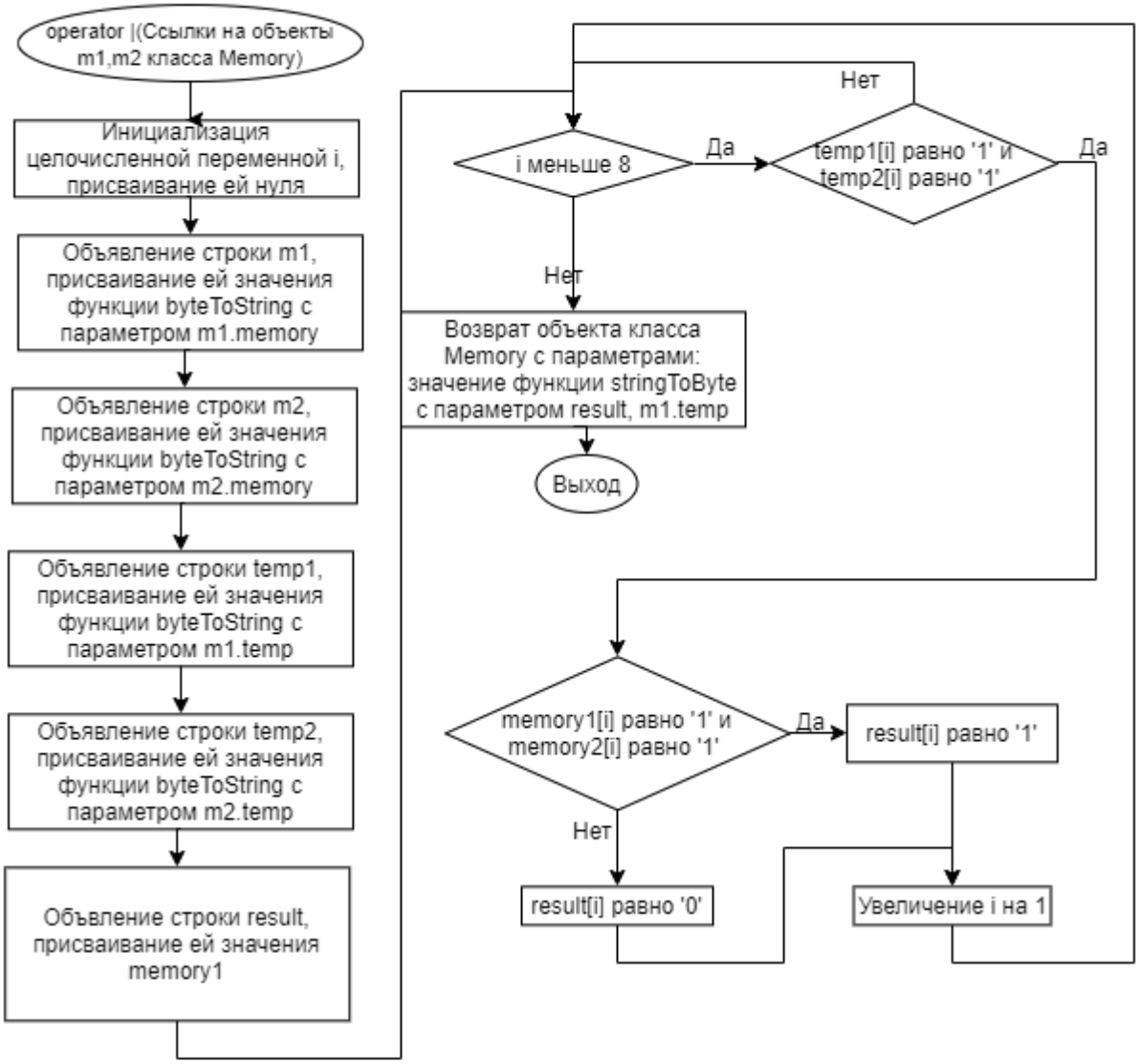

Рис. 7. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл main.cpp

```
#include "Memory.h"
#include <iostream>
#include <vector>
int main(){
        int n,m1,m2,i=0;
        char op;
        cin>>n;
        vector<Memory> vec;
        while (i<n){
                int memoryValue, tempValue;
                cin>>hex>>memoryValue>>hex>>tempValue;
                vec.push_back(Memory(memoryValue,tempValue));
                i+=1;
        }
        while (cin>>m1>>op>>m2){
                if(op=='&'){
                        vec[m1-1]=vec[m1-1]&vec[m2-1];
                }
                else{
                        vec[m1-1]=vec[m1-1]|vec[m2-1];
                }
        }
        if ((int)vec[m1-1].Get_memoryValue()<16){
                cout<<"0";
        }
        cout.setf(ios::uppercase);
        cout<<hex<<(int)vec[m1-1].Get_memoryValue();
        return 0;
}
```

Файл Memory.cpp

```
#include "Memory.h"

Memory::Memory(byte memory, byte temp){
        this->memory=memory;
        this->temp=temp;
}

byte Memory::Get_memoryValue(){
        return this->memory;
}
```

```
Memory operator &(const Memory &m1, const Memory &m2){
        int i=0;
        string memory1=byteToString(m1.memory);
        string memory2=byteToString(m2.memory);
        string temp1=byteToString(m1.temp);
        string temp2=byteToString(m2.temp);
        string result=memory1;
        while (i<8){
                if (temp1[i]=='1' and temp2[i]=='1'){
                        if (memory1[i]=='1' and memory2[i]=='1'){
                                result[i]='1';
                        }
                        else{
                                result[i]='0';
                        }
                }
                i+=1;
        }
        return Memory(stringToByte(result), m1.temp);
}

Memory operator |(const Memory &m1, const Memory &m2){
        int i=0;
        string memory1=byteToString(m1.memory);
        string memory2=byteToString(m2.memory);
        string temp1=byteToString(m1.temp);
        string temp2=byteToString(m2.temp);
        string result=memory1;
        while (i<8){
                if (temp1[i]=='1' and temp2[i]=='1'){
                        if (memory1[i]=='1' or memory2[i]=='1'){
                                result[i]='1';
                        }
                        else{
                                result[i]='0';
                        }
                }
                i+=1;
        }
        return Memory(stringToByte(result), m1.temp);
}

string byteToString(byte b){
        int n=(int)b;
        string bin;
        while (n){
                bin+=to_string(n%2);
                n/=2;
        }
        reverse(bin.begin(), bin.end());
        while (bin.length()!=8){
                bin.insert(bin.begin(),'0');
        }
        return bin;
}
```

```
byte stringToByte(string s){
        int i=7;
        byte result=0;
        for (char ch:s){
                if (ch-'0'==1){
                        result+=pow(2,i);
                }
                i-=1;
        }
        return result;
}
```

Файл Memory.h

```
#ifndef _MEMORY_H
#define _MEMORY_H
#include <string>
#include <algorithm>
#include <cmath>

using namespace std;

typedef unsigned char byte;

class Memory{
private:
        byte memory;
        byte temp;
public:
        Memory(byte memory, byte temp);
        byte Get_memoryValue();
        friend Memory operator &(const Memory &m1, const Memory &m2);
        friend Memory operator |(const Memory &m1, const Memory &m2);
};
string byteToString(byte b);
byte stringToByte(string s);
#endif
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
2 8F 1F 02 F0 1 & 2	8F	8F
2 8F 1F 02 F1 1 | 2	8F	8F
2 01 02 03 05 1 & 2	01	01
2 8F 1F 02 F0 1 | 2	8F	8F

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ (ИСТОЧНИКОВ)

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

4. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).