Балаковский инженерно-технологический институт - филиал

федерального государственного автономного образовательного учреждения

высшего образования

«Национальный исследовательский ядерный университет «МИФИ»

Факультет атомной энергетики и технологий

Кафедра «Информатика и управление в технических системах»

КОНТРОЛЬНАЯ РАБОТА

по дисциплине

«ЯЗЫКИ ПРОГРАММИРОВАНИЯ»

Вариант №9

Выполнил: студент группы ИФСТ-2з

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Никольский А.Г.

« 27 » ноября 2020г.

|  |  |
| --- | --- |
|  | Проверил ст. преподаватель каф. ИУС  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Михеев И.В.  «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 202\_\_\_г. |

Балаково 2020

Цель работы: получение практических навыков по реализации программных средств на языке программирования Си, а также закрепление знаний, полученных в результате изучения дисциплины.

1 ТЕОРИТЕЧЕСКАЯ ЧАСТЬ

Битовые(поразрядные) операторы.

Поразрядные операторы обрабатывают отдельные биты.

Поскольку C++ нацелен на то, чтобы позволить полный доступ к аппаратным средствам компьютера, важно, чтобы он имел возможность непосредственно воздействовать на отдельные биты в рамках байта или машинного слова. Именно поэтому C++ и содержит поразрядные операторы. Поразрядные операторы предназначены для тестирования, установки или сдвига реальных битов в байтах или словах, которые соответствуют символьным или целочисленным С++-типам. Поразрядные операторы не используются для операндов типа bool, float, double, long double, void или других еще более сложных типов данных. Поразрядные операторы (они перечислены в табл. 9.1) очень часто используются для решения широкого круга задач программирования системного уровня, например, при опросе информации о состоянии устройства или ее формировании. Теперь рассмотрим каждый оператор этой группы в отдельности.

Таблица 1 - Поразрядные операторы

|  |  |
| --- | --- |
| Оператор | Значение |
| & | Поразрядное И |
| | | Поразрядное ИЛИ |
| ^ | Поразрядное исключающее ИЛИ |
| >> | Сдвиг вправо |
| << | Сдвиг влево |
| ~ | Дополнение до 1 (унарный оператор НЕ) |

Поразрядные операторы И, ИЛИ, исключающее ИЛИ и НЕ (обозначаемые символами &, |, ^ и ~ соответственно) выполняют те же операции, что и их логические эквиваленты (т.е. они действуют согласно той же таблице истинности). Различие состоит лишь в том, что поразрядные операции работают на побитовой основе. В следующей таблице показан результат выполнения каждой поразрядной операции для всех возможных сочетаний операндов (нулей и единиц).

Таблица 2 - Таблица истинности поразрядных операторов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | q | p & q | p | q | p ^ q | ~p |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

Как видно из таблицы, результат применения оператора XOR (исключающее ИЛИ) будет равен значению ИСТИНА (1) только в том случае, если истинен (равен значению 1) лишь один из операндов; в противном случае результат принимает значение ЛОЖЬ (0). Поразрядный оператор И можно представить как способ подавления битовой информации. Это значит, что 0 в любом операнде, обеспечит установку в 0 соответствующего бита результата. Вот пример.

1101 0011

& 1010 1010

1000 0010

Следующая программа считывает символы с клавиатуры и преобразует любой строчный символ в его прописной эквивалент путем установки шестого бита равным значению 0. Набор символов ASCII определен так, что строчные буквы имеют почти такой же код, что и прописные, за исключением того, что код первых отличается от кода вторых ровно на 32[только для латинского алфавита]. Следовательно, как показано в этой программе, чтобы из строчной буквы сделать прописную, достаточно обнулить ее шестой бит.

// Получение прописных букв.

#include <iostream>

using namespace std;

int main()

{

char ch;

do {

cin >> ch;

// Эта инструкция обнуляет 6-й бит.

ch = ch & 223; // В переменной ch теперь прописная буква.

cout << ch;

} while (ch!= 'Q');

return 0;

}

Значение 223, используемое в инструкции поразрядного И, является десятичным представлением двоичного числа 1101 1111. Следовательно, эта операция И оставляет все биты в переменной ch нетронутыми, за исключением шестого (он сбрасывается в нуль). Оператор И также полезно использовать, если нужно определить, установлен ли интересующий вас бит (т.е. равен ли он значению 1) или нет. Например, при выполнении следующей инструкции вы узнаете, установлен ли 4-й бит в переменной status,

if(status & 8) cout << "Бит 4 установлен";

Чтобы понять, почему для тестирования четвертого бита используется число 8, вспомните, что в двоичной системе счисления число 8 представляется как 0000 1000, т.е. в числе 8 установлен только четвертый разряд. Поэтому условное выражение инструкции if даст значение ИСТИНА только в том случае, если четвертый бит переменной status также установлен (равен 1). Интересное использование этого метода показано на примере функции disp\_binary(). Она отображает в двоичном формате конфигурацию битов своего аргумента. Мы будем использовать функцию disp\_binary() ниже в этой главе для исследования возможностей других поразрядных операций.

// Отображение конфигурации битов в байте.

void disp\_binary(unsigned u)

{

register int t;

for (t = 128; t > 0; t = t / 2)

if (u & t) cout << "1";

else cout << "0 ";

cout << "\n";

}

Функция disp\_binary(), используя поразрядный оператор И, последовательно тестирует каждый бит младшего байта переменной u, чтобы определить, установлен он или сброшен. Если он установлен, отображается цифра 1, в противном случае — цифра 0. Интереса ради попробуйте расширить эту функцию так, чтобы она отображала все биты переменной u, а не только ее младший байт.

Поразрядный оператор ИЛИ, в противоположность поразрядному И, удобно использовать для установки нужных битов в единицу. При выполнении операции ИЛИ наличие в любом операнде бита, равного 1, означает, что в результате соответствующий бит также будет равен единице. Вот пример.

1101 0011

| 1010 1010

1111 1011

Можно использовать оператор ИЛИ для превращения рассмотренной выше программы (которая преобразует строчные символы в их прописные эквиваленты) в ее "противоположность", т.е. теперь, как показано ниже, она будет преобразовывать прописные буквы в строчные.

// Получение строчных букв.

#include <iostream>

using namespace std;

int main()

{

char ch;

do {

cin >> ch;

/\* Эта инструкция делает букву строчной, устанавливая ее 6-й

бит.\*/

ch = ch | 32;

cout << ch;

} while (ch != 'q');

return 0;

}

Установка шестого бита превращает прописную букву в ее строчный эквивалент. Поразрядное исключающее ИЛИ (XOR) устанавливает в единицу бит результата только в том случае, если соответствующие биты операндов отличаются один от другого, т.е. не равны. Вот пример:

0111 1111

^ 1011 1001

1100 0110

Унарный оператор НЕ (или оператор дополнения до 1) инвертирует состояние всех битов своего операнда. Например, если целочисленное значение (хранимое в переменной А), представляет собой двоичный код 1001 0110, то в результате операции ~А получим двоичный код 0110 1001.В следующей программе демонстрируется использование оператора НЕ посредством отображения некоторого числа и его дополнения до 1 в двоичном коде с помощью приведенной выше функции disp\_binary().

#include <iostream>

using namespace std;

void disp\_binary(unsigned u);

int main()

{

unsigned u;

cout << "Введите число между 0 и 255: ";

cin >> u;

cout << "Исходное число в двоичном коде: ";

disp\_binary(u);

cout << "Его дополнение до единицы: ";

disp\_binary(~u);

return 0;

} /

/ Отображение битов, составляющих байт.

void disp\_binary(unsigned u)

{

register int t; for (t = 128; t > 0; t = t / 2)

if (u & t) cout << "1";

else cout << "0";

cout << "\n";

}

Логические и поразрядные операторы выполняют различные действия. Операторы &, | и ~ применяются непосредственно к каждому биту значения в отдельности. Эквивалентные логические операторы обрабатывают в качестве операндов значения ИСТИНА/ЛОЖЬ (не нуль/нуль). Поэтому поразрядные операторы нельзя использовать вместо их логических эквивалентов в условных выражениях. Например, если значение х равно 7, то выражение х && 8 имеет значение ИСТИНА, в то время как выражение х & 8 дает значение ЛОЖЬ.

Операторы сдвига, ">>" и "<<" сдвигают все биты в значении вправо или влево. Общий формат использования оператора сдвига вправо выглядит так.

значение >> число\_бит

А оператор сдвига влево используется так.

значение << число\_бит

Операторы сдвига предназначены для сдвига битов в рамках целочисленного значения. Здесь элемент число бит указывает, на сколько позиций должно быть сдвинуто значение. При каждом сдвиге влево все биты, составляющее значение, сдвигаются влево на одну позицию, а в младший разряд записывается нуль. При каждом сдвиге вправо все биты сдвигаются, соответственно, вправо. Если сдвигу вправо подвергается значение без знака, в старший разряд записывается нуль. Если же сдвигу вправо подвергается значение со знаком, значение знакового разряда сохраняется. Как вы помните, отрицательные целые числа представляются установкой старшего разряда числа равным единице. Таким образом, если сдвигаемое значение отрицательно, при каждом сдвиге вправо в старший разряд записывается единица, а если положительно — нуль. Не забывайте, сдвиг, выполняемый операторами сдвига, не является циклическим, т.е. при сдвиге как вправо, так и влево крайние биты теряются, и содержимое потерянного бита узнать невозможно.

Операторы сдвига работают только со значениями целочисленных типов, например, символами, целыми числами и длинными целыми числами. Они не применимы к значениям с плавающей точкой.

Побитовые операции сдвига могут оказаться весьма полезными для декодирования входной информации, получаемой от внешних устройств (например, цифроаналоговых преобразователей), и обработки информация о состоянии устройств. Поразрядные операторы сдвига можно также использовать для выполнения ускоренных операций умножения и деления целых чисел. С помощью сдвига влево можно эффективно умножать на два, сдвиг вправо позволяет не менее эффективно делить на два.

Следующая программа наглядно иллюстрирует результат использования операторов сдвига.

// Демонстрация выполнения поразрядного сдвига.

#include <iostream>

using namespace std;

void disp\_binary(unsigned u);

int main()

{

int i = 1, t;

for (t = 0; t < 8; t++) {

disp\_binary(i);

i = i << 1;

} c

out << "\n";

for (t = 0; t < 8; t++) {

i = i >> 1;

disp\_binary(i);

} r

eturn 0;

} /

/ Отображение битов, составляющих байт.

void disp\_binary(unsigned u)

{

register int t;

for (t = 128; t > 0; t = t / 2)

if (u & t) cout << "1";

else cout << "0 ";

cout << "\n";

}

Результаты выполнения этой программы таковы.

0 0 0 0 0 0 0 1

0 0 0 0 0 0 1 0

0 0 0 0 0 1 0 0

0 0 0 0 1 0 0 0

0 0 0 1 0 0 0 0

0 0 1 0 0 0 0 0

0 1 0 0 0 0 0 0

1 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 1 0 0 0 0 0

0 0 0 1 0 0 0 0

0 0 0 0 1 0 0 0

0 0 0 0 0 1 0 0

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

Для удобства получения пользовательского ввода была написана шаблонная функция с следующей сингнатурой:

template <typename T>

T get\_value(std::string&& message, bool(\*compare)(const T&) = compare\_default)

Функция принимает сообщение, которое будет выведено пользователю в консоль с просьбой ввести необходимые значение, а также указатель на функцию, которая будет выполнять роль валидатора значений пользовательского ввода. По умолчанию передается «стандартный» валидатор, который всегда возвращает True. Таким образом, до тех пор, пока пользователь не введет валидные данные, он будет получать приглашение их ввести. Применяя данную функцию в основной программе, мы можем гарантировать, что данные, которые эта программа будет обрабатывать обязательно будут корректными, и нам не нужно будет загромождать наш код дополнительными проверками.

2.1 Задание 1

Написать программу вычисления площади ромба.

Окно выполнения программы изображено на рис. 1.

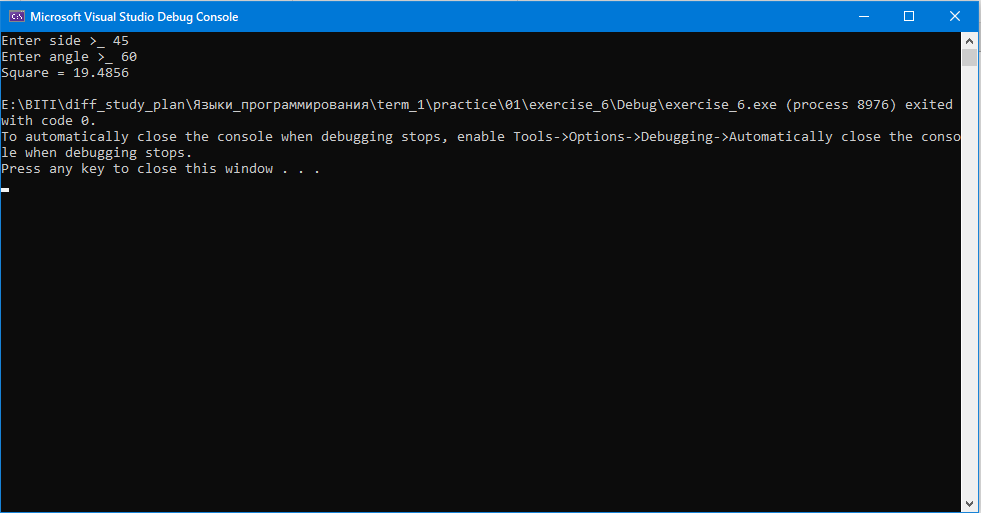


Рисунок 1 - Окно выполнение программы задания 1

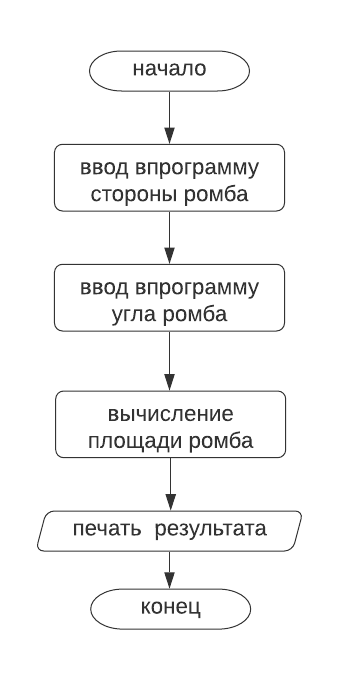
Блок схема алгоритма изображена на рис.2.

Рисунок 2 - Блок схема к заданию 1

Исходный код программы:

// created by a.nikolsky 27.11.2020

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <math.h>

#include <string>

#include <sstream>

double rhomb\_square(const double& side, const double& angle) {

return 0.5 \* side \* sin(angle \* M\_PI / 180);

}

template <typename T>

T get\_value(std::string&& message, bool(\*compare)(const T&) = compare\_default) {

T result;

std::string buff = "";

std::stringstream ss;

do {

ss.clear();

buff = "";

std::cout << message;

std::getline(std::cin, buff);

ss.str(buff);

ss >> result;

} while (ss.fail() || !compare(result));

return result;

};

template <typename T>

bool compare\_default(const T& value = 0) {

return true;

}

bool compare\_angle(const double& angle) {

if (angle >= 180 || angle <= 0) {

return false;

}

else{

return true;

}

}

bool compare\_side(const double& side) {

if (side <= 0) {

return false;

}

else {

return true;

}

}

int main()

{

double side = get\_value<double>("Enter side >\_ ", compare\_side);

double angle = get\_value<double>("Enter angle >\_ ", compare\_angle);

std::cout << "Square = " << rhomb\_square(side, angle) << std::endl;

}

2.2 Задание 2

По количеству музыкантов определить название группы (соло, дуэт, трио, квартет и т.д.)

Окно выполнения программы изображено на рис. 3

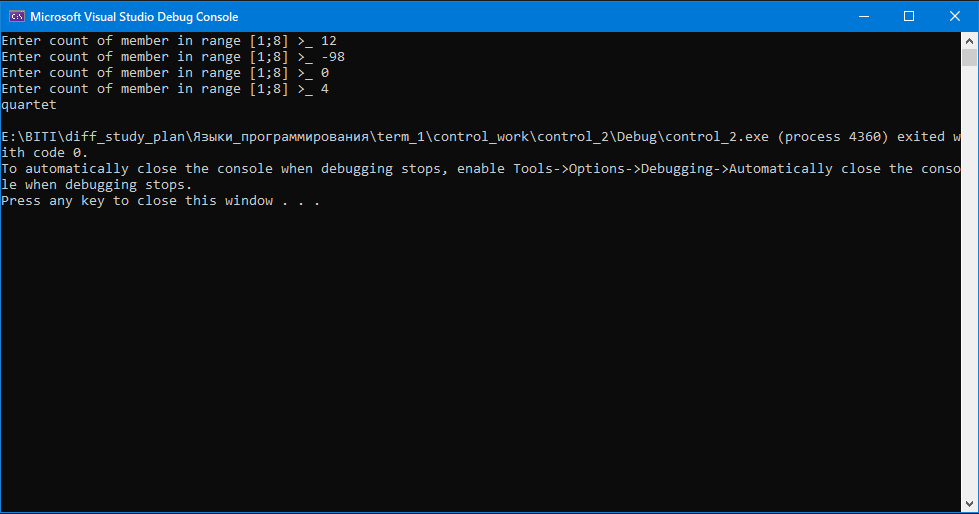


Рисунок 3 - Окно выполнения программы задания 2

Блок схема алгоритма изображена на рисунке 4.

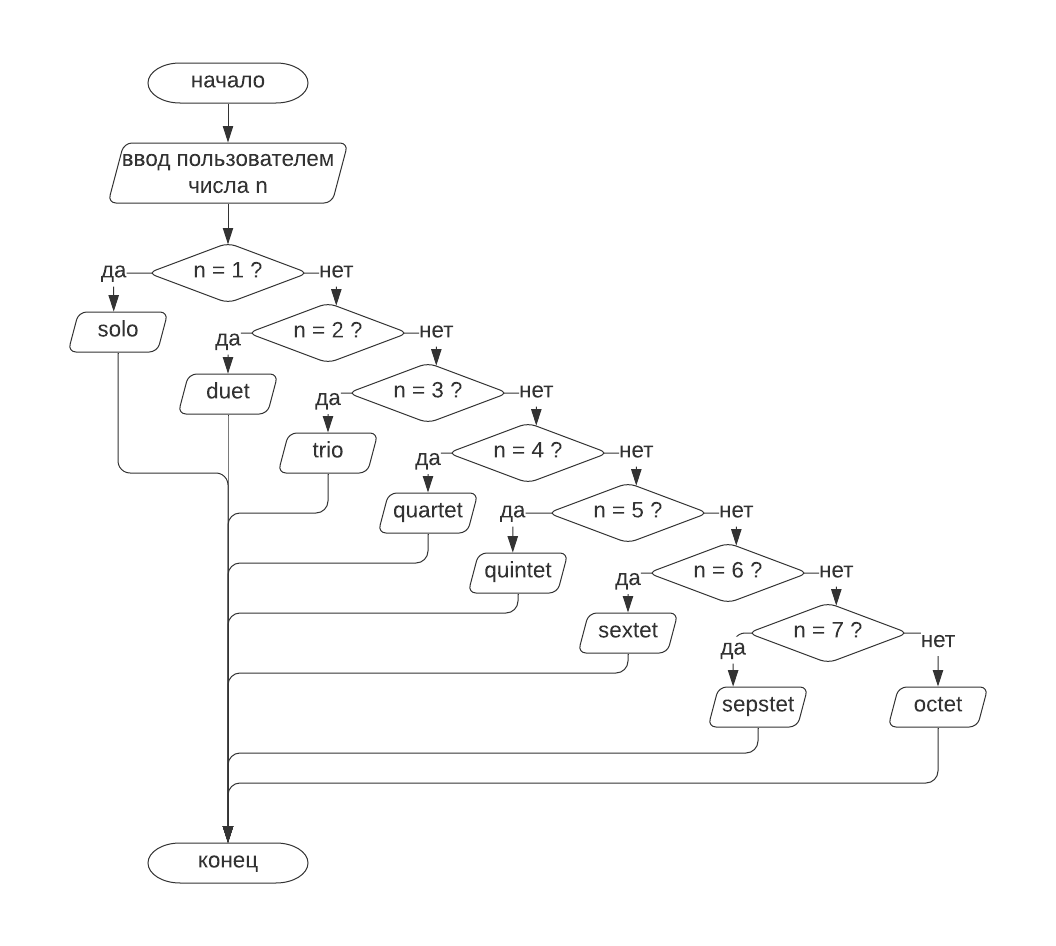


Рисунок 4 - блок cхема алгоритма для задания 2

Исходный код программы:

// created by a.nikolsky 27.11.2020

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <math.h>

#include <string>

#include <sstream>

template <typename T>

T get\_value(std::string&& message, bool(\*compare)(const T&) = compare\_default) {

T result;

std::string buff = "";

std::stringstream ss;

do {

ss.clear();

buff = "";

std::cout << message;

std::getline(std::cin, buff);

ss.str(buff);

ss >> result;

} while (ss.fail() || !compare(result));

return result;

};

template <typename T>

bool compare\_default(const T& value = 0) {

return true;

}

bool compare\_members\_count(const int& count) {

if (count <= 0 || count > 8) {

return false;

}

else {

return true;

}

}

int main()

{

int count = get\_value<int>("Enter count of member in range [1;8] >\_ ", compare\_members\_count);

switch (count) {

case 1:

std::cout << "solo\n";

break;

case 2:

std::cout << "duo\n";

break;

case 3:

std::cout << "trio\n";

break;

case 4:

std::cout << "quartet\n";

break;

case 5:

std::cout << "quintet\n";

break;

case 6:

std::cout << "sextet\n";

break;

case 7:

std::cout << "sepstet\n";

break;

case 8:

std::cout << "octet\n";

break;

}

}

2.2 Задание 2

Найти min из введенных чисел. Количество чисел задается пользователем.

Окно выполнения программы изображено на рис. 5

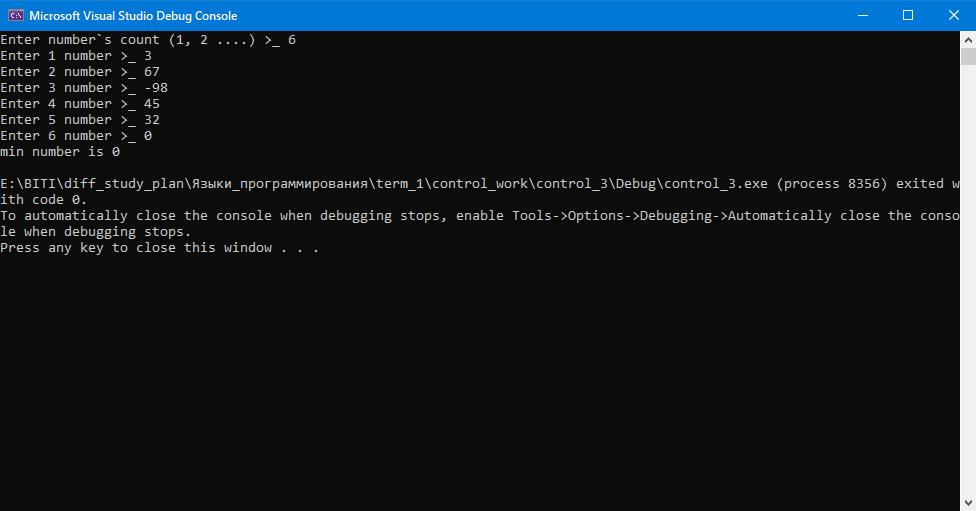


Рисунок 5 - Окно выполнения программы задания 3

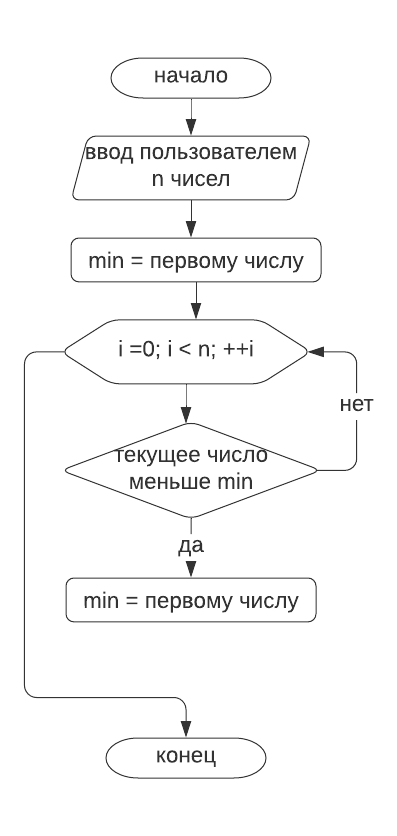
Блок схема алгоритма для задания 3 изображена на рис. 6.

Рисунок 6 - Блок схема алгоритма для задания 3

Исходный код программы:

// created by a.nikolsky 27.11.2020

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <math.h>

#include <string>

#include <sstream>

#include <vector>

template <typename T>

T get\_value(std::string&& message, bool(\*compare)(const T&) = compare\_default) {

T result;

std::string buff = "";

std::stringstream ss;

do {

ss.clear();

buff = "";

std::cout << message;

std::getline(std::cin, buff);

ss.str(buff);

ss >> result;

} while (ss.fail() || !compare(result));

return result;

};

template <typename T>

bool compare\_default(const T& value = 0) {

return true;

}

bool compare\_positive(const int& count) {

if (count <= 0) return false;

else return true;

}

int main() {

int count = get\_value<int>("Enter number`s count (1, 2 ....) >\_ ", compare\_positive);

std::vector<double> v;

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

v.push\_back(get\_value<double>("Enter " + std::to\_string(i + 1) + " number >\_ "));

}

double min = v.front();

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

if (v[i] < i) {

min = v[i];

}

}

std::cout << "min number is " << min << std::endl;

}

ВЫВОД

В данной контрольной работе подробно описали действие битовых (поразрядных) операторов в языке СИ, а также способы их применения. Написали несколько функций, которые демонстрируют применение таких операторов, а также их эффективность.

В практической части научились применять на практике навыки, полученные в ходе изучения языка программирования. В первом задании была написана простая программа, вычисляющая площадь ромба. Данная программа демонстрирует навыки работы с подсистемой ввода / вывода языка СИ (С++), основы функционального программирования, а также работу с математическими операторами.

Во втором задании научились применять управляющие конструкции языка. При решении задания было принято решение использовать конструкцию switch – case, так как все кейсы работы ветвящегося алгоритма соответствуют целым числам. Это позволило сделать код более компактным и читабельным.

В третьем задании продемонстрировали навыки использования циклов на примере цикла for. Также составили простейший однопроходный алгоритм для поиска минимального числа на заданном множестве чисел.

Цель данной работы достигнута.