|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 1** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Поразрядные операции и их применение»** | |
|  | |
| Выполнил студент группы ИКБО-11-22 | Гришин А. В. |
| Принял преподаватель | Скворцова Л. А. |

|  |  |  |
| --- | --- | --- |
| Самостоятельная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2023

Содержание

[1. Цель работы 3](#_30j0zll)

[2. Задача №1 3](#_hm0l5kud5x3d)

[2.1 Постановка задачи 3](#_wx94uo2n50lp)

[2.2 Реализация решения 3](#_2et92p0)

[2.2.1 Упражнение №1 3](#_p9s2jtqvmu90)

[2.2.1.1 Тестирование Упражнение №1 4](#_sa8kh12tq8w3)

[2.2.2 Упражнение №2 5](#_3dy6vkm)

[2.2.2.1 Тестирование Упражнение №2 6](#_bauzcbm8geft)

[2.2.3 Упражнение №3 6](#_1t3h5sf)

[2.2.3.1 Тестирование Упражнение №3 7](#_e2huxtjdxsvn)

[2.2.4 Упражнение №4 7](#_i21lxtnjr220)

[2.2.4.1 Тестирование Упражнение №4 7](#_91hw4o3byc7y)

[2.2.5 Упражнение №5 8](#_4d34og8)

[2.2.5.1 Тестирование Упражнение №5 8](#_403tnnpofh3a)

[3. Задача №2 9](#_3rdcrjn)

[3.1 Постановка задачи 9](#_ftof4i7sphdg)

[3.2 Алгоритм решения 9](#_lnxbz9)

[3.3 Описание структуры, используемой в решении, для представления данных в оперативной памяти. 11](#_35nkun2)

[3.4 Тесты 12](#_1ksv4uv)

[4. Вывод 13](#_44sinio)

# **Цель работы**

Получение навыков применения поразрядных операций в алгоритмах.

# **Задача №1**

## **2.1 Постановка задачи**

Разработать программу, которая продемонстрирует выполнение упражнений варианта. Результаты выполнения упражнения выводить на монитор.

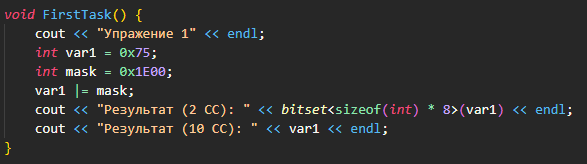
Вариант №15. Условие задания:

|  |  |  |
| --- | --- | --- |
| Упражнение 1 | Номер бита | С 9-ого бита четыре слева |
| Упражнение 2 | Номер бита | 17-ый, 15-ый, 1-ый |
| Упражнение 3 | Множитель | 1024 |
| Упражнение 4 | Делитель | 1024 |
| Упражнение 5 | Задание для выражения | Обнулить n-ый бит, используя маску (вар. 1) |

## **2.2 Реализация решения**

### **2.2.1 Упражнение №1**

Определить переменную целого типа, присвоить ей значение, используя константу в шестнадцатеричной системе счисления. Разработать оператор присваивания и его выражение, которое установит заданные в задании биты исходного значения переменной в значение 1, используя соответствующую маску и поразрядную операцию.



**Рисунок 1. Код решения Упражнение №1**

Этот код демонстрирует алгоритм для установки определенных битов в целочисленной переменной value с использованием маски и поразрядной операции "ИЛИ".

**int x = 0x75**; - Объявляется переменная x и ей присваивается значение 0x75, которое в шестнадцатеричной системе равно двоичному значению 01110101.

**int mask = 0x1E00;** - Объявляется переменная mask и ей присваивается значение 0x1E00, что представляет собой шестнадцатеричное число, равное двоичному значению 1111000000000.

**var1 |= mask;** - Выполняется поразрядная операция "ИЛИ" между x и mask.

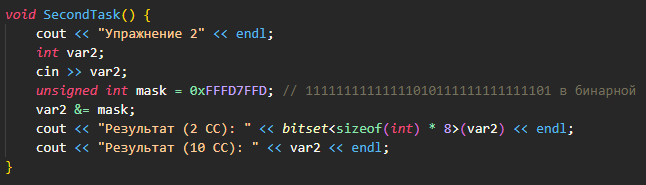
### **2.2.1.1 Тестирование Упражнение №1**



**Рисунок 2. Результаты тестирования Упражнение №1**

### **2.2.2 Упражнение №2**

Определить переменную целого типа. Разработать оператор присваивания и его выражение, которое обнуляет заданные в задании биты исходного значения переменной, используя соответствующую маску и поразрядную операцию. Значение в переменную вводится с клавиатуры.



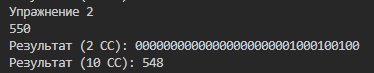
**Рисунок 3. Код решения Упражнение №2**

**int var2;** - Объявляется целочисленная переменная myVar, в которую будет введено значение с клавиатуры.

**unsigned short int mask = 0xFFFD7FFD;** - Создается маска mask, которая в шестнадцатеричной системе счисления соответствует бинарной последовательности 11111111111111010111111111111101. Эта маска обнуляет биты 17-й, 15-й и 1-й (номера битов считаются с 0).

**var2 &= mask;** - Применяется поразрядная операция И между var2 и инвертированной маской ~mask. Эта операция обнуляет в var2 только те биты, которые были установлены в 1 в маске mask, то есть 1-й, 15-й и 17-й биты.

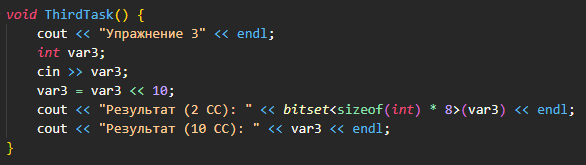
### **2.2.2.1 Тестирование Упражнение №2**



**Рисунок 4. Результаты тестирования Упражнение №2**

### **2.2.3 Упражнение №3**

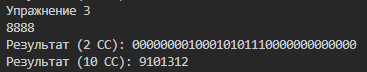
Определить переменную целого типа. Разработать оператор присваивания и выражение, которое умножает и делит значение переменной на число, указанное в третьем столбце варианта, используя соответствующую поразрядную операцию. Изменяемое число вводится с клавиатуры.



**Рисунок 5. Решение Упражнение №3**

**var3 = var3 << 10;** выполняет сдвиг влево на 10 битов, что эквивалентно умножению на 2^10, то есть на 1024. Этот сдвиг приводит к увеличению значения var3 в 2^10 раз, что фактически делает умножение на 1024.

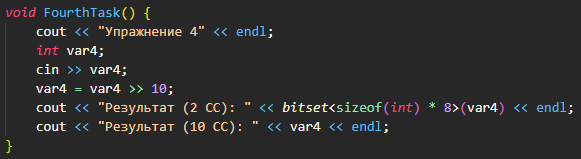
### **2.2.3.1 Тестирование Упражнение №3**



**Рисунок 6. Результаты тестирования Упражнение №3**

### **2.2.4 Упражнение №4**

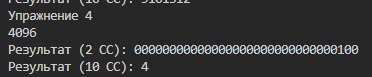
Определить переменную целого типа. Разработать оператор присваивания и выражение, которое делит значение переменной на число, указанное в третьем столбце варианта, используя соответствующую поразрядную операцию. Изменяемое число вводится с клавиатуры.



**Рисунок 7. Решение Упражнение №4**

**var3 = var3 >> 10;** выполняет сдвиг вправо на 10 битов, что эквивалентно делению на 2^10, то есть на 1024. Этот сдвиг приводит к уменьшению значения var3 в 2^10 раз, что фактически делает деление на 1024. Каждый бит, который "выталкивается" за пределы правого конца, теряется.

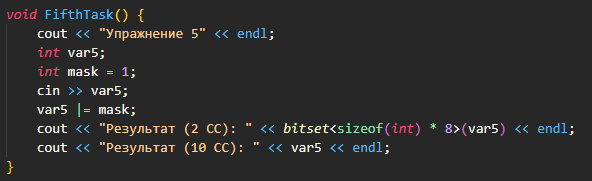
### **2.2.4.1 Тестирование Упражнение №4**



**Рисунок 8. Результаты тестирования Упражнение №4**

### **2.2.5 Упражнение №5**

Определить переменную целого типа. Разработать оператор присваивания и выражение, в котором используются только поразрядные операции. В выражении используется маска – переменная. Маска инициализирована единицей в старшем разряде. Изменяемое число вводится с клавиатуры.

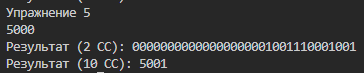


**Рисунок 9. Решение Упражнение №5**

**int mask = 1;** - Инициализируется переменная mask значением 1. Это маска, которая имеет только младший бит установленный в 1.

**var5 = var5 | mask;** - Выполняется операция поразрядного ИЛИ между var5 и mask. Эта операция устанавливает младший бит переменной var5 в 1, если он был 0, оставляя все остальные биты неизменными.

### **2.2.5.1 Тестирование Упражнение №5**



**Рисунок 10. Результаты тестирования Упражнение №5**

# **3. Задача №2**

## **3.1 Постановка задачи**

Реализовать задачу по сортировке данных файла, используя для представления данных файла (107 семизначных чисел) в памяти, массив битов.

**Дано**: Файл, содержащий не более n (n=107) целых положительных чисел, каждое из которых семизначное число, т.е. принадлежит диапазону [1000000..9999999] и среди них нет повторяющихся.

**Результат**: упорядоченный по возрастанию список чисел, сохраненных в файле.

**Ограничения**:

- Доступной оперативной памяти 1МБ.

- Дисковая память неограниченна.

- Время выполнения программы 10 секунд.

## **3.2 Алгоритм решения**

Для решения данной задачи будет использоваться бит массив. При использовании битового массива для представления сортируемых чисел, программу можно представить как последовательность из трех задач:

1. Инициализация бит массива нулевыми значениями.

2. Считывание целых чисел из файла и установка в 1 соответствующих бит.

3. Формирование упорядоченного выходного файла путем последовательной проверки бит массива и вывод в файл номеров тех бит, которые установлены в 1.

*Листинг 1 – Исходный код программы для Задания 2*

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <bitset>  #include <ctime>  #include <cstdlib>  #include <chrono>  using namespace std;  void Task2()  {  const long int size = 10000000; // Размер std::bitset (10^7 бит)  bitset<size>\* numbers = new bitset<size>; // Используем std::bitset для хранения битов  long int input;  ofstream outfile("sortedList.txt");  cout << "Введите по очереди семизначные числа (0 для завершения):\n";  while (true) {  cin >> input;  if (input == 0) {  break;  }  if (input >= size / 10 && input <= 9999999) {  numbers->set(input - 1, 1);  }  else {  cout << "Только семизначные числа!\n";  }  }  char spam;  cout << "Нужно ли дополнительно ввести 1000 семизначных чисел? (y/n): ";  cin >> spam;  if (spam == 'y' || spam == 'Y') {  srand(static\_cast<unsigned int>(time(nullptr)));  for (long int i = 999; i > 1; i--) {  long int random\_input = ((size / 10) \* (1 + rand() % (8))) + ((rand() % (i + 1)) \* (990));  numbers->set(random\_input, 1);  }  }  auto start\_time = chrono::high\_resolution\_clock::now();  for (long int i = 0; i < size; i++) {  if (numbers->test(i)) {  outfile << i + 1 << endl;  }  }  auto end\_time = chrono::high\_resolution\_clock::now();  auto exec\_time = chrono::duration\_cast<chrono::milliseconds>(end\_time - start\_time);  outfile.close();  cout << "Результат сортировки был записан в файл.\n";  cout << "На выполнение затрачено " << exec\_time.count() << " миллисекунд." << endl;  }  int main() {  setlocale(LC\_ALL, "RUS");  Task2();  return 0;  } |

## **3.3 Описание структуры, используемой в решении, для представления данных в оперативной памяти.**

В данной работе используется структура std::bitset - это структура данных в C++, предназначенная для представления фиксированного числа битов в памяти. Она обеспечивает эффективное и удобное управление битами, где каждый бит может быть установлен в 0 или 1. Важные характеристики std::bitset:

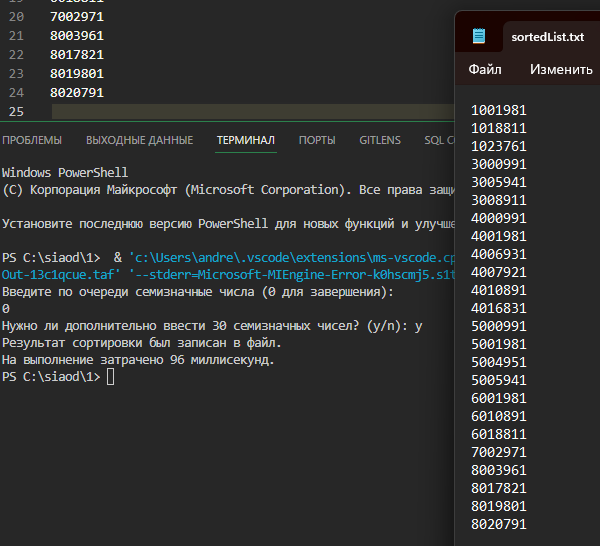
Фиксированный размер: Размер std::bitset фиксирован во время компиляции и не может изменяться во время выполнения программы. Например, std::bitset<8> будет иметь 8 битов.

Поразрядные операции: std::bitset поддерживает поразрядные операции, такие как "И", "ИЛИ", "Исключающее ИЛИ" и др., что позволяет легко выполнять битовые операции.

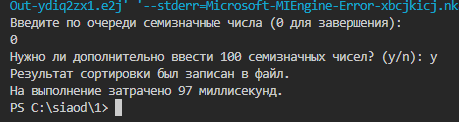
Легкий доступ: Каждый бит в std::bitset имеет индекс, который можно использовать для доступа к нему. Например, myBitset[3] обратится к третьему биту в myBitset.

Удобное использование: std::bitset часто используется для флагов, битовых масок и других задач, связанных с битами.

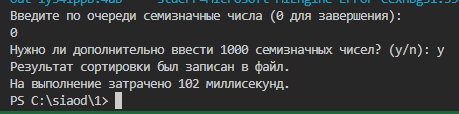
## **3.4 Тесты**



**Рисунок 6. Результат выполнения на 30 значениях**



**Рисунок 7. Результат выполнения на 100 значениях**



**Рисунок 8. Результат выполнения на 1000 значениях**

# **4. Вывод**

В результате выполнения работы я:

1. Освоил алгоритмы работы с поразрядными операциями и их реализацию на языке программирования C++

2. Научился программировать автоматическое тестирование простых программ