Минобрнауки России

Федеральное государственное автономное образовательное учреждение

высшего образования «Санкт-Петербургский государственный

электротехнический университет «ЛЭТИ»

им В. И. Ульянова (Ленина)»

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

**Зачётная работа № 2**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Множество как объект»**

Выполнили студенты группы 4315:

Данилова С.В.

Коновалова К.Л.

Принял: старший преподаватель Манирагена Валенс

Санкт-Петербург

2025

Оглавление

[Цель работы 2](#_Toc705406750)

[Задание 3](#_Toc547824656)

[Результаты эксперимента с четырьмя структурами данных на основе классов 3](#_Toc370898219)

[Результат эксперимента с отслеживанием вызовов функций-членов 5](#_Toc1098821373)

[Выводы 6](#_Toc978469696)

[Список используемых источников 6](#_Toc1694393401)

[Приложение. Текст программы 6](#_Toc1088690411)

# Цель работы

Исследование эффекта от использования классов.

# Задание

Строчные русские буквы: множество, содержащие буквы, общие для множеств A, B, C и не встречающиеся в D.

# Результаты эксперимента с четырьмя структурами данных на основе классов

Ниже представлены примеры работы программы (рис. 1–4).

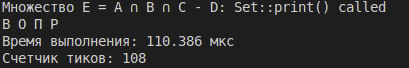


Рис.1 Выполнение программы с множеством символов

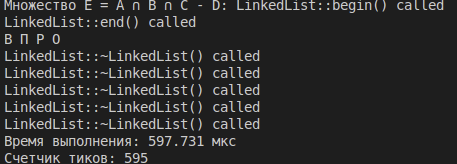


Рис.2 Выполнение программы со списками

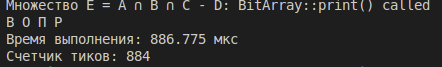


Рис.3 Выполнение программы с массивом битов

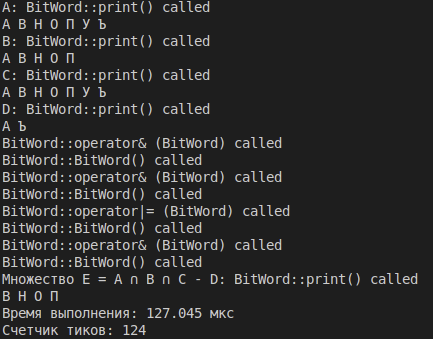


Рис.4 Выполнение программы с машинным словом

Результат измерения времени обработки для каждого из способов

Таблица 2. Результаты измерения времени обработки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Мощность множеств | t, c | | | |
| Массив символов | Список | Унивёрсум | Машинное слово |
| 2 | 8,00E-05 | 9,00E-05 | 7.7e-05 | 7.5e-05 |
| 4 | 8.9e-05 | 1.1e-04 | 7.6e-05 | 7.7e-05 |
| 6 | 1.1e-04 | 1.2e-04 | 7.8e-05 | 8,00E-05 |
| 8 | 1.1e-04 | 1.2e-04 | 8.1e-05 | 8,00E-05 |
| 10 | 1.3e-04 | 1.2e-04 | 8.1e-05 | 7.4e-05 |
| 12 | 1.4e-04 | 1.3e-04 | 8,00E-05 | 7.7e-05 |
| 14 | 1.7e-04 | 1.5e-04 | 7.6e-05 | 7.6e-05 |
| 16 | 1.7e-04 | 1.7e-04 | 7.6e-05 | 7.8e-05 |
| 18 | 1.8e-04 | 1.9e-04 | 7.8e-05 | 7.9e-05 |
| 20 | 2,00E-04 | 2.3e-04 | 7.7e-05 | 7.8e-05 |
| 22 | 2.2e-04 | 2.4e-04 | 8,00E-05 | 8.2e-05 |
| 24 | 2.3e-04 | 2.6e-04 | 8,00E-05 | 8.1e-05 |
| 26 | 2.4e-04 | 2.6e-04 | 7.6e-05 | 7.9e-05 |
| 28 | 2.7e-04 | 3,00E-04 | 7.9e-05 | 7.5e-05 |
| 30 | 2.9e-04 | 3.2e-04 | 8.3e-05 | 7.9e-05 |
| 32 | 3.4e-04 | 3.3e-04 | 8.2e-05 | 7.8e-05 |

Для получения более точных данных измеряемый процесс обработки множеств для каждого способа представления повторялся 10000 раз. При представлении множеств в виде массива символов и списка заметно, что время обработки множеств с увеличением мощности также увеличивается. Для универсума и машинного слова время обработки практически неизменно, т.е. не зависит от размера входа.

В сравнении с бесклассовой реализацией (см. Отчёт “Зачётная работа №1. Множество в памяти ЭВМ”), скорость выполнения алгоритма практически не изменяется вне зависимости от используемого типа данных. Это исходит от того, что любой класс - абстракция, занимающая минимальную память.

# Результат эксперимента с отслеживанием вызовов функций-членов

Множество: функция вставки вызывается для каждого из четырёх множеств n-ое количество раз, где n - длина множества. После этого происходит проверка на факт правильной заполненности после сохранения адресов начала и конца (для перемещения). Функция вызывается единожды для каждого из множеств. После поиска совпадений в соответствии с логикой задания set E заполняется вставкой (поиск происходит для каждого будущего элемента отдельно). Работа программы заканчивается вызовом деструктора, удаляющим множества по адресу. Использованные функции: insert(), begin(), end(), contains().

Список: в первую очередь, вызывается конструктор класса LinkedList(). При каждом вызове создаётся список (всего пять). Сразу после ввода все элементы записываются в список через push\_back. Получив адреса первого и последнего элемента каждого списка (т. е. подтвердив существование и корректную последовательность), программа схожим с множественной реализацией образом находит элементы и заполняет итоговый лист. Деструктор удаляет листы по адресам.

Унивёрсум (массив битов): в отличие от предыдущих представлений данных, заполнение происходит с помощью непостоянного оператора. Это вызвано особенностью расположения в памяти. Позиция бита в массиве определяется ASCII-кодом символа. Однако он всё ещё заполняется поэлементно, поэтому оператор не может возвращать постоянное значение. Деструктор отсутствует.

Машинное слово: после создания слов и получения данных, последние копируются через оператор. Результат инвертируется через маску до значения 33 бит (при изначальных 64) для оптимизации пространства памяти.

# Выводы

В ходе лабораторной работы было выяснено, что классы практически не имеют влияние на время выполнения программы. Это несомненное преимущество перед “голой” реализацией через структуры, как это было в прошлой работе, так как помимо экономии, объектное представление обеспечивает безопасность и удобство (в том числе и постоянном использовании одного и того же шаблона). Использование конструкторов упроща

Список используемых источников

1. Колинько П. Г. Пользовательские структуры данных: Методические указания по дисциплине «Алгоритмы и структуры данных, часть 1». –– СПб.: СПбГЭТУ «ЛЭТИ», 2025. — 32 с. (вып.2509).

2. Лафоре Р. Объектно-ориентированное программирование в С++. Классика Computer Science. 4-е изд. — СПб.: Питер, 2015. — 217 с.

3. Хагерти Р. Дискретная математика для программистов. Изд. 2‑е, испр. — М.: Техносфера, 2012. — 44 с.

4. Алгоритмы. http://old.math.nsc.ru/LBRT/k5/OR-MMF/dasgupta\_2014.pdf

# Приложение. Текст программы

#ifndef SET\_H #define SET\_H

#include #include

template class Set { private: struct Node { T data; Node\* left; Node\* right;

Node(const T& value) : data(value), left(nullptr), right(nullptr) {}  
};  
  
Node\* root;  
  
class Iterator {  
private:  
 static const int MAX\_HEIGHT = 64;  
 Node\* stack[MAX\_HEIGHT];  
 int top;  
  
 void pushLeft(Node\* node) {  
 while (node && top < MAX\_HEIGHT) {  
 stack[top++] = node;  
 node = node->left;  
 }  
 }  
  
public:  
 Iterator(Node\* root = nullptr) : top(0) {  
 pushLeft(root);  
 }  
  
 T& operator\*() {  
 return stack[top - 1]->data;  
 }  
  
 const T& operator\*() const {  
 return stack[top - 1]->data;  
 }  
  
 Iterator& operator++() {  
 if (top <= 0) return \*this;  
 Node\* current = stack[--top];  
 pushLeft(current->right);  
 return \*this;  
 }  
  
 bool operator!=(const Iterator& other) const {  
 if (top == 0 && other.top == 0) return false;  
 if (top == 0 || other.top == 0) return true;  
 return stack[top - 1] != other.stack[other.top - 1];  
 }  
};  
  
// вспомогательные функции   
Node\* insert(Node\* node, const T& value) {  
 if (!node) return new Node(value);  
 if (value < node->data) {  
 node->left = insert(node->left, value);  
 } else if (value > node->data) {  
 node->right = insert(node->right, value);  
 }  
 return node;  
}  
  
bool contains(Node\* node, const T& value) const {  
 if (!node) return false;  
 if (value == node->data) return true;  
 if (value < node->data) return contains(node->left, value);  
 return contains(node->right, value);  
}  
  
Node\* findMin(Node\* node) const {  
 while (node && node->left) node = node->left;  
 return node;  
}  
  
Node\* remove(Node\* node, const T& value) {  
 if (!node) return node;  
 if (value < node->data) {  
 node->left = remove(node->left, value);  
 } else if (value > node->data) {  
 node->right = remove(node->right, value);  
 } else {  
 if (!node->left) {  
 Node\* temp = node->right;  
 delete node;  
 return temp;  
 } else if (!node->right) {  
 Node\* temp = node->left;  
 delete node;  
 return temp;  
 }  
 Node\* temp = findMin(node->right);  
 node->data = temp->data;  
 node->right = remove(node->right, temp->data);  
 }  
 return node;  
}  
  
void inorder(Node\* node) const {  
 if (node) {  
 inorder(node->left);  
 std::cout << node->data << " ";  
 inorder(node->right);  
 }  
}  
  
void destroy(Node\* node) {  
 if (node) {  
 destroy(node->left);  
 destroy(node->right);  
 delete node;  
 }  
}

public: // конструктор Set() : root(nullptr) { std::cout << "Set::Set() called\n"; }

// конструктор с начальной инициалиацией  
Set(std::initializer\_list<T> init) : root(nullptr) {  
 std::cout << "Set::Set(initializer\_list) called\n";  
 for (const T& value : init) {  
 insert(value);  
 }  
}  
  
// деструктор  
~Set() {  
 std::cout << "Set::~Set() called\n";  
 destroy(root);  
}  
  
Iterator begin() const {  
 std::cout << "Set::begin() called\n";  
 return Iterator(root);  
}  
  
Iterator end() const {  
 std::cout << "Set::end() called\n";  
 return Iterator(nullptr);  
}  
  
// методы интерфейса  
void insert(const T& value) {  
 std::cout << "Set::insert() called\n";  
 root = insert(root, value);  
}  
  
void remove(const T& value) {  
 std::cout << "Set::remove() called\n";  
 root = remove(root, value);  
}  
  
bool contains(const T& value) const {  
 std::cout << "Set::contains() called\n";  
 return contains(root, value);  
}  
  
void print() const {  
 std::cout << "Set::print() called\n";  
 inorder(root);  
 std::cout << std::endl;  
}  
  
bool empty() const {  
 std::cout << "Set::empty() called\n";  
 return root == nullptr;  
}

};

#endif

#ifndef LIST\_H #define LIST\_H

#include #include // стандартный заголовок #include

template class LinkedList { private: struct Node { T data; Node\* next;

Node(const T& value) : data(value), next(nullptr) {}  
};  
  
Node\* head;  
size\_t size\_;  
  
class Iterator {  
private:  
 Node\* current;  
  
public:  
 Iterator(Node\* node) : current(node) {}  
  
 T& operator\*() {  
 return current->data;  
 }  
  
 const T& operator\*() const {  
 return current->data;  
 }  
  
 Iterator& operator++() {  
 if (current) current = current->next;  
 return \*this;  
 }  
  
 bool operator!=(const Iterator& other) const {  
 return current != other.current;  
 }  
};  
  
void clear() {  
 while (head) {  
 Node\* temp = head;  
 head = head->next;  
 delete temp;  
 }  
 size\_ = 0;  
}

public: // конструктор по умолчанию LinkedList() : head(nullptr), size\_(0) { std::cout << "LinkedList::LinkedList() called\n"; }

// конструктор с начальной иницализацией  
LinkedList(std::initializer\_list<T> init) : head(nullptr), size\_(0) {  
 std::cout << "LinkedList::LinkedList(initializer\_list) called\n";  
 for (const T& value : init) {  
 push\_back(value);  
 }  
}  
  
// деструктор  
~LinkedList() {  
 std::cout << "LinkedList::~LinkedList() called\n";  
 clear();  
}  
  
// копирование  
LinkedList(const LinkedList& other) : head(nullptr), size\_(0) {  
 std::cout << "LinkedList::LinkedList(const LinkedList&) called\n";  
 Node\* current = other.head;  
 while (current) {  
 push\_back(current->data);  
 current = current->next;  
 }  
}  
  
LinkedList& operator=(const LinkedList& other) {  
 std::cout << "LinkedList::operator= called\n";  
 if (this != &other) {  
 clear();  
 Node\* current = other.head;  
 while (current) {  
 push\_back(current->data);  
 current = current->next;  
 }  
 }  
 return \*this;  
}  
  
// основные методы  
Iterator begin() {  
 std::cout << "LinkedList::begin() called\n";  
 return Iterator(head);  
}  
  
Iterator begin() const {  
 std::cout << "LinkedList::begin() const called\n";  
 return Iterator(head);  
}  
  
Iterator end() {  
 std::cout << "LinkedList::end() called\n";  
 return Iterator(nullptr);  
}  
  
Iterator end() const {  
 std::cout << "LinkedList::end() const called\n";  
 return Iterator(nullptr);  
}  
  
void push\_front(const T& value) {  
 std::cout << "LinkedList::push\_front() called\n";  
 Node\* newNode = new Node(value);  
 newNode->next = head;  
 head = newNode;  
 ++size\_;  
}  
  
void push\_back(const T& value) {  
 Node\* newNode = new Node(value);  
 std::cout << "LinkedList::push\_back() called\n";  
 if (!head) {  
 head = newNode;  
 } else {  
 Node\* current = head;  
 while (current->next) {  
 current = current->next;  
 }  
 current->next = newNode;  
 }  
 ++size\_;  
}  
  
bool remove(const T& value) {  
 std::cout << "LinkedList::remove() called\n";  
 if (!head) return false;  
  
 if (head->data == value) {  
 Node\* temp = head;  
 head = head->next;  
 delete temp;  
 --size\_;  
 return true;  
 }  
  
 Node\* current = head;  
 while (current->next && current->next->data != value) {  
 current = current->next;  
 }  
  
 if (current->next) {  
 Node\* temp = current->next;  
 current->next = current->next->next;  
 delete temp;  
 --size\_;  
 return true;  
 }  
  
 return false;  
}  
  
bool contains(const T& value) const {  
 std::cout << "LinkedList::contains() called\n";  
 Node\* current = head;  
 while (current) {  
 if (current->data == value) return true;  
 current = current->next;  
 }  
 return false;  
}  
  
size\_t size() const {  
 std::cout << "LinkedList::size() called\n";  
 return size\_;  
}  
  
bool empty() const {  
 std::cout << "LinkedList::empty() called\n";  
 return head == nullptr;  
}  
  
void print() const {  
 std::cout << "LinkedList::print() called\n";  
 Node\* current = head;  
 while (current) {  
 std::cout << current->data << " ";  
 current = current->next;  
 }  
 std::cout << std::endl;  
}

};

#endif

#ifndef BITWORD\_H #define BITWORD\_H

#include #include #include

class BitWord { private: static const size\_t ALPHABET\_SIZE = 33; std::string alphabet = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"; unsigned long long data;

// вспомогательная функция: позволяет получить индекс буквы в алфавите  
size\_t getLetterIndex(char c) const {  
 for (size\_t i = 0; i < ALPHABET\_SIZE; ++i) {  
 if (c == alphabet[i]) {  
 return i;  
 }  
 }  
 throw std::invalid\_argument("Символ не является заглавной буквой русского алфавита");  
}

public: // конструктор по умолчанию — пустое множество BitWord() : data(0) { std::cout << "BitWord::BitWord() called\n"; }

void set(char letter) {  
 std::cout << "BitWord::set() called\n";  
 size\_t idx = getLetterIndex(letter);  
 if (idx >= 64) return;  
 data |= (1ULL << idx);  
}  
  
void reset(char letter) {  
 std::cout << "BitWord::reset() called\n";  
 size\_t idx = getLetterIndex(letter);  
 if (idx >= 64) return;  
 data &= ~(1ULL << idx);  
}  
  
bool test(char letter) const {  
 std::cout << "BitWord::reset() called\n";  
 size\_t idx = getLetterIndex(letter);  
 if (idx >= 64) return false;  
 return (data & (1ULL << idx)) != 0;  
}  
  
void clear() {  
 std::cout << "BitWord::clear() called\n";  
 data = 0;  
}  
  
bool empty() const {  
 std::cout << "BitWord::empty() called\n";  
 return data == 0;  
}  
  
size\_t count() const {  
 std::cout << "BitWord::count() called\n";  
 size\_t cnt = 0;  
 unsigned long long n = data;  
 while (n) {  
 cnt += n & 1;  
 n >>= 1;  
 }  
 return cnt;  
}  
  
void print() const {  
 std::cout << "BitWord::print() called\n";  
 for (size\_t i = 0; i < ALPHABET\_SIZE; ++i) {  
 if (data & (1ULL << i)) {  
 std::cout << alphabet.substr(i\*2, 2) << " ";  
 }  
 }  
 std::cout << std::endl;  
}  
  
unsigned long long getValue() const {  
 std::cout << "BitWord::getValue() called\n";  
 return data;  
}  
  
BitWord operator|(const BitWord& other) const {  
 std::cout << "BitWord::operator| (BitWord) called\n";  
 BitWord result;  
 result.data = this->data | other.data;  
 return result;  
}  
BitWord& operator|=(unsigned long long mask) {  
 std::cout << "BitWord::operator|= (unsigned long long) called\n";  
 data |= mask;  
 return \*this;  
}  
friend BitWord operator~(const BitWord& bw) {  
 std::cout << "BitWord::operator|= (BitWord) called\n";  
 const unsigned long long MASK = (1ULL << 33) - 1;  
 BitWord result;  
 result.data = (~bw.data) & MASK;  
 return result;  
}  
  
BitWord operator&(const BitWord& other) const {  
 std::cout << "BitWord::operator& (BitWord) called\n";  
 BitWord result;  
 result.data = this->data & other.data;  
 return result;  
}  
  
BitWord& operator|=(const BitWord& other) {  
 std::cout << "BitWord::operator&= called\n";  
 data |= other.data;  
 return \*this;  
}  
  
BitWord& operator&=(const BitWord& other) {  
 std::cout << "BitWord::operator~ called\n";  
 data &= other.data;  
 return \*this;  
}

};

#endif

#ifndef BITARRAY\_H #define BITARRAY\_H

#include #include #include

struct Bit { unsigned int bit : 1; // 0 или 1

Bit() : bit(0) {}  
Bit(unsigned int b) : bit(b & 1) {}

};

class BitArray { private: static const size\_t ALPHABET\_SIZE = 33; std::string alphabet = "АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯЁ"; Bit bits[ALPHABET\_SIZE];

// Прокси-класс для доступа к биту  
class BitProxy {  
 Bit& bit\_ref;  
public:  
 BitProxy(Bit& b) : bit\_ref(b) {}  
  
 // Присваивание: proxy = 0 или proxy = 1  
 BitProxy& operator=(unsigned int value) {  
 bit\_ref.bit = value & 1;  
 return \*this;  
 }  
  
 // Преобразование в bool (для чтения)  
 operator bool() const {  
 return bit\_ref.bit != 0;  
 }  
};  
  
// вспомогательная функция: позволяет получить индекс буквы в алфавите  
size\_t getLetterIndex(char c) const {  
 for (size\_t i = 0; i < ALPHABET\_SIZE; ++i) {  
 if (c == alphabet[i]) {  
 return i;  
 }  
 }  
 throw std::invalid\_argument("Символ не является буквой русского алфавита");  
}

public:

// конструктор по умолчанию — все биты 0  
BitArray() {  
 std::cout << "BitArray::BitArray() called\n";  
 for (size\_t i = 0; i < ALPHABET\_SIZE; ++i) {  
 bits[i].bit = 0;  
 }  
}  
  
BitProxy operator[](size\_t index) {  
 std::cout << "BitArray::operator[] (non-const) called\n";  
 return BitProxy(bits[index]);  
}  
  
bool operator[](size\_t index) const {  
 std::cout << "BitArray::operator[] (const) called\n";  
 if (index >= ALPHABET\_SIZE) return false;  
 return bits[index].bit != 0;  
}  
  
void set(char letter) {  
 std::cout << "BitArray::set() called\n";  
 size\_t idx = getLetterIndex(letter);  
 bits[idx].bit = 1;  
}  
  
void reset(char letter) {  
 std::cout << "BitArray::reset() called\n";  
 size\_t idx = getLetterIndex(letter);  
 bits[idx].bit = 0;  
}  
  
int get(int idx) {  
 std::cout << "BitArray::get() called\n";  
 return bits[idx].bit;  
}  
  
bool test(char letter) const {  
 std::cout << "BitArray::test() called\n";  
 size\_t idx = getLetterIndex(letter);  
 return bits[idx].bit == 1;  
}  
  
void clear() {  
 std::cout << "BitArray::clear() called\n";  
 for (size\_t i = 0; i < ALPHABET\_SIZE; ++i) {  
 bits[i].bit = 0;  
 }  
}  
  
void print() const {  
 std::cout << "BitArray::print() called\n";  
 for (size\_t i = 0; i < ALPHABET\_SIZE; ++i) {  
 if (bits[i].bit) {  
 std::cout << alphabet.substr(i\*2, 2) << " ";  
 }  
 }  
 std::cout << std::endl;  
}  
  
bool empty() const {  
 std::cout << "BitArray::empty() called\n";  
 for (size\_t i = 0; i < ALPHABET\_SIZE; ++i) {  
 if (bits[i].bit) return false;  
 }  
 return true;  
}  
  
size\_t count() const {  
 std::cout << "BitArray::count() called\n";  
 size\_t cnt = 0;  
 for (size\_t i = 0; i < ALPHABET\_SIZE; ++i) {  
 cnt += bits[i].bit;  
 }  
 return cnt;  
}

};

#endif

#include #include #include #include "./headers/bitarray.h"

#include

using namespace std;

int f(string s) { if (s[1]-'0'+160 == -15) return 32; // Ё return s[1]-'0'+160; }

void input(BitArray& b) { string line; getline(cin, line, '\n');

for (int i = 0; i < line.size(); i+=2) b[f(line.substr(i, 2))] = 1;

}

int main() { setlocale(LC\_ALL, "Russian");

BitArray bA;   
BitArray bB;   
BitArray bC;   
BitArray bD;   
BitArray bE;   
  
cout << "A: "; input(bA);  
cout << "B: "; input(bB);  
cout << "C: "; input(bC);  
cout << "D: "; input(bD);  
  
auto t1 = std::chrono::high\_resolution\_clock::now();  
auto tt1 = clock();  
  
cout << endl;  
  
cout << "A: "; bA.print();  
cout << "B: "; bB.print();  
cout << "C: "; bC.print();  
cout << "D: "; bD.print();  
  
// Находим A ∩ B ∩ C - D  
for (int i = 0; i < 33; i++) bE[i] = bA[i] && bB[i] && bC[i];   
  
for (int i = 0; i < 33; i++) bE[i] = not (bE[i] <= bD[i]);  
  
// // Вывод результата  
cout << "Множество E = A ∩ B ∩ C - D: "; bE.print();  
  
auto t2 = std::chrono::high\_resolution\_clock::now();  
auto tt2 = clock();  
  
cout << "Время выполнения: " << std::chrono::duration\_cast<std::chrono::duration<double, micro>>(t2-t1).count() << " мкс" << endl;  
cout << "Счетчик тиков: " << tt2-tt1 << endl;  
  
  
return 0;

}

#include #include #include #include "./headers/bitword.h"

#include

using namespace std;

int f(string s) { if (s[1]-'0'+160 == -15) return 32; // Ё return s[1]-'0'+160; }

void input(BitWord& w) { string line; getline(cin, line, '\n');

for (int i = 0; i < line.size(); i+=2) w |= (1LL << f(line.substr(i, 2))); }

int main() { setlocale(LC\_ALL, "Russian");

BitWord wA; BitWord wB; BitWord wC; BitWord wD; BitWord wE;

cout << "A: "; input(wA); cout << "B: "; input(wB); cout << "C: "; input(wC); cout << "D: "; input(wD);

auto t1 = std::chrono::high\_resolution\_clock::now(); auto tt1 = clock();

cout << endl;

cout << "A: "; wA.print(); cout << "B: "; wB.print(); cout << "C: "; wC.print(); cout << "D: "; wD.print();

// Находим A ∩ B ∩ C - D wE = wA & wB & wC;

wE = wE & (~wD);

// Вывод результата cout << "Множество E = A ∩ B ∩ C - D: "; wE.print();

auto t2 = std::chrono::high\_resolution\_clock::now();  
auto tt2 = clock();  
  
cout << "Время выполнения: " << std::chrono::duration\_cast<std::chrono::duration<double, micro>>(t2-t1).count() << " мкс" << endl;  
cout << "Счетчик тиков: " << tt2-tt1 << endl;

return 0; }

#include #include #include #include "./headers/new\_list.h"

#include

using namespace std;

void print(LinkedList &l) { for (auto ch : l) cout << ch << " "; cout << endl; }

void input(LinkedList &l) { string line; getline(cin, line, '\n');

for (int i = 0; i < line.size(); ) {  
 unsigned char ch = line[i];  
 size\_t len;  
  
 // Определяем длину UTF-8 символа по первому байту  
 if ((ch & 0x80) == 0) len = 1; // 0xxxxxxx  
 else if ((ch & 0xE0) == 0xC0) len = 2; // 110xxxxx  
 else if ((ch & 0xF0) == 0xE0) len = 3; // 1110xxxx  
 else if ((ch & 0xF8) == 0xF0) len = 4; // 11110xxx  
 else len = 1; // fallback (некорректный UTF-8)  
  
 // Ограничиваем длину размером строки  
 if (i + len > line.size()) len = 1;  
  
 l.push\_back(line.substr(i, len));  
 i += len;  
}

}

int main() { setlocale(LC\_ALL, "Russian");

LinkedList<string>\* listA = new LinkedList<string>;  
LinkedList<string>\* listB = new LinkedList<string>;  
LinkedList<string>\* listC = new LinkedList<string>;  
LinkedList<string>\* listD = new LinkedList<string>;  
LinkedList<string>\* listE = new LinkedList<string>;  
  
cout << "A: "; input(\*listA);  
cout << "B: "; input(\*listB);  
cout << "C: "; input(\*listC);  
cout << "D: "; input(\*listD);

auto t1 = std::chrono::high\_resolution\_clock::now(); auto tt1 = clock();

cout << endl;  
  
cout << "A: "; print(\*listA);  
cout << "B: "; print(\*listB);  
cout << "C: "; print(\*listC);  
cout << "D: "; print(\*listD);  
  
// Находим A ∩ B ∩ C - D  
bool found = 0;  
for (auto chA : \*listA) {  
 for (auto chB : \*listB) {  
 for (auto chC : \*listC) {  
 if (chA == chB && chA == chC) {  
 (\*listE).push\_back(chA); found = 1; break;  
 }  
 }  
 if (found) {found = 0; break;}  
 }  
}  
  
for (auto itE = (\*listE).begin(); itE != (\*listE).end(); ++itE) {  
 for (auto itD = (\*listD).begin(); itD != (\*listD).end(); ++itD) {  
 if (\*itE == \*itD) {(\*listE).remove(\*itE); break;}  
 }  
}  
  
  
// Вывод результата  
cout << "Множество E = A ∩ B ∩ C - D: "; print((\*listE));  
  
auto t2 = std::chrono::high\_resolution\_clock::now();  
auto tt2 = clock();  
  
delete listA;  
delete listB;  
delete listC;  
delete listD;  
delete listE;  
  
cout << "Время выполнения: " << std::chrono::duration\_cast<std::chrono::duration<double, micro>>(t2-t1).count() << " мкс" << endl;  
cout << "Счетчик тиков: " << tt2-tt1 << endl;  
  
return 0;

}

#include #include #include #include "./headers/new\_set.h" #include

using namespace std;

void input(Set& s) { string line; getline(cin, line, '\n');

for (int i = 0; i < line.size(); ) {  
 unsigned char ch = line[i];  
 size\_t len;  
  
 // Определяем длину UTF-8 символа по первому байту  
 if ((ch & 0x80) == 0) len = 1; // 0xxxxxxx  
 else if ((ch & 0xE0) == 0xC0) len = 2; // 110xxxxx  
 else if ((ch & 0xF0) == 0xE0) len = 3; // 1110xxxx  
 else if ((ch & 0xF8) == 0xF0) len = 4; // 11110xxx  
 else len = 1; // fallback (некорректный UTF-8)  
  
 // Ограничиваем длину размером строки  
 if (i + len > line.size()) len = 1;  
  
 s.insert(line.substr(i, len));  
 i += len;  
}

}

int main() { setlocale(LC\_ALL, "Russian");

Set<string> setA;  
Set<string> setB;  
Set<string> setC;  
Set<string> setD;  
Set<string> setE;  
  
cout << "A: "; input(setA);  
cout << "B: "; input(setB);  
cout << "C: "; input(setC);  
cout << "D: "; input(setD);  
  
auto t1 = std::chrono::high\_resolution\_clock::now();  
auto tt1 = clock();  
  
cout << endl;  
  
cout << "A: "; setA.print();  
cout << "B: "; setB.print();  
cout << "C: "; setC.print();  
cout << "D: "; setD.print();  
  
// Находим A ∩ B ∩ C - D  
for (const auto& ch : setA) {  
 if (setB.contains(ch) && setC.contains(ch) && !setD.contains(ch)) setE.insert(ch);  
}  
  
// Вывод результата  
cout << "Множество E = A ∩ B ∩ C - D: "; setE.print();  
  
auto t2 = std::chrono::high\_resolution\_clock::now();  
auto tt2 = clock();  
  
cout << "Время выполнения: " << std::chrono::duration\_cast<std::chrono::duration<double, micro>>(t2-t1).count() << " мкс" << endl;  
cout << "Счетчик тиков: " << tt2-tt1 << endl;  
  
  
return 0;

}

# 