**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**ОТЧЕТ**

**по лабораторной работе № 2**

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

**Тема: «Множество как объект»**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 3311 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Сапронов К. Д.  Землякова С. А. |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Преподаватель |  | Манирагена Валенс |

Санкт-Петербург

2024

**Цель работы:**

Сравнение процедурного и объектно-ориентированного подходов на примере задачи обработки множеств

**Задание (вариант 31)**

Универсум: шестнадцатеричные цифры.

Требуется вычислить: множество, содержащее цифры, имеющиеся в А или С, но отсутствующие в B или D.

**Формула для вычисления нового множества**

**E = (A ∨ C) ∧ (¬ B ∨ ¬ D), также ее аналог: E = (A ∨ C) ∧ ¬ (B ∨ D),**.

**Сравнение результатов измерения времени.**

Измерение времени производится для 1 000 000 операций.

Массивы символов – 0.967 секунд

Односвязные списки – 9.521 секунд

Односвязные списки (без перегрузки) – 9.279 секунд

Массивы битов – 0.557 секунд

Машинные слова – 0.073 секунд

**Контрольные примеры:**

Множества, используемые в примерах:

A: 1259ABC

B: 15ACEF

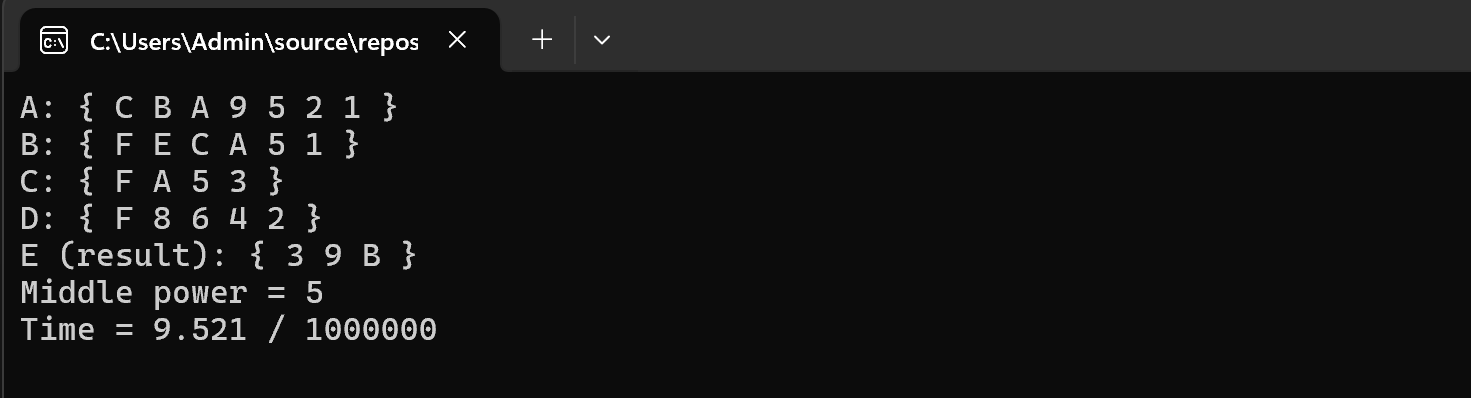
C: 35AF

D: 2468F

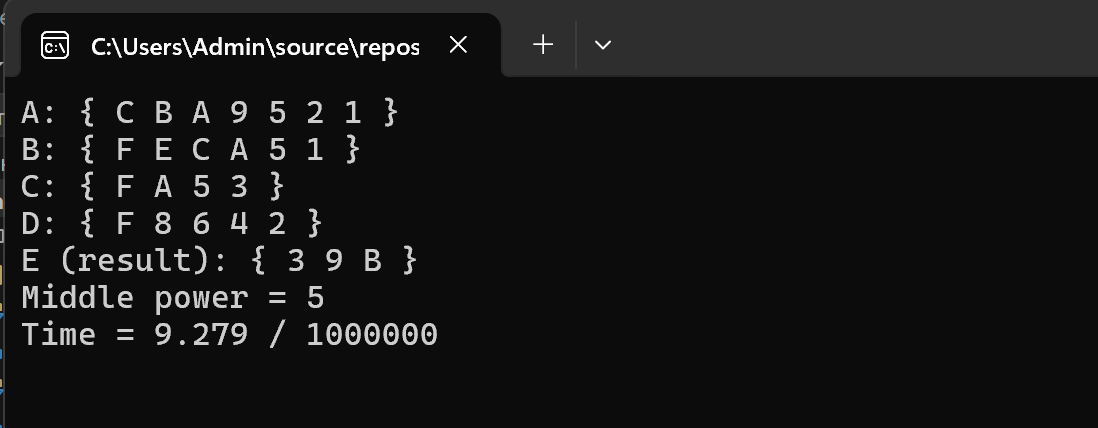
1. **Массивы символов**

****

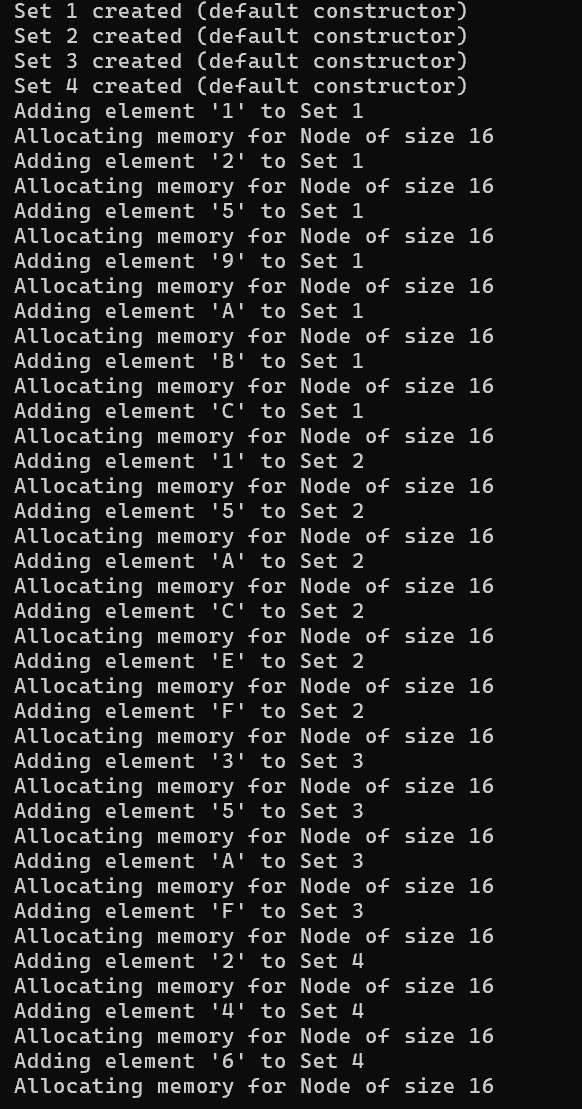
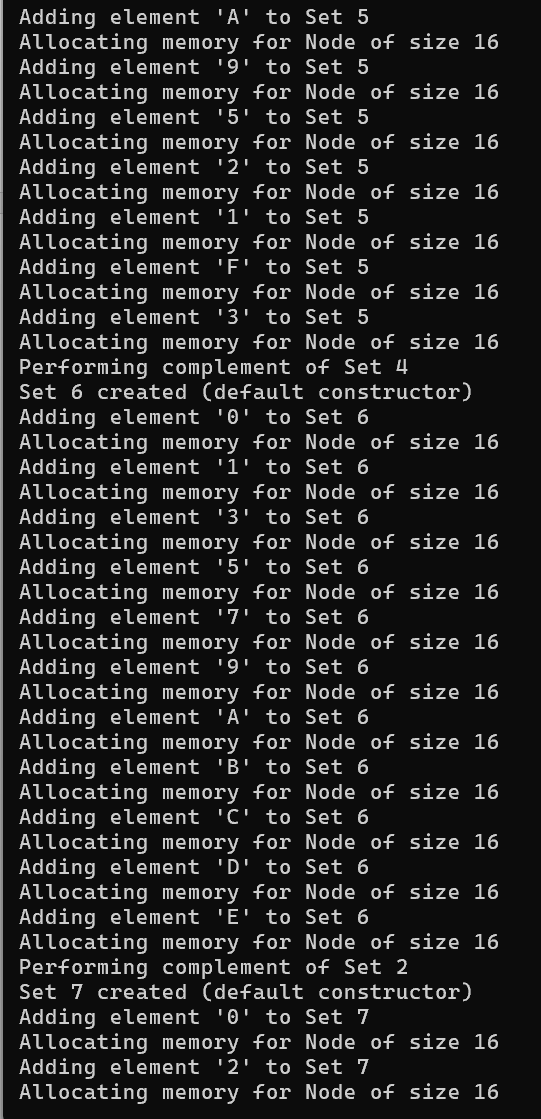
1. **Односвязные списки с перегрузкой new и delete**

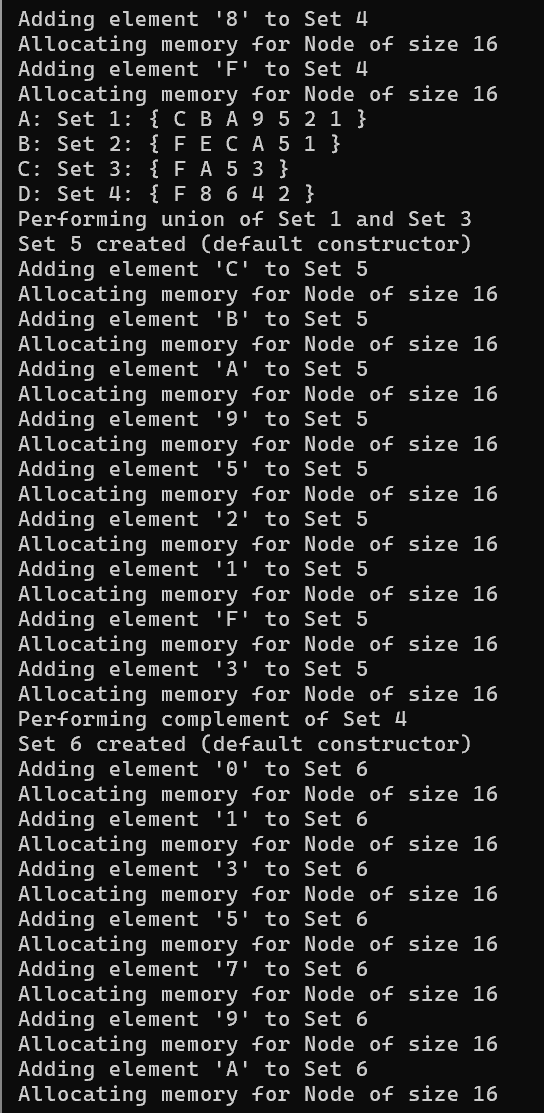
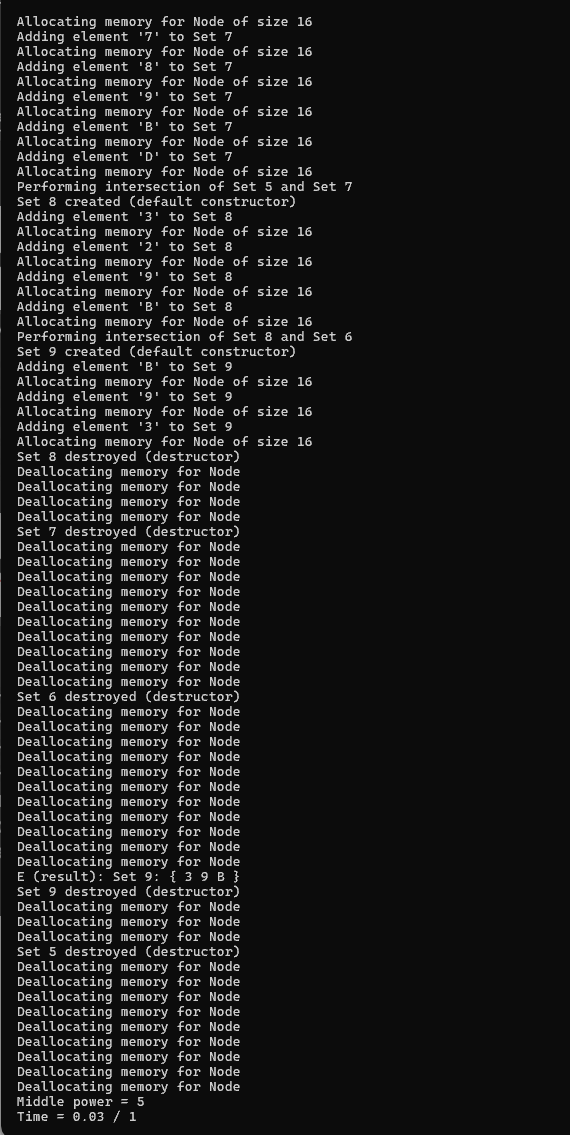
****

1. **Односвязные списки без перегрузки**

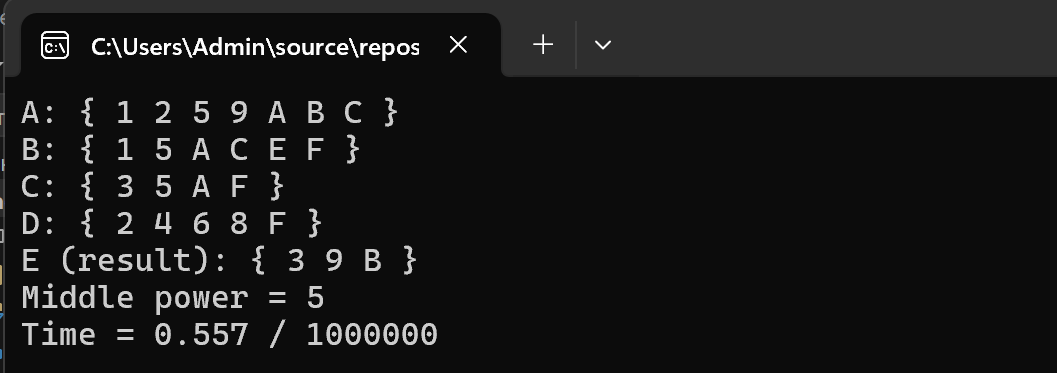
****

1. **Односвязные списки с отслеживанием вызова функций**

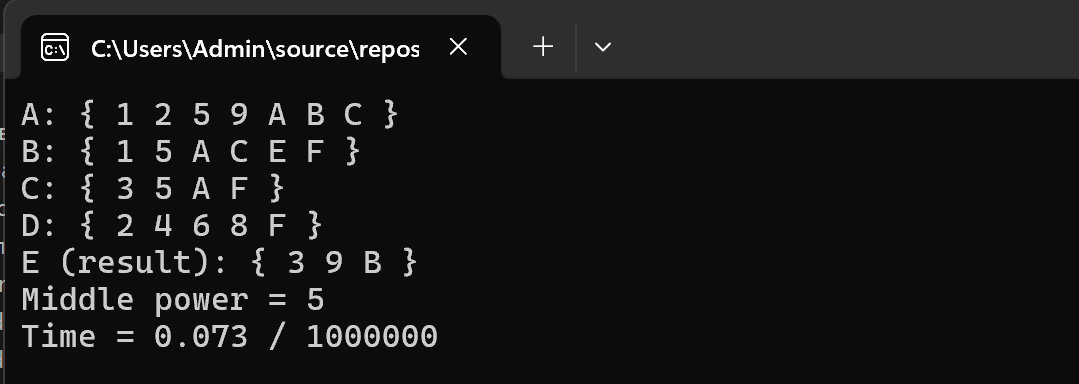
****  ****

** **

1. **Массивы битов**

****

1. **Машинные слова**

****

**Оценка сложности**

Временная сложность алгоритмов обработки сохраняется в соответствии с лабораторной работой №1.

**Результат эксперимента с отслеживанием вызовов функций-членов**

В результате эксперимента с отслеживанием вызовов функций членов был сделан вывод о вызове конструктора в последовательности от первого к последнему, а после вызов деструктора аналогично от первого к последнему.

**Заключение:**

В процессе выполнения задания успешно применялись операции над множествами с использованием различных типов данных, представленных в виде классов и вынесенных в отдельный заголовочный файл. Для каждого типа представления данных были разработаны соответствующие алгоритмы, поля и методы.

Решение задачи наглядно показало, что использование классов в данной лабораторной работе оказалось менее эффективно из-за увеличения времени вычислений. Тем не менее, реализация продемонстрировала один из ключевых принципов ООП — инкапсуляцию (защиту и скрытие данных). Сравнительный анализ времени выполнения операций для различных реализаций множеств дал ясное представление о том, как выбор структуры данных влияет на производительность и общую эффективность программы.

**Контрольные вопросы:**

**1. Класс какого типа можно использовать для создания объектов-множеств каждого из реализованных вами способов кодирования? Можно ли во всех случаях использовать класс-значение или класс-контроллер?**

Для каждого из способов кодирования множеств (массивы символов, массивы битов, машинное слово, списки) мы использовали класс Set.

Класс-значение можно использовать для большинства реализаций, поскольку множества имеют фиксированный размер (до 16 элементов) и являются копируемыми. Например, для реализации через массивы символов, массивы битов и машинное слово вполне подойдёт класс-значение. В этом случае все операции реализуются в виде стандартных операторов, а уничтожение множества не требует специальных процедур, потому что данные хранятся в пределах самого объекта.

Класс-контроллер подходит для реализации на основе связного списка, поскольку требует управления динамической памятью (каждый элемент списка выделяется через new). Класс-контроллер позволяет лучше управлять объектами, включая их создание, копирование и уничтожение. В этом случае требуется более сложное управление памятью, например, при копировании списка.

**2. Какую выгоду можно получить от применения объектов в программе обработки множеств?**

Использование объектов упрощает и систематизирует управление множествами за счёт: а) инкапсуляции данных и операций, что предотвращает случайное изменение данных напрямую; б) возможности объединить операции над множествами в одном классе, а также перегрузить операторы для интуитивного выполнения операций, таких как объединение и пересечение; в) создания независимых реализаций (через интерфейс класса Set), которые можно легко менять, не затрагивая основную логику программы.

**3. Как повлияло применение объектов на время вычисления множества-результата? Можно ли исключить такое влияние?**

Использование объектов немного увеличило время вычислений, потому что добавились вызовы конструкторов, деструкторов и других функций. В реализации со списками это особенно заметно, потому что приходится выделять и освобождать память под каждый элемент отдельно.

Частично сократить это влияние можно, например, для списка — использовать вместо new и delete пул объектов, чтобы сразу выделять память под несколько элементов. А вот для массивов битов и машинного слова влияние на производительность минимальное.

**4. Все ли созданные в программе множества действительно уничтожаются?**

Да, все множества уничтожаются, потому что для каждого из них автоматически вызывается деструктор при выходе из области видимости. Это касается и объектов, которые объявлены в main, так как они тоже выходят из области видимости после завершения main.

**5. В каком порядке происходит уничтожение множеств?**

Уничтожение объектов происходит в обратном порядке их создания. В main сначала удаляются множества, созданные позже, и только потом — те, что были созданы раньше. Это связано с тем, как объекты выходят из области видимости.

**6. Встречается ли в программе факт использования уже уничтоженного или ещё не созданного множества?**

Нет, такого в программе не встречается. Все объекты создаются и уничтожаются последовательно внутри main. Мы работаем только с теми объектами, которые уже созданы и находятся в доступной области видимости.

**7. Можно ли ожидать в программе использования конструктора копии или присваивания в варианте «с переносом»?**

Да, можно ожидать вызова конструктора копирования или оператора присваивания. Например, при передаче множества в функцию по значению, при создании нового множества на основе другого (Set E = A | B;), или при использовании оператора присваивания для копирования одного множества в другое.

**8. Наблюдалось ли такое использование, а если нет, что нужно изменить в программе, чтобы оно произошло?**

В программе используются операции копирования и присваивания, например, при выполнении операций над множествами, как в примере Set E = (AC & ~B) & ~D;

#include "setCharArray.h"

//#include "setLinkedList.h"

//#include "setLinkedListNoOverload.h"

//#include "setLinkedListTracking.h" // Version with function call tracking, set q0 = 1

//#include "setMachineWord.h"

//#include "setMassiveBits.h"

#include <string.h>

#include <stdlib.h>

#include <time.h>

#include <iostream>

using namespace std;

const long q0 = 1000000;

int main() {

srand(time(0)); // Initialize random number generator with current time

int powerE; // Variable to store the power (number of elements) of set E

Set A, B, C, D; // Initialize sets A, B, C, D

// Uncomment to define sets (char array)

// Set A ("1259ABC");

// Set B ("15ACEF");

// Set C ("35AF");

// Set D ("2468F");

// Linked list & massive of bites

// A.add('1'); A.add('2'); A.add('5'); A.add('9'); A.add('A'); A.add('B'); A.add('C');

// B.add('1'); B.add('5'); B.add('A'); B.add('C'); B.add('E'); B.add('F');

// C.add('3'); C.add('5'); C.add('A'); C.add('F');

// D.add('2'); D.add('4'); D.add('6'); D.add('8'); D.add('F');

// Machine word

// A = Set((1 << 1) | (1 << 2) | (1 << 5) | (1 << 9) | (1 << 10) | (1 << 11) | (1 << 12));

// B = Set((1 << 1) | (1 << 5) | (1 << 10) | (1 << 12) | (1 << 14) | (1 << 15));

// C = Set((1 << 3) | (1 << 5) | (1 << 10) | (1 << 15));

// D = Set((1 << 2) | (1 << 4) | (1 << 6) | (1 << 8) | (1 << 15));

// Randomly fill sets A, B, C, and D with elements (comment if defining manually)

A.randomFill();

B.randomFill();

C.randomFill();

D.randomFill();

// Display the elements of sets A, B, C, and D

cout << "A: "; A.show();

cout << "B: "; B.show();

cout << "C: "; C.show();

cout << "D: "; D.show();

// Performance measurement and set operations testing

clock\_t begin = clock(); // Start measuring time

for (long q = 0; q < q0; q++) {

// Inion of sets A and C

Set AC = A | C;

// Intersection of AC with the complements of B and D

Set E = (AC & ~B) & ~D;

// Display the result of the first iteration

if (q == 0) {

cout << "E (result): "; E.show();

powerE = E.power(); // Calculate and store the power of E

}

}

clock\_t end = clock(); // Stop measuring time

// Average power of the sets

cout << "Middle power = " <<

(A.power() + B.power() + C.power() + D.power() + powerE) / 5 <<

" \nTime = " << double(end - begin)/CLOCKS\_PER\_SEC << " / " << q0 << endl;

cin.get(); // Wait for the user to press Enter before exiting

return 0;

}

#ifndef SET\_H

#define SET\_H

#include <iostream>

#include <cstring>

using namespace std;

class Set {

private:

char A[17]; // Array to store elements (null-terminated string)

int n; // Number of elements in the set

public:

Set(); // Default constructor

Set(const char\*); // Constructor with initialization

Set operator | (const Set&) const; // Union operator

Set operator & (const Set&) const; // Intersection operator

Set operator ~ () const; // Complement operator

void show() const; // Display the set

bool contains(char) const; // Check if element exists in the set

int power() { return n; } // Return number of elements

void randomFill(); // Randomly fill the set

};

// Default constructor

Set::Set() : n(0) {

A[0] = '\0';

}

// Constructor with initialization (populate the set)

Set::Set(const char\* elems) {

n = strlen(elems);

strcpy\_s(A, elems);

}

// Union operator implementation

Set Set::operator | (const Set& B) const {

Set result;

strcpy\_s(result.A, A);

result.n = n;

// Add elements from set B that are not in A

for (int i = 0; i < B.n; ++i) {

if (!contains(B.A[i])) {

result.A[result.n++] = B.A[i];

}

}

result.A[result.n] = '\0'; // Terminate string

return result;

}

// Intersection operator implementation

Set Set::operator & (const Set& B) const {

Set result;

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

if (B.contains(A[i])) {

result.A[result.n++] = A[i];

}

}

result.A[result.n] = '\0';

return result;

}

// Complement operator implementation (relative to universal set)

Set Set::operator ~ () const {

Set result;

const char universe[] = "0123456789ABCDEF";

for (int i = 0; i < 16; ++i) {

if (!contains(universe[i])) {

result.A[result.n++] = universe[i];

}

}

result.A[result.n] = '\0';

return result;

}

// Check if element exists in the set

bool Set::contains(char elem) const {

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

if (A[i] == elem) return true;

}

return false;

}

// Display the set

void Set::show() const {

cout << "{ ";

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

cout << A[i] << ' ';

}

cout << "}" << endl;

}

// Fill the set with random elements

void Set::randomFill() {

const char universe[] = "0123456789ABCDEF";

n = 0;

int numElements = rand() % 16 + 1; // Random number of elements (1 to 16)

while (n < numElements) {

char randomChar = universe[rand() % 16]; // Select random character from universe

if (!contains(randomChar)) {

A[n++] = randomChar; // Add character if not already in the set

}

}

A[n] = '\0'; // Terminate string

}

#endif

#ifndef SET\_LINKED\_LIST\_H

#define SET\_LINKED\_LIST\_H

#include <iostream>

class Set {

private:

struct Node {

char el; // Element of the set

Node\* next; // Pointer to the next node

Node(char e, Node\* n = nullptr) : el(e), next(n) {} // Constructor for Node

// Overload new and delete operators

void\* operator new(size\_t size) {

return ::operator new(size);

}

void operator delete(void\* ptr) {

::operator delete(ptr);

}

};

Node\* head; // Pointer to the head node (start of the set)

static int tag\_counter; // Counter to generate unique tags for sets

int tag; // Unique tag for this set

public:

Set(); // Default constructor

Set(const Set&); // Copy constructor

Set(Set&&); // Move constructor

~Set(); // Destructor

Set& operator=(const Set&); // Copy assignment operator

Set& operator=(Set&&); // Move assignment operator

void randomFill(); // Fill the set with random elements

void add(char); // Add an element to the set

bool contains(char) const; // Check if an element is in the set

Set operator | (const Set&) const; // Union of two sets

Set operator & (const Set&) const; // Intersection of two sets

Set operator ~ () const; // Complement of the set

int power() const; // Size of the set (number of elements)

void show() const; // Print the set

void printTag() const; // Show the tag of the set

};

// Initialize static tag counter

int Set::tag\_counter = 0;

// Default constructor

Set::Set() : head(nullptr), tag(++tag\_counter) {}

// Copy constructor

Set::Set(const Set& other) : head(nullptr), tag(++tag\_counter) {

Node\* temp = other.head;

while (temp) {

add(temp->el);

temp = temp->next;

}

}

// Move constructor

Set::Set(Set&& other) : head(other.head), tag(++tag\_counter) {

other.head = nullptr;

}

// Destructor

Set::~Set() {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

// Copy assignment operator

Set& Set::operator=(const Set& other) {

if (this != &other) {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

head = nullptr;

Node\* temp = other.head;

while (temp) {

add(temp->el);

temp = temp->next;

}

}

return \*this;

}

// Move assignment operator

Set& Set::operator=(Set&& other) {

if (this != &other) {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

head = other.head;

other.head = nullptr;

}

return \*this;

}

// Add an element to the set

void Set::add(char elem) {

if (!contains(elem)) {

head = new Node(elem, head); // Insert the new node at the beginning

}

}

// Check if an element is in the set

bool Set::contains(char elem) const {

Node\* temp = head;

while (temp) {

if (temp->el == elem) return true;

temp = temp->next;

}

return false;

}

// Union of two sets

Set Set::operator | (const Set& B) const {

Set result;

// Add all elements from the first set

Node\* temp = head;

while (temp) {

result.add(temp->el);

temp = temp->next;

}

// Add elements from the second set that are not already in the first set

temp = B.head;

while (temp) {

result.add(temp->el);

temp = temp->next;

}

return result;

}

// Intersection of two sets

Set Set::operator & (const Set& B) const {

Set result;

Node\* temp = head;

while (temp) {

if (B.contains(temp->el)) {

result.add(temp->el);

}

temp = temp->next;

}

return result;

}

// Complement of the set

Set Set::operator ~ () const {

Set result;

const char universe[] = "0123456789ABCDEF"; // Universal set

for (int i = 0; i < 16; ++i) {

if (!contains(universe[i])) {

result.add(universe[i]);

}

}

return result;

}

// Size of the set

int Set::power() const {

int count = 0;

Node\* temp = head;

while (temp) {

count++;

temp = temp->next;

}

return count;

}

// Print the set

void Set::show() const {

Node\* temp = head;

std::cout << "{ ";

while (temp) {

std::cout << temp->el << ' ';

temp = temp->next;

}

std::cout << "}" << std::endl;

}

// Fill the set with random elements

void Set::randomFill() {

const char universe[] = "0123456789ABCDEF"; // Universal set

int numElements = rand() % 16 + 1; // Number of elements (1 to 16)

while (power() < numElements) {

char randomChar = universe[rand() % 16]; // Randomly pick a character

add(randomChar); // Add the character if it's not already in the set

}

}

// Print the tag of the set

void Set::printTag() const {

std::cout << "Set " << tag << std::endl;

}

#endif // SET\_LINKED\_LIST\_H

#ifndef SET\_LINKED\_LIST\_H

#define SET\_LINKED\_LIST\_H

#include <iostream>

class Set {

private:

struct Node { // Node structure (list element)

char el; // Element value

Node\* next; // Pointer to the next element

Node(char e, Node\* n = nullptr) : el(e), next(n) {} // Node constructor

};

Node\* head; // Head of the list (start of set)

public:

Set(); // Default constructor (empty set)

~Set(); // Destructor (cleans up memory)

void randomFill(); // Fill set with random elements

void add(char); // Add an element to the set

bool contains(char) const; // Check if an element is in the set

Set operator | (const Set&) const; // Union operator

Set operator & (const Set&) const; // Intersection operator

Set operator ~ () const; // Complement operator

int power() const; // Set power (number of elements)

void show() const; // Display the set

};

// Default constructor

Set::Set() : head(nullptr) {}

// Destructor, freeing memory

Set::~Set() {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

// Add an element to the set

void Set::add(char elem) {

if (!contains(elem)) {

head = new Node(elem, head); // Add new node at the beginning of the list

}

}

// Check if an element is in the set

bool Set::contains(char elem) const {

Node\* temp = head;

while (temp) {

if (temp->el == elem) return true;

temp = temp->next;

}

return false;

}

// Union operation

Set Set::operator | (const Set& B) const {

Set result;

// Add all elements from the current set

Node\* temp = head;

while (temp) {

result.add(temp->el);

temp = temp->next;

}

// Add elements from B that are not already in the current set

temp = B.head;

while (temp) {

result.add(temp->el);

temp = temp->next;

}

return result;

}

// Intersection operation

Set Set::operator & (const Set& B) const {

Set result;

Node\* temp = head;

while (temp) {

if (B.contains(temp->el)) {

result.add(temp->el);

}

temp = temp->next;

}

return result;

}

// Complement operation

Set Set::operator ~ () const {

Set result;

const char universe[] = "0123456789ABCDEF"; // Universe of possible elements

for (int i = 0; i < 16; ++i) {

if (!contains(universe[i])) {

result.add(universe[i]);

}

}

return result;

}

// Set power (number of elements)

int Set::power() const {

int count = 0;

Node\* temp = head;

while (temp) {

count++;

temp = temp->next;

}

return count;

}

// Display the set

void Set::show() const {

Node\* temp = head;

std::cout << "{ ";

while (temp) {

std::cout << temp->el << ' ';

temp = temp->next;

}

std::cout << "}" << std::endl;

}

// Fill the set with random elements

void Set::randomFill() {

const char universe[] = "0123456789ABCDEF"; // Universe of possible elements

int numElements = rand() % 16 + 1; // Random number of elements (1 to 16)

while (power() < numElements) {

char randomChar = universe[rand() % 16]; // Select random character from universe

add(randomChar); // Add character if not already in the set

}

}

#endif

#ifndef SET\_LINKED\_LIST\_H

#define SET\_LINKED\_LIST\_H

#include <iostream>

class Set {

private:

struct Node {

char el; // Element of the set

Node\* next; // Pointer to the next node

Node(char e, Node\* n = nullptr) : el(e), next(n) {} // Constructor for Node

// Overload new and delete operators

void\* operator new(size\_t size) {

std::cout << "Allocating memory for Node of size " << size << std::endl;

return ::operator new(size);

}

void operator delete(void\* ptr) {

std::cout << "Deallocating memory for Node" << std::endl;

::operator delete(ptr);

}

};

Node\* head; // Pointer to the head node (start of the set)

static int tag\_counter; // Counter to generate unique tags for sets

int tag; // Unique tag for this set

public:

Set(); // Default constructor

Set(const Set&); // Copy constructor

Set(Set&&); // Move constructor

~Set(); // Destructor

Set& operator=(const Set&); // Copy assignment operator

Set& operator=(Set&&); // Move assignment operator

void randomFill(); // Fill the set with random elements

void add(char); // Add an element to the set

bool contains(char) const; // Check if an element is in the set

Set operator | (const Set&) const; // Union of two sets

Set operator & (const Set&) const; // Intersection of two sets

Set operator ~ () const; // Complement of the set

int power() const; // Size of the set (number of elements)

void show() const; // Print the set

void printTag() const; // Show the tag of the set

};

// Initialize static tag counter

int Set::tag\_counter = 0;

// Default constructor

Set::Set() : head(nullptr), tag(++tag\_counter) {

std::cout << "Set " << tag << " created (default constructor)" << std::endl;

}

// Copy constructor

Set::Set(const Set& other) : head(nullptr), tag(++tag\_counter) {

std::cout << "Set " << tag << " created (copy constructor from Set " << other.tag << ")" << std::endl;

Node\* temp = other.head;

while (temp) {

add(temp->el);

temp = temp->next;

}

}

// Move constructor

Set::Set(Set&& other) : head(other.head), tag(++tag\_counter) {

std::cout << "Set " << tag << " created (move constructor from Set " << other.tag << ")" << std::endl;

other.head = nullptr;

}

// Destructor

Set::~Set() {

std::cout << "Set " << tag << " destroyed (destructor)" << std::endl;

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

// Copy assignment operator

Set& Set::operator=(const Set& other) {

if (this != &other) {

std::cout << "Set " << tag << " assigned (copy assignment from Set " << other.tag << ")" << std::endl;

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

head = nullptr;

Node\* temp = other.head;

while (temp) {

add(temp->el);

temp = temp->next;

}

}

return \*this;

}

// Move assignment operator

Set& Set::operator=(Set&& other) {

if (this != &other) {

std::cout << "Set " << tag << " assigned (move assignment from Set " << other.tag << ")" << std::endl;

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

head = other.head;

other.head = nullptr;

}

return \*this;

}

// Add an element to the set

void Set::add(char elem) {

if (!contains(elem)) {

std::cout << "Adding element '" << elem << "' to Set " << tag << std::endl;

head = new Node(elem, head); // Insert the new node at the beginning

}

}

// Check if an element is in the set

bool Set::contains(char elem) const {

Node\* temp = head;

while (temp) {

if (temp->el == elem) return true;

temp = temp->next;

}

return false;

}

// Union of two sets

Set Set::operator | (const Set& B) const {

std::cout << "Performing union of Set " << tag << " and Set " << B.tag << std::endl;

Set result;

// Add all elements from the first set

Node\* temp = head;

while (temp) {

result.add(temp->el);

temp = temp->next;

}

// Add elements from the second set that are not already in the first set

temp = B.head;

while (temp) {

result.add(temp->el);

temp = temp->next;

}

return result;

}

// Intersection of two sets

Set Set::operator & (const Set& B) const {

std::cout << "Performing intersection of Set " << tag << " and Set " << B.tag << std::endl;

Set result;

Node\* temp = head;

while (temp) {

if (B.contains(temp->el)) {

result.add(temp->el);

}

temp = temp->next;

}

return result;

}

// Complement of the set

Set Set::operator ~ () const {

std::cout << "Performing complement of Set " << tag << std::endl;

Set result;

const char universe[] = "0123456789ABCDEF"; // Universal set

for (int i = 0; i < 16; ++i) {

if (!contains(universe[i])) {

result.add(universe[i]);

}

}

return result;

}

// Size of the set

int Set::power() const {

int count = 0;

Node\* temp = head;

while (temp) {

count++;

temp = temp->next;

}

return count;

}

// Print the set

void Set::show() const {

std::cout << "Set " << tag << ": { ";

Node\* temp = head;

while (temp) {

std::cout << temp->el << ' ';

temp = temp->next;

}

std::cout << "}" << std::endl;

}

// Fill the set with random elements

void Set::randomFill() {

const char universe[] = "0123456789ABCDEF"; // Universal set

int numElements = rand() % 16 + 1; // Number of elements (1 to 16)

std::cout << "Filling Set " << tag << " with random elements" << std::endl;

while (power() < numElements) {

char randomChar = universe[rand() % 16]; // Randomly pick a character

add(randomChar); // Add the character if it's not already in the set

}

}

// Print the tag of the set

void Set::printTag() const {

std::cout << "Set " << tag << std::endl;

}

#endif // SET\_LINKED\_LIST\_H

#ifndef SET\_MASSIVE\_BITS\_H

#define SET\_MASSIVE\_BITS\_H

#include <iostream>

#include <cstdlib>

class Set {

private:

bool bits[16]; // Bit array to represent elements of the set

public:

Set(); // Default constructor (empty set)

void randomFill(); // Fill set with random elements

void add(char); // Add element to the set

bool contains(char) const; // Check if an element is in the set

Set operator | (const Set&) const; // Union operator

Set operator & (const Set&) const; // Intersection operator

Set operator ~ () const; // Complement operator

int power() const; // Set power (number of elements)

void show() const; // Display the set

};

// Default constructor with empty set initialization

Set::Set() {

for (int i = 0; i < 16; ++i) {

bits[i] = false;

}

}

// Fill set with random elements

void Set::randomFill() {

for (int i = 0; i < 16; ++i) {

bits[i] = false;

}

int numElements = rand() % 16 + 1; // Random number of elements (1 to 16)

while (power() < numElements) {

int index = rand() % 16;

bits[index] = true; // Set random element

}

}

// Add an element to the set

void Set::add(char elem) {

int index = (elem <= '9') ? elem - '0' : elem - 'A' + 10;

bits[index] = true; // Set bit for this element

}

// Check if an element is in the set

bool Set::contains(char elem) const {

int index = (elem <= '9') ? elem - '0' : elem - 'A' + 10;

return bits[index];

}

// Union operation

Set Set::operator | (const Set& B) const {

Set result;

for (int i = 0; i < 16; ++i) {

result.bits[i] = bits[i] | B.bits[i]; // Bitwise OR operation

}

return result;

}

// Intersection operation

Set Set::operator & (const Set& B) const {

Set result;

for (int i = 0; i < 16; ++i) {

result.bits[i] = bits[i] & B.bits[i]; // Bitwise AND operation

}

return result;

}

// Complement operation (universal set)

Set Set::operator ~ () const {

Set result;

for (int i = 0; i < 16; ++i) {

result.bits[i] = !bits[i]; // Invert bits

}

return result;

}

// Set power (number of elements)

int Set::power() const {

int count = 0;

for (int i = 0; i < 16; ++i) {

if (bits[i]) count++;

}

return count;

}

// Display the set

void Set::show() const {

std::cout << "{ ";

for (int i = 0; i < 16; ++i) {

if (bits[i]) {

if (i < 10) std::cout << char('0' + i) << ' ';

else std::cout << char('A' + (i - 10)) << ' ';

}

}

std::cout << "}" << std::endl;

}

#endif

#ifndef SET\_MACHINE\_WORD\_H

#define SET\_MACHINE\_WORD\_H

#include <iostream>

#include <bitset>

#include <cstring>

using namespace std;

class Set {

private:

unsigned short word; // Machine word (16 bits, represents set elements)

public:

Set(); // Default constructor

Set(unsigned short w); // Constructor with given word

void randomFill(); // Fill set with random elements

Set operator | (const Set&) const; // Union operator

Set operator & (const Set&) const; // Intersection operator

Set operator ~ () const; // Complement operator

void show() const; // Display the set

int power() const; // Set power (number of elements)

};

// Default constructor initializing with an empty set

Set::Set() : word(0) {}

// Constructor with a given word value

Set::Set(unsigned short w) : word(w) {}

// Fill set with random elements

void Set::randomFill() {

word = 0;

int numElements = rand() % 16 + 1; // Random number of elements (1 to 16)

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

word |= (1 << (rand() % 16)); // Set a random bit

}

}

// Union operation

Set Set::operator | (const Set& B) const {

return Set(word | B.word);

}

// Intersection operation

Set Set::operator & (const Set& B) const {

return Set(word & B.word);

}

// Complement operation (16-bit complement)

Set Set::operator ~ () const {

return Set(~word & 0xFFFF); // Invert bits and mask to 16 bits

}

// Display the set

void Set::show() const {

std::cout << "{ ";

for (int i = 0; i < 16; ++i) {

if (word & (1 << i)) { // Check if bit is set

if (i < 10) std::cout << char('0' + i) << ' ';

else std::cout << char('A' + (i - 10)) << ' ';

}

}

std::cout << "}" << std::endl;

}

// Set power (number of set elements)

int Set::power() const {

return std::bitset<16>(word).count(); // Count the number of set bits

}

#endif