Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Электротехнический факультет   
Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т  
по лабораторной работе**

**“Калькулятор множеств”**

Дисциплина: «Дискретная математика и математическая логика»

Выполнил   
студент группы ИВТ-23-2б  
Злыгостев Д.Н.

Проверил   
доцент кафедры ИТАС  
Рустамханова Г. И.

Пермь 2024

Оглавление

[1. Введение 3](#_Toc181348199)

[2. Теоретическая часть 3](#_Toc181348200)

[3. Описание алгоритма 3](#_Toc181348201)

[4. Исходный код 4](#_Toc181348202)

[5. Тестирование и результаты 11](#_Toc181348203)

[6. Анализ сложности 11](#_Toc181348204)

[7. Git-Hub 12](#_Toc181348205)

[8. Заключение 12](#_Toc181348206)

[9. Приложения 12](#_Toc181348207)

# 1. Введение

В данной лабораторной работе рассматривается задача реализации калькулятора множеств, который позволяет выполнять различные операции над множествами целых чисел в пределах заданного универсума от -30 до 30. Основная цель работы заключается в создании программного обеспечения, которое способно эффективно управлять множествами и выполнять над ними операции, такие как пересечение, объединение, разность, симметрическая разность и дополнение.

# 2. Теоретическая часть

Множество — это коллекция уникальных элементов. В данной работе реализованы следующие операции над множествами:

* **Объединение**: объединяет элементы двух множеств.
* **Пересечение**: включает только те элементы, которые присутствуют в обоих множествах.
* **Разность**: включает элементы одного множества, которые отсутствуют в другом.
* **Симметрическая разность**: включает элементы, которые находятся только в одном из множеств.
* **Дополнение**: включает все элементы универсума, кроме элементов заданного множества.

Универсум в данной работе определен как целые числа от -30 до 30.

# 3. Описание алгоритма

Программа реализует следующие шаги:

1. Запрос количества множеств от пользователя.
2. Запрос метода задания множества: случайно или через ввод.
3. Применение условий фильтрации к созданным множествам.
4. Ввод выражения для вычисления с использованием различных операций над множествами.
5. Вычисление результата и вывод его на экран.

# 4. Исходный код

#include <iostream>

#include <vector>

#include <set>

#include <algorithm>

#include <stack>

#include <map>

#include <cstdlib>

#include <ctime>

#include <string>

using namespace std;

// Константы универсума

const int UNIVERSE\_MIN = -30; // Минимальное значение универсума

const int UNIVERSE\_MAX = 30; // Максимальное значение универсума

const int UNIVERSE\_SIZE = UNIVERSE\_MAX - UNIVERSE\_MIN + 1; // Размер универсума

// Функция для вывода множества

void printSet(const set<int>& s) {

if (s.empty()) {

cout << "0!"; // Если множество пустое, выводим 0!

}

else {

for (int elem : s) {

cout << elem << " "; // Выводим каждый элемент множества через пробел

}

}

cout << endl;

}

// Функция для вычисления пересечения множеств

set<int> setIntersection(const set<int>& a, const set<int>& b) {

set<int> result;

// Вычисляем пересечение множеств a и b, результат сохраняем в result

set\_intersection(a.begin(), a.end(), b.begin(), b.end(), inserter(result, result.begin()));

return result;

}

// Функция для вычисления объединения множеств

set<int> setUnion(const set<int>& a, const set<int>& b) {

set<int> result(a); // Копируем множество a в result

result.insert(b.begin(), b.end()); // Добавляем элементы множества b

return result;

}

// Функция для вычисления разности множеств (a \ b)

set<int> setDifference(const set<int>& a, const set<int>& b) {

set<int> result;

// Вычисляем разность множеств a и b, результат сохраняем в result

set\_difference(a.begin(), a.end(), b.begin(), b.end(), inserter(result, result.begin()));

return result;

}

// Функция для симметрической разности множеств

set<int> setSymmetricDifference(const set<int>& a, const set<int>& b) {

set<int> result;

// Вычисляем симметрическую разность множеств a и b

set\_symmetric\_difference(a.begin(), a.end(), b.begin(), b.end(), inserter(result, result.begin()));

return result;

}

// Функция для дополнения множества относительно универсума

set<int> setComplement(const set<int>& a) {

set<int> result;

// Проходим по всем элементам универсума

for (int i = UNIVERSE\_MIN; i <= UNIVERSE\_MAX; ++i) {

if (a.find(i) == a.end()) {

// Если элемент не находится в множестве a, добавляем его в результат

result.insert(i);

}

}

return result;

}

// Функция для определения приоритета операций

int getPriority(char op) {

// Определяем приоритет операций: \* и \ имеют приоритет 2, + и ^ — приоритет 1, - (дополнение) — приоритет 3

if (op == '\*' || op == '\\') return 2;

if (op == '+' || op == '^') return 1;

if (op == '-') return 3; // Для унарной операции дополнения

return 0;

}

// Функция для выполнения операции над множествами

set<int> applyOperation(const set<int>& a, const set<int>& b, char op) {

// В зависимости от операции вызываем соответствующую функцию

switch (op) {

case '\*': return setIntersection(a, b); // Пересечение

case '+': return setUnion(a, b); // Объединение

case '\\': return setDifference(a, b); // Разность

case '^': return setSymmetricDifference(a, b); // Симметрическая разность

default: return set<int>(); // Возвращаем пустое множество в случае ошибки

}

}

// Функция для парсинга и вычисления выражений

set<int> evaluateExpression(const string& expression, const vector<set<int>>& sets) {

stack<set<int>> operands; // Стек для операндов (множеств)

stack<char> operators; // Стек для операторов

for (size\_t i = 0; i < expression.length(); ++i) {

char token = expression[i]; // Текущий символ в выражении

if (isdigit(token)) {

// Если символ — цифра (индекс множества)

int idx = token - '0' - 1; // Преобразуем в индекс множества (индексация с 0)

operands.push(sets[idx]); // Кладем соответствующее множество в стек операндов

}

else if (token == '(') {

// Если символ — открывающая скобка, добавляем её в стек операторов

operators.push(token);

}

else if (token == ')') {

// Если символ — закрывающая скобка, выполняем операции до открывающей скобки

while (!operators.empty() && operators.top() != '(') {

char op = operators.top(); // Достаем оператор из стека

operators.pop();

set<int> b = operands.top(); operands.pop(); // Достаем множество b

set<int> a = operands.top(); operands.pop(); // Достаем множество a

operands.push(applyOperation(a, b, op)); // Выполняем операцию и кладем результат в стек

}

operators.pop(); // Удаляем открывающую скобку

}

else if (token == '\*' || token == '+' || token == '\\' || token == '^' || token == '-') {

// Если символ — оператор

while (!operators.empty() && getPriority(operators.top()) >= getPriority(token)) {

// Выполняем все операции с более высоким приоритетом

char op = operators.top();

operators.pop();

if (op == '-') { // Если оператор — унарное дополнение

set<int> a = operands.top(); operands.pop(); // Достаем множество a

operands.push(setComplement(a)); // Кладем дополнение множества в стек

}

else {

set<int> b = operands.top(); operands.pop(); // Достаем множество b

set<int> a = operands.top(); operands.pop(); // Достаем множество a

operands.push(applyOperation(a, b, op)); // Выполняем операцию и кладем результат в стек

}

}

operators.push(token); // Добавляем текущий оператор в стек

}

}

// Выполняем оставшиеся операции в стеке

while (!operators.empty()) {

char op = operators.top();

operators.pop();

if (op == '-') { // Если оператор — унарное дополнение

set<int> a = operands.top(); operands.pop();

operands.push(setComplement(a)); // Кладем дополнение множества в стек

}

else {

set<int> b = operands.top(); operands.pop();

set<int> a = operands.top(); operands.pop();

operands.push(applyOperation(a, b, op)); // Выполняем операцию

}

}

return operands.top(); // Возвращаем результат

}

// Функция для генерации случайного множества

set<int> generateRandomSet(int size) {

set<int> s; // Создаем пустое множество

while (s.size() < size) {

int num = rand() % (UNIVERSE\_MAX - UNIVERSE\_MIN + 1) + UNIVERSE\_MIN; // Генерируем случайное число в универсума

s.insert(num); // Добавляем число в множество

}

return s; // Возвращаем сгенерированное множество

}

// Основная функция

int main() {

srand(static\_cast<unsigned>(time(0))); // Инициализация генератора случайных чисел

int n; // Количество множеств

cout << "Введите количество множеств: ";

cin >> n;

vector<set<int>> sets(n); // Вектор множеств

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

int choice;

cout << "Как задать множество " << (i + 1) << "? (1 - случайно, 2 - вручную): ";

cin >> choice;

if (choice == 1) {

int size;

cout << "Введите размер множества: ";

cin >> size;

sets[i] = generateRandomSet(size); // Генерация случайного множества

}

else {

int size;

cout << "Введите размер множества: ";

cin >> size;

cout << "Введите элементы множества (через пробел): ";

for (int j = 0; j < size; ++j) {

int elem;

cin >> elem; // Ввод элемента

sets[i].insert(elem); // Добавляем элемент в множество

}

}

// Выводим созданное множество

cout << "Множество " << (i + 1) << ": ";

printSet(sets[i]);

}

// Ввод выражения для вычисления

string expression;

cout << "Введите выражение для вычисления (например, 1 \* 2 + 3): ";

cin >> expression;

// Вычисление результата

set<int> result = evaluateExpression(expression, sets);

// Вывод результата

cout << "Результат: ";

printSet(result);

return 0;

}

# 5. Тестирование и результаты

Программа была протестирована с различными входными данными:

1. **Пример 1**:
   * Множества:
     + 1: {1, 2, 3}
     + 2: {3, 4, 5}
     + 3: {5, 6, 7}
   * Выражение: 1 \* 2 + 3
   * Результат: {3, 5, 6, 7}
2. **Пример 2**:
   * Множества:
     + 1: {1, 2, 3}
     + 2: {1, 2}
     + 3: {2, 3, 4}
   * Выражение: 1 + 2 ^ 3
   * Результат: {1, 4}
3. **Крайние случаи**:
   * Пустые множества: Результат соответствует ожиданиям (выводит 0!).
   * Все элементы в универсуме: Результат также соответствует ожиданиям.

# 6. Анализ сложности

* Временная сложность операций:
  + Пересечение: O(n + m), где n и m — размеры множеств.
  + Объединение: O(n + m).
  + Разность: O(n + m).
  + Симметрическая разность: O(n + m).
  + Дополнение: O(U), где U — размер универсума.

# 7. Git-Hub

Ссылка на репозиторий:

[Diskretka\_Math/лаба 1 at main · ZligostevDenis/Diskretka\_Math](https://github.com/ZligostevDenis/Diskretka_Math/tree/main/%D0%BB%D0%B0%D0%B1%D0%B0%201)

# 8. Заключение

В ходе лабораторной работы была успешно разработана программа для работы с множествами. Программа продемонстрировала свою эффективность при выполнении различных операций и справилась с проверкой на крайние случаи.

# 9. Приложения

* Скриншоты работы программы.
* Примеры входных данных и результаты.

