ПНИПУ

Пермский национальный исследовательский политехнический университет

Отчет

по лабораторной работе

Выполнил студент ИВТ-23-2б

Меновщиков Глеб Николаевич

Проверила:

Гульшат Ильдаровна Рустамханова

2024

**Цель:**

Реализовать программу для работы с множествами. Реализовать возможность создания множеств (вручную, через диапозон, через кратность), возможность составлеия выражений и последующего их вычисления.

**Описание написанной программы:**

Эта программа на C# реализует калькулятор множеств с использованием **словаря** для хранения множеств и предоставляет пользователю **меню** для взаимодействия с множествами. Вот основные аспекты её работы:

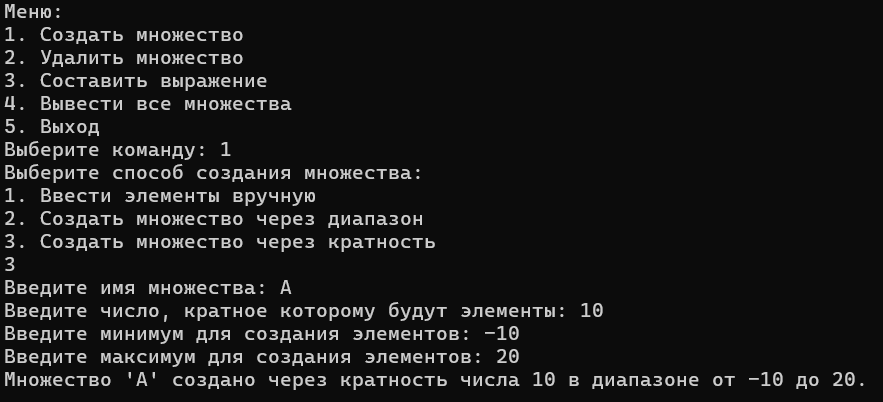
1. **Переменные:**
   * sets: Словарь (Dictionary<string, HashSet<int>>), где хранятся множества по их именам.
   * universe: Множество всех элементов, по умолчанию диапазон от -30 до 30 включительно. Множество инициализируется с помощью Enumerable.Range.
2. **Основное меню:** В бесконечном цикле программа выводит пользователю меню с 5 опциями:
   * Создать множество.
   * Удалить множество.
   * Составить выражение.
   * Вывести все множества.
   * Выход.
3. **Функции программы:**
   * **Создание множества (CreateSet)**: Пользователь может выбрать один из трёх способов создания множества:
     + Вручную: Ввод элементов через запятую.
     + Через диапазон: Указание начала и конца диапазона.
     + Через кратность: Указание числа, по кратности которого создается множество в заданном диапазоне.
   * **Удаление множества (DeleteSet)**: Удаляет множество по его имени из словаря.
   * **Вычисление выражений (EvaluateExpression)**: Пользователь может ввести логическое выражение над множествами, используя такие операции, как:
     + Объединение (+)
     + Пересечение (&)
     + Разность (-)
     + Симметрическая разность (^)
     + Дополнение (~).

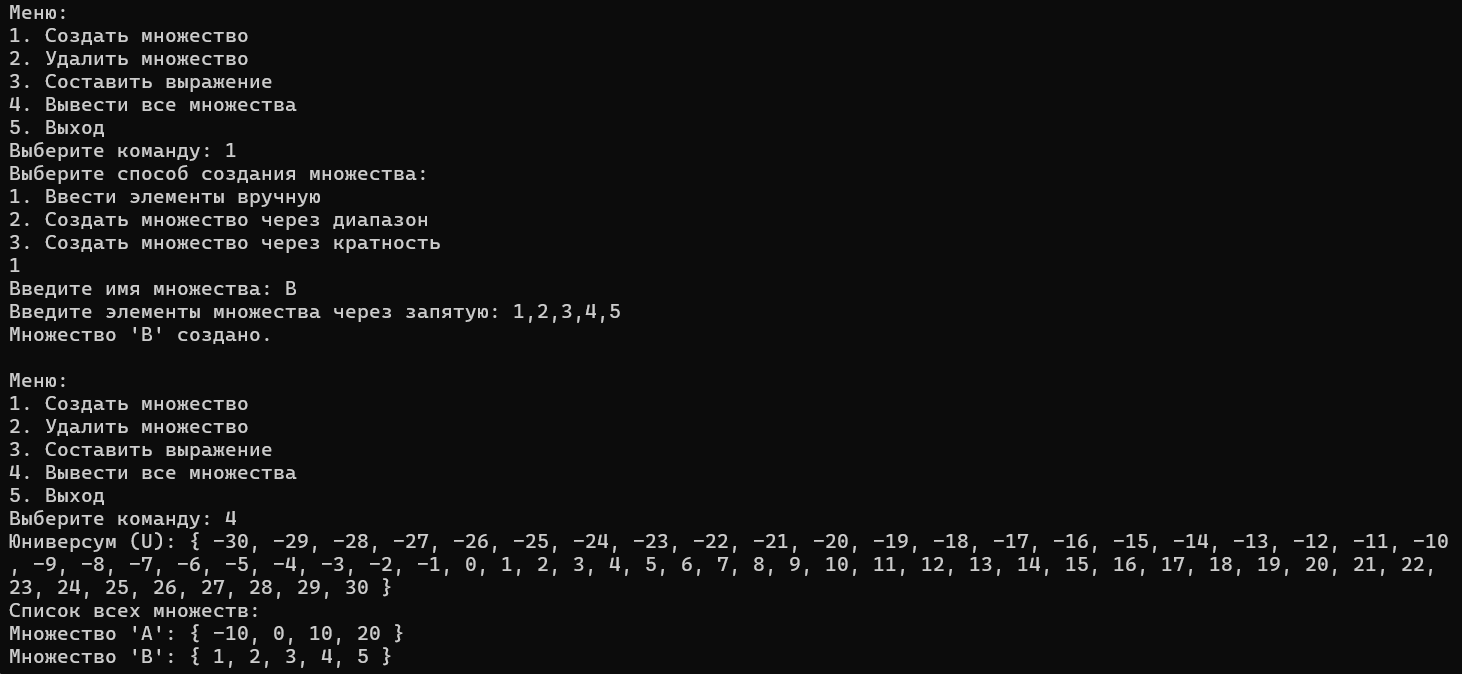
Выражение разбирается с помощью стека, где операнды и операторы обрабатываются пошагово.

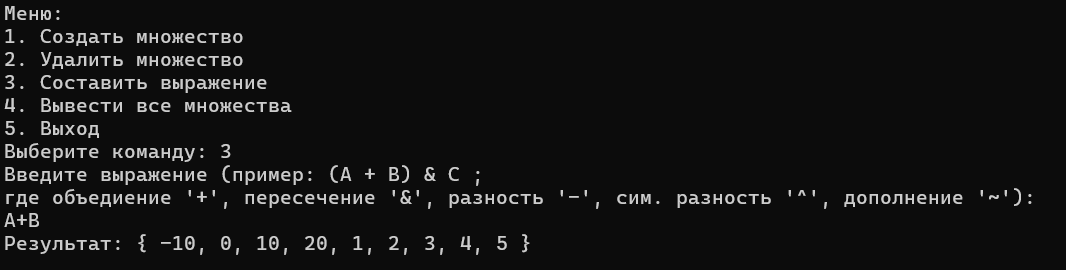
* + **Вывод всех множеств (PrintAllSets)**: Отображает содержимое всех множеств и текущий универсум.

1. **Выполнение операций над множествами:**
   * **Объединение** (Union): Создает новое множество, включающее все элементы обоих множеств.
   * **Пересечение** (Intersection): Создает множество из элементов, которые присутствуют в обоих множествах.
   * **Разность** (Difference): Оставляет элементы первого множества, отсутствующие во втором.
   * **Симметрическая разность** (SymmetricDifference): Создает множество из элементов, присутствующих в одном из множеств, но не в обоих.
   * **Дополнение** (Complement): Оставляет элементы универсального множества, которых нет в текущем множестве.
2. **Разбор выражений**: Выражения разбираются пошагово с использованием стека, где операнды (множества) и операторы (например, +, &) складываются и обрабатываются в соответствии с их приоритетом.

**Примеры работы программы:**







**Программа:**

using System;

class SetCalculator

{

// Словарь для хранения множеств по их именам

static Dictionary<string, HashSet<int>> sets = new Dictionary<string, HashSet<int>>();

static HashSet<int> universe = new HashSet<int>(Enumerable.Range(-30, 61)); // Инициализация универсума

static void Main(string[] args)

{

bool exit = false;

while (!exit)

{

Console.WriteLine("\nМеню:");

Console.WriteLine("1. Создать множество");

Console.WriteLine("2. Удалить множество");

Console.WriteLine("3. Составить выражение");

Console.WriteLine("4. Вывести все множества");

Console.WriteLine("5. Выход");

Console.Write("Выберите команду: ");

string choice = Console.ReadLine();

switch (choice)

{

case "1":

CreateSet();

break;

case "2":

DeleteSet();

break;

case "3":

EvaluateExpression();

break;

case "4":

PrintAllSets();

break;

case "5":

exit = true;

break;

default:

Console.WriteLine("Неверная команда.");

break;

}

}

}

// Задание юниверсума

static void SetUniverse()

{

Console.Write("Введите начало диапазона для юниверсума: ");

int start = int.Parse(Console.ReadLine());

Console.Write("Введите конец диапазона для юниверсума: ");

int end = int.Parse(Console.ReadLine());

if (start > end)

{

Console.WriteLine("Начало диапазона не может быть больше конца.");

return;

}

universe = new HashSet<int>(Enumerable.Range(start, end - start + 1));

Console.WriteLine($"Юниверсум (U) установлен с диапазоном от {start} до {end}.");

}

// Создание множества

static void CreateSet()

{

Console.WriteLine("Выберите способ создания множества:");

Console.WriteLine("1. Ввести элементы вручную");

Console.WriteLine("2. Создать множество через диапазон");

Console.WriteLine("3. Создать множество через кратность");

string option = Console.ReadLine();

switch (option)

{

case "1":

CreateSetManually();

break;

case "2":

CreateSetByRange();

break;

case "3":

CreateSetByMultiplicity();

break;

default:

Console.WriteLine("Неверная команда.");

break;

}

}

// Создание множества вручную

static void CreateSetManually()

{

Console.Write("Введите имя множества: ");

string setName = Console.ReadLine();

if (sets.ContainsKey(setName))

{

Console.WriteLine($"Множество с именем '{setName}' уже существует.");

return;

}

Console.Write("Введите элементы множества через запятую: ");

string elementsInput = Console.ReadLine();

try

{

HashSet<int> set = new HashSet<int>(elementsInput.Split(',').Select(int.Parse));

sets[setName] = set;

Console.WriteLine($"Множество '{setName}' создано.");

}

catch

{

Console.WriteLine("Ошибка ввода. Убедитесь, что вы ввели целые числа.");

}

}

// Создание множества через диапазон

static void CreateSetByRange()

{

Console.Write("Введите имя множества: ");

string setName = Console.ReadLine();

if (sets.ContainsKey(setName))

{

Console.WriteLine($"Множество с именем '{setName}' уже существует.");

return;

}

Console.Write("Введите начало диапазона: ");

int start = int.Parse(Console.ReadLine());

Console.Write("Введите конец диапазона: ");

int end = int.Parse(Console.ReadLine());

if (start > end)

{

Console.WriteLine("Начало диапазона не может быть больше конца.");

return;

}

HashSet<int> set = new HashSet<int>(Enumerable.Range(start, end - start + 1));

sets[setName] = set;

Console.WriteLine($"Множество '{setName}' создано через диапазон от {start} до {end}.");

}

// Создание множества через кратность

static void CreateSetByMultiplicity()

{

Console.Write("Введите имя множества: ");

string setName = Console.ReadLine();

if (sets.ContainsKey(setName))

{

Console.WriteLine($"Множество с именем '{setName}' уже существует.");

return;

}

Console.Write("Введите число, кратное которому будут элементы: ");

int multiple = int.Parse(Console.ReadLine());

Console.Write("Введите минимум для создания элементов: ");

int min = int.Parse(Console.ReadLine());

Console.Write("Введите максимум для создания элементов: ");

int max = int.Parse(Console.ReadLine());

if (multiple <= 0 || min > max)

{

Console.WriteLine("Минимум должен быть меньше или равен максимуму.");

return;

}

HashSet<int> set = new HashSet<int>(Enumerable.Range(min, max - min + 1).Where(x => x % multiple == 0));

sets[setName] = set;

Console.WriteLine($"Множество '{setName}' создано через кратность числа {multiple} в диапазоне от {min} до {max}.");

}

// Удаление множества

static void DeleteSet()

{

Console.Write("Введите имя множества для удаления: ");

string setName = Console.ReadLine();

if (sets.Remove(setName))

{

Console.WriteLine($"Множество '{setName}' удалено.");

}

else

{

Console.WriteLine($"Множество с именем '{setName}' не найдено.");

}

}

// Составление выражения

static void EvaluateExpression()

{

Console.WriteLine("Введите выражение (пример: (A + B) & C ; \nгде объедиение '+', пересечение '&', разность '-', сим. разность '^', дополнение '~'): ");

string expression = Console.ReadLine();

try

{

expression = expression.Replace("∪", "+")

.Replace("∩", "&")

.Replace("∆", "^")

.Replace("/", "-");

HashSet<int> result = ParseExpression(expression);

if (result != null)

{

Console.WriteLine($"Результат: {{ {string.Join(", ", result)} }}");

}

else

{

Console.WriteLine("Ошибка в выражении.");

}

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

}

// Вывод множества

static void PrintSet()

{

Console.Write("Введите имя множества для вывода: ");

string setName = Console.ReadLine();

if (sets.ContainsKey(setName))

{

HashSet<int> set = sets[setName];

Console.WriteLine($"Множество '{setName}': {{ {string.Join(", ", set)} }}");

}

else

{

Console.WriteLine($"Множество с именем '{setName}' не найдено.");

}

}

// Вывод всех множеств

static void PrintAllSets()

{

if (universe != null)

{

Console.WriteLine($"Юниверсум (U): {{ {string.Join(", ", universe)} }}");

}

if (sets.Count == 0)

{

Console.WriteLine("Нет сохранённых множеств.");

return;

}

Console.WriteLine("Список всех множеств:");

foreach (var setEntry in sets)

{

Console.WriteLine($"Множество '{setEntry.Key}': {{ {string.Join(", ", setEntry.Value)} }}");

}

}

// Парсинг выражения и выполнение операций

static HashSet<int> ParseExpression(string expression)

{

Stack<HashSet<int>> stack = new Stack<HashSet<int>>();

Stack<char> operators = new Stack<char>();

for (int i = 0; i < expression.Length; i++)

{

char c = expression[i];

if (char.IsWhiteSpace(c)) continue;

if (c == '(')

{

operators.Push(c);

}

else if (c == ')')

{

while (operators.Count > 0 && operators.Peek() != '(')

{

ApplyOperator(stack, operators.Pop());

}

operators.Pop();

}

else if (IsOperator(c))

{

while (operators.Count > 0 && Priority(operators.Peek()) >= Priority(c))

{

ApplyOperator(stack, operators.Pop());

}

operators.Push(c);

}

else if (c == '~')

{

operators.Push(c);

}

else if (char.IsLetter(c))

{

string setName = c.ToString();

if (setName == "U")

{

stack.Push(new HashSet<int>(universe));

}

else if (sets.ContainsKey(setName))

{

stack.Push(new HashSet<int>(sets[setName]));

}

else

{

Console.WriteLine($"Множество '{setName}' не найдено.");

return null;

}

}

}

while (operators.Count > 0)

{

ApplyOperator(stack, operators.Pop());

}

return stack.Count > 0 ? stack.Pop() : null;

}

// Проверка на оператор

static bool IsOperator(char c)

{

return c == '+' || c == '&' || c == '-' || c == '^' || c == '~';

}

// Приоритет операций

static int Priority(char op)

{

if (op == '~') return 4;

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

if (op == '&') return 2;

if (op == '^') return 3;

if (op == '-') return 3;

return 0;

}

// Применение оператора к верхним элементам стека

static void ApplyOperator(Stack<HashSet<int>> stack, char op)

{

HashSet<int> result;

if (op == '~')

{

var set = stack.Pop();

result = Complement(set);

}

else

{

var set2 = stack.Pop();

var set1 = stack.Pop();

switch (op)

{

case '+':

result = Union(set1, set2);

break;

case '&':

result = Intersection(set1, set2);

break;

case '-':

result = Difference(set1, set2);

break;

case '^':

result = SymmetricDifference(set1, set2);

break;

default:

throw new ArgumentException("Неизвестный оператор");

}

}

stack.Push(result);

}

// Реализация операций

static HashSet<int> Union(HashSet<int> set1, HashSet<int> set2)

{

HashSet<int> result = new HashSet<int>(set1);

foreach (var item in set2)

{

result.Add(item);

}

return result;

}

static HashSet<int> Intersection(HashSet<int> set1, HashSet<int> set2)

{

HashSet<int> result = new HashSet<int>();

foreach (var item in set1)

{

if (set2.Contains(item))

{

result.Add(item);

}

}

return result;

}

static HashSet<int> Difference(HashSet<int> set1, HashSet<int> set2)

{

HashSet<int> result = new HashSet<int>(set1);

foreach (var item in set2)

{

result.Remove(item);

}

return result;

}

static HashSet<int> SymmetricDifference(HashSet<int> set1, HashSet<int> set2)

{

HashSet<int> result = new HashSet<int>();

foreach (var item in set1)

{

if (!set2.Contains(item))

{

result.Add(item);

}

}

foreach (var item in set2)

{

if (!set1.Contains(item))

{

result.Add(item);

}

}

return result;

}

static HashSet<int> Complement(HashSet<int> set)

{

HashSet<int> result = new HashSet<int>(universe);

foreach (var item in set)

{

result.Remove(item);

}

return result;

}

}