Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра информационных систем и технологий

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Множества»

Выполнил:

Студент 1 курса 3 группы

Берёзка Вадим Сергеевич

Преподаватель: Белодед Н.И.

2024, Минск

**Содержание**

[Введение…………………………………………………………………..………………….………3](#введение)

[Основы множеств в C++………………………………………………………………….…………4](#основы)

[Операции над множествами………………………………………………...………………………5](#операции)

[Методы множеств в C++……………………………………………………………………………7](#методы)

[Продвинутые операции с множествами………………………………………………...…………8](#про_операции)

[Примеры использования множеств в C++…………………………………………………...……10](#примеры)

[Быстродействие операций над множествами………………………………………………….….12](#быстро)

[Заключение…………………………………………………………………………………………14](#закл)

**Введение**

Что будет рассмотрено в реферате:

* Общее понятие о множествах
* Значение множеств в программировании
* Определение и объявление множеств
* Инициализация множеств
* Добавление элементов в множество
* Удаление элементов из множества
* Проверка наличия элемента в множестве
* Размер множества и проверка на пустоту
* Очистка множества
* Объединение, пересечение и разность множеств
* Сравнение множеств
* Примеры использования множеств в C++
* Примеры кода для измерения быстродействия
* Обзор основных моментов работы с множествами в C++
* Важность понимания и использования множеств в программировании

Цели реферата:

* Понимание концепции множеств: Основной целью реферата является понимание основных концепций множеств и их применения в программировании.
* Изучение операций над множествами и освоение методов множеств в C++: Реферат должен помочь узнать о различных операциях, которые можно выполнять над множествами в и изучить различные методы, доступные для работы с множествами в C++.

**Общее понятие о множествах**

**Множество в C++** - это структура данных, которая хранит набор уникальных элементов в неупорядоченном порядке.

Множества реализуются с помощью контейнера **set** из стандартной библиотеки шаблонов (STL). Контейнер **set** обеспечивает быстрый поиск и вставку элементов за логарифмическое время. Он автоматически удаляет дубликаты и хранит элементы в отсортированном порядке для быстрого доступа.

Множества в C++ очень полезны для решения задач, где требуется быстрый поиск или удаление дубликатов. Они также используются для реализации некоторых алгоритмов и структур данных.

**Значение множеств в программировании**

1. **Удаление дубликатов**: Множества автоматически удаляют дубликаты, что делает их идеальным инструментом для обработки данных, где необходимо устранить повторяющиеся элементы.
2. **Быстрый поиск**: Множества обеспечивают быстрый поиск элементов. Это особенно полезно, когда работают с большими наборами данных и требуется быстро определить, присутствует ли элемент в множестве.
3. **Математические операции**: Множества позволяют выполнять стандартные математические операции, такие как объединение, пересечение и разность. Это может быть полезно во многих алгоритмах и приложениях.
4. **Порядок элементов**: В отличие от других контейнеров, таких как векторы и списки, множества не сохраняют порядок вставки элементов. Это может быть полезно в ситуациях, когда порядок элементов не имеет значения.
5. **Эффективность**: Множества обеспечивают эффективное использование памяти и времени выполнения, что делает их предпочтительным выбором для многих задач программирования.

**Основы множеств в C++**

**Определение и объявление множеств**

Множества в C++ объявляются и определяются с использованием контейнера set из стандартной библиотеки шаблонов (STL). Базовым синтаксисом для объявления множества является:

std::set<тип\_данных> имя\_множества;

тип\_данных - это тип элементов, которое будет храниться в множестве, а имя\_множества - это имя, которое присваивается множеству.

**Инициализация множеств**

Есть несколько способов инициализировать множества в C++.

1. **Инициализация при объявлении**Можно инициализировать множество при его объявлении, используя фигурные скобки {}.

std::set<int> mySet = {1, 2, 3, 4, 5};

1. **Использование метода insert**: Можно добавить элементы в множество после его объявления, используя метод **insert**.

std::set<int> mySet;

mySet.insert(1);

mySet.insert(2);

mySet.insert(3);

1. **Инициализация из другого контейнера**: Можно инициализировать множество из другого контейнера, такого как вектор или список.

std::vector<int> myVector = {1, 2, 3, 4, 5};

std::set<int> mySet(myVector.begin(), myVector.end());

**Операции над множествами**

**Добавление элементов в множество**

Метод **insert** используется для добавления элементов в множество. Если элемент, который надо добавить, уже существует в множестве, он не будет добавлен повторно, так как множества не могут содержать дубликаты.

#include <set>

#include <iostream>

#include <locale.h>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

// Объявляем множество целых чисел

set<int> mySet;

// Добавляем элементы в множество с помощью метода insert

mySet.insert(10);

mySet.insert(20);

mySet.insert(30);

// Выводим элементы множества

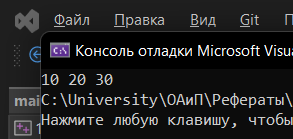
for (int element : mySet) {

cout << element << " ";

}

return 0;

}



**Удаление элементов из множества**

Метод **erase** используется для удаления элементов из множества. Можно передать ему значение, которое нужно удалить.

#include <set>

#include <iostream>

#include <locale.h>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

// Объявляем множество целых чисел и добавляем элементы

set<int> mySet = { 10, 20, 30 };

// Удаляем элемент из множества с помощью метода erase

mySet.erase(20);

// Выводим элементы множества

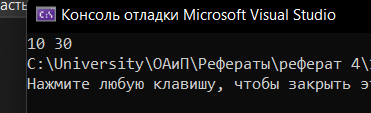
for (int element : mySet) {

cout << element << " ";

}

return 0;

}



**Проверка наличия элемента в множестве**

Метод **find** используется для поиска элемента в множестве. Он возвращает итератор к найденному элементу. Если элемент не найден, метод **find** возвращает итератор, указывающий на конец множества.

#include <set>

#include <iostream>

#include <locale.h>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

// Объявляем множество целых чисел и добавляем элементы

set<int> mySet = { 10, 20, 30 };

// Проверяем наличие элемента в множестве с помощью метода find

if (mySet.find(20) != mySet.end()) {

cout << "Элемент найден" << endl;

}

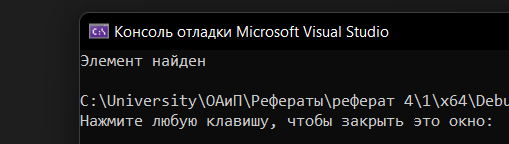
else {

cout << "Элемент не найден" << endl;

}

return 0;

}

****

**Методы множеств в C++**

**Размер множества, проверка на пустоту и очистка множества**

Метод **size()** возвращает количество элементов в множестве, что позволяет узнать его размер.

Метод **empty()** используется если надо проверить пусто ли множество, который возвращает **true**, если множество пусто, и **false** в противном случае.

Метод **clear()** используется для удаления всех элементов из множества, что делает его пустым.

#include <set>

#include <iostream>

#include <locale.h>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

// Объявляем множество целых чисел и добавляем элементы

set<int> mySet = { 10, 20, 30 };

// Выводим размер множества

cout << "Размер множества: " << mySet.size() << endl;

// Проверяем, пусто ли множество

if (mySet.empty()) {

cout << "Множество пусто" << endl;

}

else {

cout << "Множество не пусто" << endl;

}

// Очищаем множество

mySet.clear();

// Проверяем, пусто ли множество после очистки

if (mySet.empty()) {

cout << "Множество пусто после очистки" << endl;

}

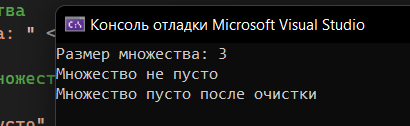
else {

cout << "Множество не пусто после очистки" << endl;

}

return 0;

}

****

**Продвинутые операции с множествами**

**Объединение, пересечение и разность множеств**

#include <set>

#include <iostream>

#include <locale.h>

#include <algorithm>

#include <iterator>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

// Объявляем два множества целых чисел

// В каждом множестве все элементы уникальны и автоматически отсортированы

set<int> set1 = { 1, 2, 3, 4, 5 };

set<int> set2 = { 4, 5, 6, 7, 8 };

// Создаем множество для результата

// Это множество будет использоваться для хранения результатов операций над множествами

set<int> resultSet;

// Выполняем операцию объединения

// Объединение множеств включает в себя все элементы из обоих множеств

set\_union(set1.begin(), set1.end(), set2.begin(), set2.end(), inserter(resultSet, resultSet.begin()));

cout << "Объединение множеств: ";

for (int element : resultSet) {

cout << element << " ";

}

cout << endl;

// Очищаем resultSet и выполняем операцию пересечения

// Пересечение множеств включает только те элементы, которые присутствуют в обоих множествах

resultSet.clear();

set\_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(), inserter(resultSet, resultSet.begin()));

cout << "Пересечение множеств: ";

for (int element : resultSet) {

cout << element << " ";

}

cout << endl;

// Очищаем resultSet и выполняем операцию разности

// Разность множеств включает элементы, которые присутствуют в первом множестве, но отсутствуют во втором

resultSet.clear();

set\_difference(set1.begin(), set1.end(), set2.begin(), set2.end(), inserter(resultSet, resultSet.begin()));

cout << "Разность множеств (set1 - set2): ";

for (int element : resultSet) {

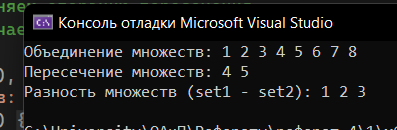
cout << element << " ";

}

cout << endl;

return 0;

}

****

**Сравнение множеств**

#include <set>

#include <iostream>

#include <locale.h>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

// Объявляем два множества целых чисел

set<int> set1 = { 1, 2, 3, 4, 6 };

set<int> set2 = { 1, 2, 3, 4, 5 };

// Сравниваем два множества

if (set1 == set2) {

// Если множества равны, выводим соответствующее сообщение

cout << "Множества равны" << endl;

}

else {

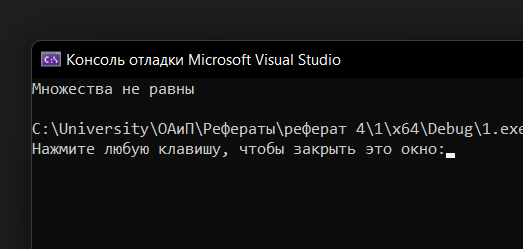
// Если множества не равны, выводим соответствующее сообщение

cout << "Множества не равны" << endl;

}

return 0;

}



**Примеры использования множеств в C++**

**1. Удаление дубликатов из вектора**

Множества в C++ автоматически удаляют дубликаты, что делает их идеальным инструментом для удаления дубликатов из вектора.

#include <set>

#include <vector>

#include <iostream>

#include <locale.h>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

// Объявляем вектор с дубликатами

vector<int> myVector = { 1, 2, 2, 3, 3, 3, 4, 4, 4, 4 };

// Создаем множество из вектора, что автоматически удаляет дубликаты

set<int> mySet(myVector.begin(), myVector.end());

// Выводим элементы множества

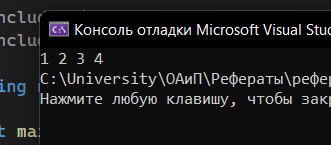
for (int element : mySet) {

cout << element << " ";

}

return 0;

}



**2. Проверка на подмножество**

Можно проверить, является ли одно множество подмножеством другого, используя методы **set\_intersection** и **size**.

#include <set>

#include <iostream>

#include <locale.h>

#include <algorithm>

#include <iterator>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

// Объявляем два множества

set<int> set1 = { 1, 2, 3, 4, 5 };

set<int> set2 = { 1, 2, 3 };

// Создаем множество для результата пересечения

set<int> intersection;

// Находим пересечение двух множеств

set\_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(), inserter(intersection, intersection.begin()));

// Если размер пересечения равен размеру второго множества, то второе множество является подмножеством первого

if (intersection.size() == set2.size()) {

cout << "set2 является подмножеством set1" << endl;

}

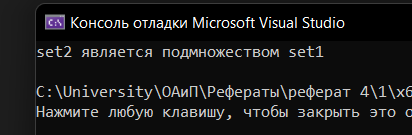
else {

cout << "set2 не является подмножеством set1" << endl;

}

return 0;

}



**Быстродействие операций над множествами**

**Измерение времени выполнения операции вставки в множество**

#include <set>

#include <iostream>

#include <locale.h>

#include <chrono>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

// Объявляем множество целых чисел

set<int> mySet;

// Засекаем время начала операции

auto start = chrono::high\_resolution\_clock::now();

// Добавляем элементы в множество

for (int i = 0; i < 1000000; ++i) {

mySet.insert(i);

}

// Засекаем время окончания операции

auto end = chrono::high\_resolution\_clock::now();

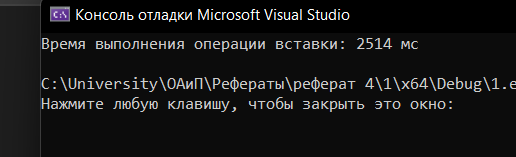
// Вычисляем и выводим затраченное время

auto duration = chrono::duration\_cast<chrono::milliseconds>(end - start);

cout << "Время выполнения операции вставки: " << duration.count() << " мс" << endl;

return 0;

}

****

**Измерение времени выполнения операции удаления в множестве**

#include <set>

#include <iostream>

#include <locale.h>

#include <chrono>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

// Объявляем множество целых чисел и добавляем элементы

set<int> mySet;

for (int i = 0; i < 1000000; ++i) {

mySet.insert(i);

}

// Засекаем время начала операции

auto start = chrono::high\_resolution\_clock::now();

// Удаляем элементы из множества

mySet.erase(500000);

// Засекаем время окончания операции

auto end = chrono::high\_resolution\_clock::now();

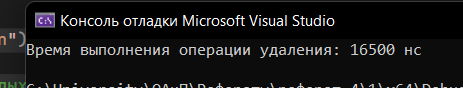
// Вычисляем и выводим затраченное время

auto duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

cout << "Время выполнения операции удаления: " << duration.count() << " нс" << endl;

return 0;

}

****

**Измерение времени выполнения операции поиска элемента в множестве**

#include <set>

#include <iostream>

#include <locale.h>

#include <chrono>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

// Объявляем множество целых чисел и добавляем элементы

set<int> mySet;

for (int i = 0; i < 1000000; ++i) {

mySet.insert(i);

}

// Засекаем время начала операции

auto start = chrono::high\_resolution\_clock::now();

// Ищем элемент в множестве

mySet.find(500000);

// Засекаем время окончания операции

auto end = chrono::high\_resolution\_clock::now();

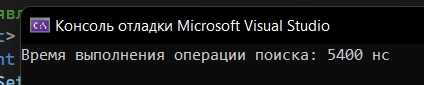
// Вычисляем и выводим затраченное время

auto duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

cout << "Время выполнения операции поиска: " << duration.count() << " нс" << endl;

return 0;

}



**Заключение**

Множества в C++ представляют собой мощный и гибкий инструмент, который может быть использован для решения широкого спектра задач. Они обеспечивают быстрый доступ к данным, автоматическую сортировку и удаление дубликатов, что делает их идеальным выбором для многих сценариев программирования.

Я рассмотрел и представил в реферате основные операции над множествами, такие как вставка, удаление и поиск элементов, а также более продвинутые операции, такие как объединение, пересечение и разность множеств. И дополнительно измерил быстродейсвтие выполнения операций над множествами.