# ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ «ВОЛГОГРАДСКИЙ СОЦИАЛЬНО – ПЕДАГОГИЧЕСКИЙ КОЛЛЕДЖ»

Кафедра информационных технологий обучения  
Специальность 09.02.07 Информационные системы и программирование

**Курсовая работа**

**«Абстрактные типы данных»**

**Студента группы: 21 «Д»**

Малашин Андрей Андреевич

**Специальность:** 09.02.07 «Информационные системы и

программирование»

**Руководитель:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/** Бетиров А. М.

**Работа допущена к защите:**

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(роспись руководителя)

Волгоград, 2023 г.

**Оглавление**

[Содержание 2](#_3znysh7)

[Введение: 3](#_30j0zll)

[Глава 1.Теоритическая часть 5](#_2et92p0)

[1.1Теоретические основы 5](#_tyjcwt)

[1.2Принципы работы абстрактных типов данных 9](#_3dy6vkm)

[1.3Классификация абстрактных типов данных 11](#_1t3h5sf)

[Глава 2.Практичсекая часть 12](#_4d34og8)

[2.1Реализация абстрактных типов данных на языке С# 12](#_2s8eyo1)

[2.2Применение абстрактных типов данных в практических задачах 15](#_17dp8vu)

[2.3 Сравнение эффективности применения абстрактных типов данных с другими видами данных 16](#_3rdcrjn)

Заключение: 17

# 

# **Введение**:

Абстрактные типы данных (Abstract Data Types, ADT) - это типы данных в программировании, которые определяются с точки зрения поведения и операций, которые можно выполнить с этим типом данных, а не с точки зрения его внутренней реализации и представления. Такой подход позволяет создавать удобные и гибкие абстракции, которые могут быть использованы в различных приложениях с минимальными изменениями. Примерами Абстрактных типов данных являются стек, очередь, хеш-таблица и множество.

Актуальность исследования абстрактных типов данных (АТД) является фундаментальной концепцией в программировании, которая позволяет разделять описание структуры данных и алгоритмы, работающие с этими данными. Абстрактные типы данных являются абстрактным понятием, описывающим, как данные могут быть организованы и как к ним можно обращаться, не зависимо от фактической реализации.

**Цель курсовой работы:** Изучить абстрактные типы данных и их применение в программировании.

**Задачи курсовой работы:**

1. Изучить теорию абстрактных типов данных;
2. Исследовать поведение абстрактных типов данных в разных ситуациях ;
3. Показать на практике примеры применения абстрактных типов данных;

**Методы исследования** включают в себя теоретическую и практическую часть.

**К теоретической части относится:**

1. Определение и описание базовых абстрактных типов данных, таких как стеки, очереди, списки, деревья, хэш-таблицы и графы.
2. Рассмотрение методов доступа к элементам абстрактных типов данных, таких как вставка, удаление, поиск и изменение элементов.
3. Анализ сложности операций на данных абстрактных типов, вычисление временных и пространственных характеристик алгоритмов работы с данными типами.

**К практической относятся следующее:**

1. Реализовать стек на C# с методами добавления, удаления, проверки пустоты и определения размера стека.
2. Реализовать иерархию классов на C# с абстрактным классом "Фигура" и конкретными классами для круга, квадрата и прямоугольника с переопределением методов расчета площади и периметра.
3. Написать программу на C#, использующую абстрактный тип данных "очередь"

**Практическая значимость:**

Абстрактные типы данных помогают получить более гибкий и эффективный код, создать чистый и документированный код, оптимизировать производительность, управлять изменениями в программе и добавлять новые функции.

# 

# **ГЛАВА 1.ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

# **1.1 Теоретические основы**

Абстрактные типы данных (АТД) - это специализированные типы данных, которые предоставляют некоторый интерфейс, определяющий операции, которые могут быть выполнены с данными этого типа. АТД определяются в терминах их свойств, а не в терминах их реализации. Они предоставляют абстракцию связанную с конкретными методами реализации, что означает, что конкретные подробности реализации скрыты от пользователя АТД. Это позволяет скрыть сложные детали реализации и упростить процесс разработки программного обеспечения. Некоторые примеры АТД: стек, очередь, массив, список, дерево и т.д.

К Абстрактным типам данных относиться:

1. Стек- это структура данных, которая позволяет добавлять и удалять элементы только с одного конца - вершины стека. Это означает, что последний добавленный элемент всегда будет первым удаленным (Last-In-First-Out, LIFO).

Стек используется в различных задачах, таких как:

Рекурсия: когда функция вызывает саму себя, используется стек, чтобы хранить информацию о вызываемых функциях и их параметрах.

- Обход дерева: при обходе дерева в глубину используется стек для хранения узлов, которые необходимо обойти.

- Вычисление выражений: скобки в математических выражениях могут быть обработаны с помощью стека.

Операции, которые можно выполнить со стеком, включают в себя:

1. Стек в программировании - это структура данных, которая позволяет добавлять и удалять элементы только с одного конца - вершины стека. Это означает, что последний добавленный элемент всегда будет первым удаленным (Last-In-First-Out, LIFO).

Стек используется в различных задачах, таких как:

- Рекурсия: когда функция вызывает саму себя, используется стек, чтобы хранить информацию о вызываемых функциях и их параметрах.

- Обход дерева: при обходе дерева в глубину используется стек для хранения узлов, которые необходимо обойти.

- Вычисление выражений: скобки в математических выражениях могут быть обработаны с помощью стека.

Операции, которые можно выполнить со стеком, включают в себя:

- Push (добавление элемента на вершину стека)

- Pop (удаление элемента с вершины стека)

1. Очередь – это структура данных, которая хранит элементы в определенном порядке (обычно первым в очереди приходит элемент, который был добавлен первым). Это аналогия с очередью людей, стоящих в очереди на кассу, например.

Очередь может быть реализована на основе массива или связного списка. Она поддерживает две основные операции: добавление элемента в конец очереди (enqueue) и удаление элемента из начала очереди (dequeue).

Очередь широко используется в программировании, например, для буферизации ввода-вывода в операционных системах, для реализации алгоритмов обхода графов в алгоритмах поиска кратчайшего пути и т.д.

1. Список (list) в программировании - это структура данных, которая позволяет хранить упорядоченный набор элементов. Элементы списка могут быть любого типа данных (числа, строки, другие списки и т.д.). Основные операции с списками включают добавление элемента, удаление элемента, получение элемента по индексу, изменение элемента по индексу, сортировку списка, нахождение длины списка и т.д. В разных языках программирования существуют различные реализации списков (например, в Python это встроенный тип данных, в С++ это массивы и векторы, в Java это класс ArrayList и т.д.). Списки являются важным инструментом для работы с коллекциями данных в программировании.
2. Множество (set) в программировании - это структура данных, которая содержит уникальные элементы в произвольном порядке. Она поддерживает операции добавления, удаления, поиска элемента и проверки на вхождение элемента. Множество является одним из базовых типов данных во многих языках программирования, таких как Python, Java, C++, C# и других. В языке Python, например, множество определяется при помощи фигурных скобок {} или функции set(). Обычно множество используется для проверки уникальности элементов, фильтрации повторяющихся значений и реализации алгоритмов, связанных с графами и математикой.
3. Граф в программировании - это структура данных, которая представляет собой множество вершин и ребер, связывающих эти вершины. Граф может быть ориентированным, когда направление ребер имеет значение, или неориентированным, когда ребра не имеют направления.

Графы используются в программировании для моделирования многих задач. Например, они могут использоваться для моделирования сетей сообщений, графических пользовательских интерфейсов, графиков вызовов функций, компьютерных игр и т.д.

В программировании графы могут храниться в различных структурах данных - списки смежности, матрицы смежности, списки ребер и т.д. Различные алгоритмы и операции могут быть применены к графам, такие как поиск в глубину и ширину, кратчайший путь, минимальное остовное дерево и т.д.

1. Дерево в программировании - это структура данных, которая состоит из узлов, связанных друг с другом в виде ветвей. Каждый узел может иметь несколько потомков (дочерних узлов), но только одного родителя. Узлы в дереве бывают двух типов: корневой и листовой. Корневой узел не имеет родителя, а листовой - не имеет потомков.

Деревья используются в программировании для организации данных, которые имеют иерархическую структуру, например, файловой системе операционных систем, организации баз данных, построении деревьев решений в машинном обучении, построения графических интерфейсов и т.д.

Одной из важных операций над деревом является обход его элементов. Существует три основных способа обхода дерева: прямой (pre-order), симметричный (in-order) и обратный (post-order). Прямой обход дерева начинается с корневого узла, затем обходятся левое и правое поддерево. В симметричном обходе узлы обходятся в порядке левое поддерево, корень, правое поддерево. В обратном обходе сначала обходятся левое и правое поддерево, затем корень.

Деревья могут быть реализованы на разных языках программирования с использованием разных структур данных, например, списков, указателей и т.д. Одним из известных способов реализации деревьев является использование классов и объектов в объектно-ориентированном программировании.

1. Хеш таблица (hash table или hash map) – это структура данных, которая позволяет хранить пары ключ-значение и обеспечивает быстрый доступ к значению по его ключу. Она использует хеш-функцию для преобразования ключа в индекс таблицы, по которому сохраняется значение.

Преимущества хеш таблицы:

- быстрый доступ к элементам;

- эффективное добавление и удаление пар ключ-значение;

- возможность хранения большого объема данных.

Хеш таблицы широко применяются в программировании, например, для реализации кэша, баз данных, языковых моделей и других приложений. Однако, при выборе хеш функции необходимо учитывать ее коллизионность, т.е. вероятность того, что разные ключи могут быть преобразованы в один и тот же индекс таблицы. Коллизии могут вызывать проблемы с производительностью и достоверностью хранения данных, поэтому для минимизации их количества используются различные методы, например, метод цепочек или метод открытой адресации.

Реализация абстрактных типов данных может происходить как на разных языках программирования так и на разных уровнях программирования. Например, одним из способов описания АТД являются интерфейсы, которые определяют список методов и операций, доступных для работы с данными. Также, для реализации АТД может использоваться классы, структуры, перечисления и другие средства, которые позволяют описать хранимые данные и способы их обработки.

# **1.2 Принципы работы абстрактных типов данных**

Принципы работы абстрактных типов данных основаны на инкапсуляции, что позволяет скрыть детали реализации и предоставить необходимые операции для работы с данными.

Первым принципом является принцип абстракции. Он заключается в представлении АДТ только через его основные операции, без указания деталей его реализации. Принцип абстракции помогает упростить работу с АДТ и избежать сложности в работе с деталями его реализации.

Вторым принципом является принцип наследования. Он заключается в использовании уже существующих АДТ для реализации новых ADT. Принцип наследования помогает упростить разработку программных продуктов и повторно использовать существующий код.

Третьим принципом является принцип полиморфизма. Он заключается в возможности использования одного и того же АДТ для работы с различными типами данных. Принцип полиморфизма позволяет унифицировать работу с данными и повышает гибкость программы.

Таким образом, принципы работы абстрактных типов данных основаны на принципах инкапсуляции, наследования и полиморфизма. Эти принципы позволяют упростить разработку программных продуктов и повысить их гибкость и надежность.

Принципы работы абстрактных типов данных также помогают разработчикам сконцентрироваться на бизнес-логике и функциональности приложения, вместо того чтобы тратить время на детали реализации. Объектно-ориентированный подход АДТ также облегчает командную работу и повышает читаемость кода, что делает его более понятным и доступным для других разработчиков.

АДТ также является основой для разработки библиотек и фреймворков. Благодаря этому, АДТ может использоваться для создания множества приложений, что делает его очень мощным инструментом для разработчиков.

Важно отметить, что принципы работы абстрактных типов данных не являются новыми, они существуют уже давно и были разработаны еще в начале 1970-х годов. Однако, они до сих пор остаются актуальными и полезными для различных типов приложений и программного обеспечения.

Таким образом, использование принципов работы абстрактных типов данных является важным шагом для создания качественного и эффективного программного обеспечения. АДТ помогает упростить разработку, усовершенствовать тестирование и повысить надежность и гибкость программы в целом.

**1.3 Классификация абстрактных типов данных**

Абстрактные типы данных (АТД) являются важной составной частью программирования и используются для описания объектов и операций, которые могут быть выполнены над ними. Существует несколько способов классификации АТД:

Простые и сложные АТД. Простые АТД включают в себя базовые типы данных, такие как целочисленные, вещественные числа и символы, а сложные АТД представляют собой упорядоченные или неупорядоченные наборы простых АТД.

1. АТД с понятием совместного использования и без понятия совместного использования. АТД с понятием совместного использования позволяют нескольким объектам использовать один объект данных одновременно. АТД без понятия совместного использования могут использовать только один объект в один момент времени.
2. АТД с фиксированной и изменяемой длиной. АТД с фиксированной длиной имеют фиксированный размер, в то время как АТД с изменяемой длиной могут изменять свой размер в процессе выполнения программы.
3. АТД с одной и несколькими структурами. АТД с одной структурой имеют только один тип структуры данных, в то время как АТД с несколькими структурами могут иметь несколько типов структур данных.
4. АТД с определенной и неопределенной структурой. АТД с определенной структурой имеют заранее определенный способ организации данных, а АТД с неопределенной структурой могут иметь любую организацию данных без заранее известного шаблона.
5. АТД с ограниченной и неограниченной структурой. АТД с ограниченной структурой имеют предельный размер, который не может быть превышен, а АТД с неограниченной структурой могут иметь любой размер, в том числе и бесконечный.

При выборе АТД для решения задачи в программировании, важно учитывать его классификацию в зависимости от задачи, которую необходимо решить.

**Вывод:**

Абстрактные типы данных позволяют определить сущности и операции, которые могут быть выполнены над ними, но не определяют, как эти операции будут реализованы. Они являются основой для построения различных структур данных, которые используются для хранения, обработки и управления информацией. Важно выбирать правильный тип данных в зависимости от задачи, чтобы обеспечить эффективность и оптимальность алгоритмов и программ.

# **ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ**

**2.1 Реализация абстрактных типов данных на языке С#**

using System;

using System.Collections;

class Program

{

static void Main(string[] args)

{

Queue queue = new Queue();

Random rnd = new Random();

// заполним очередь случайными числами

for (int i = 0; i < 10; i++)

{

int num = rnd.Next(1, 100);

queue.Enqueue(num);

}

// выведем на экран все элементы очереди

Console.WriteLine("Элементы очереди:");

foreach (int num in queue)

{

Console.Write(num + " ");

}

// извлечем два элемента из очереди

int first = (int)queue.Dequeue();

int second = (int)queue.Dequeue();

Console.WriteLine("\n\nПервый извлеченный элемент: " + first);

Console.WriteLine("Второй извлеченный элемент: " + second);

// выведем на экран все элементы очереди после извлечения двух элементов

Console.WriteLine("\nЭлементы очереди после извлечения двух элементов:");

foreach (int num in queue)

{

Console.Write(num + " ");

}

// добавим еще один элемент в очередь

int newNum = rnd.Next(1, 100);

queue.Enqueue(newNum);

Console.WriteLine("\n\nНовый элемент, добавленный в очередь: " + newNum);

// выведем на экран все элементы очереди после добавления нового элемента

Console.WriteLine("\nЭлементы очереди после добавления нового элемента:");

foreach (int num in queue)

{

Console.Write(num + " ");

}

Console.ReadKey();

}

}

В данной программе создается очередь queue, которая заполняется случайными числами при помощи метода Enqueue. Затем выводятся все элементы очереди на экран, используя цикл foreach и метод Peek.

Далее из очереди извлекаются два элемента методом Dequeue и выводятся на экран. Затем выводятся все элементы очереди после удаления двух элементов.

Наконец, в очередь добавляется новый элемент методом Enqueue, который также выводится на экран, а затем выводятся все элементы очереди после добавления нового элемента.

Таким образом, работа программы показывает, что очередь работает по принципу "первым пришел - первым ушел" и что элементы могут быть извлечены из очереди только в порядке их добавления.

**2.2 Применение абстрактных типов данных в практических задачах**

Абстрактные типы данных (АТД) используются в решении многих практических задач в программировании. Наиболее распространенными примерами применения АТД являются:

1. Стеки: АТД стек используется в программировании для решения задач, связанных с рекурсией, обходом деревьев, поиска в глубину, вычислений с постфиксной записью выражений и т.д. Стеки используются в различных языках программирования, таких как Си, C++, Java, Python и других.

2. Очереди: АТД очередь используется для решения задач, связанных с обработкой сообщений, задач, запросов и т.д. Очереди также используются в различных языках программирования и базах данных.

3. Списки: АТД список используется для хранения и обработки данных в виде последовательности элементов. Списки могут использоваться для хранения информации о контактах, товарах, друзьях и т.д.

4. Деревья: АТД дерево используется для хранения и обработки иерархических данных. Деревья используются в базах данных, для построения алгоритмов поиска, сортировки, обхода и т.д.

5. Хеш-таблицы: АТД хеш-таблица используется для быстрого поиска и вставки элементов. Хеш-таблицы используются в базах данных, поисковых системах, библиотеках и других приложениях.

6. Графы: АТД граф используется для представления и обработки связей между элементами. Графы используются в расписаниях, социальных сетях, картографии и т.д.

Кроме того, АТД можно использовать при решении задач, связанных с математическими структурами, подсчетом, алгоритмами машинного обучения и другими областями.

**2.3 Сравнение эффективности применения абстрактные типы данных с другими структурами данных**

Сравнение эффективности АТД с другими структурами данных зависит от конкретных условий использования. Например, список может быть быстрее вставки и удаления элементов, чем массив, но медленнее доступа к конкретному элементу. Также, АТД могут быть не оптимальны для хранения больших объемов данных, где более простые структуры данных могут справляться лучше.

**Вывод:**

В целом, использование АТД при проектировании программного обеспечения может увеличить эффективность и качество кода, но выбор конкретной структуры данных должен зависеть от требований к производительности и функциональности программы.

В данной главе были рассмотрены основы работы с абстрактными типами данных (АТД), такими как стек, очередь, список, дерево, хеш-таблица и граф. Были описаны основные действия, которые можно выполнять с каждым типом данных, и приведены примеры кода на языке программирования C#.

Также было проанализировано применение АТД в практических задачах и представлены примеры использования каждого типа данных в различных областях программирования.

Наконец, было отмечено, что выбор структуры данных для конкретной задачи зависит от ее требований к производительности и функциональности, и использование АТД может увеличить эффективность и качество кода в программировании.

**­Заключение:**

В заключении курсовой работы на тему абстрактные типы данных, можно сделать вывод, что данные типы представляют собой важное средство для организации информации в программировании. Они позволяют упростить процесс работы с данными и повысить производительность программы.

Кроме того, использование абстрактных типов данных дает возможность лучшего контроля за данными, так как они могут быть скрыты от пользователя и обеспечивают более безопасную обработку.

Таким образом, абстрактные типы данных имеют важное значение в программировании и являются необходимым инструментом для создания эффективных и безопасных программных решений.