

**ПРОФЕСИОНАЛНА ГИМНАЗИЯ**

**ПО ЕЛЕКТРОТЕХНИКА И ЕЛЕКТРОНИКА “АПОСТОЛ АРНАУДОВ”**

гр. Русе, ул. “Потсдам” № 3; п.к. 7005, тел. 082/84-60-96; e-mail: [info-1806301@edu.mon.bg](mailto:info-1806301@edu.mon.bg)

Д И П Л О М Е Н П Р О Е К Т

**ЗА ПРИДОБИВАНЕ НА**

**ТРЕТА СТЕПЕН НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ**

**по професия код 481020 „Системен програмист“**

**специалност код 4810201 „Системно програмиране“**

**ТЕМА:**

**Механизъм за управление на данни чрез интеграция на CRUD операциите в C# колекциите**

**Ученик**: Константин Христианов Ганев

**Ръководител-консултант:** Милена Дамесова-Христова

гр. Русе

2024

**ГЛАВА ПЪРВА**

**Структури от данни в C# и CRUD операции**

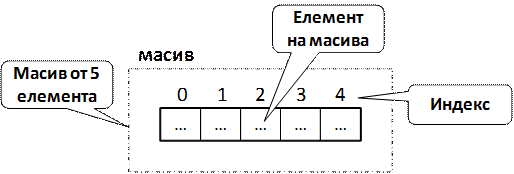
1. **Въведение в C#**

C# е език за програмиране, разработен от Microsoft. Той е създаден през 2000 г. и е част от Microsoft's .NET платформа. Езикът е предназначен за разработка на разнообразни приложения, включително уеб, настолни, мобилни и облачни приложения. Ето някои ключови характеристики и концепции на C#:

* Обектно-ориентиран програмен език: C# е изцяло обектно-ориентиран, което означава, че програмите се структурират чрез обекти, които съдържат данни и методи за работа с тези данни.
* Силно типизиран език: Променливите в C# трябва да бъдат декларирани с определен тип преди да бъдат използвани, и компилаторът извършва строга проверка на типовете по време на компилацията.
* Управление на паметта: C# използва автоматично управление на паметта чрез Garbage Collector, който автоматично освобождава ресурсите, които вече не се използват.
* Интеграция с .NET Framework и .NET Core: C# е част от .NET платформата, която предоставя обширен набор от библиотеки и инструменти за разработка на различни видове приложения.
* Събитиен програмен модел: В C# събитията позволяват на програмите да реагират на събития като бутонове, мишка, клавишни комбинации и др.
* Многонишково програмиране: C# поддържа многонишково програмиране чрез използването на класовете от пространството на имена System.Threading.
* LINQ (Language Integrated Query): Предоставя декларативен начин за работа с данни, който е интегриран директно в езика.
* ASP.NET: C# се използва широко за уеб разработка чрез технологиите на ASP.NET, включително ASP.NET MVC и ASP.NET Core.

1. **Различните структури от данни в C#**
   1. **Масиви в C#**

Масивите в езика C# представляват съвкупност от няколко еднотипни променливи. Те се наричат елементи на масива. Ето какво представляват масивите:



Фигура 1: Масив

Характеристики на масивите:

* Масивът може да бъде **едномерен**, **многомерен** или **масив от масиви**.
* Базирани са на **нулево индексиране** – това означава, че в масив с N елемента, първият елемент ще е с индекс нула, а последният с индекс N-1.
* Елементите на масива могат да бъдат от всякакъв тип, включително от тип масив.
* Стойността по подразбиране на елементите от числен тип е *нула*, за референтните типове е *null*, а за булевите типове е *false*. При масив от масиви елементите са от референтен тип и по подразбиране са *null*.
* Редът на елементите и дължината на масива са фиксирани.

A computer screen with numbers and letters

Description automatically generated

Фигура 2: Код за масиви

* 1. **Двумерни и многомерни масиви в C#**

В C#, многомерните масиви са структури от данни с две или повече измерения. Най-често срещаните са двумерните масиви, но е възможно да създадете и масиви с по-голям брой измерения. Ето и някои предимства на многомерните и двумерните масиви:

* **Таблична организация:** Както двумерните, така и многомерните масиви предоставят таблична организация на данните, което улеснява работата с таблични структури.
* **Ефективен достъп до елементите:** Масивите предоставят константно време за достъп до елементите с помощта на индексацията, което ги прави ефективни при работа с данни в различни измерения.
* **Подходящи за математически операции:** Когато имаме задачи, свързани с математика и линейна алгебра, многомерните масиви са полезни за представяне на триизмерни или четириизмерни структури от данни.
  + 1. **Двумерен масив**

Двумерният масив представлява таблица с редове и колони. За да създадете двумерен масив с 3 реда и 4 колони в C#, използвате следния синтаксис:



Фигура 3: Двумерен масив

* + 1. **Многомерен масив**

Можете да създавате масиви с повече от две измерения, като промените броя на индексите при деклариране и инициализация. Ето пример за тримерен масив с 2 “слоя“, всеки с по 3 реда и 4 колони:



Фигура 4: Триизмерен масив

* 1. **List в C#**

В C#, List представлява динамичен масив, който може динамично да променя своя размер по време на изпълнение на програмата. Той е част от пространството от имена System.Collections.Generic. List предоставя разнообразие от методи за работа с данните, като Add, Remove, IndexOf, Contains и много други. Ето някои основни характеристики на List в C#:

* **Динамичен размер:** List автоматично увеличава своя размер при добавяне на елементи.
* **Генеричен тип (Generics):** List използва генерични типове за гарантиране на типовата безопасност. Това означава, че по време на създаване на списъка задавате типа данни, който ще се съхранява в него.
* **Индексиране:** Елементите в List се индексират от 0, и можете да получите достъп до елемент по неговия индекс.

Пример за създаване на List и добавяне на елементи:

A computer screen with white text

Description automatically generated

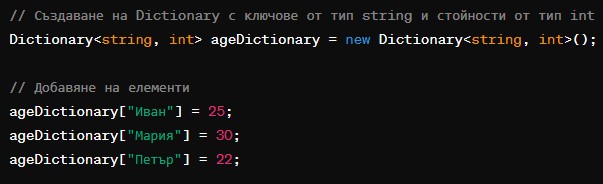
Фигура 5: List

* 1. **Dictionary в C#**

В C#, Dictionary е структура от данни, която представлява колекция от ключ-стойност. Тя позволява бързо търсене на стойности (елементи) по ключ. Dictionary също така е част от пространството от имена System.Collections.Generic. Ето някои основни характеристики на Dictionary:

* **Ключ-стойност:** Всяка стойност в Dictionary е свързана с уникален ключ.
* **Бързо търсене:** Dictionary осигурява бърз достъп до стойности по ключ.
* **Обобщения (Generics):** Dictionary използва генерични типове за гарантиране на типовата безопасност.

Пример за създаване и добавяне на елементи в Dictionary:



Фигура 6: Dictionary

В този пример се създава Dictionary, където ключовете са низове (string), а стойностите са цели числа (int).

* 1. **Stack в C#**

В C#, Stack е структура от данни, която представлява стек – колекция от елементи, където последният добавен елемент е първият, който може да бъде извлечен. Stack е част от пространството от имена System.Collections.Generic. Ето някои основни характеристики на Stack:

* **Last In, First Out (LIFO):** Stack следва принципа "последен влиза, първи излиза", където последният добавен елемент е първият, който може да бъде извлечен.
* **Обобщения (Generics):** Stack използва генерични типове за гарантиране на типовата безопасност.
* **Методи за манипулация:** Stack предоставя методи като Push за добавяне на елемент, Pop за извличане на последния добавен елемент и Peek за връщане на стойността на последния елемент без да го извлича.

Пример за създаване, добавяне и извличане на последния елемент:

A computer screen with text and numbers

Description automatically generated

Фигура 7: Stack

В този пример се създава Stack от цели числа, елементи се добавят с метода Push и се извлича последния добавен елемент с метода Pop.

* 1. **Queue в C#**

Опашката в програмирането е вид абстрактна структура от данни и е представител на абстрактните типове данни (АТД). Опашките спадат към линейните (списъчни) структури от данни, заедно със списъците и стековете. Опашката представлява крайно, линейно множество от елементи, при което елементи се добавят само най-отзад (enqueue) и се извличат само най-отпред (dequeue). Абстрактната структура опашка изпълнява условието „първият влязъл първи излиза“ (FIFO: First-In-First-Out). Това означава, че след като е добавен един елемент в края на опашката, той ще може да бъде извлечен (премахнат) единствено след като бъдат премахнати всички елементи преди него в реда, в който са добавени.

Структурата опашка и поведението на нейните елементи произхождат от ежедневната човешка дейност. Например опашка от хора, чакащи на каса за билети. Опашката има начало и край. Новодошлите хора застават последни на опашката и изчакват докато постепенно се придвижат към началото. Когато стигнат до самото начало на опашката си купуват билет и напускат опашката. По този начин опашката изпълнява функцията на буфер.

Пример за създаване, добавяне и извличане на първия добавен елемент:

A computer screen with white text

Description automatically generated

Фигура 8: Queue

В този пример се създава Queue от цели числа, елементи се добавят с метода Enqueue и се извлича първия добавен елемент с метода Dequeue.