1. **Пояснити схему виконання програми в .NET.**

Основні етапи виконання програми в .NET включають наступні кроки:

* Компіляція: Вихідний код програми, написаний на мові C#, компілюється в інтермедіатний мовний код (IL - Intermediate Language), відомий також як MSIL (Microsoft Intermediate Language) або CIL (Common Intermediate Language). Цей інтермедіатний код не прив'язаний до конкретної архітектури чи платформи.
* Інтерпретація або JIT-компіляція: Під час виконання програми інтермедіатний код перетворюється в машинний код за допомогою JIT-компілятора (Just-In-Time Compiler). Цей процес дозволяє оптимізувати виконання програми на певній апаратній платформі, забезпечуючи оптимальну продуктивність.
* Виконання коду: Після перетворення в машинний код програма виконується на віртуальній машині .NET (Common Language Runtime - CLR). CLR відповідає за керування виконанням програм, а також збиранням сміття, що дозволяє автоматично вивільняти пам'ять від об'єктів, які більше не використовуються.
* Взаємодія зі збірниками і класами .NET: Під час виконання програма взаємодіє з бібліотеками класів .NET (наприклад, класи у просторах імен System) та зовнішніми збірками, які використовуються для різноманітних завдань, таких як введення/виведення, мережева взаємодія, робота з базами даних та багато іншого.
* Обробка винятків та виняткових ситуацій: В разі виникнення помилок або виняткових ситуацій .NET Framework надає механізми для їх обробки, що дозволяє забезпечити стабільність та безпеку програми.

1. **Що таке типи даних? Які типи даних є в мові С#? Як вони сполучаються з варіантами збереження інформації в оперативному запам’ятовуючому пристрої? Розкрити поняття «перетворення типів». Яким воно може бути? Які типи даних не можуть бути перетворені неявно ні в які інші типи?**

Типи даних - це важливий аспект будь-якої мови програмування, який визначає вид і обсяг даних, які можуть бути збережені та оброблені в програмі. Вони визначають обмеження, правила та операції, які можуть бути виконані над цими даними. У мові C# існує кілька типів даних, які можна класифікувати наступним чином:

* Типи значень (Value Types): Ці типи даних зберігають фактичне значення і зазвичай знаходяться в стеку. Деякі приклади цих типів: цілі числа (int, long), числа з плаваючою точкою (float, double), символи (char), булеві значення (bool) тощо.
* Типи посилань (Reference Types): Ці типи даних зберігають посилання на об'єкт, який знаходиться в купі (heap). До цих типів відносяться класи, інтерфейси, делегати, а також масиви.
* Типи перечислень (Enumerations): Це спеціальні типи, які дозволяють створювати набір іменованих констант. Вони допомагають зробити код більш зрозумілим та зменшити кількість помилок, пов'язаних з магічними числовими значеннями.
* Типи значення Nullable (Nullable Value Types): Це розширення для типів значень, яке дозволяє їм приймати значення null.
* Типи даних рядків (String Data Type): Це особливий тип, який використовується для збереження послідовностей символів.

Перетворення типів (Type Casting) в мові C# - це процес перетворення одного типу даних на інший. Це може бути неявне або явне. Неявне перетворення відбувається автоматично для сумісних типів, тоді як явне перетворення вимагає явного вказівника типу. Наприклад, ви можете перетворити ціле число в число з плаваючою точкою явно за допомогою (float), якщо ви впевнені, що втрата точності не є проблемою.

Деякі типи даних не можуть бути перетворені неявно в інші типи через потенційну втрату даних або некоректність перетворення. Наприклад, не можна неявно перетворити типи даних з більшою точністю, такі як double, в менш точні типи, такі як int, оскільки це може призвести до втрати дробової частини даних.

1. **Які умовні конструкції є в мові С#?**

Основні умовні конструкції в мові C# включають:

Оператор if-else: Цей оператор дозволяє виконувати певний блок коду, якщо задана умова є істинною, і інший блок коду, якщо умова є хибною.

if (условие)

{

// виконується, якщо умова істинна

}

else

{

// виконується, якщо умова хибна

}

Оператор switch: Цей оператор дозволяє вибрати один з багатьох шляхів виконання коду, залежно від значення змінної.

switch (змінна)

{

case значення1:

// виконується, якщо змінна дорівнює значенню1

break;

case значення2:

// виконується, якщо змінна дорівнює значенню2

break;

default:

// виконується, якщо жоден з варіантів не відповідає значенню змінної

break;

}

Тернарний оператор: Це коротка форма if-else, яка дозволяє вибрати одне з двох значень на основі заданої умови.

результат = (умова) ? значення1 : значення2;

// якщо умова істинна, результат дорівнює значенню1, інакше дорівнює значенню2

1. **Які циклічні конструкції є в мові С#?**

Цикл while: Цей цикл виконує блок коду, доки задана умова є істинною.

while (умова)

{

// виконується, доки умова є істинною

}

Цикл do-while: Цей цикл виконує блок коду принаймні один раз, а потім продовжує виконувати його, доки задана умова є істинною

do

{

// виконується принаймні один раз

} while (умова);

Цикл for: Цей цикл виконує блок коду певну кількість разів, яка визначається початковим значенням, умовою і кроком.

for (ініціалізація; умова; крок)

{

// виконується, доки умова є істинною

}

Цикл foreach: Це спеціальний цикл, який використовується для ітерації через елементи колекцій або масивів.

foreach (тип\_елементу змінна in колекція\_або\_масив)

{

// виконується для кожного елементу у колекції або масиві

}

1. **Що таке метод? В чому полягає різниця між функціями та процедурами? Що таке сигнатура та семантика метода?**

У мові програмування C# метод - це підпрограма, яка виконує певне завдання. Метод може містити ряд інструкцій, які виконуються при його виклику. Методи отримують параметри, виконують певну дію і повертають значення.

Різниця між функцією і процедурою полягає в тому, що функція завжди повертає значення, а процедура - ні; в C# функцією вважається метод, який повертає значення, а процедурою - метод, який не повертає значення.

Сигнатура методу визначається ім'ям методу та списком параметрів. Іншими словами, сигнатура методу залежить від імені методу та типу параметрів. Семантика методу визначає його поведінку та взаємодію з рештою програми. Це передається через його ім'я та дію, яка виконується при його виклику.

Методи C# є важливим інструментом для структурування коду та забезпечення його повторного використання Методи C# є важливим інструментом для структурування коду та забезпечення його повторного використання. Методи дозволяють розбити програму на менші, більш керовані частини, що сприяє підвищенню читабельності та зручності обслуговування коду.

1. **В чому різниця між передачею параметра за значенням і за посиланням? В чому різниця між ref та out параметрами?**

У C# передача параметрів може відбуватись за значенням або за посиланням. Основна різниця між ними полягає в тому, як змінюється значення параметра під час його передачі в метод.

Передача параметра за значенням (Passing by Value): При передачі параметра за значенням копіюється значення змінної, і ця копія використовується в методі. Це означає, що будь-які зміни, які вносяться у цю змінну всередині методу, не впливають на вихідне значення змінної.

Передача параметра за посиланням (Passing by Reference): При передачі параметра за посиланням передається посилання на оригінальний об'єкт, а не його копія. Це дозволяє методу змінювати значення змінної навіть поза межами методу, що дозволяє впливати на оригінальне значення змінної.

Різниця між ключовими словами ref та out також полягає в тому, як вони взаємодіють з передачею параметрів:

ref параметр: Ключове слово ref вказує, що параметр буде переданий за посиланням. Він повинен бути ініціалізований до початкового значення до передачі у метод, і метод може змінити його значення. Метод також може читати його початкове значення.

out параметр: Ключове слово out вказує, що параметр повинен бути ініціалізований в методі, перед тим як він буде використаний. Він використовується для повернення декількох значень з методу. Параметр out не потребує початкового значення при його передачі у метод.

Ці різниці дозволяють програмістам ефективно керувати передачею параметрів у методи та забезпечують гнучкість у використанні та зміні значень змінних всередині та поза методами.

1. **Як працює рекурсія?**

Рекурсія - це процес, коли функція викликає сама себе. У рекурсивних функцій використовується підхід, коли завдання розбивається на менші підзадачі, які розв'язуються за допомогою того ж самого алгоритму. Рекурсія може бути безпосередньою або непрямою.

Процес рекурсії включає наступні етапи:

* Базовий випадок (Base Case): Це умова, яка зупиняє рекурсивний процес. Без базового випадку рекурсивна функція буде викликана нескінченну кількість разів, що може призвести до переповнення стека та помилки переповнення стека.
* Рекурсивний виклик (Recursive Call): Це момент, коли функція викликає сама себе для обробки меншої версії початкової задачі. Кожен рекурсивний виклик зменшує розмір задачі до базового випадку.
* Продвиження до базового випадку (Progress towards Base Case): Кожен рекурсивний виклик повинен наближати виконання до базового випадку. Це допомагає гарантувати, що рекурсивний процес завершиться.

Правильно використана рекурсія дозволяє розв'язувати складні задачі шляхом декомпозиції їх на більш прості підзадачі. Проте неправильне використання рекурсії може призвести до переповнення стека та інших помилок виконання, тому важливо уникати безкінечної рекурсії та забезпечувати наявність базового випадку.

1. **Які різновиди масивів реалізуються в мові С#?**

Одновимірні масиви (One-Dimensional Arrays): Це найпростіший тип масивів, який містить один рівень елементів. Елементи зберігаються в рядковому порядку, і до них можна отримати доступ за допомогою індексу.

int[] одновимірнийМасив = new int[5];

Двовимірні масиви (Two-Dimensional Arrays): Ці масиви мають два рівні індекса, що дозволяє організувати елементи у вигляді таблиці з рядками та стовпцями.

int[,] двовимірнийМасив = new int[3, 3];

Масиви більшої вимірності (Multidimensional Arrays): У мові C# можна створювати масиви з більшою кількістю рівнів, які називаються багатовимірними масивами.

int[,,] багатовимірнийМасив = new int[3, 3, 3];

Зубчасті масиви (Jagged Arrays): Це масиви масивів, де кожен підмасив може мати різну довжину. Вони дозволяють створювати структури даних, які мають різну кількість елементів у кожному рядку.

int[][] зубчастийМасив = new int[3][];