Homework Primitive Data Types and Variables

**Topic 1: Research IEEE-754**

IEEE-754 е технически стандарт за реални числа с плаваща запетая. Стандартът определя:

- Аритметични формати: набор от двоични и десетични данни с плаваща запетая, които са съставени от краен брой цифри.

- Формати за обмен: кодировки (битови низове), които могат да бъдат използвани за обмен на данни в една ефективна и компактна форма.

- Алгоритми за закръгляване: методи, които се използват за закръгляване на числата по време на изчисления.

- Операции: аритметика и други операции на аритметичните формати.

Реалните числа в плаваща запетая се състоят от три компонента: знак (1 или -1), мантиса и порядък (експонента), като стойността им се изчислява по формула.

Съгласно IEEE-754 стандарта произволно реално число R може да бъде представено във вида:

R = M \* qp

където *M* e мантисата на числото, а *p* е порядъкът му (експонента), и съответно *q* е основа на бройната система, в която е представено числото.

**Topic 2: Which is the best way to copy data from integer type to short type?**

Най-добрият начин да се копира integer type към short type е с explicit casting. Причината е, че се конвертират тип данни с по-голям диапазон на стойностите към тип данни с по-малък такъв.

**Topic 3: What are the 2 kinds of data type conversions in C#?**

Има два типа преобразуване на типове данни в C# - implicit typecasting и explicit typecasting.

Implicit typecasting имаме, когато преобразуваме от тип с по-малък обхват към тип с по-голям обхват, например от short към int. Използва се, когато няма опасност от загуба на данни при конвертиране. Става автоматично от компилатора, без да се използват специални символи или оператори. Понякога е възмонжо, въпреки че се конвертира от тип с по-малък обхват към такъв с по-голям, да има загуба на данни, пример за това е конверсия от тип int към тип float или тип long към тип double. Причината е, че int и long използват всичките си битове за представяне на едно целочислено число, докато типовете float и double използват част от битовете си за плаващата запетая.

Explicit typecasting имаме, когато има опасност от загуба на данни при преобразуването и когато се конвертира от тип с по-голям обхват към тип с по-малък. С оператора (тип) става възможно еxplicit конвертирaнето. Пример:

long a = 19999999;

int b = (long)a;

Тук също има е възможна загуба на данни, тогава когато по-големият тип данни е със стойност по-голяма от максималната за по-малкия тип данни, към който се преобразува.

Възможно е и преобразуване на всеки един тип към тип string. Преобразуването става автоматично с оператора (+) или с метода ToString().

**Topic 4: Can DateTime be null?**

Да, възможно е, ако ако се използва nullable type. Пример:

DateTime? a = null;

Причината компилаторът да позволява това е, защото *Nullable types* дават възможност *value types(struct)* да приемат стойност *null*.

**Topic 5: One of the most important features of the CLR is type safety. Explain what Type Safety means and how casting between types works**

Код, който е Type Safety има достъп само до памет, до която му е позволено. Такъв код не може да чете стойности от fields с право на достъп private на други обекти. Той ги достъпва само по точно определен, допустим начин.

По време на Just in Time компилирането, се прави проверка, по избор, която преглежда Metadata-та и Microsoft Intermediate Language (MSIL) на даден метод, дали могат да се компилират до native (машинен) код, с цел да се потвърди, че са type safe. Въпреки че тази проверка не е задължителна, type safety е много важна част, когато става дума за изолацията на едно assembly и секюритизирането. Когато кодът е type safety, CLR напълно изолира assemblies-та едно от друго. Това позволява assemblies да не си влияят негативно и повишава надежността на приложенията. Когато кодът не е Type Safety, могат да се появят нежелани странични ефекти, като това при изпълнението си managed кодът да се преобразува до native код и да предизвика операции, които са вредни за системата. При type safe код, изпълнението на секюритизирането позволява той да не достъпва native код, освен ако няма право.

Type casting е механизъм за конвертиране на стойността на един тип данни към друг. Конвертирането е възможно само ако двата типа данни са съвместими едни с друг, в противен случай се получава InvalidCastException.

**Topic 6: Explain Namespaces and Assemblies.**

Namespaces (пространства от имена) декларира пространство, което съдържа набор от свързани имена на класове, методи, обекти. Имена декларирани в един namespace няма да са в противоречие със същите имена декларирани в друг namespace. Namespace, което се използва в библиотеката на .Net Framework е System.

Namespaces са важни, защото има изключително много имена на класове, свойства, методи, променливи. Без Namespaces, всички тези имена биха породили конфликти помежду си. Това се предодвратява благодаренение на това, че Namespaces ограничава вдимостта на имената декларирани в него.

Assemblies (асемблита) са колекция от типове и ресурси, която формира логическа единица функционалност. Всички типове в C# и изобщо в .NET Framework могат да съществуват само в асемблита. При всяка компилация на .NET приложение се създава асембли. То се съхранява като файл с разширение .exe или .dll.

**Topic 7: Explain how reference and value types work.**

Value types – променлива от value type съдържа директно стойността си в паметта, наречена стек (stack), примерно 3,1456 или 222. Когато методът, с който е дефинирана дадена променлива от value type, завърши изпълнението си, тя се освобождава от стека. Value type са примитивните числови типове, символният тип и булевият тип: sbyte, byte, short, ushort, int, long, ulong, float, double, decimal, char, bool.

Reference types – съдържат в стека за изпълнение референция към динамичната памет (heap), където се съхранява стойността им. Референцията е указател, който сочи към реалното местоположение на стойността в динамичната памет. Динамичната памет, която се заделя от референтните типове не се освобождава, когато методът, в който е декларирана променливата от референтен тип, завърши изпълнението си, а от garbage collector, когато той прецени, че тази рефернция не се използва от програмата. Референтни типове са инстанции на даден клас, интерфейсите, масивите, стринг-овете, обектите.

**Topic 8: Explain how Boxing and Unboxing value types work.**

Boxing – процес, при който се конвертира value type в object type или към всеки друг interface type, който е имплементирн от този value type. Когато имаме boxing CLR обвива стойността в System.Object и я съхранява в динамичната памет. Това е implicit конверсия.

Unboxing – извлича value type от object. Това е еxplicit конверсия.

**Topic 9: Research and explain Object Hash Codes**

Всеки object, който се създава в . Net има GetHashCode метод, заедно с Equals, GetType, и ToString методите. Този метод е метод на Object класа който наследяват всички други класове.

Хаш-кодът е числена стойност (int), която е служи за идентификатор на обект в hash-based колекции като Dictionary<TKey, TValue> класът, Hashtable класът.

GetHashCode методът предоставя хаш-кода за алгоритми, които имат нужда от бързи справки за равенство между обекти.

Два обекта, които са еднавки, връщат еднакви хаш-кодове. Два еднакви хаш-кодове обаче, не означават, че обектите са еднакви, защото различни обекти, могат да имат идентични хаш-кодове.

**Topic 10: Research and explain the dynamic type**

Променлива от тип Dynamic може да се инициализира с всеки друг тип. Този тип може да се променя по време на изпълнението на програмата. Недостатък на Dynamic типа е, че производителността намалява и не се извършва проверка за грешки по време на compile time на програмата. При изпълнението на дадена програма, обаче, ако има грешки, те се появяват.

Dynamic типът дава достъп до Dynamic Language Runtime, функционалност над Common Language Runtime. Това означава, че dynamic е tимплементиран с много поставен код, който използва CLR.