Prague parking 1

**Version 1.0**

Array för parkeringsplatser & textbaserad meny

Jag skapar en array för att kunna arbeta med en fast struktur direkt, och för att det blir lättare att hantera platserna eftersom varje index motsvarar en plats. Jag väljer att göra en sträng för lagring eftersom det är enklast för att representera registreringsnummer.

Jag börjar med att bygga en textbaserad meny som användaren kan navigera i. Menyn innehåller alternativ för att parkera, hämta ut, flytta, söka och skriva ut hela parkeringsplatsen. Detta gör jag genom en snurra. Jag börjar menyn med Console.Clear() för att menyn ska vara lättläst för användaren.

Menyhantering genom switch

Jag börjar med menyhanteringen för att programmet ska köra den kod som tillhör alternativet användaren väljer. Jag gör först if-satser men inser att koden blir rätt ”ful” och svår att läsa. Så jag ändrar till att göra en switch. Jag skriver in alla case som ska finnas med i menyn. Sedan avslutar jag varje case med ett break, för att programmet inte ska fortsätta köra nästa case med. Jag lägger även till en default så att om användaren skriver in något annat än alternativen i menyn står det felaktigt val, försök igen.

Efter switch-blocket lägger jag till:

Console.WriteLine("\nPress any key to continue...");

För att skriva ut ett meddelande till användaren och \n för att hoppa till en ny rad.

&

Console.ReadKey();

För att programmet ska pausa tills användaren trycker på en tangent för att ge användaren tid att läsa meddelandet som visas.

Metod för att parkera fordon

Jag gör en void för att metoden ska parkera fordon. Först frågar systemet efter fordonstyp och registreringsnummer. Jag använder (Console.ReadLine() ?? "").Trim().ToUpperInvariant(); i strängarna där användaren skriver in fordonstyp och registreringsnummer. Console.ReadLine() för att läsa in vad användaren skriver som text. ?? ”” för om värdet blir null (om exempelvis användaren inte skriver något utan bara trycker på enter) så används en ny sträng i stället för att programmet kraschar. .Trim() för att ta bort eventuella mellanslag och andra osynliga tecken i början och slutet av en textsträng. .ToUpperInvariant() är en unicode-case mapping. Alltså är den oberoende av datorns språkinställningar och gör att all text ska konverteras till versaler, för att göra det enklare att läsa och enklare att jämföra text. Jag lägger även till en if-sats i type strängen för att säkerställa att användaren inte ska kunna skriva in någon annan fordonstyp än CAR eller MC. I strängen string vehicle = type + "#" + regNr; kombineras typen av fordonstyp och regnummer i formatet "TYPE#REGNR".

Jag väljer att i första versionen av programmet börjar med att göra en for-loop: for (int i = 0; i < parkingLot.Length; i++) som gör att fordonet parkeras på första bästa plats.

Metod för att hämta ut fordon

Jag gör en void för att hämta ut fordon. Användaren skriver in fordonets registreringsnummer. Programmet går igenom alla parkeringsplatser och jämför varje plats exakt mot CAR#REGNR eller MC#REGNR (inte med Contains) för att undvika falska träffar. Om fordonet inte finns meddelas detta, annars hämtas fordonet ut.

Metod för att flytta fordon

Jag skapar en void här för att kunna flytta fordon inom parkeringsplatsen. Användaren skriver in registreringsnumret för det fordon som ska flyttas. Programmet söker igenom alla parkeringsplatser tills det hittar rätt fordon genom exakt jämförelse (CAR#REGNR eller MC#REGNR). Om fordonet hittas sparas platsens index i variabeln oldIndex. Sedan frågar programmet vilken plats användaren vill att fordonet flyttas till. Om platsen är giltig och ledig flyttas fordonet från den gamla platsen till den nya.

Metod för att söka efter fordon

Jag skapar en void för att söka efter fordon. Användaren skriver in registreringsnumret. Loopen går igenom alla platser på parkeringsplatsen och jämför exakt varje plats med CAR#REGNR eller MC#REGNR. Om fordonet hittas skriver programmet ut platsnumret. Om inget hittas skriver programmet ut att fordonet inte finns på parkeringsplatsen.

Metod för att skriva ut hela parkeringsplatsens innehåll

Jag skapar en void som skriver ut hela parkeringsplatsen. Jag använder en for-loop som går igenom alla parkeringsplatser index 0-99. Jag använder null operatorn ?? för att skriva ut om en plats är tom. Om ett fordon står på en plats skrivs TYPE#REGNR ut intill platsnummer.

**Version 1.1**

Påbyggnad metoder

Jag lägger till två metoder för att förhindra dubbletter innan parkering. Först lägger jag till en bool som heter MatchVehicle() för att kolla en plats i taget om den matchar registreringsnumret. Sedan även en bool som heter Exists() som kollar hela parkeringsplatsen om fordonet finns någonstans. Exists() använder MatchVehicle() för varje plats i loopen. Sedan lägger jag in en kontroll genom en if-sats i metoden ParkVehicle().

Säkra upp användarinput

Jag läser igenom koden och inser att jag använder .Trim() för att förhindra mellanslag i början eller slutet av strängen. Men jag har inte förhindrat att användaren kan skriva in ett mellanslag och dyl mitt i strängen. Jag lägger därför in hjälpfunktion i form av en bool: IsValidReg() under boolen Exists(). Sedan lägger jag till if-satsen (!IsValidReg(regNr)) i metoderna ParkVehicle, RemoveVehicle, MoveVehicle och SearchVehicle. Jag ändrar koden så att programmet inte längre avslutas när användaren matar in fel data. Istället använder jag while(true)-loopar vid inmatning av fordonstyp, registreringsnummer och platsnummer. Programmet upprepar frågan tills användaren skriver in giltig data.

Dubbelparkering av MC & Optimeringsrutin

Jag låter varje plats i arrayen fortfarande vara en sträng, men nu kan en plats innehålla två MC samtidigt. För att skilja dem åt använder jag tecknet | som avgränsare. Jag lägger till hjälpfunktioner som känner av om en plats innehåller en bil eller motorcykel och om det finns plats för ytterligare en MC. I metoden ParkVehicle ändrar jag koden så att en MC först försöker parkera i en ruta som redan innehåller en annan MC, om det finns utrymme. Om inte, parkeras den som tidigare på första lediga plats. I metoderna RemoveVehicle, MoveVehicle och SearchVehicle justerar jag logiken så att programmet kan hantera att två motorcyklar delar en ruta. När en MC tas bort lämnas den andra kvar. I utskriften (PrintParkingLot) visas båda motorcyklarna i samma ruta separerade med |.

Jag lägger in en optimeringsrutin som flyttar ihop lösa MC, så att så många MC som möjligt blir dubbelparkerade. Detta gör jag genom att först lägga till en metod som skapar arbetsordrar, GenerateMcOptimizationWorkOrders(). Sedan lägger jag till den som en ny punkt i menyn och som ett nytt case i min switch.

Visualisering av vad som finns på parkeringsplatsen

Jag börjar med att lägga till datatypen enum för att definiera ett antal förbestämda värden. Enumen kallas SpotStatus och representerar statusen på varje parkeringsruta. De enda giltiga värdena är Empty, MC1, MC2 och CAR. Jag skriver enumen sist i koden under PrintParkingLot. Detta gör jag eftersom datatyper som enum måste komma efter top-level-satserna i C#. Jag skapar två hjälpfunktioner för att kunna visa statusen på parkeringsplatserna. Den första, GetSpotStatus(), går igenom en plats och avgör om den är tom, har en bil, en motorcykel eller två motorcyklar. Den andra, CountStatuses(), räknar hur många platser som är tomma, halvfulla (en MC), fulla med två MC, eller upptagna av en bil. Sedan skapar en enkel översikt symboler och färger genom att lägga till en till metod ShowOverview()som visar en grafisk översikt av hela parkeringsplatsen med symboler och färger. Varje ruta skrivs ut med en symbol som visar status: E för tom, m för en MC, M för två MC och C för bil. Jag använder olika färger för varje status (grön för tom, gul för en MC, röd för två MC och blå för bil. Sedan läger jag till en ny punkt i menyn Console.WriteLine("6. Overview map (colors)"); och ett nytt case i min switch case "6": ShowOverview(); break;.

Jag lägger även till rapporter för bilar, motorcyklar och tomma platser, utöver standardlistan och overview.