**Sammanfattning**

Jag har utvecklat en textbaserad applikation, Prague Parking, som hanterar en parkeringsplats med 100 rutor. Systemet tar emot bilar och motorcyklar, kan flytta och hämta ut fordon, söka efter fordon samt visa en översikt av parkeringsplatsen. I version 1.1 har jag utökat lösningen med färgvisualisering, separata rapporter (alla bilar, alla MC, tomma rutor), en optimeringsrutin som föreslår arbetsorder för att dubbelparkera MC samt säkrare användarinput (loopar tills korrekt data). Jag har följt uppgiftens lagringsmodell (strängar) och byggt upp metoder som gör koden läsbar och utbyggbar.

**Hur jag löste uppgiften**

Version 1.0

Jag började med en enkel version som:

* representerar parkeringsplatsen som string?[] parkingLot = new string?[100];
* lagrar fordon som text enligt mallen CAR#REGNR eller MC#REGNR
* använder en meny med switch och top-level statements
* implementerar grundfunktioner: parkera, hämta ut, flytta, söka och skriva ut alla platser

Version 1.1

Därefter byggde jag vidare på en mer utvecklad och användarvänlig version genom att lägga till:

* **Visualisering**: lade till enum SpotStatus { Empty, Mc1, Mc2, Car } och hjälpfunktioner GetSpotStatus och CountStatuses. Metoden ShowOverview() ritar en kompakt karta (10 rutor per rad) med symboler (E/m/M/C) och färger för snabb överblick.
* **Rapporter**: metoderna ListCars, ListMotorcycles, ListEmptySpots ger filtrerade rapporter utöver standardlistan.
* **Dubbelparkering MC**: jag tillät att en ruta kan innehålla två MC med | som avgränsare, t.ex. MC#ABC123|MC#DEF456. Hjälpfunktioner som IsCarSpot, IsMcSpot, GetMcTokens, TryAddMcToSpot och RemoveMcFromSpot hanterar inläggning/urplock.
* **Optimering (arbetsorder)**: GenerateMcOptimizationWorkOrders() letar upp singel-MC-rutor och parar ihop dem till arbetsorder (“Flytta MC XYZ från plats A till plats B”). (Jag lät personalen utföra flytten – enligt kravtexten.)
* **Säker input**: ersatte engångskontroller med inmatningsloopar. Fordonstyp valideras till CAR/MC; registreringsnummer valideras med IsValidReg (A–Z, ÅÄÖÜ, 0–9, inga mellanslag); platsnummer läses tills det är ett tal 1–100.

**Utmaningar och hur de löstes**

1. **Top-level statements & deklarationsordning**  
   Jag stötte på compiler-fel när jag placerade enum och metoder på fel plats. Lösning: all programlogik (metoder) före enum, och inga metoder efter typen. Jag dubbelkollade klammerpar så att metoder inte hamnade inuti varandra.
2. **Null-varningar och typer**  
   Jag valde string?[] eftersom tomma rutor representeras av null. Det eliminerade varningar och gjorde logiken tydlig: null = ledig.
3. **Exakt matchning vs. Contains**  
   För att undvika falska träffar bytte jag från substring-sök till exakt jämförelse: en bil matchar **bara** CAR#REGNR, en MC matchar **bara** en av MC-tokens i rutan.
4. **Validering & Unicode**  
   Jag använde ToUpperInvariant() för att undvika kulturspecifika “i/I”-problem och stödja Å/Ä/Ö/Ü. IsValidReg() blockerar mellanslag och icke tillåtna tecken. Kravet “be om nya data” löstes med while(true)-loopar.
5. **MC-delning och uppdelning**  
   Jag behövde en robust hantering av strängformatet för en eller två MC: Split('|') och Join('|'). Hjälpmetoderna kapslar detta så att huvudflödena (parkera/flytta/ta bort) förblir läsbara.
6. **Bygg/kör gamla binärer**  
   Vid ett tillfälle kördes en gammal build. Jag använde **Build → Clean** och **Rebuild**, samt såg till att alla filer var sparade.
7. **Användarupplevelse (“dubbel paus”)**  
   ShowOverview() har egen paus och loopen har en global paus. Jag löste det med continue; i menycaset för att undvika två tangenttryck.

**Metoder och modeller jag använde**

* **Små, fokuserade metoder**: varje metod gör en sak, t.ex. Exists, MatchVehicle, TryAddMcToSpot.
* **Hjälpfunktioner för strängformat**: gör det lätt att byta intern representation vid behov.
* **Enum för status**: ger tydlig semantik mellan tom/MC1/MC2/bil, vilket förenklar översikten och rapporterna.
* **Defensiv programmering**: inmatningsloopar, validering, tidiga returer vid fel.
* **Enkel greedy-heuristik** för optimeringsförslag: para ihop singel-MC-rutor från ytterkanterna in mot mitten för att maximera dubbelparkeringar på ett deterministiskt och lättförklarat sätt.

**Hur jag skulle lösa uppgiften nästa gång**

* **Separera UI och logik mer**: lyfta ut användarinteraktion (Console I/O) till egna metoder och låta “kärnlogiken” bli testbar utan konsol.
* **Införa en liten modelltyp** (t.ex. en struct/klass) för att representera rutor och fordon i minnet, och mappa till/från strängar i ytterkant. Det skulle minska risken för strängfel.
* **Tidsstämpling**: bygga in datum/tid i formatet, t.ex. CAR#REGNR#2025-10-11T14:23:11 eller hålla en parallell array/dictionary för in-tid, och visa parkeringstid vid uttag.
* **Enhetstester**: testa IsValidReg, TryAddMcToSpot, RemoveMcFromSpot, MatchVehicle och optimeringsförslagen.
* **Mer avancerad optimering**: ex. först fyll alla Mc1 till Mc2 närmsta granne (minimera gångsträckor), eller skriva ut rutter i lämplig ordning.

**Slutsats hemuppgift**

Version 1.0 uppfyller grundkraven: parkera, hämta, flytta, söka, lista med en enkel meny. Version 1.1 innehåller förbättringar: visualisering i form av symboler + färg + rapporter, och optimering i form av arbetsorder för att para ihop singel-MC, samt säker input som loopar tills korrekt. Jag har följt uppgiftens formatval genom strings i en array, men samtidigt byggt hjälpfunktioner som gör koden robust och relativt lätt att vidareutveckla. Programmet går att köra på en annan dator (ingen extern lagring krävs), och README kan instruera hur man startar via Visual Studio.

**Slutsats kurs**

Jag har fått praktisk träning i:

* att bryta ned problem i små metoder,
* hantera C#-detaljer (top-level statements, enum-placering, nullbarhet),
* designa enkel men robust användarinput,
* och resonera kring datastrukturer även när formatet (strängar i array) är begränsat av uppgiften.

Nästa steg för mig är att strukturera kodbasen ännu tydligare (UI vs. logik), lägga till tidsstämpling och börja testa metoder isolerat. Jag känner att uppgiften gett en bra grund att stå på inför kommande moment (ex. databaser) där modellen kan bli renare och uthålligare.