INLÄMNINGSUPPGIFT 1

Prague Parking V1

Inledning

Kund önskar ett stöd för en parkering vid slottet i Prag.

Parkeringsplatsen är s.k. valet parking. Kunden lämnar nyckel och fordon samt får ett kvitto vilket ger rätten att hämta ut fordonet. Parkeringsplatsen sköts av "finniga studenter" och pensionärer så systemet måste vara enkelt.

Parkeringsplatsen tar emot bilar och motorcyklar.

I dagsläget hämtas alla fordon ut före 00.00 när parkeringen stänger. Ej uthämtade fordon körs till en parkering utanför stan och kunderna får betala straffavgift för att få ut sitt fordon. (Hanteras ej av det aktuella systemet)

Kundens krav på systemet

- Systemet skall kunna ta emot ett fordon och tala om vilken parkeringsplats den skall köras till.
- Manuellt flytta ett fordon från en plats till en annan.
- Ta bort fordon vid uthämtning.
- Söka efter fordon.
- Kunden önskar en textbaserad meny

I dagsläget behövs ingen sparfunktion. Datorn slås på när parkeringsplatsen öppnar och slås av när man går hem.

Tekniska krav

- All identifiering av fordon sker genom registreringsnummer
- Registreringsnummer är alltid strängar med maxlängd 10 tecken.
- På parkeringsplatsen finns 100 parkeringsrutor
- En parkeringsruta kan innehålla
    - 1 bil eller
    - 1 mc eller
    - 2mc eller

- vara tom

Parkeringsrutorna skall hanteras som en endimensionell vektor (array) av strängar. Vektorn skall hantera 100 element. Kundens personal är människor och förväntar sig att platserna numreras 1–100 i in- och utmatningar i systemet.

Loggbok

Förutom den rena applikationen behöver du under projektets gång föra en dagbok, eller loggbok. I den noterar du varje dag vad du har gjort i projektet. Du kan notera sådant som problem som uppstod och hur de löstes. Loggboken kommer att vara en del av din inlämning.

Github

All inlämning ska göras via Github.

Gör era repon **publika**, så att läraren (och eventuellt klasskamraterna) kan läsa vad som finns. Privata repon gör det väldigt besvärligt för läraren att granska ert arbete.

Kom ihåg att ni ska skicka in länken till ert repo, inte till någon enskild fil. Om ni har problem så fråga läraren eller klasskamraterna.

Inlämning

Inlämning sker individuellt. Dock är det inget som hindrar att ni arbetar tillsammans.

Inlämning görs i lärplattformen. Där bifogar ni en länk till ert repo i GitHub. Dessutom laddar ni upp filen med er loggbok.

Betygskrav

För G

- För betyget G skall alla kraven ovan vara uppfyllda.
- En individuell loggbok ska lämnas in. I något lämpligt format.
- Applikationen skall gå att köra på en dator annan än er egen (dvs på lärarens dator)
- Om det behövs några speciella handgrepp för att köra applikationen (utöver att trycka F5 eller Ctrl-F5 i Visual Studio) så skall dessa dokumenteras. Använd README.MD i GitHub för ändamålet.

Prague Parking 1.1 - För er som vill ha VG på uppgiften.

När ni är klara, dvs har ett fungerande program med en enkel meny, med Prague Parking1.0:

Skapa ett nytt projekt i er Solution med lämpligt namn. Det skall givetvis framgå tydligt att det är version 1.1

Kopiera innehållet från 1.0-projektet till 1.1-projektet och fortsätt att arbeta därifrån.

Lägg till ***minst två*** av nedanstående funktioner.

1. **Visualisering av vad som finns på parkeringsplatsen**. Dvs vilka platser är lediga, halvfulla och fyllda så att personalen får en god överblick. Pluspoäng för kreativa lösningar och färgmarkering. Ni får lov att göra mer än en rapport tex alla motorcyklar, alla bilar, alla tomma platser etc. En lista som visar vilka regnr och fordonstyper som finns på vilken plats är minimum.
2. **Skapa en optimeringsrutin** som "flyttar ihop" lösa mc så att vi får så många mc som möjligt dubbelparkerade. När optimeringen fungerar, glöm inte att personalen måste göra jobbet, dvs ni måste skriva ut arbetsordrar *flytta MC abc123 från plats 3 till plats 7*, *flytta MC bbb222 från plats 12 till plats 16* osv.
3. **Säkra upp användarinput** - tex be om nya data om användare matar in felaktiga data när registreringsnummer och eventuellt platsnummer skall anges. P-platsnummer måste vara tal i intervallet 1–100, registreringsnummer får inte innehålla mellanslag o. dyl., samt måste hantera västeuropeiska alfabeten samt siffror. Tex Ü, Å, Ä, Ö etc. Titta i Wikipedia: https://sv.wikipedia.org/wiki/Registreringsskylt
4. **Utöka lagringsformatet** för fordon så att man även får med datum och tid när ett fordon kom in. (Tidsstämplingen skall ske automatiskt) När fordonet tas ut ur systemet så skall man få reda på hur lång tid det varit parkerat. Detta kan nu vara mer än ett dygn. Använd lämplig representation av datum och tid.

Ytterligare krav för VG

För VG i betyg krävs det, förutom en fungerande version 1.1 av systemet enligt ovan, att ni skriver en personlig reflektion över kursen och projektet. Reflektionen kan lämpligen struktureras enligt nedan:

1. Sammanfattning
2. Hur vi/jag löste uppgiften
3. Utmaningar i uppgiften och hur de löstes
4. Metoder och modeller som använts för att lösa uppgiften
5. Hur du skulle lösa uppgiften nästa gång, givet vad du vet nu

6. Slutsats hemuppgift
7. Slutsats kurs, så här långt

En sådan reflektion behöver inte bli särskilt omfattande, men ett par – tre sidor är normalt.

Tips

I denna uppgiften gäller det bland annat att lagra flera bitar information en ett enda element i en vektor. I en kommande kurs i databaser kommer vi att lära oss varför detta är en dålig idé, men för stunden har vi inget val.

Antag att P-huset modelleras som en array av string:

```
string[] parkingGarage = new string[100];
```

Varje element i parkingGarage motsvarar då en P-plats. Indexet är P-platsens nummer minus 1 (index i arrayer börjar på noll, men den första P-platsen heter nummer 1)

I varje element skall vi lagra

- En bil, eller
- En MC, eller
- Två MC

Varje fordon har

- Fordonstyp (till exempel ”CAR” eller ”MC”)
- Registreringsnummer (en kombination av bokstäver och siffror)

Ett sätt att få ihop detta är att vi bildar en sträng av (fordonstyp + skiljetecken + registreringsnummer), där skiljetecknet är et tecken som garanterat inte finns med på en registreringsskylt, till exempel ”#”. Då skulle en bil med registreringsnummer ABC123 bilda strängen **CAR#ABC123**. På motsvarande sätt skulle en MC med registreringsnumret KLM789 beskrivas av strängen **MC#KLM789**.

Om man sedan tar denna idén ett steg vidare så skulle två MC med registreringsnumren KLM789 och FTP666 kunna slås ihop med ett annat skiljetecken, till exempel ”|”. P-platsen med de två MC skulle då innehålla strängen **MC#KLM789|MC#FTP666**.

För att göra denna ihop- och isärslagning används med fördel metoderna String.Join respektive String.Split.

Programstruktur

Fundera på vilka metoder som behövs. Typiskt behövs metoder för

- Utskrift av meny

- Inläsning av menyval
- Menyhantering/styrning av programflöde
- Sökning
    - Efter givet fordon
    - Efter ledig plats för given fordonstyp
- Utskrift
    - Av enskild P-plats
    - Av hela P-huset

Ni kommer att upptäcka fler metoder – var inte rädda för att skapa många små metoder. Kom ihåg att en metod skall göra **en** sak. Undvik till exempel att ha en massa kod för användargränssnitt eller skärmdialog i metoder som inte måste ha det. Istället för att fråga efter ett registreringsnummer inuti en sökmetod så skicka in registreringsnumret som en parameter till metoden.

Inte alla metoder behöver, eller skall vara, av typen **void**. Metoder kan returnera allt möjligt. Med en out parameter kan man till och med skicka tillbaka mer än en sak (Jämför med hur TryParse() fungerar). Detta skulle kunna vara användbart i en sökfunktion, till exempel.