Laboration 3 - Bilregister

Programmeringsteknik

Skriv ett program som representerar ett fordonsregister, d.v.s. ett register över fordon och dess ägare. Fordonsregistret ska ha ett textbaserat gränssnitt, likt det ni använde i laboration 2. Nedan beskrivs den data som ska finnas representerad i programmet, samt de funktioner som ska vara implementerade.

Data

Person Denna typ ska användas i programmet för att lagra information om ett fordons ägare. Informationen vi behöver är ägarens namn och ålder.

Fordon Denna typ ska användas i programmet för att lagra information om ett fordon. Ett fordon ska ha egenskaperna fordonstyp, märke, registreringsnummer och ägare. Alla fordon har en ägare.

Programmet ska kunna innehålla max 10 fordon. Programmet ska behålla informationen om registret om det avslutas och startas igen. På grund av detta måste informationen lagras i en fil när programmet avslutas, och läsas in igen när programmet startas. Om ingen fil finns, ska ett nytt bilregister skapas.

Funktioner

När programmet kör ska användaren få välja mellna följande val:

1. Lägg till ett fordon. All information om fordonet och dess ägare ska läsas in från användaren, och fordonet ska läggas till i registret.
2. Ta bort ett fordon. Användaren ska ange en position (1-10) och fordonet på den positionen ska tas bort. Tänk på att vektorer börjar på 0 och inte på 1, och hur du ska kompensera för detta.
3. Sortering efter bilmärke. Registret ska sorteras i bokstavsordning efter bilmärke. Funktionen kan ha valfri skiftlägeskänslighet[1].
4. Skriv ut information om ett fordon. Användaren ska ange en position (1-10) och fordonet på den positionen ska skrivas ut, samt information om dess ägare.
5. Skriv ut hela fordonsregistret. Endast ett fordon per rad. Fundera på hur ni ska presentera informationen på ett så läsligt sätt som möjligt. Ingen information om ägare ska vara med.

0. Avsluta programmet.

Om användaren försöker göra något som inte är korrekt, exempelvis lägga till ett fordon när registret är fullt, ska ett tydligt felmeddelande visas.

[1] `https://sv.wikipedia.org/wiki/Skiftl%C3%A4gesk%C3%A4nslighet`

Kontroller

Gå igenom dessa kontroller innan ni ber om provkörning.

- Ta bort registerfilen och starta programmet. Programmet ska inte sluta fungera, utan visa ett tomt register.
- Lägg till ett fordon när registret är fullt.
- Försök visa eller ta bort en bil på en ogiltig plats, t.ex. -1 och 13.
- Försök visa eller ta bort en bil på en plats där ingen bil finns, t.ex. 7 om det bara finns 5 bilar i registret.
- Försök skriva in oväntad inmatning. Exempelvis text som ålder, text istället för ett val, eller en text som är för lång. Programmet får inte sluta fungera (t.ex. krasha).

Krav på programkoden

Inga globala variabler

Variabler får inte deklareras utanför en funktion. Exempelvis:

```
#include<stdio.h>
int antal; //Felaktig kod
int main() {
    ...
}
```

Är ej korrekt, utan variabeln behöver flyttas in i en funktion:

```
#include<stdio.h>
int main() {
    int antal; //Korrekt kod
}
```

Inga magiska tal

Alla tal, vars syfte inte är uppenbart från sammanhanget i programkoden, måste defineras som konstanter. Dessa tal kallas *magiska tal*[2] Exempelvis:

```
#include<stdio.h>
int main() {
    int vektor[100]; //Ej korrekt, 100 = magiskt tal.
}
```

Flytta istället talet till en konstant.

```
#include<stdio.h>
#define SIZE 100
int main() {
    int vektor[SIZE]; //Korrekt
}
```

En vanlig tumregel är alla tal som inte är 1, 0 eller -1 är magiska tal, men det finns självklart undantag.

[2] https://en.wikipedia.org/wiki/Magic_number_(programming)

Säker inmatning

All inläsning av information från användaren ska ske säkert, alltså programmet ska inte krasha eller ha odefinerade beteenden. Detta betyder att informationen måste läsas in med funktionen `fgets`. Här är exempel på hur man läser in en text:

```
char text[32];
fgets(text, 31, stdin);
strtok(text, "\n");
```

Och här är ett exempel på hur man läser in ett tal:

```
char buf[32];
fgets(buf, 31, stdin);
int number = atoi(buf);
```

Modularisering

Projektet ska ha minst fyra filer: `main.c`, `fil.c`, `fil.h` och `makefile`. Rätt sak ska vara i rätt fil (se era anteckningar från föreläsningen om modularisering). Kommandot `make` ska korrekt kompilera programmet, med separata regler för varje filpar. Det är valfritt att ha med `main.h`. Alla funktioner för filhantering ska vara i `fil.c` eller `fil.h`.

Exempelkörning

Här visar jag bara en kort körning på hur programmet kompileras och körs, men inte vad valen gör, då ni har lite frihet i hur ni designar utmatningen, och vi vill se er kreativitet lite i den sista laborationen.

```
$ make
gcc -c file.c
gcc file.o main.c -o lab3
$ ./lab3
1. Add Vehicle
2. Remove Vehicle
3. Sort
4. Show Vehicle
5. Show Registry
0. Quit
>
```