# Obligatorisk oppgave nr.4

## PROG1001 - Grunnleggende programmering, høsten 2023

**Innleveringsfrist: 14.november 2023 kl.11:00 (*må* overholdes)**
***i Blackboard på pdf-format***

Lag et program som holder orden på *ett* arrangements ulike (dugnads)oppgaver/gjøremål og personene som er satt til å utføre/betjene hver av dem. Det er max. seks personer på hver oppgave. Disse har ikke noe navn, men identifiseres bare via et unikt nummer/ID (1-1000), og disse ligger lagret i `hvem`. Det *aktuelle* antallet (altså 6 eller mindre) på en oppgave ligger lagret i `antallTotalt`. Antall personer foreløpig registrert på oppgaven ligger lagret i `antallNaa` (som da alltid skal være `<= antallTotalt`). Til alt dette har vi altså **struct'en**:

```
struct Oppgave {
  char* navn;
  int   antallTotalt,
        antallNaa;
  int   hvem[MAXPERS];              //  #define MAXPERS   6
};
```

Vi har også *kun* de **to globale variablene** (muligens trenger du også noen flere **const'er**):

```
struct Oppgave* gOppgavene[MAXOPPG];  //  #define MAXOPPG  20
int gSisteOppgave = 0;
```

**Lag** `main`, som i `EKS_27.c` og `EKS_35.c`, som styrer hele programmets funksjonalitet.
**Lag også** en `skrivMeny()`, og **bruk gjerne** `LesData.h` som introduseres i `EKS_34.c`.

**Lag funksjonene (10 stk)**:

`void nyOppgave()`

Funksjonen kommer med en melding om det *ikke* er plass til flere oppgaver i datastrukturen. I motsatt fall opprettes det en ny struct som neste ledige peker i arrayen settes til å peke på. Husk å telle opp antall pekere brukt i arrayen. Lag også:
`void oppgaveLesData(struct Oppgave* oppgave)`
Som sørger for at den nyopprettede structen får innlest sine *to første* datamedlemmer. Den tredje nullstilles, og den fjerde fylles først i den fjerde funksjonen nedenfor.

`void skrivOppgaver()`

Funksjonen går gjennom *alle nåværende* oppgaver, og sørger for at *alle* data om hver av dem blir skrevet ut. Til dette siste bruker den funksjonen:
`void oppgaveSkrivData(const struct Oppgave* oppgave)`

`void ledigeOppgaver()`

Funksjonen går gjennom *alle nåværende* oppgaver, og sørger for at *alle* data om hver av dem som *ikke er oppfylt med nok personer* (dvs. `antallNaa < antallTotalt`) blir skrevet ut. For å gjøre alt dette, bruker den funksjonen:
`bool oppgaveLedigPlass(const struct Oppgave* oppgave)` og den ovenfor.

`void personerTilknyttesOppgave()`

Funksjonen kommer med en melding om datastrukturen er tom. I motsatt fall leses et oppgavenummer (0-`gSisteOppgave`). Velges 0 (null) betyr det at brukeren angrer, og intet mer gjøres. Ellers tilknyttes personer til oppgaven vha. funksjonen:
`void oppgaveTilknyttPersoner(struct Oppgave* oppgave)`
som først skriver *alle* oppgavens data (vha. allerede laget funksjon). Er det allerede fullt med personer på oppgaven kommer det en egen melding. I motsatt fall spørres det om antall personer som skal tilknyttes oppgaven. Dette er et tall mellom 0 og det antall personer som mangler for at det skal bli nok personer på oppgaven. Deretter gås det gjennom dette antallet, og det spørres etter personers nummer (1-1000). Det trengs *ikke* å sjekke at disse numrene er ulike, eller om denne personen allerede er tilknyttet en annen oppgave. Til slutt skrives alle dataene om oppgaven igjen (på samme måten som tidligere), samt at nåværende antall på oppgaven telles opp med det antallet som nettopp er lest inn.

`void fjernOppgave()`

Funksjonen kommer med en melding om datastrukturen er tom. I motsatt fall leses ønsket nummer for oppgave som skal fjernes/slettes. Dette er et tall mellom 0 og så mange oppgaver som før øyeblikket er aktuelle/i bruk. Skriver brukeren 0, hoppes det bare over alt annet innhold i funksjonen, og det skrives en melding om at ingen slettes/fjernes. Brukeren må så bekrefte at virkelig *vil* fjerne. Alle andre svar enn 'J' gjør at det samme som rett ovenfor skjer. Ønsker brukeren å fjerne den *eksakt* siste, så kommer det en melding om at dette har skjedd. Ønskes en annen fjernet, kommer det en annen melding, og den bakerste/siste flyttes til den fjernedes/slettedes plass (peker oppdateres bare til å peke på den siste). I begge tilfeller må `gSisteOppgave` telles ned med minus en. *Husk å frigi allokert memory* for den fjernede/slettede struct'en, samt *all* allokert memory inni selve struct'en. Dette aller siste gjøres vha:
`void oppgaveSlettData(struct Oppgave* oppgave)`

***Husk god kommentering etter Doxygen.***

**Lykke til!**

**FrodeH**