NTNU
Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

Eksamensoppgave i PROG1001
– Grunnleggende programmering

Faglig kontakt under eksamen:	**Frode Haug**
Tlf:	**950 55 636**
Eksamensdato:	**8.desember 2022**
Eksamenstid (fra-til):	**09:00-13:00 (4 timer)**
Hjelpemiddelkode/Tillatte hjelpemidler:	**F - Alle trykte og skrevne.** (kalkulator er *ikke* tillatt)
Annen informasjon:	
Målform/språk:	**Bokmål**
Antall sider (inkl. forside):	**8**

Informasjon om trykking av eksamensoppgaven

Originalen er:

1-sidig X **2-sidig** ☐

sort/hvit X **farger** ☐

Skal ha flervalgskjema ☐

Kontrollert av:

_____

Dato Sign

NB: Oppgave 1a, 1b og 2 er totalt uavhengige og kan derfor løses separat.

Oppgave 1 (30%)

a) Hva blir utskriften fra følgende program (litt hjelp: det blir 5 linjer):

```
#include <stdio.h>
#include <string.h>


char txt[] =
        "SAMMENKOMSTER-SAMMEN-SAMLER-SALMER-OG-SANGER-SANNELIG";


int main() {

  int i = 7,  j = i*2,  k = i*3;

  while (txt[i] != '\0')  {
    if (txt[i] == 'S') printf("%c ", txt[i+2]);
    ++i;
  }
  printf("\n");


  i +=10;  i %= 9;
  while (txt[i] == txt[j]  &&  txt[j] == txt[k])  {
    i++;  ++j;  ++k;
  }
  printf("%c %c %c\n", txt[i], txt[j], txt[k]);


  char* t = txt;
  k = strlen(txt)-3;  j = 0;
  for (i = 0;  i < k;  i++, t++)
      if (!strncmp(t, "SAM", 3) ||  !strncmp(t, "SAN", 3)) j++;
  printf("%i\n", j);


  i = j = k/2;
  while (txt[--i] != txt[++j])  ;
    printf("%i %i\n", i, j);


  t = &txt[j-i];
  if (strncmp(t, txt, 6)) printf("Martin\n");
  else                    printf("Odegaard\n");

  return 0;
}
```

b) **Hva blir utskriften fra følgende program (litt hjelp: det blir 5 linjer):**

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

char farger[][10]= { "Blaa", "Gronn", "Gul", "Hvit", "Lilla", "Rod", "Svart"};

struct Lag  {
  char nvn[40];
  int  farge, antall;
};

void H22Funk1(const struct Lag lag)  {
  printf("%s, %s    ", lag.nvn, farger[lag.farge]);
}

int H22Funk2(const struct Lag* l1, const struct Lag* l2)  {
  return ((l2->antall > l1->antall) ? l2->antall : l1->antall);
}

bool H22Funk3(const struct Lag l1, const struct Lag* l2)  {
  return (!strcmp(farger[l1.farge], farger[l2->farge]));
}

struct Lag H22Funk4(const struct Lag l1, const struct Lag l2)  {
  struct Lag lag;
  strcpy(lag.nvn, l1.nvn);  lag.farge = l2.farge;
  lag.antall = (l1.antall > l2.antall) ? l1.antall : l2.antall;
  return lag;
};

int H22Funk5(const struct Lag* l1, const struct Lag* l2)  {
  if (strlen(l1->nvn) >= 4  &&  strlen(l2->nvn) >= 4)  {
     if (l1->nvn[0] > l2->nvn[0])  return 19;
     else {
        if (l2->nvn[2] < l1->nvn[1] &&  l1->nvn[2] < l2->nvn[1]) return 71;
        else if (l1->nvn[3] != l2->nvn[4]) return 94;
        else return 0;
     }
  } else return 0;
}

int main()  {
  struct Lag lag1 = { "Arsenal", 5, 60260 },
             lag2 = { "Tottenham", 3, 62850 },
             lag3 = { "Norwich", 2, 27244 },
             lag4 = { "Manchester United", 5, 74879 },
             lag5;

  H22Funk1(lag1);   H22Funk1(lag3);   H22Funk1(lag2);   printf("\n");

  printf("%i\n", H22Funk2(&lag1, &lag2));

  printf("%i %i\n", H22Funk3(lag2, &lag3), H22Funk3(lag4, &lag1));

  lag5 = H22Funk4(lag2, lag3);  H22Funk1(lag5);
                                printf("%i\n", lag5.antall);

  printf("%i %i\n", H22Funk5(&lag2, &lag3), H22Funk5(&lag1, &lag4));

  return 0;
}
```

Oppgave 2 (70%)

Les *hele* teksten for denne oppgaven (2a-2g) *nøye*, før du begynner å besvare noe som helst. Studér vedlegget (som også er på utdelte papirer), som inneholder mange viktige opplysninger som du trenger/skal bruke. Legg spesielt merke til `#define`/`const`, `enum`en, structen med datamedlemmer, funksjoner, globale variable, `main()` og *tre ferdiglagde funksjoner*. Husk også på de ferdiglagde funksjonene for å lese inn data på `LesData.h`. Bruk *alt* dette *svært* aktivt.

Det holdes orden på ulike hendelser (overnattinger, aktiviteter/gjøremål og bespisning) under en biltur (f.eks. i Alpene).

Datastrukturen

Datastrukturen består (se vedlegget) av arrayen `gHendelser`. I denne er indeksene fra 0 til `gAntallHendelser-1` i bruk. Vedlegget angir også hvilke datamedlemmer structen inneholder.
NB: Legg merke til hvilket format `dato` og `klokken` er på.

*Vedlegget inneholder alt du trenger av struct, datamedlemmer og globale variable for å løse denne eksamensoppgaven. Dessuten er **prototyper** for alle funksjoner også ferdig deklarert/definert.*

Oppgaven

a)

Skriv innmaten til `void nyHendelse()` **og**
`void hendelseLesData(struct Hendelse* hendelse)`

Den første funksjonen kommer med en egen melding om det allerede er fullt med hendelser. I motsatt fall skriver den ut den nye hendelsen sitt nummer (*en* høyere enn den sist registrerte). Det opprettes en ny hendelse, som legges inn bakerst. Denne leser selv inn verdier til alle sine datamedlemmer (vha. den andre funksjonen), og antall hendelser telles opp med en.
Den andre funksjonen leser inn tre tekster, to tall (dato: 101-1231, klokken: 0-2359), samt typen hendelse (bruk funksjonen som lages i oppgave 2C). Brukeren kan skrive inn klokken 0 (null) som verdi når tidspunktet er uinteressant (f.eks. ankomst til et hotell).
NB: Vi har ellers ingen sjekk på dato eller klokkeslett sin gyldighet. F.eks. vil dato 1167 og klokken 1583 være i rett intervall, men selvsagt finnes ikke 67.nov eller kl.15:83.

b)

Skriv innmaten til `void skrivAlt()` **og**
`void hendelseSkrivData(const struct Hendelse* hendelse)`

Den første funksjonen går igjennom alle registrerte hendelser og skriver deres nummer (nummerert fra 1 (en) og oppover) og *alle* hendelsenes data vha. den andre funksjonen.
Den andre funksjonen skriver ut *alle* hendelsens data på to linjer. Datoen skal skrives på formen DD/MM ('/' imellom) og klokkeslettet som TT:MM (':' imellom). Bruk ferdiglaget funksjon for å skrive ut `type`.

c)

Skriv innmaten til `enum Type lesHendelsesType()`

Funksjonen leser inn fra brukeren, og godtar *kun*, bokstavene A, F, H, I, M og S.
Dvs. forbokstavene i `enum`-verdiene. Deretter returnerer den aktuell enum-verdi ut fra bokstaven.

d) **Skriv innmaten til** `void nyHendelse2()`

Her skal det lages en ny versjon av funksjonen fra oppgave 2A. Dette for at brukeren selv kan velge at en ny hendelse ikke nødvendigvis alltid må legges inn bakerst, men kan bli smettet inn mellom andre. Det kommer først en melding om det er fullt. Ellers blir *alle* hendelsene skrevet ut på skjermen. Brukeren får deretter selv angi hvilket nummer den nye hendelsen skal inn som. Dette er et tall i intervallet 1 til `gAntallHendelser`+1 (dersom den tross alt ønskes innlagt helt bakerst). Deretter flyttes om nødvendig aktuelle hendelser opp ett hakk/en indeks, den nye opprettes, får sine data lest inn (jfr. andre funksjon i oppgave 2A), og settes på plass/smettes inn.
NB: Vi *forutsetter* at brukeren på denne måten sørger for at hendelser på samme dato blir liggende samlet, og datoene er i stigende rekkefølge. Men det er ingen sjekk på dette i selve koden.

e) **Skriv innmaten til** `void skrivGitteHendelser()`

Er ingen hendelser registrert kommer det en egen melding. I motsatt fall blir brukeren bedt om å skrive inn 'D' (dato) eller 'T' (type). Velges 'D' spørres det etter en dato. Deretter gås det gjennom *alle* hendelsene, og *alle* hendelser på denne datoen blir skrevet ut med *alle* sine data. For 'T' spørres det etter en hendelsestype (jfr. oppgave 2C), og *alle* hendelser av denne typen får *alle* sine data skrevet ut. I begge tilfellene skal det komme en egen melding om *ingen* aktuelle hendelser er å finne/blir skrevet ut.

f) **Skriv innmaten til** `void datoMedFlestHendelser()`

Er ingen hendelser registrert kommer det en egen melding. I motsatt fall finner funksjonen ut hvilken dato det er flest hendelser på, og skriver ut dette antallet og datoen (på formen DD/MM). Er det flere datoer med dette antallet, skrives bare den første datoen ut.
NB: Husk at fra oppgave 2D har vi forutsatt at alle like datoer ligger samlet!

g) **Skriv innmaten til** `void lesFraFil()` **og**

```
void hendelseLesFraFil(FILE* inn, struct Hendelse* hendelse)
```

Funksjonene sørger til sammen for at *alle hendelsene* blir lest inn fra filen «HENDELSER.DTA», `gAntallHendelser` *skal* ligge aller først på filen. **Filformatet** ellers bestemmer du helt selv, men ***skal*** **oppgis i besvarelsen**.

Annet (klargjørende):

- Det skal altså ikke skrives kode/funksjoner for å f.eks: skrive til fil, editere/flytte/slette en hendelse.
- Gjør dine egne forutsetninger og presiseringer av oppgaven, dersom du skulle finne dette nødvendig. Gjør i så fall klart rede for disse *i starten* av din besvarelse av aktuell deloppgave.
- **NB:** Det skal *ikke* brukes C++-kode, dvs. slikt som f.eks: string-klassen, kode fra STL, templates eller andre større hjelpebiblioteker. Men, de vanligste includer brukt i hele høst er tilgjengelig.

Lykke til på (eksamens)turen!
FrodeH

Vedlegg til PROG1001, desember 2022: Halvferdig programkode

```
#include <stdio.h>               //  printf, scanf, FILE
#include <stdlib.h>              //  sizeof, malloc
#include <string.h>              //  strcpy, strlen
#include <stdbool.h>             //  bool, true, false
#include "LesData.h"             //  Verktøykasse for lesing av diverse data

#define   MAXHENDELSER   100     ///<  Max. antall hendelser på turen.
const int STRLEN  =       80;    ///<  Max. tekstlengde.

/**
 *  Type hendelse (ulike type hendelser som kan skje på turen).
 */
enum Type { Attraksjon, Fjelltopp, Hotell, Idrettsarena, Museum, Spisested };

/**
 *  Hendelse (med navn, sted, webside, dato, klokkeslett og hendelsestype).
 */
struct Hendelse {             //  Hendelsens:
  char* navn,                 //     - navn
      * sted,                 //     - sted
      * webside;              //     - webside/URL
  int   dato,                 //     - dato, på formen: MMDD (måned/dag)
        klokken;              //     - klokkeslett, på formen: TTMM (time/min)
  enum Type type;             //     - jfr. ovenstående 'enum'
};

void skrivMeny();                                                  //  | Ferdig-
void skrivTypeTekst(const enum Type type);                         //  | laget:
enum Type typeFraChar(const char tegn);                            //  |
void nyHendelse();                                                 //  Oppgave 2A
void hendelseLesData(struct Hendelse* hendelse);                   //  Oppgave 2A
void skrivAlt();                                                   //  Oppgave 2B
void hendelseSkrivData(const struct Hendelse* hendelse);           //  Oppgave 2B
enum Type lesHendelsesType();                                      //  Oppgave 2C
void nyHendelse2();                                                //  Oppgave 2D
void skrivGitteHendelser();                                        //  Oppgave 2E
void datoMedFlestHendelser();                                      //  Oppgave 2F
void lesFraFil();                                                  //  Oppgave 2G
void hendelseLesFraFil(FILE* inn, struct Hendelse* hendelse);      //  Oppgave 2G

int    gAntallHendelser;             ///<  Antall hendelser hittil registrert.
struct Hendelse* gHendelser[MAXHENDELSER];  ///<  Alle hendelsene.

/**
 *  Hovedprogrammet:
 */
int main()  {
    char kommando;

    lesFraFil();                                              //  Oppgave 2G
    skrivMeny();   kommando = lesChar("\nØnske");

    while (kommando != 'Q') {
        switch (kommando) {
          case 'N':  nyHendelse2();             break;    //  Oppgave 2A og 2D
          case 'S':  skrivAlt();                break;    //  Oppgave 2B
          case 'G':  skrivGitteHendelser();     break;    //  Oppgave 2E
          case 'F':  datoMedFlestHendelser();   break;    //  Oppgave 2F
          default:   skrivMeny();               break;
        }
        kommando = lesChar("\nØnske");
    }
    return 0;
}
```

```
/**
 *  Skriver/presenterer programmets muligheter/valg for brukeren.
 */
void skrivMeny() {
    printf("\nFØLGENDE KOMMANDOER ER LOVLIG:\n");
    printf("\tN   = Ny hendelse\n");
    printf("\tS   = Skriv alle hendelser\n");
    printf("\tG   = skriv Gitte hendelser\n");
    printf("\tF   = skriv datoen med Flest hendelser\n");
    printf("\tQ   = Quit/avslutt\n");
}


/**
 *  Skriver ut en Type på skjermen som tekst.
 *
 *  @param   type  -  Hendelsestypen som skrives ut
 */
void skrivTypeTekst(const enum Type type)  {
   switch (type)  {                          //  Skriver enum-verdi som tekst:
     case Attraksjon:    printf("Attraksjon    ");   break;
     case Fjelltopp:     printf("Fjelltopp     ");   break;
     case Hotell:        printf("Hotell        ");   break;
     case Idrettsarena:  printf("Idrettsarena  ");   break;
     case Museum:        printf("Museum        ");   break;
     case Spisested:     printf("Spisested     ");   break;
   }
}


/**
 *  Gjør om en lovlig bokstav til en aktuell enum-verdi.
 *
 *  @param    tegn  -  Bokstav/tegn som skal konverteres til enum-verdi
 *  @return   Enum-verdi ut fra en bokstav/tegn
 */
enum Type typeFraChar(const char tegn)  {
  switch (tegn)  {
    case 'A':  return  Attraksjon;              //  Ut fra bokstav returneres
    case 'F':  return  Fjelltopp;               //    aktuell enum-verdi:
    case 'H':  return  Hotell;
    case 'I':  return  Idrettsarena;
    case 'M':  return  Museum;
    case 'S':  return  Spisested;
  }
}


/**
 *  Oppgave 2A - Legger inn (om mulig) en ny hendelse inn i datastrukturen.
 *
 *  @see      hendelseLesData(...)
 */
void nyHendelse()  {                                        /*   LAG INNMATEN  */  }


/**
 *  Oppgave 2A - Leser inn ALLE datamedlemmene i EN hendelse.
 *
 *  @param   hendelse  -  Hendelsen som får innlest sine data
 *  @see     lesHendelsesType()
 */
void hendelseLesData(struct Hendelse* hendelse)  {          /*   LAG INNMATEN  */  }
```

```
/**
 *  Oppgave 2B - Skriver ALT om ALLE hendelsene.
 *
 *  @see   hendelseSkrivData(...)
 */
void skrivAlt()  {                                          /*   LAG INNMATEN  */  }


/**
 *  Oppgave 2B - Skriver ALT om EN hendelse ut på skjermen.
 *
 *  @param   hendelse  -  Hendelsen som skrives ut
 */
void hendelseSkrivData(const struct Hendelse* hendelse)  {  /*   LAG INNMATEN  */  }


/**
 *  Oppgave 2C - Leser forbokstaven i de ulike Typene, returnerer aktuell enum.
 *
 *  @return  Enum-verdien ut fra forbokstaven i selve enum-verdien
 */
enum Type lesHendelsesType()  {                             /*   LAG INNMATEN  */  }


/**
 *  Oppgave 2D - SMETTER INN (om mulig) en ny hendelse inn i datastrukturen.
 *
 *  @see      skrivAlt()
 *  @see      hendelseLesData(...)
 */
void nyHendelse2()  {                                       /*   LAG INNMATEN  */  }


/**
 *  Oppgave 2E - Skriver aktuelle hendelser på ønsket dato eller av type.
 *
 *  @see      hendelseSkrivData(...)
 */
void skrivGitteHendelser()  {                               /*   LAG INNMATEN  */  }


/**
 *  Oppgave 2F - Skriver (den første) datoen med flest hendelser.
 */
void datoMedFlestHendelser()  {                             /*   LAG INNMATEN  */  }


/**
 *  Oppgave 2G - Leser ALLE hendelsene inn fra fil.
 *
 *  @see   hendelseLesFraFil(...)
 */
void lesFraFil() {                                          /*   LAG INNMATEN  */  }


/**
 *  Oppgave 2G - Leser ALT om EN hendelse inn fra fil.
 *
 *  @param  inn       -  Filen det skal leses fra
 *  @param  hendelse  -  Hendelsen som får innlest sine data
 */
void hendelseLesFraFil(FILE* inn, struct Hendelse* hendelse) {  /* LAG INNMATEN */  }
```