NTNU
Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

Kontinuasjonseksamensoppgave i PROG1003 – Objekt-orientert programmering

Faglig kontakt under eksamen:	**Frode Haug**
Eksamensdato:	**4.august 2026**
Eksamenstid (fra-til):	**09:00-13:00 (4 timer)**
Hjelpemiddelkode/Tillatte hjelpemidler:	**I - Alle trykte og skrevne.** (kalkulator er *ikke* tillatt)
Annen informasjon:	
Målform/språk:	**Bokmål**
Antall sider (inkl. forside):	**9**

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 <iostream>
#include <string>
using namespace std;

class A  {
  protected:
    string txt;

  public:
    A(const string str)
       {  txt = str;  }
    bool funk3(const string s)
       {  return (txt.find_first_of(s, 0) != string::npos);  }
    virtual void funk4(const char c)
       {  txt = (c <= 'K') ? "Arsenal" : "Wrexham";  }
    virtual void funk5()  {  cout << 'A';  }
};

class B : public A {
  private:
    char ch;
    int tall;

  public:
    B(const string s, const char c, const int t) : A(s)
       {  ch = c;  tall = t;  }
    void funk1()
       { cout << (txt + ',' + ch + '-' + to_string(tall));  }
    void funk2()
       {  txt.clear();  for (int i = 0; i < 5; i++) txt += ch;   ch = 'E';  }
    void funk4(const char c)
       {  ch = (c < 'S') ? 'X' : 'Y';   A::funk4(c); }
    void funk5()  {  cout << 'B';  }
};


int main() {
  B  b1("25/1-ManU", 'U', 116);
  B* b2 = new B("24/1-QPR", 'C', 82);

  b1.funk1();  cout << ' ';  b2->funk1();  cout << '\n';

  b2->funk2();   b2->funk1();  cout << '\n';

  cout << b1.funk3("-/") << ' ' << b1.funk3("u") << ' '
       << (*b2).funk3("-/") << '\n';

  b2->funk4('P');   b2->funk1();  cout << '\n';

  delete b2;
  A* a = new A("Banan");
  b2 = reinterpret_cast <B*> (a);
  b2->funk5();    cout << '\n';

  return 0;
}
```

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

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
using namespace std;


int main()  {
  vector <int> tall  {  29, 6, 22, 7, 17  };
  vector <int> tall2 (3, 4);
  map <char, int> hvem;

  hvem['F'] = tall[1];    hvem['S'] = tall[3];    hvem['M'] = tall[0];
  hvem['I'] = tall[2];    hvem['A'] = tall[4];

  cout << tall.size() << ' ' << tall2.size() << ' ' << hvem.size() << '\n';

  for (auto const & val : hvem)
      cout << val.first << '-' << val.second << ' ';    cout << '\n';

  auto it = hvem.begin();
  while (it != hvem.end())
    cout << (it++)->first << ' ';   cout << '\n';

  for (int i = 0;  i < tall2.size();  i++)
      tall.push_back(tall2[i]);
  for (int i = 0;  i < tall.size();  i++)
      cout << tall[i] << ' ';     cout << '\n';

  sort(tall.begin(), tall.end());
  auto it2 = unique(tall.begin(), tall.end());
  tall.erase(it2, tall.end());
  for (auto const & val : tall)
      cout << val << ' ';    cout << '\n';

  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 inneholder mange viktige opplysninger som du trenger/skal bruke.
Legg spesielt merke til `const`'ene, klassene med arv, datamedlemmer og (ferdiglagde/private) funksjoner inni klassene, global variabel, `main` og `skrivMeny`.
Husk også på funksjonene på `LesData2.h`. Bruk *alt* dette *svært* aktivt.

Det skal holdes orden på en restaurant sine ulike kunder/gjester på ulike bord og deres bestillinger/kjøp. (Bordbooking/-reservasjon og betalingen for måltider foregår begge i andre programmer/systemer.)

Datastrukturen

Datastrukturen består *kun* av `vectoren gBordene`. Denne er alltid `ANTBORD` lang, dvs. det faste antall bord i restautranten. Bordene er nummerert 1 til `ANTBORD`, liggende i vectoren på indeksene 0 (null) til `ANTBORD`-1. Om bordet er i bruk, pekes det til aktuelt kunde-objekt, eller pekes det til `nullptr`. Legg merke til `mapen kjop` inni `Kunde`.

Oppgaven

a) **Skriv innmaten til** `void skrivEttBord()`
og de to virtuelle `void skrivData()`
Den første funksjonen spør først om et bordnummer. Er dette ikke i bruk, kommer det en melding. I motsatt fall skrives *absolutt alle* bordets data ut på skjermen (vha, de andre funksjonene). Har det ikke forekommet noen kjøp/bestillinger hittil, kommer det en egen melding om dette. Er det en `Stamgjest` ved bordet, skrives bl.a. teksten «STAMGJEST».

b) **Skriv innmaten til** `void nyKunde()` **og de to virtuelle** `void lesData()`
Den første funksjonen spør først om det bordnummer. Er dette allerede i bruk, kommer det en egen melding. I motsatt fall leses *og sikres* det om den som kommer er en vanlig `Kunde` eller en `Stamgjest`. Aktuelt objekt opprettes, *alle* dets data (unntatt verdier i `kjop`, jfr. oppg.2D) leses inn (vha. de andre funksjonene), og til slutt legges objektet inn i datastrukturen.
NB1: Det skal altså registreres *kun ett* navn ifm. hvem som sitter på bordet.
NB2: Det er servitørens ansvar å sørge for at det er plass til alle ved/på bordet.

c) **Skriv innmaten til** `bool Kunde::lovligKjop(string & str)`
Denne *private* funksjonen skal sjekke og returnere om `str` er på et lovlig format.
Lovlig format er: Minst to tegn lang, der det er *en* startende bokstav ('F', 'H', 'D' eller 'R), men denne *kan* det hende er i lowercase, og må da gjøres om til stor bokstav (derfor er `str` referanse-overført). Deretter *skal* `str` kun inneholde sifre ('0'-'9'), men *ikke* ha en innledende '0' (null). **NB:** 'FHDR' står for: Forrett, Hovedrett, Dessert og dRikke.

d) **Skriv innmaten til** `void kjop()` **og** `void Kunde::kjopForetas()`
Den første funksjonen leser et bordnummer, og kommer med en melding om det er tomt.
I motsatt fall kalles den andre funksjonen, og til slutt blir *hele* datastrukturen *alltid* skrevet ut til fil (jfr. oppgave 2F). Den andre funksjonen leser kjøpskoder *inntil en ulovlig skrives*.
Koden bør altså være: *En* av bokstavene 'FHDR' tett etterfulgt av et tall, f.eks: F4, H12, D7.
Deretter leses det inn antallet som er bestilt/kjøpt av denne retten/drikke. Dette kan maksimum være antallet som sitter ved bordet. Når alle kjøpene/bestillingene er registrert, skrives antallet på dem ut. Husk at om en identisk kode allerede er kjøpt, så skal antallet av denne koden *plusses på*, og *ikke* legges inn som et nytt kjøp.

NB1: Vi forutsetter at *alle* servitørene kan *alle* numrene på *alle* rettene/drikkevarene, evt. enkelt kan slå opp og finne dette i menyen.
NB2: Du kan bare kalle/bruke funksjonen i 2C, selv om du ikke har kodet/laget denne.

e) **Skriv innmaten til** `void bestillingerTotalt()` **og**
`void Kunde::bestillinger(const char kode, map <string, int> & best)`
Den første funksjonen går igjennom de fire lovlige kjøpsbokstavene. *For hver bokstav skrives totalt antall kjøp/bestillinger som for øyeblikket alle bordene til sammen har for alle de ulike retter/drikker som starter med denne bokstaven.* Dette bør samles inn ved å lage en lokal `map` inni funksjonen. Den andre funksjonen kalles med bl.a. denne som parameter, og `map`en nullstilles (cleares) mellom hver bokstav. Altså skrives først alle kjøp/bestillinger som starter med 'F' ut, deretter de med 'H' osv. For hver bokstav stanser programmet, og venter på at brukeren skriver ENTER. Er det ingen kjøp for en bokstav, så kommer det en egen melding.

f) **Skriv innmaten til** `void skrivTilFil()`,
de to virtuelle `void skrivTypeTilFil(ofstream & ut)`
og de to virtuelle `void skrivTilFil(ofstream & ut)`
Funksjonene sørger til sammen for at *hele* datastrukturen skrives ut til «RESTAURANT.DTA». Den andre funksjonene skriver *kun* ut noe om hva slags type objekt som nå kommer.
Alle bordene og deres numre *skal* skrives ut, også de som det *ikke* sitter noe ved (er tomme). Formatet på filen bestemmer du helt selv.

g) **Skriv innmaten til** `void lesFraFil()`
og de to constructorene med `inn` som parameter
Disse funksjonene sørger til sammen for at *hele* datastrukturen leses inn fra filen som det ble skrevet til i forrige oppgave. **Oppgi her filformatet du bruker.**

Annet (klargjørende):

- Ordene «kunde» og «gjest» brukes synonymt. Det samme gjør «kjøp» og «bestilling».
- Dette programmet har faktisk ingen funksjonalitet for at noe drar fra et bord (at det blir ledig/ tomt igjen). Dette skyldes at det da bare hadde vært å delete kunden, så lenge all betaling foregår i et annet system/program.
- Du *skal* bruke `LesData2.h` ifm. løsningen av denne oppgaven. Du får nok også bruk for (deler av) pensumets temaer innen STL, men *ikke* bruk saker fra STL, templates eller stoff/biblioteker utenfor pensum.
- Gjør dine egne forutsetninger og presiseringer av oppgaven, dersom du skulle finne dette nødvendig. Gjør i så fall klart rede for disse der det gjelder i besvarelsen din av oppgaven(e).

Bon Appétit!

FrodeH

Vedlegg til PROG1003, august 2026: Halvferdig programkode

```
#include <iostream>             //  cout, cin
#include <fstream>              //  ifstream, ofstream
#include <string>
#include <vector>
#include <map>
#include <cctype>               //  toupper, isdigit
#include "LesData2.h"           //  Verktøykasse for lesing av diverse data
using namespace std;


const int ANTBORD =   30;   ///<  Antall nummererte bord i restauranten.
const int RABATT  =   30;   ///<  Max. prosentrabatt for stamgjest/-kunde.
const int ANTALL  =   20;   ///<  Max. antall på ett bord.
const int RETTER  =   50;   ///<  Max. antall valg innen ulike retter/drikker.


/**
 *  Baseklassen 'Kunde' med dens navn, totalantallet vedkommendes bord og
 *  kjøpene (mat og drikke) som bordet totalt har foretatt.
 */
class Kunde  {
  private:
    string navn;            //  Hovedkundens/bestillerens navn.
    int  antall;            //  Totalantallet som er tilknyttet bordet.
    map <string, int> kjop; //  Kjøpene/bestillingene som bordet har foretatt.
    bool lovligKjop(string & str) const;                        //  Oppgave 2C
  public:
    Kunde()  { antall = 0;  }                                   //  Ferdiglaget
    Kunde(ifstream & inn);                                      //  Oppgave 2G
                                                                //  Oppgave 2E:
    void bestillinger(const char kode, map <string, int> & best) const;
    void kjopForetas();                                         //  Oppgave 2D
    virtual void lesData();                                     //  Oppgave 2B
    virtual void skrivData() const;                             //  Oppgave 2A
    virtual void skrivTilFil(ofstream & ut) const;              //  Oppgave 2F
    virtual void skrivTypeTilFil(ofstream & ut) const;          //  Oppgave 2F
};


/**
 *  Avledet klasse 'Stamgjest' med prosentrabatt og annen spesiell service.
 */
class Stamgjest : public Kunde  {
  private:
    int rabatt;             //  Rabatt (i prosent).
    string service;         //  Beskrivelse av ekstra service vedkommende får.
  public:
    Stamgjest()  {  rabatt = 0;  }                              //  Ferdiglaget
    Stamgjest(ifstream & inn);                                  //  Oppgave 2G
    virtual void lesData();                                     //  Oppgave 2B
    virtual void skrivData() const;                             //  Oppgave 2A
    virtual void skrivTilFil(ofstream & ut) const;              //  Oppgave 2F
    virtual void skrivTypeTilFil(ofstream & ut) const;          //  Oppgave 2F
};


void bestillingerTotalt();                                      //  Oppgave 2E
void kjop();                                                    //  Oppgave 2D
void lesFraFil();                                               //  Oppgave 2G
void nyKunde();                                                 //  Oppgave 2B
void skrivEttBord();                                            //  Oppgave 2A
void skrivMeny();                                               //  Ferdiglaget
void skrivTilFil();                                             //  Oppgave 2F
```

```
vector <Kunde*> gBordene(ANTBORD);          ///<  ALLE bordene i restauranten.


/**
 *  Hovedprogrammet.
 */
int main() {
  char valg;
  int nr;

  lesFraFil();                                                    //  Oppgave 2G

  skrivMeny();
  valg = lesChar("\nKommando");

  while (valg != 'Q') {
     switch (valg) {
       case 'S': skrivEttBord();              break;          //  Oppgave 2A
       case 'N': nyKunde();                   break;          //  Oppgave 2B
       case 'K': kjop();                      break;          //  Oppgave 2D
       case 'B': bestillingerTotalt();        break;          //  Oppgave 2E
       default:  skrivMeny();                 break;
     }
     valg = lesChar("\nKommando");
  }

  skrivTilFil();                                                  //  Oppgave 2F

  cout << "\n\n";

  return 0;
}



// ------------------------------------------------------------------------
//                        DEFINISJON AV KLASSE-FUNKSJONER:
// ------------------------------------------------------------------------

/**
 *  Oppgave 2C - Sjekker og returnerer om kjøpskode er korrekt.
 *
 *  @param    str  -  Tekst som skal sjekkes, og evt. få sitt 1.tegn endret
 *  @return   Om 'str' er en lovlig kjøpskode eller ei (på formen: Xnnn)
 */
bool Kunde::lovligKjop(string & str) const  {  /*    LAG INNMATEN    */     }


/**
 *  Oppgave 2G - Leser ALLE data om EN kunde/ETT bord inn fra fil.
 *
 *  @param   inn  -  Filen det leses inn fra
 */
Kunde::Kunde(ifstream & inn)  {                /*    LAG INNMATEN    */     }


/**
 *  Oppgave 2E - Finner og legger alle bestillinger/kjøp av en viss kategori
 *               inn i en referanse-overført map.
 *
 *  @param   best  -  Map med alle bestillinger med en viss kode
 */
void Kunde::bestillinger(const char kode, map <string, int> & best) const  {
                                               /*    LAG INNMATEN    */     }
```

```
/**
 *  Oppgave 2D - Registrerer nye kjøp på et bord/kunde inntil ulovlig skrives.
 *
 *  @see   lovligKjop(...)
 */
void Kunde::kjopForetas()  {                    /*    LAG INNMATEN    */      }

/**
 *  Oppgave 2B - Leser inn 'Kunde' sine hoveddata.
 */
void Kunde::lesData()  {                        /*    LAG INNMATEN    */      }

/**
 *  Oppgave 2A - Skriver ut ALT om 'Kunde' på skjermen.
 */
void Kunde::skrivData() const  {                /*    LAG INNMATEN    */      }

/**
 *  Oppgave 2F - Skriver ALLE objektets data ut til fil.
 *
 *  @param   inn  -  Filen det skrives ut til
 */
void Kunde::skrivTilFil(ofstream & ut) const  {  /*    LAG INNMATEN    */    }

/**
 *  Oppgave 2F - Skriver objekt-typen ut til fil.
 *
 *  @param   inn  -  Filen det skrives ut til
 */
void Kunde::skrivTypeTilFil(ofstream & ut) const  {   /*   LAG INNMATEN   */   }


// ---------------------------------------------------------------------------

/**
 *  Oppgave 2G - Leser inn ALLE data om EN kunde/ETT bord fra fil.
 *
 *  @param   inn  -  Filen det leses inn fra
 */
Stamgjest::Stamgjest(ifstream & inn) : Kunde(inn)  {   /*   LAG INNMATEN    */   }

/**
 *  Oppgave 2B - Leser inn 'Stamgjest' sine data.
 */
void Stamgjest::lesData()  {                          /*    LAG INNMATEN    */      }

/**
 *  Oppgave 2A - Skriver ut ALT om 'Stamgjest' på skjermen.
 */
void Stamgjest::skrivData() const  {                  /*    LAG INNMATEN    */      }

/**
 *  Oppgave 2F - Skriver ALLE objektets data ut til fil.
 *
 *  @param   inn  -  Filen det skrives ut til
 */
void Stamgjest::skrivTilFil(ofstream & ut) const  {  /*    LAG INNMATEN    */      }

/**
 *  Oppgave 2F - Skriver objekt-typen ut til fil.
 *
 *  @param   inn  -  Filen det skrives ut til
 */
void Stamgjest::skrivTypeTilFil(ofstream & ut) const  {   /*   LAG INNMATEN   */   }
```

```
// ------------------------------------------------------------------------
//                        DEFINISJON AV ANDRE FUNKSJONER:
// ------------------------------------------------------------------------

/**
 *  Oppgave 2A - Skriver ut ALT om EN kunde/ETT bord.
 *
 *  @see   Kunde::skrivData(),  Stamgjest::skrivData()
 */
void skrivEttBord()  {                                  /*    LAG INNMATEN    */      }


/**
 *  Oppgave 2B - Legger inn (om mulig) en ny kunde på et bord.
 *
 *  @see   Kunde::Kunde(),    Stamgjest::Stamgjest()
 *  @see   Kunde::lesData(),  Stamgjest::lesData()
 */
void nyKunde()  {                                       /*    LAG INNMATEN    */      }


/**
 *  Oppgave 2D - Registrerer (om mulig) nye kjøp på et bord.
 *
 *  @see   Kunde::kjopForetas()
 *  @see   skrivTilFil()
 */
void kjop()  {                                          /*    LAG INNMATEN    */      }


/**
 *  Oppgave 2E - Finner og skriver alle nåværende aktuelle bestillinger/kjøp.
 *
 *  @see   Kunde::bestillinger()
 */
void bestillingerTotalt()  {                            /*    LAG INNMATEN    */      }


/**
 *  Oppgave 2F - Skriver ALT om ALLE kundene/bordene til fil.
 *
 *  @see   Kunde::skrivTypeTilFil(...),   Stamgjest::skrivTypeTilFil(...)
 *  @see   Kunde::skrivTilFil(...),       Stamgjest::skrivTilFil(...)
 */
void skrivTilFil()  {                                   /*    LAG INNMATEN    */      }


/**
 *  Oppgave 2G - Leser ALLE bordenes nåværende data/status inn fra fil.
 *
 *  @see   Kunde::Kunde(...),   Stamgjest::Stamgjest(...)
 */
void lesFraFil()  {                                     /*    LAG INNMATEN    */      }


/**
 *  Skriver programmets menyvalg/muligheter på skjermen.
 */
void skrivMeny()  {
  cout << "\nFolgende kommandoer er tilgjengelige:\n"
       << "   S  - Skriv ALT om ETT bord\n"
       << "   N  - Ny gjest/kunde ankommer\n"
       << "   K  - Kjop/bestillinger registreres\n"
       << "   B  - Bestillinger totalt for oyeblikket\n"
       << "   Q  - Quit / avslutt\n";
}
```