BOKMÅL

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

KONTERINGSEKSAMEN I FAG
TDT4102 Prosedyre og objektorientert programmering

Onsdag 6. august 2008
Kl. 09.00 – 13.00

Faglig kontakt under eksamen:
Trond Aalberg, tlf (735) 9 79 52 / 976 31 088

Tillatte hjelpemidler:

- Absolute C++, Savitch
- C++ Pocket Reference

Sensurdato:
Sensuren faller 3 uker etter eksamen og gjøres deretter kjent på http://studweb.ntnu.no/.

Prosentsatser viser hvor mye hver oppgave teller innen settet.

Merk: Alle programmeringsoppgaver skal besvares med kode i C++.

Lykke til!

Generelt for alle oppgaver

- Merk at ikke alle deloppgaver krever at de foregående er løst, så ikke hopp over de resterende deloppgavene om én blir for vanskelig.
- Der det spesifikt står definer eller deklarer er vi kun interessert i funksjondeklarasjonen eller klassedeklarasjoner. Der det står implementer skal du vise implementasjon av funksjoner.
- Der det spesifikt spørres etter forklaringer og beskrivelser er det et av kravene at dette skal være med.
- NB! Det er ikke et krav at alle funksjoner/klasser skal være komplette, implementer kun det som er nødvendig for å besvare en oppgave!

OPPGAVE 1 (30 %): Funksjoner og operatorer

a) Implementer funksjonen **`int findfirst(int a[], int size, int value)`**som leter gjennom en tabell (array) av heltall og returnerer indeksen til første forekomst av **`value`**. Hvis **`value`** ikke finnes i indeksen skal du returnere -1. Parameter **`size`** angir størrelsen på tabellen.

b) Implementer en funksjon med navn **`rightShift`** som du kan bruke til å endre verdiene til tre heltallsvariabler (**`int`**). Etter funksjonskallet **`rightSshift(a, b, c)`** skal variabel **`b`** ha fått verdien til **`a`**, variabel **`c`** skal ha fått verdien til **`b`** og variabel **`a`** skal ha verdien til **`c`**. Funksjonen skal ikke returnere noe (**`void`**).

c) Implementer funksjonen **`bool XOR(bool a, bool b)`** som returnerer true når den ene parameteren er **`true`** og den andre er **`false`**. NB! I implementasjonen får du kun lov til å benytte de logiske operatorene AND (**`&&`**), OR (**`||`**) og NOT (**`!`**) – ingen andre operatorer er tillatt. Funksjonen skal gi følgende resultat:

```
XOR(true, true) -> false
XOR(false, false) -> false
XOR(true, false) -> true
XOR(false, true) -> true
```

Implementer en funksjon **`void divideByTwo(int x)`** som ved hjelp av rekursjon skriver ut tallrekken (inklusive tallet selv) du får ved gjentatte deleoperasjoner med 2. Tallene skal skrives ut med det minste tallet først. Hvis funksjonen kalles med tallet 64 skal den skrive ut tallrekken: 1, 2, 4, 8, 16, 32, 64. Hvis funksjonen kalles med tallet 60 skal den skrive ut 15, 30, 60 osv.

d) I funksjons-headeren i koden under under er det benyttet referanse-parameter. Forklar hva dette er og vis hvilke endringer som du må gjøre i swap hvis du i stedet hadde benyttet peker-parameter i swap-dekarasjonen.

```
void swap(int& x, int& y){
   int temp = y;
   y = x;
   x = temp;
}


int main(){
   int x = 1;
   int y = 2;
   swap (x, y);
   //nå skal x ha verdien 2 og y ha verdien 1
}

// Hva må du endre i swap hvis header i stedet
// hadde vært swap(int* x, int* y) og
// funksjonskallet hadde vært swap(&x, &y)?
```

OPPGAVE 2 (40 %): Klasser og operatorer

I denne oppgaven skal du lage klasser for kort i en kortstokk. Et kort har en farge (spar, hjerter, ruter, kløver) og en verdi (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, knekt, dame, konge, ess). Du skal implementere en supertype kalt **`Card`** og subtypene **`Spade`**, **`Heart`**, **`Diamond`**, **`Club`** (hhv. spar, hjerter, ruter kløver). Lovlige kortverdier skal implementeres ved hjelp av en enumeration (**`enum`**) kalt **`Value`** bestående av verdiene: **`TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING, ACE`**.

a) Deklarer enum-typen **`Value`** og vis hvordan du kan sørge for at det er samsvar mellom konstanten og heltallsverdien for denne; dvs. TWO med verdien 2, THREE med verdien 3, osv, JACK er 11, QUEEN er 12, KING er 13 og ACE er 14.

b) Implementer supertypen **`Card`** og en eller flere av subtypene **`Spade`**, **`Heart`**, **`Diamond`** og **`Club`**. Alle kortene skal ha en innkapslet medlemsvariabel for verdi (av typen **`Value`**) og kortverdien skal kun settes av konstruktøren. Vis fornuftig bruk av arv og konstruktører og forklar koden din kort.

c) Du ønsker å kunne skrive ut en tekstreng bestående av verdi og farge for hvert kort ved hjelp av operatoren **`cout`** og operatoen **`<<`**. Kortverdien skal skrives ut som heltall for kortene 2-10 og teksten ”Jack”, ”Queen”, ”King” og ”Ace” for billedkortene og essene. Kortfarge skal skrives ut med teksten ”Spades”, ”Heart”, ”Diamonds” og ”Clubs” (se eksempelet under). Implementer operatoren og nødvendig tillegsfunksjonalitet i klassene slik at du oppnår utskriften som forklart i koden over. NB! Her er vi ute etter både implementasjon av operatoren og bruk av virtuelle metoder. Gi en kort forklaring på koden din (hvorfor du velger å gjøre det slik du gjør).

```
Club c1(TWO);
Heart c2(JACK);

cout << c1 << endl; // skal skrive ut ”2 of Clubs”
cout << c2 << endl; // skal skrive ut “Jack of Hearts”
```

d) Implementer et eksempel på en sammenligningsoperatorene slik at du kan teste om et kort er større enn, mindre enn eller er lik et annet kort. Også i denne oppgaven er vi ute etter både operatorimplementasjonen og annen funksjonalitet som er nødvendig for å få dette til å fungere.

Et kort sammenlignes både på grunnlag av verdi og farge. Hvis kortene har ulik verdi er det verdien som teller, hvis de har samme verdi er det fargen som avgjør:
spar **>** hjerter **>** ruter **>** kløver.

```
Club c1(TWO);
Heart c2(JACK);

if (c1 > c2){
   cout << c1 << "er høyere enn" << c2 << endl;
}else{
   cout << c1 << "er ikke høyere enn" << c2 << endl;
}
```

OPPGAVE 3 (30 %): Set av kort

Du skal implementere en klasse kalt **`Cards`** som skal kunne brukes til organisere et sett av kort-objekter (Card-objekter fra forrige oppgave). I denne oppgaven er det viktig å skjønne kravene, og du skal kun implementere noen av egenskapene til klassen.

`Cards`-klasse skal ha funksjoner for å legge til kort og ta ut kort fra **`Cards.`** I tillegg skal klassen ha funksjoner for å organisere kortene i tilfeldig rekkefølge (stokke om) eller sortere kortene. **`Cards`**-objekter kan enten inneholde et begrenset antall Card-objekter eller et ubegrenset antall (avhengig av hvilken konstruktør som kalles). Generelle krav til klassen er at rekkefølgen til kortene er signifikant. Når du legger til kort skal du kunne hente dem ut i samme rekkefølge som de ble lagt til, og hvis du stokker kortene får de en ny rekkefølge. Hvis kortene sorteres skal de etterpå være i sortert rekkefølge. Cards-klassen skal også kunne støtte unike sett av kort (alle kort må være forskjellig) eller ikke-unike sett av kort (kortene kan være like - for eksempel hvis de kommer fra 2 eller flere kortstokker). En parameter i konstruktør-kallet avgjør om instansen sjekker på unikhet eller ikke. Følgende kode er en ufullstendig deklarasjon av Cards-klassen og i deloppgavene under er det spesifisert hva du skal svare på og gjøre i denne oppgaven (kun deler av klassen skal implementeres).

```
class Cards{

public:
   Cards(bool unique, int maxsize);
   Cards(bool unique);
   void add(....) throw (UniqueException, MaxSizeException);
   void sort;
   void shuffle;
};
```

a) I implementasjonen av klassen Cards har du bruk for noen medlemsvariabler. Forklar hvilke variabler du trenger, hvilke datatyper du bruker og bruk av pekervariabler kontra verdi-variabler. Ta hensyn til kravene som er beskrevet i introduksjonen til denne oppgaven og begrunn dine valg.

b) Implementer konstruktørene til klassen. Unique-paramenteren brukes for å spesifisere om Cards-objektet kan inneholde 2 eller flere like kort (samme farge og verdi) eller om Cards-klassen kun skal kunne inneholde unike kort (alle kort skal ha forskjellig farge og verdi). Hvis konstruktøren som har en maxsize-parameter benyttes skal objektet som instansieres kun inneholde opp til maxsize antall kort. Hvis konstruktøren uten maxsize benyttes er det ingen begrensinger på antall kort som kan lagres.

c) Implementer **`add`**-funksjonen. I denne oppgaven er vi spesielt interessert i unntaksmekanismen. For Cards-objekter med maxsize skal unntak av typen **`MaxSizeException`** kastes hvis du prøver å legge til flere kort en lovlig. For Cards-objekter som er instansierte med **`unique`** = **`true`** skal det kastes et unntak av typen **`UniqueException`** hvis du prøver å legge til et kort som har samme farge/verdi som et annet kort som allerede finnes.

d) Implementer en mekanisme for å hente ut og fjerne kortene fra et Cards-objekt ett for ett. Mekanismen skal kunne brukes til å gå igjennom alle kortene i den rekkefølgen de til enhver tid er organisert i. Mekanismen trenger ikke ta hensyn til stokking og sortering siden vi overlater til de som bruker klassen å sørge for at de ikke stokker eller sorterer mens de går igjennom sekvensen av kort. Her er vi primært ute etter hvilke(n) funksjon(er) du vil basere deg på deg og det holder at du skriver funksjonsdeklarasjonene og implementer/forklarer basisprinsippet ditt.

e) Du har bestemt deg for at Cards-klassen er en nyttig type klasse som kan generaliseres og benyttes til mange forskjellige typer objekter og ønsker å lage en template-klasse kalt MyCollectionClass. Vis med kode og forklar hvordan du kan lage en template-klasse som har de samme funksjoner som Cards-klassen. Forklar eventuelle endringer du må gjøre i koden og mulige begrensinger som kan finnes i bruken av denne template-klassen.