Inlämningsuppgifter

1. ***Gymnastiktävling*** Skriv ett C++ program som avgör en tävling i gymnastik. Tre tävlande deltar i tävlingen. De får sina poäng av 3 olika domare. Poängen ska ligga mellan 0 och 10. Skapa en array över poäng för varje tävlande, där varje element i arrayen är en domares poäng. Programmet ska simulera domarnas poänggivning med slumptal. Poängen ska summeras till en totalpoäng för varje tävlande. Slutligen ska programmet skriva ut både varje tävlandes totalpoäng och utropa tävlingens vinnare.

Ledning:

Steg 1 Skapa tre arrays, en till varje tävlande.

Steg 2 Skapa tre variabler för de tävlandens totalpoäng, en för varje tävlande, och initiera dem till 0.

Steg 3 Repetera ***3.9 Hantering av slumptal*** i ***kursboken***, sid 69.

Steg 4 Skriv en **`for`**-loop med tre varv för att fylla arrays från ***Steg 1*** med slumpvärden i intervallet mellan 0 och 10. Så kan domarnas poänggivning simuleras.

Steg 5 Summera varje tävlandes poäng till en totalpoäng för varje tävlande. Tilldela dessa totalpoäng till de variabler du skapade i ***Steg 2***.

Steg 6 Bestäm den största totalpoängen.

Steg 7 Skriv ut både de tävlandes totalpoäng och vilken av dem som vunnit gymnastiktävlingen.

****

2. ***Master Mind*** är ett litet spel som låter användaren gissa ett slumpmässigt genererat fyrsiffrigt heltal genom att leda spelaren med en inbyggd hjälpprocedur vars regler är beskrivna nedan.

Ledning:

Börja med att behandla *fyr*siffriga heltal som en serie av *fyra en*siffriga tal dvs som en array av heltal med fyra element.

Skriv först en funktion med huvudet **`void skapa(int secretNo[])`** som ska generera det hemliga fyrsiffriga talet och lagra det i en **`int`**-array, säg **`secretNo`**, med 4 element. Varje element i arrayen **`secretNo`** kan genereras som ett slumptal mellan 0 och 9. Dessutom ska funktionen **`skapa()`** kontrollera spelets regel enligt vilken alla fyra siffror måste vara olika.

Skriv sedan en funktion med huvudet **`void help(int guessedNo[], int secretNo[])`** som ska bearbeta spelarens gissning enligt följande regler:

För varje rätt siffra på rätt plats från vänster till höger skrivs ut ett	**`R`**
För varje rätt siffra på fel plats från vänster till höger skrivs ut ett	**`S`**
För varje fel siffra från vänster till höger skrivs ut ett frågetecken	**`?`**

Är t.ex. det hemliga talet 4693 och spelaren gissar 7498, så erhålls hjälpen:

`? S R ?`

När hjälpen skriver ut **`RRRR`** har spelaren lyckats och programmet avslutas med att skriva ut ett lämpligt meddelande. Skriv ett program som tillåter flera spelomgångar.

Frivilligt: Skriv en pekarversion av projektet *Master Mind*.

****

3. ***Palindrom*** Skriv en funktion **`bool palindrom(char*)`** som avgör om en sträng är en *palindrom* eller ej. T.ex. är ”rar”, ”död” och ”radar” palindromer då de inte ändras när de läses baklänges. Men även en text som ”ni talar bra latin” är en palindrom om man ignorerar mellanslagen. Vid behandling av sådana texter låt programmet först ta bort alla mellanslag.

****

4. ***Automaten*** Skriv ett program som simulerar interaktionen med en automat. Följande klass beskriver det som pågår i automaten efter att användaren lagt in pengar och valt en vara:

```
class Automataktion
{
   string varunamn;
   float  pris;
   float  betalning;
   float  vaxel;

public:
   Automataktion(float pengar, char vara)
   {
      switch(vara)
      {
         . . .
      }
      betalning = pengar;
      vaxel = betalning - pris;
   }



   void vaxel_i_mynt()
   {
         . . .
```

```
    }
};
```

switch-satsen i konstruktorn ska tilldela datamedlemmarna **varunamn** och **pris** värden beroende på valet av vara och skriva ut ett meddelande om inlagt belopp samt varan som ska levereras. Metoden **vaxel_i_mynt()** är till för att dela upp växeln i ”tillåtna” myntslag (endast 10-kr, 5-kr, 1-kr och 50-öringar) och skriva ut hur många av varje ”tillåtet” myntslag som ska ges tillbaka.

Börja i **main()** med att skriva en meny över alla varor samt priserna. Låt sedan användaren lägga in pengar. Läs in beloppet till en **float**-variabel. Låt användaren även välja en vara. Sedan kan ett objekt av den ovan deklarerade klassen **Automataktion** skapas och samtidigt konstruktorn **Automataktion()** anropas. Vid detta anrop skickas de inlästa värdena till det inlagda beloppet och den valda varan som aktuella parametrar till **Automataktion()**. Efter att objektet skapats och datamedlemmarna initierats via konstruktorn, kan metoden **vaxel_i_mynt()** anropas.

Komplettera programmet med att ta hand om en eventuellt felaktig eller otillräcklig betalning från användarens sida.

****

5. ***Anställda*** Modellera en arvhierarki över olika typer av anställda och använd på ett polymorft sätt metoden **salary()** i alla klasser för att beräkna lönen för de olika anställdtyper. Skriv en superklass **Anställd** som ärvs av tre subklasser **FastAnställd**, **Säljare** och **TimAnställd**. Varje subklass ska ha privata datamedlemmar, en konstruktor, set/getmetoder till varje ny datamedlem, en **toString()**-metod som skriver ut den anställdas typ, namn och anställningsnummer samt metoden **salary()** som i varje subklass definierar om (överskuggar) superklassens metod **salary()**. Typiska nya privata datamedlemmar till klassen **FastAnställd** kan vara **månadslön**, till klassen **Säljare** kan tänkas **bonus** och **säljvolym** och till klassen **TimAnställd** t.ex. **timlön** och **antalTimmar**. Skriv dessutom en subklass **FastSäljare** som ärver klassen **Säljare** och har den nya privata datamedlemmen **fastLön**. Testa dina subklasser i **main()** som skrivs i en testklass.