Labb 4 Algoritmer

Uppgift: Använd STL (Standard Template Library) för att implementera några algoritmer genom att använda de färdiga algoritmerna och containerna i STL.

I flera av uppgifterna nedan behövs testdata som är oordnad. Det finns två sätt att framställa dessa:

- Man kan generera slumpvisa data - lätt när det gäller heltal, svårare när vi ska sortera Personer.
- Man kan ta en lista med personer och sedan ”blanda” dem – detta går att göra med alla objekt som kan sorteras. Det finns en standard funktion för detta: random_shuffle

Några användbara std funktioner och i vilka bibliotek de finns:

Random numbers: C++ bibliotek är krångligt, det gamla C varianten enkel:

```
#include <cstdlib>
int r = rand();
```

och nu är r ett random positivt heltal (0 <= r <= RAND_MAX == 32767)

Fylla en container med något:

```
iota(v.begin(), v.end(), 101);
```

fyller v med 101, 102, 103 etc.

```
random_shuffle(v.begin(), v.end());
```

“blandar” v

Lambdafunktioner är anonyma funktioner t.ex.:

```
[](int x){return i%10==7; }
```

//ger true för alla tal som har 7 som sista siffra.

I <vector> finns vector klassen som bl.a. har metoderna:

- push_back
- erase

I <algorithm>

- sort
- stable_sort
- remove

Uppgifter:

Uppgift 1: Sortering

Uppgift 1a: Sortering av heltal i vector

Gör en funktion som:

1. Skapar en std::vector<int> med heltal i slumpvis ordning
2. Skriver ut containern.
3. Sorterar den med std::sort
4. Skriver ut containern.

Uppgift 1b: Sortering av heltal i int[]

Gör en funktion som:

1. Skapar en c-array (int []) med heltal i slumpvis ordning
2. Skriver ut containern.
3. Sorterar den med std::sort
4. Skriver ut containern.

Uppgift 1c: Sortering av heltal i vector i sjunkande ordning

Gör en funktion som:

1. Skapar en std::vector<int> med heltal i slumpvis ordning
2. Skriver ut containern.
3. Sorterar den baklänges med std::sort (använd rbegin och rend)
4. Skriver ut containern.

Uppgift 1d: Sortering av heltal i int[] i sjunkade ordning

rbegin och rend ger ”reverse_iterrators” som resultat, detta är ett objekt som innehåller en pekare men överlagrarar alla operationer så de går ”baklänges” t.ex så gör överlagringen av ++ operatorn operationen – på den underliggande pekaren.
Om vi har en c-array (int[]) så finns inger enkelt sätt att göra det men vi akn i stället ge sort en jämförelse funktion som tredje parameter:

```
bool Greater(int x, int y) {return x>y; }
```

sortera vektorn från uppgift 1b med hjälp av funktionen Greater och std::sort.

Uppgift 2a: Sortering av personer

Det är naturligt att tänka sig en sorteringsfunktion i ett personregister. Den kan fungera genom att sortera den c-array som personerna ligger ifrån Laboration 3.
I PersonReg klassen lägg till funktionerna:

```
Person* PersonReg::begin() {return personer; }

Person* PersonReg::end() {return personer + currenNumberOfPersons; }
```

Om din implementation av PersonReg inte har de aktuella personerna samlade i början av arrayen så sortera hela arrayen, även de tomma personerna.

För att kunna sortera något så måste det kunna jämföras, lägg till jämförelseoperatorn i Person klassen:

```
bool Person::operator<(const Person& that) {
    return this->name < that.name;
}
```

Gör en funktion som:

1. Skapar ett PersonReg som innehåller ett antal personer. Se till att du har många personer med samma namn men olika adress.
2. Kör random_shuffle.
3. Skriver ut containern.
4. Sorterar den med std::sort
5. Skriver ut containern.

Uppgift 2b: Sortera baklänges på adress

Det är vår jämföresle operator som bestämmer hur sort gör.
Ändra definitionen så att den i stället jämför tvärtom på adresserna. T.ex.

```
bool Person::operator<( const Person& that) {
    return this->adress > that.adress;
}
```

Gör en funktion som:

1. Skapar ett PersonReg som innehåller ett antal personer. Se till att du har många personer med samma namn men olika adress.
2. Kör random_shuffle.
3. Skriver ut containern.
4. Sorterar den med std::sort
5. Skriver ut containern.

Uppgift 3:

Ett problem med att algoritm-biblioteket använder sig av iteratorer och inte själva containers är att en iterator inte kan ta bort element i en container. Gör ett program som:

1. Skapar en container med slumpvisa tal (C-objekt) eller bestämda tal med slumpvis ordning. (random_shuffle)
2. Skriver ut containern.
3. Tar bort alla jämna tal med hjälp av STL (använd ”remove_if” och ”erase”).
4. Skriver ut containern.

För att anropa remove_if gör en funktion i stil med:

```
bool Even(int i) { return i%2==0; }
```