Laboration 3, Personregister

Implementera ett enkelt personregister som består av två klasser och ett testprogram.
Klassen Person innehåller namn och adress. Klassen PersonReg innehåller själva registret.
Allmänt gäller för hela uppgiften att om inte annat anges nedan så är det fritt fram att använda standardbiblioteket. I std::string så finns t.ex. `compare` och `replace` mm.

Krav:

1. Ni ska skriva alla loopar med pekare, inte med heltal:
   Om vi antar att vi har deklarerat:

   ```
   int storlek=10;
   person* personer;
   ```

   och sedan i initieringen kört:

   ```
   personer=new Person[storlek];
   ```

   så skulle en loop som skriver ut hela arrayen vara:

   ```
   for (Person* ptr = personer; ptr!=personer+storlek; ++ptr) {
       ptr->Print();
   }
   ```

2. Registret ska vara strukturerat som en array av Person, inte som en array av Personpekare. Ni får inte använda std::vector eller liknande utan det ska var "Person[]"

3. Det ska finnas nödvändig debugkod för att skriva ut ev. minnesläckor och det får inte finnas några minnesläckor. Koden för att hitta minnesläckor finns på Canvas.

4. Const på alla rätta ställen. Dvs. om en parameter inte förändras i metoden skall den vara const.

5. Packa data i början av registret och håll reda på hur många personer som finns i registret.

6. Ingen kod i .h filerna utan den ska vara i .cpp filerna.

Uppgift 1. Personregister

PersonReg objektet ska ha följande metoder:

- Konstruktor som tar max antal personer
  `PersonReg(int maxSize)`
- Lägg till ett personobjekt, tar en pekare till ett personobjekt som parameter, detta objektet får sedan kopieras in på rätt plats i arrayen.
- Ta bort ett personobjekt, tar en pekare till det personobjekt som ska tas bort som parameter. Obs. att pekaren skall peka direkt på det objekt som skall tas bort.
- Sök på namn, ger en pekare till objektet som svar.
- Print som anropar Print funktionen för alla Personobjekt.
- Destruktor, det som körs när man gör delete på PersonRegistret
  `~PersonReg()`

PersonReg objektet ska lagra personerna i en private variabel som deklareras:

- `Person* personer;  //pekare till själva registret`

Personklassen ska innehåll minst namn och adress. Den behöver även en konstruktor och en Print funktion.

Sen ska ett litet testprogram skrivas om skapar ett personregister och provar system - ska skriva ut resultat på ett begripligt sätt.

Det finns ett testprogram ”PersonRegTest.cpp” som ni kan använda om ni vill, speciellt för Fritextsökning nedan så ger den ett tips på hur det ska fungera.

1.1 Inläsning

Det finns en fil med några personer: *PersonExempel.txt*
Nedan är ett exempel på hur man kan läsa in den filen till sitt register.
Det är lite olika i olika Visual Studio versioner i vilket directory programmet ”är” när det startar. I en del versioner så är det på samma plats som ”exe”-filen ligger, i andra på samma ställe som källkoden till programet, dvs. där .cpp filerna ligger.
När jag kör med Visual Studio 2017 och lägger *PersonExempel.txt* i den mapp som .cpp filerna ligger så räcker det med "PersonExempel.txt". Om du i stället vill lägga det högre upp i mapp hierarkin så skriv "/../PersonExempel.txt".

```
bool ReadReg(PersonReg& reg, string fileName) {
	string line;
	ifstream myfile(fileName);
	if (myfile.is_open())
	{
		while (getline(myfile, line))
		{
			while (line.length() == 0 && getline(myfile, line))
				;
			string name(line);
			string adress;
			getline(myfile, adress);
			reg.LäggTill(&Person(name, adress));
		}
		myfile.close();
		return true;
	}
	else {
		cout << "Unable to open file";
		return false;
	}
}
```

Uppgift 2. Fritextsökning

Bygg ut PersonReg med en fritextsökning som tar en sträng som argument och om den strängen finns i namnet eller adressen så blir det träff. Med fritextsökning så blir det vanligt att vi får flera träffar och vi väljer att göra det genom att kunna upprepa sökning och söka efter nästa träff. Sökfunktionen skall ta en sträng som förta argument och som andra argument en pekare till den senaste träffen. För att kunna starta om sökningen från början av registret så sätt andra parametern till nullptr. Exempel på anrop:

```
Person* x1 = myPersonReg.SökFritt(”olle”, nullptr); //x1 blir första träffen
Person* x2 = myPersonReg.SökFritt(”olle”, x1);      //x2 blir andra träffen
Person* x3 = myPersonReg.SökFritt(”olle”, x2);      //x3 blir tredje träffen
```

Observera att Sökfritt ska lämna `nullptr` som resultat om inget mer hittas.

Tänk på att ni ska testa att programmet klarar upprepade sökningar i stil med ovan.

Uppgift 3. Arv

Vi kanske vill kunna lägga till telefonnummer till en del personer. Definiera en ny klass PersonMedTel som ärver från Person:

```
class PersonMedTel: publik Person {
    std::string nummer;
    void Print();
    …
}
```

För att få detta att fungera rätt så måste Print i Person deklareras virtual. Även destruktorn måste deklareras virtual i Person för annars kan det uppstå minnesläckor när fel destruktor anropas. Prova sedan att tillverka ett objekt av typen PersonMedTel och anropa Print funktionen, t.ex.

```
Person * myPerson = new PersonMedTel(…);
myPerson -> Print();    //Om Print är virtual i Person så anropas
PersonMedTel::Print
```

Prova sen att lägga in objektet i PersonReg och skriv ut PersonReg.
Fast ni lagt in ett PersonMedTel så kommer ändå inga telefonnummer att skrivas ut, förklara vad som hänt.

Diverse:

- Om ni vill ha svenska bokstäver skriv först i main:
  `locale::global(locale("swedish"));`
  Det kommer inte att fungera perfekt, att få inmatning, utmatning och konstanter i koden att fungera för svenska tecken är svårt!

- När ni skapar ett nytt projekt så bocka av precompiled headers! Det underlättar felsökning.