Mittuniversitetet
MID SWEDEN UNIVERSITY

Laboration 1 - Structar
Objektbaserad programmering i C++

Jimmy Åhlander*

26 april 2021

Innehåll

1 Introduktion

Primitiva datatyper är användbara för att lagra enkel data. Ett simpelt medelvärde kan till exempel lagras i en `double`, ett villkor i en `bool` och en bokstav i en `char`. Ibland önskar vi lagra mer komplicerad data. Till exempel en person som har ett namn, en ålder och ett telefonnummer. För att lagra sammansatt data behöver vi egendefinierade datatyper.

*jimmy.ahlander@miun.se. Avdelningen för informationssystem och -teknologi (IST).
Baserad på laborationen *att hantera textposter med hjälp av structar* av Martin Kjellqvist.

Det finns flera olika sätt att skapa egendefinierade datatyper i C++. Ett sådant är genom en *struct*. En struct tillåter dig att skapa en egen sammansatt datatyp som består av andra primitiva datatyper eller till och med andra structar.

I den här laborationen kommer du deklarera och definiera ett par structar för att lagra personuppgifter som en användare kan söka i.

2 Mål

Efter genomförd laboration ska du förstå hur du kan skapa och använda structar med överlagrade operatorer. Nya begrepp som används i laborationen är *struct*, *medlemsvariabel* (medlemmar) och *operatoröverlagring*.

Följande lärandemål är kopplade till laborationen. Du ska kunna:

- använda funktionsöverlagring.
- använda objekt av andra typer som datamedlemmar i en klass.

3 Teori

En struct i C++ [1] är i dagsläget inget annat än en lätt förklädd klass. De är syntaktiskt utbytbara med varandra och skiljer sig endast avseende medlemmarnas synlighet; i en struct syns alltid medlemmarna utåt, om vi inte sagt något annat. Synligheten går att styra med nyckelorden *public* och *private*, men du kan bortse från dem i denna labb.

En struct må fungera som en klass men de används på helt olika sätt. Klasser används när vi vill specifiera både data och ett beteende för ett objekt, till exempel en spelkaraktär med data som health och med funktioner för att förflytta karaktären. En struct används istället bara för att inkapsla data.

Inför den här laborationen bör du ha läst om följande:

- struct, se [1].
- operatoröverlagring, se [2, s. 556-560].
- `std::size_t`
- `std::string`
- `std::vector`

Du kan också komma att ha nytta av:

- `std::getline`
- `std::find`, eller
- `std::string::find`

- `std::count`
- `tolower`
- `toupper`

4 Uppgift

Din uppgift är att skapa ett program som läser in personuppgifterna ifrån textfilen `names.txt`[1] till en vektor av structen `person`. Textfilen har följande struktur:

```
Namn\n
Personnummer\n
Gatuadress, postnummer[2 blanksteg]stad[utfyllnad]\n
```

Du måste skapa två olika structar för att lagra personuppgifterna: `person` och `address`.

Structen `person` ska ha följande medlemmar:

- `name:std::string`,
- `id:std::string`,
- `location:address`

Structen `address` ska ha följande medlemmar:

- `street:std::string`,
- `zip:int`,
- `city:std::string`

Användaren av ditt program ska därefter genom en meny kunna söka efter namn eller stad och erhålla ett resultat.

Din uppdragsgivare har tillhandahållit följande API (*application programming interface*). Ditt uppdrag är att implementera de specificerade funktionerna för att lösa uppgiften.

```
1   /*
2   * Reads from file <filename> and returns the results in
3   *  a vector. Uses overloaded operator>>.
4   * @param filename The path to the file to be read.
5   * @returns vector<person> containing all the people listed
6   *  in the file. An empty vector is returned if no people
7   *  are found.
8   */
9   std::vector<person> read_file(string filename);
10
11
```

[1] URL till textfil: http://apachepersonal.miun.se/~jimahl/DT019G/names.txt

```
12      /*
13      * Searches in vector <haystack> for names containing the
14      *  substring <name_part>. Note that <name_part> is a
15      *  a substring meaning exact matches are not necessary.
16      * The search is case insensitive.
17      * @param haystack The vector containing the people to
18      *  search through.
19      * @param name_part The name to look for.
20      * @returns the number of people where name_part is found and
21      *  occurs at least once.
22      */
23      size_t find_in_names(vector<person> haystack, string name_part)
        ;
24
25      /*
26      * Searches in vector <haystack> for people living in a
27      *  particular <city>.
28      * The search is case insensitive.
29      * Exact matches only (except case).
30      * @param haystack The vector containing the people to
31      *  search through.
32      * @param city The city to look for.
33      * @returns a vector containing all persons with exactly
34      *  matching cities.
35      */
36      vector<person> find_person_from_city(vector<person> haystack,
        string city);
37
38      /*
39      * Overloads the operator >> to read from an istream, e.g. a
40      * file, to a person. Sets all the attributes of a person.
41      * @param in The istream data is read from.
42      * @param p The person to store data in.
43      * @returns a reference to the stream (for operator chaining).
44      */
45      istream& operator>>(istream& in, person& p);
```

När programmet startar har alltså användaren tre olika alternativ:

1. Sök del av personnamn.
2. Sök efter personer i stad (endast exakta matchningar).
3. Avsluta.

Användaren ska erhålla följande resultat från vardera alternativ:

1. Användaren anger del av ett namn och får tillbaks:

   [Antal matchningar]\n

2. Användaren anger en stad och får för varje matchad person tillbaks:

   [Personnummer], [Namn], [Stad]\n

3. Programmet avslutas.

Restriktioner

När du får i uppdrag att implementera ett API efter en kravspecifikation förväntas det normalt av dig att du följer API:t till punkt och pricka. Om andra systemutvecklare utgår ifrån samma API och förväntar sig kunna anropa dina funktioner i sina egna program är det inte särskilt behjälpligt om du ändrar funktionsnamn, parametrar eller returvärden på eget bevåg. Tänk dig själv att du försöker anropa `std::string::find` utifrån dokumentationen för stdlib och att det inte funkar för att utvecklaren istället döpt funktionen till `findstr`. I arbetslivet kommer du bli jagad med högaffel av dina kollegor om de måste skriva om sin kod för att din kod inte följer överenskomna specifikationer. Du får gärna göra tillägg, t.ex. i form av extra funktioner men inte förändra det som redan fastställts.

Observera därför följande krav:

- Båda strukturerna måste deklareras och definieras med exakt de namn och exakt de medlemmar och datatyper som angivits.
- Samtliga funktioner måste implementeras exakt som specificerats. Notera funktionsnamn, parametrar och returtyper.
- Funktionerna i API:t ska varken printa något eller fråga om input. Det sköts av anroparen.
- Funktionerna får inte kringgås utan ska användas för sina tilltänkta syften - ingen allt-i-main-gulaschsoppa.
- Inläsningen från fil till person måste ske genom att inströmsoperatorn (>>) överlagrats.
- Sökning efter städer är exakt. Sökningen efter namn är inte det.
- Utdata till användaren måste vara exakt som angivet ovan.
- Utdata får inte vara destruerat. Nikki heter Nikki och inte nikki eller NIKKI.
- Textfilen får inte modifieras på något sätt.[2]
- Koden ska vara kommenterad till en rimlig grad. Förklara varför, inte vad.

Följande är inte krav, men är fördelaktiga för lösningen:

- Stöd för å, ä och ö.
- Felhantering för när användaren skriver felaktig input.
- Överlagrad inströmsoperator för `address`.
- Överlagrad utströmsoperator för `person`. Kan med fördel användas för utskriften efter stadssökningen.

[2] Textfilen är sparad i UTF-8 med UNIX line endings. Ladda ner den och öppna med valfri *vettig* editor. Använd absolut inte copy, paste.

5 Genomförande

Börja med att skapa dina structar. När dina structar är på plats kan du påbörja arbetet med `read_file()`. Öppna filströmmen i `read_file()` och försök först att läsa från strömmen till en instans av en person med inströmsoperatorn, t.ex. `file >> p`. Detta kommer att anropa inströmsoperatorn som du nu måste överlagra. I inströmsoperatorn kommer du behöva saxa, strängmanipulera och konvertera de olika raderna i filströmmen för att på ett snyggt sätt tilldela alla personens medlemmar. Större delen av arbetet ligger alltså här. När du vet att du kan läsa in data för en person (kom ihåg att testa!) upprepar du processen för hela textfilen och sparar alla personerna i en vektor. Den vektorn kan nu `read_file()` returnera för att senare användas till de olika sökfunktionerna. Att implementera sökfunktionerna blir därför det sista steget tillsammans med användarmenyn.

6 Examination

Redovisa laborationen under ett av laborationstillfällena som ges under kursens gång genom att presentera funktionaliteten och förklara koden. När du fått klartecken från labbhandledaren att redovisningen är godkänd kan du lämna in i inlämningslådan på lärplattformen där en slutgiltlig bedömning utförs av examinator.

Din inlämning ska bestå av en eller flera cpp-filer. I de fall där du arbetar med flera filer bör du döpa huvudfilen till `main.cpp`. Varje fil ska innehålla en header med åtminstone ditt namn och aktuellt datum.

6.1 Bedömning och återkoppling

Uppgiften bedöms med betygen *Godkänd (G)*, *Komplettering (Fx)*, och *Underkänd (U)*. Bedömningen baseras i huvudsak på huruvida fullständig funktionalitet uppnåtts och om du under den muntliga delen av examinationen kunnat förklara innebörden av din kod.

Återkoppling erhåller du i första hand muntligt under redovisningen. Normalt lämnas ingen skriftlig återkoppling vid godkänt resultat för denna uppgift.

Referenser

[1] cplusplus.com team, “Data structures.” [Online]. Available: http://www.cplusplus.com/doc/tutorial/structures/

[2] S. B. Lippman, *C++ primer*, 5th ed. Upper Saddle River, N.J.: Addison-Wesley, 2013.