Mittuniversitetet
MID SWEDEN UNIVERSITY

DT019G Objektbaserad programmering i C++

Projekt:
Labyrintlaborerande

Martin Kjellqvist*

maze.tex 1056 2013-05-16 06:04:48Z martin

Innehåll

1 Introduktion

Labyrinter kommer i många olika former. Vi kommer att betrakta labyrinter som inte innehåller loopar, dessa labyrinter har en unik lösning för vägen från start till slut. Konstruktion av en labyrinter innehåller flera intressanta delar som är användbara till flera ändamål. En labyrint utan loopar kan betraktas som ett träd där startpunkten är roten och de olika vägarna är desss grenar.

Detta projekt är utformat så att det ska ta runt 3 dagar att genomföra.

2 Syfte

Syftet är att fördjupa dina kunskaper inom och utveckla din vana för programmering. Du kommer att få fördjupade kunskaper om hur strukturerna stack och kö fungerar i konkreta tillämpningar.

3 Teori

3.1 Labyrintrepresentation

En labyrint representeras enkelt genom en matris med symboler exempelvis:

```
struct labyrinth
{
  const char WALL = '*';
  const char PATH = ' ';
  const size_t SIZE = 100;
  char maze[SIZE][SIZE];
}
```

Istället för att använda en klumpig array kan man använda `vector<vector<char>>`.

En enkel 7x7 labyrint.

```
*******
S+    *
* *** *
*   * *
*** ***
*     X
*******
```

Lägg märke till att storleken ska vara udda i en dylik representation. `+` märker ut startpunkten.

3.2 Labyrintgenerering

Att generera en labyrint görs enkelt genom en DFS, depth first search, metod:

```
- Lägg startpunkten (1, 1) till en stack.
- Ta en punkt från stacken märk punkten som besökt. Detta är nuvarande punkt P.
  - I slumpmässig ordning lägg till alla obesökta grannar till P till stacken.
    - Tag första elementet på stacken och ta bort väggen mellan P och
    elementet.  Det nya elementet är nu P. Märk P som besökt.
```

Metoden använder en stack och går på djupet först. Man kan även använda en kö med liknande resultat. Strukturen växer då på bredden.

3.3 Labyrintlösare

Att hitta vägen ut ur en labyrint följer samma mönster:

```
- Lägg startpunkten (1, 1) till en stack.
- Ta en punkt från stacken märk punkten som besökt. Detta är nuvarande punkt P.
  - Lägg till alla nåbara obesökta grannar till stacken.
    - Tag första elementet på stacken. Har du nått slutet är du klar.  Stacken
    visar vägen du tagit.
```

4 Uppgift

Projektet är uppdelat i olika betygssteg. För betyget D krävs att du genomför grunduppgiften perfekt utan anmärkningar enligt granskningsprotokollet. Vid maximalt två anmärkningar ges betyget E, vid fler anmärkningar betyget F.

För betygen C-A krävs de extrauppgifter som ges efter grunduppgiften, för ett betyg krävs att samtliga föregående extrauppgifter även är genomförda. (Även dessa ska vara utan anmärkningar, vid anmärkning sänks betyget fortfarande till E.)

4.1 Grunduppgift

Skapa ett program som är avsett att anropas från kommandoraden. Programmet ska antingen konstruera eller lösa en labyrint. Dialogen löser du på valfritt sätt om inte extrauppgift A görs.

Om du väljer att lösa en labyrint tas labyrinten emot som en textfil och lösningen skrivs ut till `cout` på lämplig form, exempelvis:

```
*******
Sx    *
*x*** *
*xxx* *
***x***
*  xxxX
*******
```

Om du väljer att skapa en labyrint ska användaren kunna ange storleken på labyrinten.

Programmet ska inte under några omständigheter krasha. Felhantering måste vara användarvänlig. Om någon indata inte är giltig ska användaren få beskrivande felmeddelanden och kunna åtgära felet.

Koden ska vara välkommenterad med beskrivande kommentarer som gör koden lättläst.

4.2 Extrauppgift C

Programmet ska kunna utföra både konstruktion och lösning. Programmet ska kunna ta emot indata och utdata via kommandoradsargument.

Exempelvis

```
$./maze -input maze.txt
*******
Sx    *
*x*** *
*xxx* *
***x***
*  xxxX
*******
$./maze -size 7
*******
S     *
* *** *
*   * *
*** ***
*     X
*******
```

Programmet ska vara uppdelat i lämpliga header- och implementationsfiler med relaterade klasser/funktioner. Sorteringsrelaterade göromål hanteras av en separat klass. Klasser och funktioner ska inte vara specialiserade i onödan.

4.3 Extrauppgift A

Programmet ska använda sig av `getopt`. Se projektbeskrivningen för sort. getopt är ett GNU-bibliotek och kan hämtas för Windows. Programmet har ingen interaktiv dialog med användaren alls. Allt görs med argument till programmet.

Följande argument ska stödjas.

- `--version | -v`. Skriver ut versionsnummer.
- `--help | -h`. Skriver ut tillgängliga argument.
- `(--size | -s)N`. Skapa en labyrint med storleken `N`.
- `(--columns | -c)W`. Skapa en labyrint med bredden `W`.
- `(--rows | -r)H`. Skapa en labyrint med höjden `N`.
- `(--input | -i)file`. Använd filen `file` som indata.
- `(--output | -o)file`. Använd filen `file` för utdata. Annars `cout`.
- `--check | -b`. Skriver ut endast Solution foundöm en lösning finnes, annars Solution not found".

Felhanteringen ska göras enligt följande:

- Om ingen lösning finns ska programmet inte presentera någon labyrint som resultat.
- Om `file` inte finns eller inte har korrekt indata skrivs ett felmeddelande till `cerr`.
- Om argumenten ger felaktiga eller motsägelsefulla instruktioner till programmet ska ett meningsfullt felmeddelande skrivas ut.

Om ett fel inträffat ska main returnera `EXIT_FAILURE` annars `EXIT_SUCCESS`.

Extra fokus på struktur och välskriven kod. Endast `main` ligger som fri funktion.

5 Examination

Ditt färdiga projekt ska först granskas av en annan student på kursen med hjälp av det granskningsprotokoll som återfinns på lärplattformen. Efter att du åtgärdat alla påpekanden som uppkommit under granskningen ska du tillsammans med granskaren redovisa ditt projekt och granskningen för en av kursens lärare vid något av de inbokade redovisningstillfällena. När du redovisat och fått godkänt laddar du upp din källkod med tillhörande byggskript[1] till inlämningslådan i lärplattformen.

[1] Notera att det är obligatoriskt att ha ett byggskript för programmet.