Inlämningsuppgift 1 - Prioritetskö
DV1490

Christian Nordahl, Andreas Jonasson, & Joakim Ståhle Nilsson

March 29, 2019

1 Uppgift

Ni skall implementera den abstrakta datatypen (ADT) prioritetskö, där det element med **lägst** värde är det med **högst** prioritet, som implementerar följande API:

- `void enqueue(T element);` - Lägger till ett element i er prioritetskö.
- `void dequeue();` - Tar bort det högst prioriterande elementet från er prioritetskö.
- `T peek();` - Hämtar det högst prioriterade elementet i er prioritetskö.
- `size_t size();` - Returnerar antalet element som finns i er prioritetskö.
- `bool isEmpty();` - Visar om er prioritetskö är tom eller inte.

Er klass skall ha namnet PriorityQueue och den skall kunna hantera en generell datatyp, d.v.s. ni skall implementera en klassmall. Ni ska anta att mindre-än-operatorn (`operator<`) är implementerad för alla datatyper som används i samband med prioritetskön.

Prioritetskön skall sortera på datan själv, det kommer ingen extra nyckel som representerar dess prioritet, och den skall kunna hantera en godtycklig mängd data. Ni skall även se till att hantera eventuella fel som kan uppstå vid användning av er prioritetskö genom att kasta ett exception. Tänk noga igenom vilka fel som kan uppstå vid användning.

När er lösning är färdigimplementerad, och ni klarar de tester som är givna, skall ni se till att er lösning klarar av det testscript som ges vid sidan av denna uppgiftsspecifikation. Mer information kring detta testscript finns under Appendix A.

2 Medföljande Projektfiler

Tillsammans med inlämningen medföljer två projektmappar, en för *Visual Studio 2017* och en för *Visual Studio Code*. Båda dessa projekt har konfigurerats för att vara likvärdiga. De konfigurationer som är gjorda och beskrivningar av dessa går att läsa i Appendix B. Vänligen notera att vi inte garanterar att dessa projektstommar fungerar utanför skolans datormiljö. Medföljande i dessa projektstommar är tester för er kod.

2.1 Tester

De medföljande projektstommarna innehåller tester för att du iterativt skall kunna säkerhetsställa att din kod uppfyller de specificerade kraven. Dessa tester kräver att namngivningen är konsekvent med detta dokument, men givet att klassen `PriorityQueue` är deklarerad i filen `PriorityQueue.h` så kommer dessa tester att kompilera och informera om eventuella fel i din implementation. Det är starkt rekommenderat att du *inte* sätter dig in i denna kod, utan enbart fokuserar på att implementera den specificerade klassen och passera testerna enligt utskrifter. Du är fri att skapa dina egna tester för din implementation, men det är starkt rekommenderat att använda de medföljande stommarna för de inställningar som är gjorda. Testscriptet är baserat på de inställningar som finns i dessa stommar.

2.2 Körning av det medföljande projektet

Den fil som projektet kompileras ner till förväntar sig ett input för antal iterationer som testerna skall köra. I Visual Studio är detta specificerat till **15** iterationer, medans det för g++ projektet kan specificeras vid körning. Detta görs genom att skriva önskat antal iterationer efter .exe-filen från kommandotolken. Om inga iterationer anges kommer även g++ projektet att köra med **15** iterationer.

3 Kravspecifikation och Inlämning

- All kod skall skrivas av dig själv och inlämningarna är individuella. Alla inlämningar kommer genomgå en plagieringskontroll.
  - Du som student förvntas läsa informationen som presenterats på Canvas angående plagiering.
- Följande, och endast följande, bibliotek är tillåtna att användas:
  - <memory>
  - <cstdlib>
  - <exception>/<stdexcept>
  - <utility>
  - <iterator>
- Prioritetskön skall uppfylla alla de krav som specificerats under Rubrik 1.
- Inga kompileringsvarningar eller kompileringsfel får finnas i den lösning som ni skickar in. Dessa skall sesöver och hanteras innan inämning.
- Innan inlämning skall det medföljande testscriptet köras. Passerar inte de tester som utförs av detta script är din implementation ej redo för inlämning.
  - De studenter som lämnar in en implementation som inte klarar av detta script kommer automatiskt att bli underkända.
  - För körinstruktioner vänligen se Appendix A.
  - Notera att detta script *endast* är ämnat att fungera på skolans datormiljö. Studenter som önskar köra scriptet på egen dator erbjuds ingen hjälp med detta.
- Ni skall *endast* lämna in filen `PriorityQueue.h.txt` som innehåller klassmallen `PriorityQueue`. **Ingen main-funktion skall finnas i er fil!** Om en fil med felaktigt namn eller med en main-funktion implementerad lämnas in kommer denna inlämning automatiskt att bli underkänd, då den varken kommer passera ert givna testscript eller vårt rättningsscript.

A Körinstruktioner för testscript

Det medföljande testscriptet skall köras och passeras innan inlämning. Detta script kontrollerar så att er inlämning har korrekt namn, inte använder några otillåtna bibliotek, med mera. Detta script tillhandahålles för att göra rättningsprocessen transparent. Inlämnad kod som *inte* klarar av detta script kommer inte rättas.

Notera att er implementation inte garanterat uppfyller de krav som ställs i uppgiften även om testerna passerar. Detta script är det första och mest grundläggande test som utförs. Ett passerat script är alltså inte ett garanterat godkänt på uppgiften.

För att köra testscriptet kopierar ni filen `PriorityQueue.h` till mappen *Student_Folder*. Byt sedan filändelsen från `.h` till `.h.txt`. Högerklicka på filen *Submission_Test.ps1* och välj *Kör med PowerShell*. Skrivs texten *"Submission script succesfully finished."* ut är er implementation redo för inlämning.

B Projektkonfigurationer

De medföljande projekten har konfigurerats med följande inställningar (eller likvärdiga för Visual Studio). Det är viktigt att dessa inställningar lämnas orörda, då viss funktionalitet kräver dessa:

- -std=c++17
  - Kompilerar projektet med `C++`17-standarden.
  - Denna flagga behövs för att de medföljande Unit Test-filerna skall kunna användas.
- -Wall
  - Säkerhetsställer att inga viktiga varningar uteblir.
- -Wextra
  - Lägger till några varningar som annars överses.
- -Wpedantic
  - Säkerhetsttäller att den förväntade `C++` standarden efterföljs.
- -pedantic-errors
  - Som -Wpedanitc, men rapporterar det som fel istället för att ge en varning.
- -Werror
  - Markerar ***alla*** varningar som fel.
- -O0
  - Säger åt kompilatorn att optimisera koden minimalt.
  - Denna flagga används för att exekveringstiden skall vara baserad på koden och inte hur effektivt kompilatorn kan optimera.