UMEÅ UNIVERSITET
Institutionen för datavetenskap
Obligatorisk uppgift 1

1 februari 2024

5DV149

Datastrukturer och algoritmer

Obligatorisk uppgift 1 — Testning av stack

version 1.0

Namn Nils Sjölund
Användarnamn id23nsd

Innehåll

1 Introduktion

Denna laboration genomfördes i syfte att skapa två program som testar olika implementationer av datastrukturen stack. Testerna är utformade efter datastrukturens gränsyta och har som syfte att säkerställa att implementationerna följer gränsytans struktur.

2 Gränsyta till datatypen Stack

Nedan följer gränsytan för den abstrakta datatypen stack. Denna laboration berör två implementationer: `stack` och `int_stack`, där den stora skillnaden mellan implementatinerna är att `stack` använder sig av `void`-pekare för att hantera sina element, vilket innebär att `stack` kan hantera alla typer av data så länge användaren allokerar minne för den. `int_stack` kan däremot bara hantera heltal.

- **Empty()** — Returnera en tom stack
- **Push(s, v)** — Stoppar värdet **v** överst på stacken **s** och returnerar den modifierade stacken.
- **Isempty(s)** — Returnerar en bool som beskriver om stacken **s** är tom eller inte.
- **Top(s)** — Returnerar det översta värdet på stacken
- **Pop(s)** — Tar bort det översta elementet på stacken och returnerar den modifierade stacken.
- **Print(s)** — Skriver ut stacken **s**.
- **Kill(s)** — Förstör den givna stacken **s**. **s**.

3 Dokumentation av testerna

Nedan följer beskrivningar av alla test-programmets implementerade tester .

3.1 `int_stack`

3.1.1 Test 1 - `is_empty_returns_false_test()`

Testar `stack_is_empty()`

Testet börjar med att skapa en tom stack och kontrollerar att `stack_is_empty()` returnerar `TRUE`. Om `stack_is_empty()` returnerar `FALSE` innebär det att `stack_is_empty()` inte fungerar som den ska, då avslutas programmet och ett felmeddelande skrivs ut.

3.1.2 Test 2 - `is_empty_returns_true_test()`

Testar `stack_is_empty()` och `stack_push()`

Testet börjar med att skapa en tom stack och därefter anropas `stack_push()` för att lägga ett element på stacken. Den modifierade stacken används sedan i `stack_is_empty()` för att kontrollera om den returnerar `TRUE`. Om så är fallet innebär det att antingen `stack_push()` eller `stack_is_empty()` inte fungerar som de ska, vilket resulterar i att programmet avslutas och ett felmedelande skrivs ut.

3.1.3 Test 3 - `increasing_loop_test()`

Testar `stack_push()` och `stack_top()`.

Testet börjar med att starta en tom stack och går sedan in i en for-loop som loopas 5 gånger. I loopen anropas `stack_push()` för att pusha i elementet `expected_value` som ökas från 0 till 4 för varje iteration av loopen. Direkt efter att värdet pushas in i stacken anropas `stack_top()` och jämför det med det nuvarande `expected_value`. Om de inte stämmer överens innebär det att `stack_push()` eller `stack_top()` fungerar som de ska , vilket resulterar i att programmet avslutas och ett felmeddelande skrivs ut.

3.1.4 Test 4 - `decreasing_loop_test()`

Testar `stack_push()`, `stack_top()` och `stack_pop()`.

Testet börjar med att starta en tom stack och går sedan in i en for-loop som pushar in värden från 0 till 4 i stacken.

Därefter initieras en variabel `expected_value` och en while loop som pågår tills `stack_is_empty()` returnerar `TRUE` på stacken. I loopen jämförs `stack_top()` med `expected_value` och om de stämmer överens anropas `stack_pop()` på stacken och `expected_value` minskar med ett. Om `stack_top()` och `expected_value` inte stämmer överens innebär det att `stack_top()` eller `stack_pop()` inte fungerar som de ska, vilket resulterar i att programmet avslutas och ett felmeddelande skrivs ut.

3.2 `stack`

3.2.1 Test 1 - `empty_test()`

Testar stack_empty().

Funktien anropar `stack_empty()` och kontrollerar i en if-sats om funktionen har returnerat NULL, om den har det så innebär det att `stack_empty()` inte fungerar som den ska, programmet avslutas och ett felmedelande skrivs ut.

3.2.2 Test 2-5

Testerna är logiskt sett uppbyggda på samma sätt som test 1-4 för `int_stack()` med skillnaden att varje gång ett element pushas in i stacken allokeras plats för elementet i minnet, och varje gång ett element popas avallokeras den platsen. Varje test avslutas också med `kill_stack()` för att avallokera allt som inte redan har blivit det, och förebygger därmed minnesläckor.

4 Resultat

4.1 Testkörningar

Nedan följer exempel på två körningar av test-programmet: en på en korrekt stack och en på en trasig stack. Figur 1 visar utskrifter för en korrekt implementerad stack. Figur 2 visar utskrifter för en implementation där `stack_top()` är felimplemnterad.

```
Testing if stack_empty returns NULL pointer...
Testing if is_empty returns false on empty stack...
Testing if is_empty returns true on non-empty stack...
Testing stack with increasing values...
Testing stack with decreasing values...
SUCCESS: Implementation passed all tests. Normal exit.


[1] + Done                       "/usr/bin/gdb" --interpreter=mi --t
```

Figur 1: Testkörning 1. Testkörning mot en korrekt stack-implementation. Utskrifterna visar att koden klarade alla testerna.

```
Testing if stack_empty returns NULL pointer...
Testing if is_empty returns false on empty stack...
Testing if is_empty returns true on non-empty stack...
Testing stack with increasing values...
FAIL: expected 1, got 256

[1] + Done                       "/usr/bin/gdb" --interpreter=mi --t
ty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-ppbp51mp.arx" 1>"/tmp/Mi
crosoft-MIEngine-Out-exs3s4mf.0ky"
```

Figur 2: Testkörning 2. Testkörning mot en inkorrekt stack-implementation, där top returnerar det översta värdet - 1. Utskrifterna visar att koden klarade de tre första testerna och stannade vid `increasing_loop_test()` eftersom `stack_top()` inte returnerade det förväntade värdet.

4.2 Summering

Laborationens syfte var att skapa program som testar två implementationer av datastrukturen stack. Resultatet blev två program som bestod av 4, respektive 5 tester som tillsammans kontrollerar att implementationernas funktioner beter sig enligt gränsytans specifikation.