Laborationsrapport **4 maj 2025**

5DV149 Datastrukturer och algoritmer

Obligatorisk uppgift 1 — Testning av stack

version 1.0

Marc Meunier	Marc Meunier
tfy22mmr	tfy22mmr

Innehåll

1 Introduktion

I denna laboration så skriver vi två olika koder för att testa en implementation av stack. Vi började med att skriva självaste testet för en stack som bara hanterar heltal (int), int_stack.h filen är given. I mitt fall anpassades koden för att hantera generiska pekare via stack.h för att så den klarar av att spara annat än bara ints. För att testa att implementationen fungerar för både stack.h och int_stack.h skriver vi olika funktioner som testar att stacken fungerar på de sätt som den är beskriven och implementerad.
Ett exempel på ett test som görs är ifall funktionen stack_empty() skapar en stack eller inte. När de visat sig att stacken skapats kan man sen gå vidare till stack_is_empty(**s**) och sedan se om den returnerar korrekta värde när den blir matad med en fungerande stack. På liknande sett fortsattes testerna tills att vi antingen hittat fel på implementeringen, eller att stacken fungerar som den ska.

2 Gränsyta till datatypen Stack

Skillnaden på dem båda är att stack.h använder sig av en kill funktion för att radera stacken i efterhand medans int_stack.h. stack.h modifierar också pekaren direkt, medans int_stack.h returnerar hela stacken varje gång några av modifieringskommandona körs på stacken.

- **stack_empty(kill_func)** — Returnera en tom stack. beroende på vilken implementation kan en kill function va bra att ha så att man kan deallokera minnet innuti stacken.
- **stack_kill(s)** — Raderar stacken **s** och ger tillbaka de utrymme som var upptaget av den.
- **stack_push(s, v)** — Sätter in värdet / pointern **v** överst på stacken **s** och returnerar den nya stacken. Den stack som skickades in ska anses vara invalid efter att ha kört denna funktion.
- **stack_pop(s)** — Tar bort de översta elementet på stacken **s** och returnerar den nya stacken. Den stack som skickades in ska anses vara invalid efter att ha kört denna funktion.
- **stack_top(s)** — Kollar på de översta värdet på stacken och returnerar värdet. Ifall den används på en tom stack returneras värdet undefined.
- **stack_is_empty(s)** — Kollar ifall stacken **s** är tom eller ej. Returnerar true ifall **s** är tom, annars false.
- **stack_print(s, print_func)** — Printar ut hela stacken **s** med funktionen **print_func**

3 Dokumentation av testerna

Det här är den viktigaste delen! Skriv några korta ord kring varje test, vad du förutsätter ska fungera (redan testat av tidigare tester) och hur du har tänkt kring testet. Undvik att använda kod i din beskrivning — det blir bara svårläst!

Skriv t.ex. “Testet lägger ett element på en tom stack och kollar att stacken inte är tom” eller “testet placerar fyra värden på stacken och kontrollerar att dom hamnar i rätt ordning”. (Plus fler detaljer om vilka funktioner du anropar i vilken ordning och vad det förväntade resultatet är.)

3.1 `int_stack`

3.1.1 Test 1 - isempty_return_true

Här testar jag att initiera en tom stack med stack_empty och kollar ifall stack_is_empty på den stacken returnerar true eller false. Förväntad output är True och jag printar ett fel ifall den skulle bli false. Sedan lägger jag till ett värde till stacken med stack_push och kollar igen vad stack_is_empty returnerar. Nu är förväntade värdet som den skall returnera false då stacken inte längre är empty och ifall den svarar fel printar jag ett fel.

3.1.2 Test 2 - push_stack_on_top

Testet initialiserar en tom stack med stack_empty därefter loopar jag över stack_pushfunktionen för att lägga in 10 värden i stacken. Efter de så kollar jag värdena i stacken med stack_top och jämför dem med value_equal funktionen så jag ser att de ligger i rätt ordning för att sen radera de översta elementet i stacken med stack_pop. Ifall ett element ligger felaktigt printar jag ut ett felmeddelande. Testar också så stacken är tom med hjälp av redan testade funktion stack_is_empty så inga element finns kvar i funktionen. Funktionen förväntas därför printa felmeddelande ifall de är felaktig ordning på stacken.

3.1.3 Test 2 - top_read_wrong

Liknande till förra så initializerar de här testet en stack, lägger in värden i den och sen kollar den ifall de läses av i rätt ordning. Ifall den läser av i annan ordning än vad den stoppat in värdena i kommer funktionen att printa ut ett fel. Vi förväntar oss att båda push placerar i rätt ordning. Stacken töms sen på de inlagda elemente och vi kollar ifall stack_is_empty rapporterar yom stack.

3.1.4 Test 2 - pop_remove_wrong

De här testet vill kolla så att vi tar bort rätt värde från stacken. Vi förväntar oss att tidigare funktioner fungerar, alltså är de bara pop som kan va fel. Vi börjar med att skapa en tom stack som vi sen fyller med 3 heltal genom att köra en for loop. Sen kör vi en till for loop och jämför topelementet som returneras av stack_top, för att sedan ta bort översta elementet med stack_pop. Ifall fel tal returneras printar funktionen ett felmeddelande. Testar också så stacken är tom med hjälp av redan testade funktion stack_is_empty för att verifiera att inga element finns kvar i stack.

3.2 stack

3.2.1 Test 1 - test_stack_empty

Funktionern ska generera en tom stack och även kolla så att stacken faktist sparades, dvs så ska stacken s inte vara null. Ifall stacken inte sparas printar den ett felmeddelande. Stacken raderas sedan genom att anropa funktionen stack_kill.

3.2.2 Test 2 - isempty_return_true

Detta test skapar en stack, och kollar sedan ifall funktionen stack_is_empty ger rätt värde för den tomma stacken. Den testar även att lägga till ett värde i stacken genom att använda stack_push och int_create som allokerar minne till stacken så att stacken inte ska va tom. Sedan testas stack_is_empty igen på den fyllda stacken och printar fel ifall stacken felaktigt rapporterar att den är tom. Vi förlitar oss på att stack_push fungerar. stacken raderas sedan med stack_kill.

3.2.3 Test 3 - push_stack_on_top

Detta test är tänkt för att sätta in värden i stacken och kolla att de placeras i rätt ordning. Jag börjar som tidigare med att skapa en tom stack för att sedan köra en for loop som matar in olika heltal. Först kommer heltalen efter varann, men efter värde 5 sätter jag in heltal med 5 i mellanrum. Detta görs för att vi sedan ska kunna se att den inte bara sätter in värden utefter den position i stacken den har. Efter att ha satt in värdena kontrollerar jag med en ny for loop ifall värdena är samma som de jag satte in. Här anropas value_equal for att ta reda på ifall värdena är lika. Vi gör på liknande steg med 5 i början för att sen gå till nästa heltal och ifall de inte skulle överrensstämma så printar den ut ett felmeddelande.
Vid varje loop så raderas också översta elementet med stack_pop så den kan fortsätta jämföra värdena. Vi förväntar oss att både stack_top och stack_pop fungerar under detta test. Testar också så stacken är tom med hjälp av redan testade stack_is_empty. När loopen är klar raderas sedan stacken med stack_kill.

3.2.4 Test 4 - top_read_wrong

Hör initieras en stack likt tidigare, för att sen köra en forloop som fyller stacken med 4 tal. Varje gång vi fyller stacken kollar vi även att talet är längst upp genom att anropa value_equal med värdet på stacken och de vi satt in. Ifall värdet inte är de printar vi felmeddelande. Vi förväntar oss att tidigare funktioner fungerar. Vi raderar sedan stacken med stack_kill.

3.2.5 Test 5 - pop_remove_wrong

Här skapar vi en tom stack på nytt, som vi fyller med en forloop och stack_push funktionen. När vi har en fylld stack kör vi en for loop som testar att ta bort värden ifrån stacken och kollar om de värdet som ska ligga längst upp är på rätt plats. Detta jämförs igen med value_equal, och ifall dem inte överränsstämmer printas ett fel ut. Vi förväntar oss att tidigare test fungerar när vi kör detta test. Testar också så stacken är tom med hjälp av stack_is_empty. I slutet raderar vi stacken med stack_kill.

4 Resultat

Som vi kan se i första körningen 1 så fungerade stacken utan problem, alla tester startar och avslutar. För den andra körningen 2 så matar jag in fel värden i stacken i självaste push_stack_on_top testet och där ser vi att programmet skriver ut ett fel. Självaste felet är att programmet har sparat 50 i stacken medans 10 var de den skulle spara på den positionen. Alltså avslutas programmet då den ser att något är fel och vi får reda på att de är något fel i stackimplementationen som körs. Liknande felmeddelanden borde komma upp ifall ett annat test misslyckas.

```
marc@fedora:~/Documents/Umeå/DOA/OU1$ ./stack_test
stack_test.c, v1.0 2025-04-07: Test program for the typed int_stack datatype.
Starting test_stack_empty()...
done.
Starting isempty_return_true()...
done.
Starting push_stack_on_top()...
done.
Starting top_read_wrong()...
done.
Starting pop_remove_wrong()...
done.

SUCCESS: Implementation passed all tests. Normal exit.
marc@fedora:~/Documents/Umeå/DOA/OU1$
```

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

```
marc@fedora:~/Documents/Umeå/DOA/OU1$ ./stack_test
stack_test.c, v1.0 2025-04-07: Test program for the typed int_stack datatype.
Starting test_stack_empty()...
done.
Starting isempty_return_true()...
done.
Starting push_stack_on_top()...
FAIL: Expected <10>, got <50>.
marc@fedora:~/Documents/Umeå/DOA/OU1$
```

Figur 2: Testkörning 2. Testkörning mot en inkorrekt stack-implementation. Utskrifterna visar att koden misslyckas med att skriva rätt värde till stacken, och därefter avslutas direkt.