

UMEÅ UNIVERSITET
Institutionen för datavetenskap
Obligatorisk uppgift 1

31 januari 2023

Pretitle
**5DV149 Datastrukturer och
algoritmer**

Obligatorisk uppgift 1 — Testning av stack

version 1.0

Namn	Name
Användarnamn	CS username

Labrättare
Graders

Innehåll

1	Introduktion	1
2	Gränsyta till datatypen Stack	1
2.1	stack_empty()	1
2.2	stack_push()	2
2.3	more commands...	2
3	Dokumentation av testerna	2
3.1	int_stack	2
3.1.1	Test 1 - namn	2
3.1.2	Test 2 - namn	2
3.2	stack	2
3.2.1	Test 1 - namn	2
3.2.2	Test 2 - namn	2
4	Resultat	2

1 Introduktion

Skriv en kort introduktion till din rapport. I denna laboration räcker det med kanske ett eller två stycken då målgruppen för din rapport är handledarna och dom förväntas vara väl förtrogna med uppgiften och den aktuella datatypen.

Viktigt! Om du väljer att ladda ner detta dokument och kompilera det på din dator, använd kommandot

```
pdflatex -shell-escape rapport.tex
```

Kom ihåg att köra kommandot två gånger för att uppdatera innehållsförteckning, figurreferenser, osv.

2 Gränsyta till datatypen Stack

Beskriv gränsytan till datatypen Stack. Beskrivningen ska vara strukturerad, men du får själv bestämma hur, t.ex. en undersektion per funktion, en punktlista, en tabell med funktionerna, e.dyl.

I din beskrivning, utgå helst från det som är publicerad i `stack.h` eller `int_stack.h`. Som ett alternativ får du välja en beskrivning som motsvarar den informella specifikationen av datatypen från boken och föreläsningsanteckningarn.¹

Du kommer att behöva skriva en liten beskrivning av skillnaden mellan hur `stack` och `int_stack` är implementerade, men i övrigt bör du undvika att kommentera vad du vet om hur datatypen är implementerad.

Det viktiga är att det ska vara *lätt att läsa* din beskrivning. Exempelvis ska eventuell kod som du skriver bland löptext ha annat ett **fixt typsnitt**, vilket går att lösa med `Empty()` eller `Empty()`. I icke- \LaTeX -system kan du använda typsnittet `Courier`.

Nedan skriver jag några *exempel* för att visa på några tekniska lösningar i Latex och Overleaf.

- **Empty()** — Returnera en tom stack
- **Push(s, v)** — Stoppa värdet `v` överst på stacken `s` och returnerar den modifierade stacken.

<code>stack_empty(free)</code>	Returnerar en tom stack.
<code>stack_push(s, v)</code>	Stoppa värdet <code>v</code> överst på stacken <code>s</code> och returnerar den modifierade stacken.

2.1 `stack_empty()`

```
stack *stack_empty(free_function free_func);
```

¹Denna möjlighet kommer att tas bort i framtida kurser. Den finns kvar endast då det var tillåtet i den ursprungliga specifikationen.

Funktionen `stack_empty(free)` returnerar en tom stack där stacken har övertagit ansvaret för att avallokera elementvärden som läggs på stacken.

2.2 `stack_push()`

```
stack *stack_push(stack *s, void *v);
```

Funktionen `stack_push(s, v)` placerar pekaren `v` överst på stacken `s` och returnerar den modifierade stacken. Ansvaret för att avallokera `v` övertas av stacken.

2.3 more commands...

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 - namn

3.1.2 Test 2 - namn

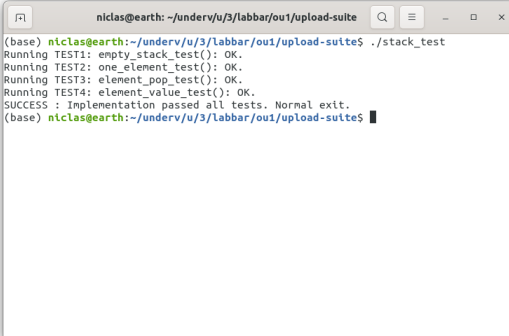
3.2 `stack`

3.2.1 Test 1 - namn

3.2.2 Test 2 - namn

4 Resultat

Specifikationen kräver två körningar: en på en korrekt stack och en på en trasig stack. Skriv lite om dom här och referera t.ex. till skärmdumpar som den i Figur 1. Kom ihåg att skriva en summering i löptexten också!

A terminal window titled 'niclas@earth: ~/underv/u/3/labbar/ou1/upload-suite' showing the execution of a test suite. The prompt is '(base) niclas@earth:~/underv/u/3/labbar/ou1/upload-suite\$'. The command './stack_test' has been entered. The output shows four tests: 'Running TEST1: empty_stack_test(): OK.', 'Running TEST2: one_element_test(): OK.', 'Running TEST3: element_pop_test(): OK.', and 'Running TEST4: element_value_test(): OK.'. The final output is 'SUCCESS : Implementation passed all tests.. Normal exit.' followed by a new prompt '(base) niclas@earth:~/underv/u/3/labbar/ou1/upload-suite\$' with a cursor.

```
niclas@earth: ~/underv/u/3/labbar/ou1/upload-suite
(base) niclas@earth:~/underv/u/3/labbar/ou1/upload-suite$ ./stack_test
Running TEST1: empty_stack_test(): OK.
Running TEST2: one_element_test(): OK.
Running TEST3: element_pop_test(): OK.
Running TEST4: element_value_test(): OK.
SUCCESS : Implementation passed all tests.. Normal exit.
(base) niclas@earth:~/underv/u/3/labbar/ou1/upload-suite$
```

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