3+2=5/20

Betriebssysteme

Übungsblatt 6

Anne Rossl Diana Hörth

December 2, 2022

Aufgabe 1

st(z) = (const, int, 2)
st(y) = (var, int, 129)
st(x) = (var int, 128)

PC	Befehl	Kommentar
0	LOADIN SP IN1 2	Lade y in IN1.
1	LOADIN SP IN2 1	Lade x in IN2
2	LOADIN SP ACC 3	Lade z in ACC.
3	MUL ACC IN1	Multipliziere y und z..
4	SUB ACC IN2	Subtrahiere das Ergebnis der vorherigen Multiplikation mit x
5	$\text{JUMP}_{>}$ 5	Falls das Ergebnis größer 0 ist bende Schleife.
6	LOAD ACC IN2	Überschreibe ACC mit x.
7	SUBI ACC 3	Ziehe drei von x ab.
8	LOAD IN2 ACC	x aktualisieren.
9	JUMP_{-7}	Wieder hochspringen und Schleifenbedingung überprüfen.
10	SUBI SP 1	SP wir eins nach oben verschoben.
11	STOREIN SP IN2 1	Ergebnis wir drauf gelegt
12	JUMP 0	beenden

Aufgabe 2

1.5 + 0.5 = 2/10

0.5 + 1 für die Mühe = 1.5/3

Aufgabe 3

-1.5 man sollte hier die konkrete Speicheraddresse als Zahlenwert nennen (in den Kommentaren)

Das übersieht man leider leicht.

a)

- (struct point p2) : Gelesen, weil der Typ eine selbstdefinierte Klasse ist, und nach geschaut werden muss was für Attribute die Klasse hat.
- (p2.x = 7) : Hier wird auf die Speicheradresse geschrieben und zwar eine 7.
- (p2.y = 4) : Hier wird auf die Speicheradresse geschrieben und zwar eine 4.
- (a = (p2.x)) : Auf den Pointer a wird die Referenz von p2.x geschrieben.

p2.x gelesen

- ((*p1).y = *a) : Hier wird gelesen worauf a zeigt.

und geschrieben wohin p1 auf dem Heap zeigt

- p3 = p1 : p1 wird in p3 geschrieben.
- p1 = p2 : Die Referenz von p2 wird in p1 geschrieben
- if((*p1).y > 5) : wird abgelesen, um es mit 5 zu vergleichen.
- *a = 42 : hier wird auf den Speicher zugegriffen auf die der Pointer zeigt und überschreibt es mit 42.
- *a = 1 : hier wird auch wie der auf den Speicher zugeriffen auf den a zeigt und der Inhalt dort wird mit 1 überschrieben.
- p1 = (struct point *) malloc(sizeof(struct point) : hier wird gelesen und geschrieben. Erst wird mit sizeof() gelesen um herauszufinden wie groß strsuct point ist, damit sie wissen wie viel Platz gebraucht wird. Später wird noch geschrieben.
- free(p3) : hier wird geschrieben und weil hier was der Inhalt gelöscht wird.

und dazu muss p1 gelesen werden

und dazu muss p2 gelesen werden

man sollte hier die konkrete Adresse der Speicheradresse nennen. Folgefehler

-0.5 das wird aus der Symotlabelle gelesen

korrekt, es wird p1 geschrieben

leider nicht, es wird einfach aus der internen...

-0.5

Datenstruktur von malloc rausgelöscht, aber die Speichzerlle behält den Wert

b) 0.5/1

0/6

c)

-0.5 hier war eine Konrette Speicherzelle verlangt und es ist ein Speicherbereich und keine Speicherzelle

Die letzte Zeile ist zulässig. Hier wird die Speicherzelle auf die der Zeiger zeigt freigegeben.

die Bibltiothekfunktion malloc hat eine interne Daternstruktur in der es die Speicherbereiche abspeichert

```
# Exp(Call(Name('input'), []))
LOADI ACC 3
SUBI SP 1;
STOREIN SP ACC 1;
# Assign(Global(Num('1')), Stack(Num('1')))
LOADIN SP ACC 1;
STOREIN DS ACC 1;
ADDI SP 1;
# // Assign(Name('x'), Call(Name('input'), []))
# Exp(Call(Name('input'), []))
LOADI ACC 15
SUBI SP 1;
STOREIN SP ACC 1;
# Assign(Global(Num('0')), Stack(Num('1')))
LOADIN SP ACC 1;
STOREIN DS ACC 0;
ADDI SP 1;
# -1.5 diesmal solltet ihr das gesamte Programm übersetzen >_<
# -2 ich muss leider Punkte abziehen, weil die Aufgabe so gedacht war, dass ihr die Patterns aus der Vorlesung verwenden sollt: "Werten Sie die Ausdrücke und Anweisungsfolgen aus, wie Sie es in der Vorlesung gelernt haben". In der Klausur könnte es dafür einen größeren Punktabzug geben, da es nicht spezifziert war, dass es dafür überhaupt Punkte geben sollte
LOADIN DS IN2 1
LOADIN DS ACC 0 # -0.5 das ist die main-Funktion
# LOADIN DS ACC 3 -0.5 z ist eine Konstante, die steht nicht im Hauptspeicher
MULTI IN2 2 # der Befehl heißt MULT nicht MUL ^^
SUB ACC IN2
# JUMP> 5 -0.5 falsche Relation
JUMP< 5
# LOAD ACC IN2 # -0.5 mal wieder ein neuer erfunder Befehl, vielleicht sollte ich Scholl vorschlagen den mitaufzunehmen, weil ihr Studies MOVE ACC IN2 nicht nutzen wollt ^^
LOADIN DS ACC 0
SUBI ACC 3 # so wie ihr euch das gedacht habt, funktioniert das nicht, weil ihr bei der Auswertung der Schleifenbedingungen ACC bereits überschreibt, dafür mal kein Punktabzug
STOREIN DS ACC 0
# bei dieser Aufgabe wären die Patterns aus der Vorlesung sicherer
JUMP -6
SUBI SP 1
# JUMP 0
```

dieses Programm wurde so abgeändert, dass es mit dem RETI-Interpreter korrekt durchläuft. Ihr könnt es durch Erstellen einer .reti Datei und ausführen mit dem RETI-Interpreter im Show-Mode -S mal selbst ansehen picoc_compile program.reti -S

1 + 2 für die Mühe = 3/6 bei dieser Aufgabe wäre es tatsächlich sicherer die Patterns aus der Volresung zu verwenden, weil es bei denen nicht zum Überschreiben von Registern kommen kann in denen eine Varaible für spä er gespeichert ist