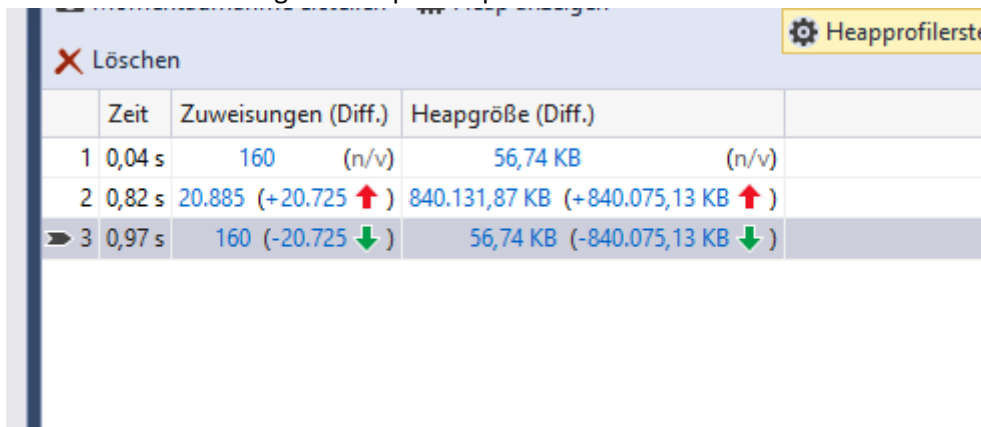


1. Dreiecksmatrix

n = 60	n = 1	n = 61
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59		1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59

Nach der Berechnung der benötigten Reihen(y) wird auf dem Heap ein Pointer Array angelegt mit der Größe der Reihen. Jedes Feld bekommt nun ebenfalls ein Integer Array welches die maximalen zu speichernden Zeichen enthält (Reihe 0 = 1 Feld; Reihe 1 = 2 Felder; ...). Eine Ausnahme ist nur die letzte Reihe wo die die Anzahl der Felder, aufgrund von n, Variabel ist. Anschließend werden die Felder befüllt und ausgegeben.

Eine weitere Funktion gibt den Speicherplatz anschließend wieder frei.



	Zeit	Zuweisungen (Diff.)	Heapgröße (Diff.)
1	0,04 s	160 (n/v)	56,74 KB (n/v)
2	0,82 s	20.885 (+20.725 ↑)	840.131,87 KB (+840.075,13 KB ↑)
3	0,97 s	160 (-20.725 ↓)	56,74 KB (-840.075,13 KB ↓)

Heap Größe bei n= 429496720 und 20724 Reihen ~ 840MB

2. Zeigermanipulation

a) Eine Zeiger Variable bzw. ein Pointer beinhalten eine Adresse die auf eine Speicheradresse im Speicher zeigt. Ein Pointer besitzt als Wert eine Adresse und keinen Wert. Wird ohne Dereferenzierungsoperator (*) gearbeitet, kann auf die Adresse die im Pointer gespeichert ist zugegriffen werden. Wenn mit Dereferenzierungsoperator gearbeitet wird gibt der Pointer den Wert an der angegebenen Adresse zurück.

b)

- 1- Es wird eine Integer Variable i erstellt und der Wert 5 drauf zugewiesen
- 2- Es wird eine Integer Variable j erstellt und der Wert 6 drauf zugewiesen
- 3- Es wird eine Integer Variable t erstellt und der Wert 0 drauf zugewiesen

- 4- Es wird ein Integer Pointer pI erstellt und die Adresse NULL darauf zugewiesen
- 5- Es wird ein Integer Pointer pJ erstellt und mithilfe des &-Operators die Adresse von j darin gespeichert
- 6- Es wird ein Float Pointer pF erstellt und mithilfe des &-Operators die Adresse von i darin gespeichert. Die Umwandlung von int zu float wird über einen Typecast ((float*)) realisiert. Dieser Typecast betrifft aber nur wie auf den Speicherbereich zugegriffen wird. Die Zahl dahinter (falls vom Pointer darauf zugegriffen wird) wird nicht gleich sein da Integer und floats verschieden aufgebaut sind. Auf 16 Bit Systemen entsteht ein weiteres Problem, da ein Float 32Bit belegt und, auf 16Bit Systemen, ein Integer nur 16Bit besitzt. Dadurch wird ein Adressbereich gelesen der zu etwas anderem gehört.
- 7- Der Pointer pI wird mithilfe des & Operators die Adresse von i zugewiesen.
- 8- Die Pointer pI und pJ werden mittels des *-Operators dereferenziert und ausgegeben. Erwartete Werte: *pI = 5 *pJ = 6
- 9- Die Variable t bekommt den mit Hilfe des *-Operators dereferenzierten Pointer wert von pI. t =
- 10- Die Variable I bekommt den Wert von J. I = 6
- 11- J bekommt den Wert von t. J = 5
- 12- Die Pointer pI und pJ werden mittels des *-Operators dereferenziert und ausgegeben. Erwartete Werte: *pI = 6 *pJ = 5
- 13- Der Pointer pJ erhält die Adresse von pI.
- 14- Die Pointer pI und pJ werden mittels des *-Operators dereferenziert und ausgegeben. Erwartete Werte: *pI = 6 *pJ = 6
- 15- Der Adresse in pJ wird der Wert 10 zugewiesen.
- 16- Die Pointer pI und pJ werden mittels des *-Operators dereferenziert und ausgegeben. Erwartete Werte: *pI = 10 *pJ = 10

c)

Adresse p1 = 00000000	Wert von p1 = -	Wert von i = 1234	Wert von k = 5678
Adresse p1 = 0061FF28	Wert von p1 = 1234	Wert von i = 1234	Wert von k = 5678
Adresse p1 = 0061FF28	Wert von p1 = 2323	Wert von i = 2323	Wert von k = 5678
Adresse p1 = 0061FF28	Wert von p1 = 2324	Wert von i = 2324	Wert von k = 5678
Adresse p1 = 0061FF24	Wert von p1 = 5678	Wert von i = 2324	Wert von k = 5678

3. Tokenizer

Testfälle:

Text	Dies	ist	ein	Probetext
Indices	0	5	9	13
Adressen	0040EA40	0040EA45	0040EA49	0040EA4D

Text	H
Indices	0
Adressen	0040EA40

Text
Indices
Adressen