Übung 10

Arbeitsaufwand insgesamt: 3h

Inhaltsverzeichnis

Teil 1 – Stackmachine

Ziel der Übung ist das Implementieren einer Stackmaschine die laut den Angaben im Beilagedokument operiert. Diese soll Kalkulationen die in Infix-Schreibweise vorliegen, lösen können. Dafür sollen auch ein Ringpuffer und ein Stack verwendet werden.

Lösungsidee:

Für die Erstellung der Stackmaschine wurden bereits in der Übungsstunde die 4 Module queue, stack, data_type und stackmachine angelegt.

Von diesem Punkt ausgehend, würde ich zuerst die Queue umschreiben, um diese zu einem Ringpuffer umzufunktionieren. Dafür wird das „shift_left()“ entfernt und zwei Variablen b (Beginn) und e (Ende) zur Struktur der Queue hinzugefügt. „b“ ist dabei immer der Index für das Element, das als nächstes herausgenommen wird. „e“ ist der Index für das Element das als nächstes dazukommt. Bei „enqueue“ wird der Wert auf den Vektor an der Stelle „b“ gesetzt und „b“ erhöht. Das gleiche passiert, für „e“ bei „dequeue“.

Beim Stack habe ich nichts weiter verändert. Dieser funktioniert ähnlich wie ein (nicht Ring-) Puffer. Die Werte werden beim Herausnehmen („pop“) aber von hinten/oben herausgenommen. Das was als letztes hineingeschrieben wird („push“), wird also als Erstes herausgenommen. Zusätzlich gibt es noch eine Funktion „top“ mit der das nächste Element geholt werden kann, ohne es aus dem Stack zu nehmen.

In den Stacks/Queues liegen Vektoren die mit speziellen Strukturen „data_type“s gefüllt sind. Ein data_type enthält dabei immer den Typ (Zahl, Rechenoperator, Klammer) und den Wert (für den Fall dass es eine Zahl ist).

Im Modul Stackmachine, werden dann die arithmetischen Ausdrücke mit denen man die Funktionen aufruft, ausgewertet und ein Wert zurückgegeben.

Das geschieht auf folgende Weise:

1. Zuerst wird der String der eingelesen wurde in die Eingabequeue geschrieben. Jeder Char wird dabei in einen data_type konvertiert! Schlägt dies fehl, endet der Algorithmus.
2. Ist die Eingabequeue mit allen Chars befüllt, so wird daraus die UPN-Queue erstellt:
   Es wird wieder jedes Element in einer Schleife aus der Eingabequeue genommen. Zahlen kommen direkt in die UPN-Queue, während Operatoren zuerst auf dem Hilfsstack landen. Wird eine rechte Klammer in der Eingabequeue erkannt, so wird der letzte Operator am Hilfsstack zur UPN-Queue hinzugefügt. Bei diesem Prozess gehen die Klammern verloren und es bleibt nur der Ausdruck in UPN-Form in der Queue übrig.
3. Um den Ausdruck auszuwerten, werden immer so lange Werte aus der UPN-Queue auf den Hilfstack geschrieben, bis ein Operator gelesen wird. Dieser nimmt dann die letzten zwei Zahlen und führt die jeweilige Operation aus. Wichtig dabei ist, dass die Zahl die als zweites aus dem Hilfsstack genommen wird, der linke Operand ist. „6/3“ würde sonst nämlich zu „3/6“ was die Rechnung verfälscht. Gibt es außerdem nicht genug Operanden für den Operator (nur eine Zahl auf dem Hilfsstack) endet der Algorithmus, weil der Ausdruck fehlerhaft ist.

Lösung:

Als C++ Projekt „Teil_1“ im Archiv

Testfälle:

- Leeres Eingabefile
- Ungültige Zeichen in File
- Division durch 0
- Zu wenige Operatoren im Ausdruck
- Zu wenige Operanden für Operator
- Geklammerte Ziffern
- Nicht einstellige Ziffern
- Beliebig komplexe Ausdrücke

Leeres Eingabefile:

```
Please enter the filename containing the expression: calc.txt

The given expression is empty.
```

Unültige Zeichen in File:

```
Please enter the filename containing the expression: calc.txt

The entered expression is:
(8+4)@0

ERROR! Invalid character (@) found.
Result: 0
```

Division durch 0:

```
Please enter the filename containing the expression: calc.txt

The entered expression is:
(8+3)/(9-(3*3))

ERROR! Division by 0.
Result: 0
```

Zu wenige Operatoren im Ausdruck:

```
Please enter the filename containing the expression: calc.txt

The entered expression is:
84+4

ERROR! Insufficient amount of operators in expression.
Result: 0
```

Zu wenige Operanden für Operator:

```
Please enter the filename containing the expression: calc.txt

The entered expression is:
(8+)+3

ERROR! Insufficient amount of parameters on help stack.
Result: 0
```

Durch das Auftreten der geschlossenen Klammer steht 8+ in der UPN-Queue. „+“ benötigt aber zwei Operanden die zuvor in der Queue stehen und es kann folglich auch nichts ausgerechnet werden.

Geklammerte Ziffern:

```
Please enter the filename containing the expression: calc.txt

The entered expression is:
(8)+4/3

ERROR! Insufficient items on help stack.
Result: 0
```

```
Please enter the filename containing the expression: calc.txt

The entered expression is:
8+(4)/3

Result: 4
```

```
Please enter the filename containing the expression: calc.txt

The entered expression is:
8+4/(3)

Result: 9
```

Probleme entstehen nur durch das Klammern der ersten Zahl oder Klammern von Zahlen, welche zwischen zwei nicht gleichrangigen Operatoren stehen. Durch die Klammerung beim zweiten Versuch wird z.B. „+“ gegenüber „/“ priorisiert.

Nicht einstellige Ziffern:

```
Please enter the filename containing the expression: calc.txt

The entered expression is:
237+10

ERROR! Insufficient amount of operators in expression.
Result: 0
```

Beliebig komplexe Ausdrücke:

```
Please enter the filename containing the expression: calc.txt

The entered expression is:
7*(((((5+5)*(5+5))*4)/2)-9)

Result: 1337
```