Fakultät IV - Elektrotechnik und Informatik
Einführung in die Programmierung WS 2017/18
Feldmann / Foucard / Lichtblau / Streibelt / Krüger / Enghardt / Palmer

Aufgabenblatt 5

letzte Aktualisierung: 08. Dezember, 12:06 Uhr

(dae868dc0f5c9ce3f350645f2079c7b73b07a261)

Ausgabe: *Freitag, 08.12.2017*
Abgabe: *spätestens Mittwoch, 20.12.2017, 18:00 Uhr*

Thema: *Trees, tree traversal, reverse polish notation*

Hinweise zur Gruppenarbeit, beginnend mit *diesem* Aufgabenblatt

Zur Erinnerung: ab diesem Blatt werden die Aufgaben grundsätzlich in festen Gruppen, die aus zwei Studierenden bestehen, gemeinsam bearbeitet und eingereicht.

Abgabemodalitäten

- Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des eecsIT mittels `gcc -std=c99 -Wall` kompilieren.
- Für die Hausaufgaben im Semester gibt es ein eigenes Subversion-Repository (SVN). Bitte die Hinweise auf Blatt 1 dazu beachten!
- Die Abgaben für Blatt 1 bis 4 erfolgen als Einzelabgaben. Ab Blatt 5 erfolgen Abgaben in festen Gruppen à 2 Personen.
- Die Gruppen werden vor Ausgabe von Blatt 5 gebildet. Solltet ihr keine Gruppe gebildet haben, werdet ihr einer Gruppe zugewiesen.
- Alle Abgaben müssen an der dafür vorgesehenen Stelle die Namen aller Gruppenmitglieder bzw. bei Einzelabgaben des Autors/der Autorin enthalten!
- Die Abgabe erfolgt ausschließlich über unser SVN im Abgaben-Ordner. Die finale Abgabe
  - für Gruppenabgaben erfolgt im Unterordner `Tutorien/t<xx>/Gruppen/g<xx>/Abgaben/Blatt<xx>/`
  - für Einzelabgaben erfolgt im Unterordner `Tutorien/t<xx>/Studierende/<tuBIT-Login>/Abgaben/Blatt<xx>/`

  WICHTIG: Die Abgabeordner werden immer von uns erstellt, sonst kommt es zu Konflikten! Benutzt zum Aktualisieren `svn up` im obersten Verzeichnis des Repositories!
- Benutzt für alle Abgaben die in der jeweiligen Aufgabe angegebenen Dateinamen (die von uns vorgegebenen Dateien haben das Wort ”vorgabe” im Dateinamen, du musst die Datei mit deiner Lösung also entsprechend umbenennen.)
- Gib bei den Programmieraufgaben den C-Quellcode ab und achte darauf, dass die Datei mit .c endet

- Bei Textaufgaben sind, wenn nicht anders angegeben, .txt Dateien zugelassen, die als Plaintext-Datei zu speichern sind. (Keine Word-Dateien umbenennen, etc.!)

1. Aufgabe: Implementieren eines Taschenrechners (3 Punkte)

In dieser Aufgabe sollst Du einen einfachen Taschenrechner implementieren, der Addition, Subtraktion und Multiplikation auf Fließkommawerten beherrscht. Um das Einlesen der Eingabe einfach zu gestalten, verwendet dieser, anstatt der geläufigen Infix Notation, die auch als *Reverse Polish Notation* bekannte Postfix Notation:

Postfix Notation: $38\ 4\ +$
Infix Notation: $38 + 4$

Diese Notation bietet den Vorteil, dass die Eingabe schrittweise mit Hilfe eines Stacks abgearbeitet werden kann und die Ausdrücke auch ohne Klammern eindeutig sind.

Postfix Ausdruck	equivalenter Infix Ausdruck		Wert
`1 2 3 + +`	`1 + (2 + 3)`	=	6
`1 2 + 3 +`	`(1 + 2) + 3`	=	6
`1 2 3 + *`	`1 * (2 + 3)`	=	5
`3.1415 2 3 - *`	`3.1415 * (2 - 3)`	=	-3.1415
`4.0 0.2 + 3.0 * 5 +`	`((4.0 + 0.2) * 3.0) + 5`	=	17.6

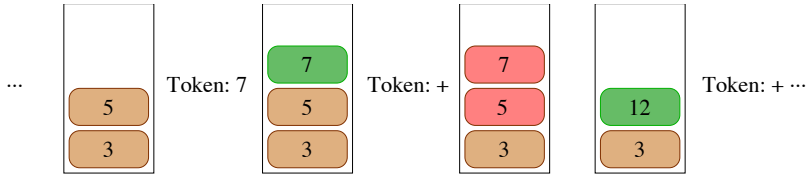

Abbildung 1: Stack während der Berechnung von “3 5 7 + +”
Grüne Zahlen werden in dem Schritt auf dem Stack abgelegt und rote Zahlen vom Stack genommen.

Die Berechnung eines in Postfix Notation geschriebenen Ausdrucks erfolgt wie folgt. Zunächst wird der als Zeichenkette bestehende Ausdruck (z.B. “`0.3 -2 +`”) in die von Leerzeichen getrennte Abschnitte unterteilt (“`0.3`”, “`-2`” und “`+`”). Jede dieser Einheiten, die selbst wieder eine Zeichenkette ist, bezeichnen wir als ein *Token*. Die Token werden dabei nacheinander, von links nach rechts, verarbeitet:

- Wenn es sich bei dem Token um eine, als Zeichenkette repräsentierte, Zahl handelt, dann konvertiere die Zahl in ein Zahlenformat (siehe `atof`) und platziere sie auf dem Stack (engl. push).
- Wenn es sich bei dem Token um einen Operator (`+`, `-` und `*`) handelt, dann nehme die obersten zwei Elemente vom Stack (engl. pop), führe die dem Operator entsprechende Operation aus und lege das Resultat der Operation wieder auf den Stack.
  - Wenn der Stack zur Zeit der Abfrage kein Elemente enthält, dann gibt er ein `NAN` (“not-a-number”) zurück. Dieses Symbol ergibt bei jeder Berechnung ebenfalls ein `NAN` und ermöglicht es damit, Fehler schnell zu finden.
- Wenn es sich bei dem Token um ein anderes (als die genannten) Zeichen (oder Zeichenketten: Kommazahlen) handelt, dann ignoriere das Token und fahre mit dem nächsten Token fort.

Das Programm endet, wenn alle Token abgearbeitet wurden.

Um das Programm zu vereinfachen, muss die Vorgabe verwendet werden, zusätzlich muss die Bibliothek `introprog_input_stacks-rpn.c` und `introprog_input_stacks-rpn.h` wie angegeben eingebunden werden. (Da die Ausgabe zu groß ist, wurde sie weggelassen.)

Listing 1: Programmbeispiel

```
> gcc -std=c99 -Wall introprog_stacks-rpn.c introprog_input_stacks-rpn.c \
      -o introprog_stacks-rpn
> ./introprog_stacks-rpn
```

Zusätzlich benötigst du in dieser Aufgabe den Befehl **double** `atof(`**const char** `*string)`. Die Funktion `atof()` wandelt die Zeichenkette, auf die `string` zeigt, in eine Zahl des Formats **double** um.

Listing 2: Beispiel für atof

```
char *string = "-1.2";
float number = atof(string);
printf("String: %s Zahl: %d\n", string, number);
```

Dein Programm soll die folgenden Bedingungen erfüllen:

- **Funktionen:**
  Programmiere die folgenden vier Funktionen und verändere nicht die anderen Funktionen in der Vorgabe:
  - `stack_push(stack *astack,` **float** `value)`
    Füge Element mit dem Wert `value` am Anfang des Stacks ein.
  - `stack_pop(stack *astack)`
    Nehme das zuletzt eingefügte Element vom Stack und gebe den enthaltenen Wert zurück. Gebe `NAN` zurück, wenn der Stack leer ist.
  - `stack* stack_erstellen()`
    Erstelle ein Stack (dynamische Speicherreservierung & Initialisierung) und gebe einen Pointer auf den Stack zurück.
  - **void** `process(stack *astack,` **char*** `token)`
    Verarbeite die Token wie oben beschrieben.
- **Datenstrukturen:**
  Mache dabei Gebrauch von den bestehenden Datenstrukturen:

Listing 3: Vorgabe `introprog_stacks-rpn_vorgabe.c`

```
typedef struct _stack stack;
typedef struct _stack_element stack_element;

struct _stack {
    stack_element* top;
};

struct _stack_element {
    stack_element* next;
    float value;
};
```

- **Speicherverwaltung:**
  Der Speicher soll zum Ende des Programms vollständig freigegeben sein.

Verwende folgende Vorgabe und füge Deinen Code an den entsprechenden Stellen ein:

Listing 4: Vorgabe `introprog_stacks-rpn_vorgabe.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>   //definiert den speziellen Wert NaN für floats
#include "introprog_input_stacks-rpn.h"

typedef struct _stack stack;
typedef struct _stack_element stack_element;

struct _stack {
    stack_element* top;
};

struct _stack_element {
    stack_element* next;
    float value;
};

/* Füge Element am Anfang des Stacks ein
 *
 * astack    - Ein Pointer auf den Stack.
 * value     - Zahl die als neues Element auf den Stack gelegt werden soll.
 */
void stack_push(stack* astack, float value)
{
    /* HIER implementieren */
}

/* Nehme das letzte eingefügte Element vom Anfang des Stack
 * Gebe NaN zurück, wenn keine Element vorhanden ist.
 *
 * astack    - Ein Pointer auf den Stack
 *
 * Gebe die im Element enthaltenen Zahl zurück
 */
float stack_pop(stack* astack)
{
    /* HIER implementieren */
}

/*
 * Führt abhängig von dem Token eine entsprechende Operation auf dem Stacks
    ↪ aus.
 * Wenn es sich bei dem Token um
 *  -> eine Zahl handelt, dann case die Zahl mithilfe von atof zu einem float
 *     und lege sie auf den Stack.
 *  -> einen Operator handelt, dann nehme zwei Zahlen vom Stack, führe die
 *     Operation aus und lege das Resultat auf den Stack.
 *  -> eine nichterkennbare Zeichenkette handelt, dann tue nichts.
 *
 * astack    - Ein Pointer auf den Stack
```

```
 * token     - Eine Zeichenkette
 */
void process(stack* astack, char* token)
{
    /* HIER implementieren */
    printf("\n<Logik fehlt!>\n");
    return;
    /* Du kannst zur Erkennung der Token folgende Hilfsfunktionen benutzen:
     *
     * Funktion         | Rückgabewert von 1 bedeutet
     * -----------------|---------------------------
     * is_add(token)    | Token ist ein Pluszeichen
     * is_sub(token)    | Token ist ein Minuszeichen
     * is_mult(token)   | Token ist ein
     *                  |  Multiplikationszeichen
     * is_number(token) | Token ist eine Zahl
     */
}

/* Debugausgabe des Stack
 * Diese Funktion kannst du zum debugging des Stack verwenden.
 *
 * astack    - Ein Pointer auf den Stack
 */
void print_stack(stack *astack) {
    int counter = 0;
    printf("\n |xxxxx|xxxxxxxxxxxxxxxxxx|xxxxxxxxxxxxxxxxxx|xxxxxxxxx|\n");
    printf(" | Nr. | Adresse           | Next              | Wert    |\n");
    printf(" |-----|-------------------|-------------------|---------|\n");
    for (stack_element* elem=astack->top; elem != NULL; elem = elem->next) {
        printf(" | %3d | %17p | %17p | %7.3f |\n", counter, elem, elem->next,
            ↪  elem->value);
        counter++;
    }
    printf(" |xxxxx|xxxxxxxxxxxxxxxxxx|xxxxxxxxxxxxxxxxxx|xxxxxxxxx|\n");
}

/* Kreiere einen Stack mit dynamischen Speicher.
 * Initialisiere die enthaltenen Variablen.
 *
 * Gebe einen Pointer auf den Stack zurück.
 */
stack* stack_erstellen() {
    /* HIER implementieren */
}

int main(int argc, char** args)
{
    stack* astack = stack_erstellen();
    char zeile[MAX_STR];
    char* token;

    intro();
    while (taschenrechner_input(zeile) == 0) {
```



```
        // Erstes Token einlesen
        token = strtok(zeile, " ");

        while (token != NULL) {
            printf("Token: %s\n", token);
            // Stackoperationen durchführen
            process(astack, token);
            // Nächstes Token einlesen
            token = strtok(NULL, " ");
            print_stack(astack);
        }

        printf("\nExtrahiere Resultat\n");
        float result = stack_pop(astack);
        print_stack(astack);

        if (astack->top != NULL) {
            while (astack->top != NULL) {
                stack_pop(astack);   //Räume Stack auf
            }
            printf("\nDoes not Compute: Stack nicht leer!\n");
        } else if (result != result) {
            printf("\nDoes not Compute: Berechnung fehlgeschlagen!\n");
        } else {
            printf("\nDein Ergebnis:\t%7.3f\n\n", result);
        }
    }
    free(astack);
}
```

Hinweise:

- Diese Aufgabe verwendet gerade in der Vorgabe einige Funktionen, die Du noch nicht kennengelernt hast. Falls Du wissen möchtest, was diese tun, so kannst Du das mit dem folgendem Befehl in der Kommandozeile in Erfahrung bringen:

  `> man <Funktionsname>`

- In dieser Aufgabe wird der Wert `NAN` verwendet. Diese Nicht-Zahl wird in der Bibliothek `math.h` definiert. Für eine beliebige Zahl x gilt: NAN $+ x =$ NAN, NAN $- x =$ NAN und NAN $* x =$ NAN. Diese Eigenschaft machen wir uns in dieser Aufgabe zu Nutze, um Fehler zu finden. Allerdings hat `NAN` noch eine weitere Eigenschaft: NAN $\neq$ NAN, d.h. der Wert `NAN` ist nicht nur ungleich jeder Zahl, sondern unterscheidet sich ebenso von sich selbst. Diese Eigenschaft benötigen wir in dieser Aufgabe nicht, aber sie kann, wenn nicht beachtet, zu Problemen führen.

Gib Deinen Quelltext in einer Datei mit folgendem Namen ab:
`introprog_stacks-rpn.c`

2. Aufgabe: Theorieaufgaben (1 Punkt)

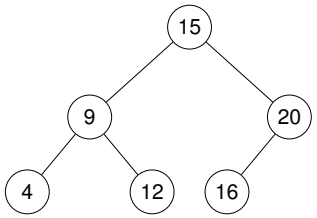

Abbildung 2: Binärbaum

2.1. In-order-tree-walk (0,5 Punkte) Durchlaufe den gegebenen Baum nach dem in der Vorlesung vorgestellten Verfahren des In-Order-Tree-Walks. Gib die Reihenfolge an, in der die Knoten bearbeitet werden (auf den Slides wurden sie ausgegeben).

2.2. Queue (0,5 Punkte) Füge unter der Verwendung der (in der Vorlesung vorgestellten) `enqueue`-Funktion die folgenden Werte in dieser Reihenfolge in eine initial leere FIFO Queue ein: `15, 3, 6, 10, 11`. Gib die Reihenfolge an, in welcher die Elemente durch die dequeue-Funktion entnommen werden.

Hinweis: Die Abgabe der Theorieaufgaben erfolgt wiederum über SVN. Dazu findet ihr die Datei `introprog_walk_queue_vorgabe.txt` im `Vorgaben`-Ordner dieses Blatts. In der Datei befinden sich genaue Angaben wie die Abgabe zu erfolgen hat. Gebt eure Abgabe als `introprog_walk_queue.txt` ab.