SYSPROG BEISPIEL 3

Calculator

Aufgabenstellung

Schreiben Sie einen Taschenrechner als Vaterprozess, der seinen Input von *stdin* als String einliest und diesen String an einen Kindprozess weiterleitet. Dieser wandelt diesen String in Zahlen um, berechnet daraus das Ergebnis und gibt dieses wieder zurück. Dieses Ergebnis wird dann vom Vaterprozess anschließend am Bildschirm wieder ausgegeben. Da als Zahlen Integerwerte (maximal 65536) angenommen werden sollen, kann die Eingabezeichenkette auf 15 Zeichen beschränkt werden.

```
SYNOPSIS:
        calculator
        $> <zahl1> <zahl2> <operator>

BNF:
        <zahl>     ::= -?[0-9]+
        <operator> ::= +|-|*|/
```

Beispielsweise soll die Eingabe ’10 15 +’ das Ergebnis ’25’ liefern.

Anleitung

Das Programm soll nach dem Starten in einer Schleife wiederholt Rechnungen von der Tastatur (*stdin*) einlesen. Zur Berechnung soll Ihr Programm einen Kindprozess erzeugen, an den es die Eingabe über eine Pipe weiterleitet. Dieser Kindprozess leitet nach der Umwandlung und Berechnung das Ergebnis über eine zweite Pipe an den Vaterprozess zurück. Der Vaterprozess gibt dieses Ergebnis wieder am Bildschirm (*stdout*) aus.

Dieser Vorgang soll solange wiederholt werden, bis der Vaterprozess EOF (Ctrl-D von der Tastatur) liest. In diesem Fall ist die Pipe zum Kindprozess zu schließen. Der Kindprozess erhält dadurch beim Lesen ebenfalls EOF und terminiert. Der Vaterprozess soll auf die Terminierung des Kindprozesses warten, alle benötigten Resourcen (Pipes) an das System zurückgeben (schließen) und dann ebenfalls terminieren.

Sie können davon ausgehen, dass alle Rechnungen richtig eingegeben werden - dh. es ist keine Syntaxüberprüfung notwendig. Auch können Sie die Ergebnisse abrunden (Kommastellen abschneiden) - dh. die Eingabe ’5 2 /’ ergibt als Ergebnis einfach ’2’ anstelle von ’2.5’.

Bitte beachten Sie auch die Allgemeinen Hinweise zur Beispielgruppe 3 und die Richtlinien für die Erstellung von C-Programmen auf der Übungs-Website.

Richtlinien für die Erstellung von C-Programmen

Gehen Sie, bevor Sie Ihr Programm abgeben, alle diese Richtlinien noch einmal durch und überprüfen Sie, ob Ihr Programm auch alle diese Richtlinien erfüllt. Sind Sie sich bei einem Punkt im Unklaren, ob er von ihrem Programm erfüllt wird, so fragen Sie einen Betreuer.

Es gibt fünf Gründe, weshalb bei der Beispielabgabe Punkte abgezogen werden können bzw. weshalb ein Beispiel zurückgewiesen wird:

- Das Programm ist funktional nicht korrekt.
- Sie wissen über die Programmlogik nicht oder zu wenig gut Bescheid.
- Fehlendes Verständnis für das Stoffgebiet des Beispiels.
- Die formalen Anforderungen an C-Programme wurden nicht erfüllt.
- Verspätete Abgabe.

Jedes Beispiel kann maximal 1x zurückgewiesen werden (Reject).

Verpflichtende Anforderungen

Ein Programm, das diesen Anforderungen nicht genügt, wird nicht abgenommen.

1. Das Programm muss mit gcc -ansi -pedantic -Wall -g -c filename.c ohne Warnings und Info-Meldungen compiliert werden können (siehe Buch S. 182!). Sehen Sie keine Möglichkeit, eine Compilerwarnung zu entfernen, so fragen Sie einen Betreuer.
2. Es muss für jedes Programm ein Makefile geben, das mindestens folgende Einträge enthält:
   - all zum Erzeugen des Programms aus den Sourcen. Dies muss das erste Target im Makefile sein.
   - clean zum Löschen aller Files, die aus den Sourcen erzeugt werden können.
3. Im fehlerfreien Fall muß das Programm mit dem Exit-Code 0 terminieren, im Fehlerfall mit einem Exit-Code größer als 0.
4. Die Funktionen gets, scanf, fscanf, atoi und atol dürfen nicht verwendet werden (Absturzsicherheit bei falschen Eingaben!).

verboten	**stattdessen zu verwenden**
gets	fgets
scanf	fgets, sscanf
fscanf	fgets, sscanf
atoi	strtol
atol	strtol

5. Argumente müssen korrekt nach den in UNIX gültigen Konventionen (siehe Skriptum) behandelt werden (dies wird z.B. durch die Verwendung von getopt gewährleistet).
6. “Busy waiting” (das wiederholte Abfragen einer Bedingung in einer Schleife zur Synchronisation von Prozessen) darf nicht verwendet werden.

7. “Synchronisation” mittels sleep ist verboten.
8. Prozeduren (d.h. Funktionen, die keine Werte zurückliefern) sind als *void* zu deklarieren.
9. Bei Funktionsdefinitionen und Prototypen sind die Typen der Parameter innerhalb der Funktionsklammern anzugeben (new style, ANSI-C style).
10. Der Kommentar am Kopf eines Moduls hat folgende Punkte zu enthalten:
    - Name des Moduls.
    - Name und Matrikelnummer des Autors.
    - Zweck des Moduls.
    - Datum der Erstellung des Moduls.
    - Nummer des Beispiels (nur im Modul des Hauptprogramms).

    Auch das Makefile hat einen Modulkopf zu enthalten.

Anforderungen, deren Verletzung zu Punkteabzügen führen

Werden die folgenden Anforderungen nicht erfüllt, so führt dies zu Punkteabzügen.

1. Fehlermeldungen sind auf die Standardfehlerausgabe (*stderr*) zu schreiben und haben den Programmnamen (*argv[0]*) zu enthalten.
2. Bei fehlerhaftem Aufruf des Programmes ist eine usage Meldung mit dem Programmnamen und der korrekten Aufrufsyntax auszugeben.
3. Vor jede Funktion ist ein Kommentar zu stellen, der folgende Punkte enthält:
    - Name der Funktion.
    - Zweck der Funktion.
    - Beschreibung der Parameter der Funktion.
    - Beschreibung des Return-Wertes der Funktion.
    - Beschreibung der globalen Variablen, die die Funktion benutzt.
4. Kann eine Funktion einen Fehlercode zurückliefern, so ist dieser abzufragen. (Zu dieser Regel gibt es einige Ausnahmen: *fprintf(stderr, ...)*, *fclose(...)* auf ein File, das zum Lesen geöffnet wurde, ...). Überall, wo es sinnvoll ist, muss der Return-Code abgefragt werden!
5. Alle Beispiele sind soweit als möglich mit ANSI-C Bibliotheksfunktionen zu lösen. Sind keine äquivalenten ANSI-C Funktionen vorhanden, können Sie auch andere Funktionen verwenden. Vermeiden Sie, das Rad zweimal zu erfinden! (z.B. Nachprogrammierung von *strcmp*).
6. Konstantendefinitionen sind gross zu schreiben, Namen von Variablen klein (ev. mit grossem Anfangsbuchstaben).
7. Seiteneffekte in mit && und || zusammengesetzten logischen Ausdrücken sind (ausser im am weitesten links stehenden Teil des Ausdrucks) zu vermeiden.
8. Dateien sind nach Verwendung wieder zu schließen. Ebenso sind angelegte Resourcen nach Verwendung wieder zu löschen.

9. Willkürliche Grenzen sind soweit als möglich zu vermeiden. Ist es notwendig, Grenzen einzuführen, so ist die Überschreitung dieser Grenzen zu behandeln. Alle diese Grenzen sind als symbolische Konstante zu vereinbaren.
10. Wichtige Konstanten sollen symbolische Namen erhalten (Makrodefinitionen!).
11. Jede switch Anweisung soll einen default: Zweig enthalten. Kann dieser nicht erreicht werden, so ist an dieser Stelle *assert(0)* zu schreiben (defensive Programmierung!).
12. Logische Werte sind mit logischen Operatoren zu behandeln, numerische Werte mit arithmetischen Operatoren (z.B. nicht *!strcmp(...)* verwenden!).
13. Wo es sinnvoll ist, sind sprechende Variablennamen zu verwenden.
14. Sinnvolle Kommentare sind zu verwenden, solche, die nur wiederholen, was im Code ohnehin steht, sind zu vermeiden (z.B. *i = i + 1; /* i wird um eins erhoeht */*).
15. 'Trickreiche' arithmetische Ausdrücke sind zu vermeiden. Das Programm ist nicht umso besser, je kürzer es ist!
16. Eine konsistente Konvention zum Einrücken ist zu verwenden.
17. Die Verwendung globaler Variablen ist soweit als möglich zu vermeiden.
18. Resourcenverschwendung und umständliche Programmierung sind zu vermeiden. Codestücke dürfen nicht im Headerfile stehen (Ausnahme: Makrodefinitionen).

Allgemeine Hinweise zum Beispiel 3

- **Argumentbehandlung:** Vergessen Sie auch bei diesem Beispiel nicht auf die Argumentbehandlung (auch bei einem Programm welches keine Argumente erhält ist eine Argumentbehandlung durchzuführen)!
- **Pipes:** Falls Sie Pipes erzeugen, sollte das wie im Übungsskriptum beschrieben geschehen. Für sämtliche I/O-Operationen (Lesen/Schreiben von *stdin*, *stdout* Pipes usw.) sind ausschließlich die Standard I/O-Funktionen (*fdopen(3)*, *fopen(3)*, *fgets(3)*, etc.) zu verwenden. Bedenken Sie hierbei, dass die Standard I/O Funktionen gepuffert sind.
- **Ressourcen:** Alle Ressourcen (wie z.B. Pipes) müssen ordnungsgemäß vor Terminierung entfernt werden.
- **Terminierung:** Die Terminierung aller Kindprozesse sicherzustellen ohne *kill(2)* oder *killpg(2)* zu verwenden. Der Exit Status beendeter Kindprozesse muss vom Vaterprozess abgeholt werden (*wait(2)*, *waitpid(2)*, *wait3(2)*).
- **Signalbehandlung:** Eine Signalbehandlung ist für diese Beispielgruppe (mit Ausnahme von einem Beispiel, bei dem explizit eine Signalbehandlung gefordert wird) nicht erforderlich!