TECHNISCHE UNIVERSITÄT
CHEMNITZ

Professur Betriebssysteme

Algorithmen und Programmierung, WS 2022/2023

Belegaufgabe #7: **Alternativer Stringtyp**

Deadline: **29. Januar 2023, 23.00 Uhr**
Erreichbare Punkte: **11**

1 Motivation

Ziel dieser Aufgabe ist…

- …Modularisierung von Code;
- …der Umgang mit make und Makefiles;
- …der Umgang mit Archiven (statische Bibliotheken);

2 Alternativer Stringtyp

Bekanntlich nutzt C anstatt eines Stringtyps ein Array von `char`, dessen letztes Element ein `'\0'` ist. Andere Sprachen machen dies anders, beispielsweise werden in Pascal stets 256 Bytes für einen String reserviert, wobei die tatsächliche Länge des Strings (d. h. des Textes) im Typ codiert wird. Das hat Vorteile: Beispielsweise kann ein String auch das ASCII-Steuerzeichen `NUL` enthalten, oder Operationen wie eine Stringverkettung können viel leichter durchgeführt werden. Dem stehen Nachteile gegenüber: Einerseits sind keine längeren Strings als mit 255 Zeichen möglich, andererseits wird meist zuviel Speicher verschwendet. In dieser Aufgabe sollen Sie eine Reihe von Funktionen schreiben, die mit einem pascal-ähnlichen Stringtyp arbeiten.

2.1 C-Typ

Der alternative Stringtyp `astr` soll sich nutzen lassen, auch ohne Interna wie z. B. das Speicherlayout (siehe Abschnitt 2.2) zu kennen. Daher wird er als Zeigertyp auf ein dynamisches Array von `char` (Forward Deklaration) zur Verfügung gestellt:

```
typedef char* astr;
```

Eine direkte Nutzung von `char*` wurde vermieden, da dabei viele C-Programmierer vermuten würden, dass es sich um einen (nullterminierten) C-String handelt.

2.2 Speicherlayout

Im Speicher hat der alternative String folgenden Aufbau:

- Die ersten zwei Byte bilden einen Header, der die Größe des gespeicherten Strings angibt. Die maximale Länge des Textes beträgt somit 2^{16} Bytes.
- Der eigentliche String beginnt mit dem dritten Byte und ist genauso lang, wie die aktuelle Stringlänge im Header angibt.
- Der angelegte String ist jetzt nicht mehr zwingend Null-terminiert, da die Information über seine Länge in den ersten zwei Bytes gespeichert ist.
- Es wird stets genau so viel Speicher reserviert, wie auch von dem gespeicherten String und den Header-Bytes benötigt wird.

Die Abbildungen 1 und 2 zeigen das Gesamtlayout sowie eine Beispielbelegung mit dem Text „Hello, World!“.

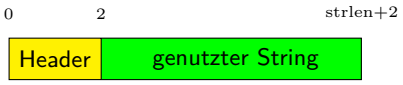

Abbildung 1: Layout des gesamten String-Typs

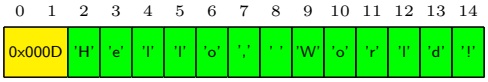

Abbildung 2: Beispiel: String „Hello, World“

3 Aufgabenstellung

Die Aufgabe besteht aus einer Reihe von Unteraufgaben, die einzeln bepunktet werden. Sie können also auch Punkte erhalten, wenn einzelne Unteraufgaben nicht (vollständig) gelöst werden. Allerdings muss auch bei dieser Aufgabe der Validierungstest bestanden werden. Dies ist nicht der Fall, wenn Ihr Code Übersetzungsfehler erzeugt oder Ihr Makefile nicht lauffähig ist.

Punkte für übergreifende Fehler (wie der Verstoß gegen formale Regeln oder die Nutzung verbotener Konstrukte) werden *nach* den Einzelaufgaben abgezogen.

Jede zu schreibende C-Funktion muss in einer eigenen Datei definiert werden, die genauso heisst, wie die darin definierte Funktion (zuzüglich Dateiextension '".c"').

a) *(1 Punkt)* Schreiben Sie eine Funktion

```
astr asfromcstr(char *cstr);
```

Diese gibt einen `astr`-String zurück, der mit dem Inhalt des C-Strings `cstr` initialisiert wird. Der neu angelegte String soll möglichst speichereffizient sein. Lässt er sich nicht anlegen oder hat `cstr` den Wert `NULL`, soll `NULL` zurückgegeben werden. Das abschließende `'\0'` von `cstr` wird nicht übernommen.

b) *(1 Punkt)* Schreiben Sie eine Funktion

```
int aslen(astr str);
```

Diese gibt die Länge der Zeichenkette zurück, die `str` beinhaltet. Ist keine Bestimmung der Länge möglich, gib -1 zurück.

c) *(1 Punkt)* Schreiben Sie eine Funktion

```
int asgetc(astr str, int index);
```

Diese gibt den ASCII-Code des Zeichens an der Stelle `index` innerhalb der Zeichenkette in `str` zurück. Die Indexzählung beginnt wie in C üblich mit 0 (ab dem ersten Zeichen nach den Header-Bytes). Wenn `index` ungültig ist, gebe -1 zurück.

d) *(1 Punkt)* Schreiben Sie eine Funktion

```
int assetc(astr str, int index, char c);
```

Diese setzt das Zeichen an der Stelle `index` der Zeichenkette in `str` auf `c`. Die Indexzählung beginnt wie in C üblich mit 0 (ab dem ersten Zeichen nach den Header-Bytes). Falls `index` ungültig ist, gebe -1 zurück, sonst 0.

e) *(1 Punkt)* Schreiben Sie eine Funktion

```
int asconcat(astr *str1, astr str2);
```

Diese hängt die Zeichenkette von `str2` an die Zeichenkette von `str1`. Die Länge von `str1` muss entsprechend aktualisiert werden. `str2` wird nicht freigegeben. Falls ein Zusammenfügen nicht möglich ist, wird -1 zurückgegeben, sonst 0.

f) *(2 Punkte)* Schreiben Sie eine Funktion

```
void fputas(FILE *stream, astr str);
```

Diese gibt die Zeichenkette von `str` in die Datei `stream` aus. Wenn `stream` eine Konsolenausgabe, d. h. `stdout` oder `stderr` ist, werden alle nichtdruckbaren Zeichen (ASCII-Steuerzeichen) durch „_“ (Unterstrich) ersetzt.

g) *(4 Punkte)* Schreiben Sie eine Datei „`Makefile`“, die folgende Ziele (*goals*) realisiert:

   a) `objs`: Baut alle Objektdateien, die noch nicht aktuell übersetzt vorliegen.

   b) `lib`: Baut eine statische Bibliothek[1] „`libastring.a`“, die alle Funktionen beinhaltet.

[1] Die Erzeugung von Bibliotheken wird im Kapitel 15 behandelt.

c) `clean`: Löscht alle angelegten Dateien.

d) Zusätzlich soll es möglich sein zu den Zielen `objs` und `lib` den Ausdruck `DEBUG=1` anzugeben (Beispiel: `make lib DEBUG=1`). Hierdurch sollen die entsprechenden Versionen mit Debugsymbolen[2] erzeugt werden. Die Bibliothek *mit* den Debugsymbolen soll den Namen `libastring-deb.a` tragen. Ohne die Angabe von `DEBUG=1` dürfen keine Debugsymbole in den Objektdateien oder der Bibliothek enthalten sein.

3.1 Anmerkungen

Bei Bedarf dürfen Sie weitere (Hilfs-)funktionen schreiben. Diese dürfen aber nicht nach außen sichtbar sein (siehe Abschnitt 3.2).

Die Deklaration von `astr` sowie alle Funktionsdeklarationen sind in einem Headerfile „`astring.h`“ enthalten, das Ihnen über folgende URL zur Verfügung gestellt wird `https://osg.informatik.tu-chemnitz.de/lehre/aup/support/astring.h`. Sie dürfen dieses File nutzen, es aber nicht verändern. Dieses Headerfile wird auf der Testmaschine zur Verfügung gestellt, sie brauchen es also nicht Ihrem Abgabearchiv beifügen. Sie können aber bei Bedarf ein eigenes Headerfile (z. B. mit internen Deklarationen) anlegen, nutzen und der Abgabe beifügen.

3.2 Einschränkungen

Bei der Lösung der Aufgabe unterliegen Sie Einschränkungen:

- Sie dürfen **keine** der vorgegeben Signaturen verändern.
- Sie dürfen **in keinem der Module** globale oder statische Variablen benutzen.
- Sie dürfen **nicht** das Schlüsselwort `goto` verwenden.
- Sie dürfen **keine** externen Funktionen außer den im C99-Standard vorgesehenen Funktionen der Standardbibliothek nutzen. Insbesondere dürfen Sie keine anderen Bibliotheken einbinden.
- Ihr Code darf **keine** Ausgabe tätigen, mit Ausnahme der beabsichtigten Ausgabe bei `fputas()`.

4 Hinweise zur Aufgabe

- Ein wesentlicher Teil dieser Aufgabe ist der Umgang mit Makefiles. Es emfiehlt sich daher, Ihr Wissen über make/Makefiles zu festigen bzw. zu vertiefen, z. B. über „`GNU make`“ (siehe `https://www.gnu.org/software/make/manual/make.pdf`)
- Testen Sie gründlich (siehe nächster Abschnitt)!
- Beachten Sie, dass Speicherfehler (z. B. Dereferenzierung von *dangling* Zeigern oder *memory leaks*) Programmierfehler darstellen.

[2] Zur Erinnerung: sowohl `gcc` als auch `clang` erzeugen Debugsymbole bei Angabe der Compileroption „`-g`“

Testen

Beachten Sie, dass Ihre Abgabe der Quellcode zur Erschaffung einer Bibliothek und kein vollständiges Programm ist. Da Sie für den Test Ihrer Funktion jedoch ein ausführbares Programm benötigen, können Sie zum Testen z. B. folgenden Code nutzen, zu dem Sie Ihre Bibliothek mit der Option „`-lastring`“ hinzulinken.

```c
#include <stdio.h> // for stdout
#include <stdlib.h>

#include "astring.h"

int main() {
    astr str1 = asfromcstr("Hello!");
    astr str2 = asfromcstr("World ");
    int len_str1 = aslen(str1);
    printf("Length of str1 is %u\n", len_str1);
    assetc(str2, 5,
           asgetc(str1,5));  // str2: "World " -> "World!"
    assetc(str1, 5, ' ');    // str1: "Hello!" -> "Hello "
    asconcat(&str1, str2);   // str1: "Hello " -> "Hello World!"
    int concat_len = aslen(str1);
    printf("Length of str1 now is %u\n", concat_len);
    assetc(str1,5,'\0');     // str1: "Hello\0World!"
    fputas(stdout, str1);    // "Hello_World!"
    printf("\n");
    free(str1);
    free(str2);
    return 0;
}
```

5 Einreichung

- Erstellen Sie ein Archiv in Form einer .zip- oder einer .tar-Archivdatei, welche alle Lösungsdateien (sechs C-Files, ein Makefile, ggf. ein eigenes Headerfile) direkt (also nicht in einem Unterordner) enthält.
- Fügen Sie dem Archiv keine weiteren Dateien oder Ordner hinzu, insbesondere auch nicht Objectdateien oder Archiv-/Bibliotheksdateien.
- Reichen Sie Ihre Lösung unter `https://osg.informatik.tu-chemnitz.de/submit` ein.
- Bis zum Abgabeende (29. Januar 2023, 23.00 Uhr) können beliebig neue Lösungen eingereicht werden, die die jeweils älteren Versionen ersetzen.
- Jede Ihrer C-Dateien muss auf der Testmaschine mit den Optionen „`-std=c99 -Wall -Werror`“ übersetzbar sein[3], ebenso muss das Makefile ausführbar sein. Details zur Testmaschine sind auf dem OpenSubmit-Dashboard verfügbar.

[3] Es wird daher empfohlen, dass Sie diese Optionen auch in Ihrem Makefile benutzen.

- Ihre Lösung wird automatisch auf die Einhaltung formaler Kriterien validiert. Ihre Lösung wird nur bewertet, wenn die Validierung erfolgreich war.

14. Januar 2023