Universität Hamburg, Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen
E. Betke, N. Hübbe, M. Kuhn, J. Lüttgau, J. Squar

Übungsblatt 12 zum Praktikum
C-Programmierung
im WiSe 2019/2020
Abgabe: 02.02.2020, 23:59

Auf diesem Übungsblatt soll eine Datenstruktur implementiert werden, deren Schnittstelle bereits fest vorgegeben ist. Die Datenstruktur finden Sie in den Dateien `data.c` und `data.h`; die Datei `data.h` und die Signaturen der Funktionen in der Datei `data.c` dürfen **nicht** verändert werden.

Die Materialien enthalten außerdem eine naive Implementierung einer Hash-Tabelle (`hash.c` und `hash.h`) sowie ein Anwendungsprogramm, das die Datenstruktur nutzt (`exercise.c`); auch diese Dateien dürfen **nicht** verändert werden.

1 Implementierung

Konkret handelt es sich bei der zu implementierenden `data`-Datenstruktur um eine opake Datenstruktur, so dass dem Anwendungsprogramm der interne Aufbau verborgen bleibt. Der Header enthält daher nur eine Deklaration der Datenstruktur:

```
1 typedef struct data data;
```

1.1 Struktur (15 Punkte)

Die Grundlage der zu implementierenden Datenstruktur bildet die `data`-Struktur, deren interner Aufbau nur der Implementierung der Datenstruktur bekannt sein soll:

```
1 struct data
2 {
3 };
```

Analysieren Sie die restlichen Funktionen, um die notwendigen Komponenten der Struktur zu bestimmen.

1.2 Erstellung (30 Punkte + 30 Bonuspunkte)

Die Datenstruktur soll sowohl Strings (Basis) als auch Blobs (Bonus) enthalten können. Für beide Datentypen muss jeweils eine `new`-Funktion implementiert werden. Beachten Sie dabei, dass ein übergebener Blob nicht unbedingt null-terminiert sein muss.

```
1 /* "content" is a null-terminated string. */
2 data* data_new_string (char const* content)
3 {
4     return NULL;
5 }
```

```c
/* "content" is a blob of length "length". */
data* data_new_blob (char const* content, unsigned int length)
{
    return NULL;
}
```

1.3 Reference Counting (20 Punkte)

Zur Verwaltung der Datenstruktur soll Reference Counting benutzt werden, so dass der allokierte Speicher erst freigegeben wird, wenn der Reference Count `0` erreicht.

```c
data* data_ref (data* data)
{
    return NULL;
}

/* Frees memory allocated by "data" if reference count reaches 0. */
void data_unref (data* data)
{
}
```

1.4 Hilfsmittel (30 Punkte)

Um den Inhalt der Datenstruktur ausgeben zu können, soll eine `as_string`-Funktion implementiert werden. Enthält die Datenstruktur einen String, soll das Präfix `String:` gefolgt vom eigentlichen String ausgegeben werden (Beispiel: `String: Hallo Welt #0!`). Enthält die Datenstruktur einen Blob, soll das Präfix `Blob:` gefolgt von der Speicheradresse des Blobs ausgegeben werden (Beispiel: `Blob: 0xdeadc0de`). Der Speicher für den String soll dabei innerhalb der Funktion allokiert und an den Aufrufer übergeben werden.

```c
/* Returns a newly-allocated string that must be freed by the caller. */
char* data_as_string (data const* data)
{
    return NULL;
}
```

1.5 Hashing (30 Punkte)

Um die Datenstruktur in eine Hash-Tabelle einfügen zu können, muss außerdem eine Hashfunktion implementiert werden. Dabei soll basierend auf dem Inhalt der Datenstruktur ein vorzeichenfreier Integer zurückgegeben werden.

```c
unsigned int data_hash (data const* data)
{
```

```
3     return 0;
4 }
```

1.6 Sortierung (30 Bonuspunkte)

Um ein Array der Datenstruktur mit `qsort` sortieren zu können, muss eine Vergleichsfunktion implementiert werden. Diese wird durch die vorgegebene Wrapper-Funktion `data_cmp_qsort_cb` aufgerufen und bekommt zwei Datenstrukturen übergeben. Implementieren Sie die Funktion so, dass das Array längen-lexikographisch sortiert wird. Das Verhalten der Funktion (insbesondere der erwartete Rückgabewert) ist in der Manpage zu `qsort` beschrieben.

```
1 int data_cmp (data const* a, data const* b)
2 {
3     return 0;
4 }
```

2 Evaluation (60 Punkte)

Kompilieren Sie das Anwendungsprogramm mithilfe des mitgelieferten Makefiles. Wenn Ihre Implementierung der Datenstruktur korrekt funktioniert, sollte die Anwendung 128 Array-Einträge ausgeben. Überprüfen Sie außerdem mithilfe von Valgrind, dass keine Speicherlecks auftreten (`valgrind --leak-check=full --show-leak-kinds=all`).

Abgabe

Abzugeben ist ein gemäß den bekannten Richtlinien erstelltes und benanntes Archiv. Das enthaltene und gewohnt benannte Verzeichnis soll folgenden Inhalt haben:

- Alle Quellen, aus denen Ihr Programm besteht (`data.c`, `data.h`, `exercise.c`, `hash.c`, `hash.h`, `Makefile`); gut dokumentiert (Kommentare im Code!)

Senden Sie Ihre Abgabe an `cp-abgabe@wr.informatik.uni-hamburg.de`.