C Standardbibliothek

Software Entwicklung 1

Annette Bieniusa, Mathias Weber, Peter Zeller

Dieses Dokument gibt einen kurzen Überblick über die Funktionen der C Standardbibliothek, welche für diese Vorlesung und Übungen relevant sind.

1 Eingabe und Ausgabe

Zur Verwendung dieser Funktionen muss die Header-Datei `stdio.h` eingebunden werden.

```
#include <stdio.h>
```

Aus dieser Bibliothek verwenden wir die folgenden Funktionen:

```
int printf(const char* format, ...);
```

Nimmt einen Format-String und eine beliebige Anzahl von Werten, die für die Platzhalter im Format-String eingesetzt werden. Für die Vorlesung sind die folgenden Platzhalter relevant:

`%s`	Wert vom Typ `char*` mit `\0` als Markierung für das Zeilenende
`%d`	Dezimaldarstellung eines `int`-Wertes
`%f`	Wert vom Typ `float` mit 6 Nachkommastellen
`%.2f`	Wert vom Typ `float` mit 2 Nachkommastellen

Der Rückgabewert von `printf` ist die Anzahl der erfolgreich geschriebenen Zeichen oder eine negative Zahl, wenn Fehler aufgetreten sind.

```
int scanf(const char* format, ...);
```

Nimmt einen String, welcher spezifiziert, wie die Eingabe zu lesen ist, sowie eine beliebige Anzahl von Speicheradressen, an denen die gelesene Eingabe gespeichert wird. Wir verwenden `scanf` nur mit jeweils einem Platzhalter im Eingabe-String und einer Speicheradresse:

`scanf("%d", &x);`	Liest einen `int`-Wert ein und speichert ihn in Variable `int x`.
`scanf("%f", &x);`	Liest einen `float`-Wert ein und speichert ihn in Variable `int x`.

Der Platzhalter `"%s"` sollte aus Sicherheitsgründen nicht verwendet werden, da er nicht überprüft, ob der eingelesene String in den gegebenen Buffer passt. Außerdem können Strings mit Leerzeichen (bzw. Whitespace-Zeichen) nicht mit

dieser Funktion eingelesen werden. Auch mit Parameter `"%s\n"` ließt `scanf` somit **nicht** eine Zeile.
Der Rückgabewert ist Anzahl der erfolgreich gelesenen Eingabe-Parameter oder die negative Konstante `EOF`, wenn die Eingabe zu Ende ist.

```
char *fgets(char *str, int count, FILE *stream);
```

Liest Zeichen aus dem Eingabestrom `stream`, bis das Ende des Stroms, das Ende der Zeile oder die maximale Zeichenzahl `count-1` erreicht ist, oder ein Fehler auftritt.
Die gelesenen Zeichen werden (inklusive Zeilenumbruch) mit einem `\0`-Zeichen am Ende in den Buffer `str` geschrieben, welcher daher mindestens `count` Zeichen lang sein muss.
Als Eingabestrom `stream` verwenden wir hier immer die Konstante `stdin`, also die Standardeingabe.
Als Rückgabewert gibt die Funktion den Pointer `str` zurück, wenn keine Fehler aufgetreten sind und einen `NULL`-Pointer, wenn es einen Fehler gab.

2 Speicherverwaltung

Zur Verwendung dieser Funktionen muss die Header-Datei `stdlib.h` eingebunden werden.

```
#include <stdlib.h>
```

Diese Bibliothek verwendet und definiert den Typ `size_t`, was ein `unsigned` Integer-Typ und der Typ von `sizeof`- Ausdrücken ist. Die Größe des Typs ist abhängig vom System, unter 64bit-System aber in der Regel `long unsigned int`.
Zur Speicherverwaltung verwenden wir die folgenden Funktionen:

```
void *malloc(size_t size);
```

Alloziert einen Speicherbereich mit `size` Bytes und gibt eine Referenz auf diesen Speicherbereich zurück. Im Fehlerfall wird der Nullpointer zurückgegeben (wenn zum Beispiel `size == 0` oder nicht genug Speicherplatz vorhanden ist).

```
void free(void *ptr);
```

Gibt den Speicherbereich auf den `ptr` verweist wieder frei. Wenn `ptr` der Nullpointer ist passiert nichts. Wenn der angegebene Speicherbereich nicht gültig ist, ist das Verhalten undefiniert. Das ist zum Beispiel der Fall, wenn er schon zuvor freigegeben oder nicht mit `malloc` oder einer ähnlichen Funktion angelegt wurde.

```
void *calloc(size_t num, size_t size);
```

Wie `malloc`, aber initialisiert den neuen Speicherbereich mit `0`-Werten. Außerdem wird nicht die Gesamtgröße in Bytes, sondern die Anzahl der Einträge `num` und die Größe der einzelnen Einträge `size` als Parameter übergeben.

```
void *realloc(void *ptr, size_t new_size);
```

Die Funktion `realloc` erlaubt es, die Größe von Speicherbereichen zu erweitern bzw. zu verkleinern. Sie liefert einen Zeiger auf einen Speicherbereich der Größe `size`, der die gleichen Daten wie der durch `pointer` referenzierte Bereich enthält (bis zum Minimum der Größe von ursprünglichen und des neuen Bereichs). Neue Speicherregionen werden dabei nicht initialisiert. Der von `ptr` referenzierte Speicherbereich muss wie bei `free` ein gültiger Speicherbereich oder `NULL` sein. Falls `NULL` übergeben wird, verhält sich `realloc` wie `malloc`.
Falls die Allokation fehlschlägt oder `new_size == 0`, wird der ursprüngliche Bereich nicht verändert und `NULL` zurückgegeben.