Dynamischer Speicher

C-Kurs 2012, 3. Vorlesung

Tino Kutschbach
tino.kutschbach@campus.tu-berlin.de

http://wiki.freitagsrunde.org

13. September 2012

Inhaltsverzeichnis

Inhaltsverzeichnis

Char-Arrays

- C bietet keinen eigenen Datentyp für Zeichenketten/Strings an
- Strings sind Arrays von einzellnen char-Zeichen
- Deklaration einer Zeichenkette: `char string [10];` (1)
- Besonderheit:
  - Das Ende eines Strings wird durch das Nullzeichen (\0) markiert
  - Da das Nullzeichen immer mitgespeichert werden muss, kann der in (1) deklarierte String höchstens 9 Zeichen aufnehmen

Initialisierung von Strings

Listing 1: src/init.c

```c
// Den Zeichen lassen sich nacheinander einzelne Werte zurordnen
char string[10] = { 'Z', 'e', 'i', 'c', 'h', 'e', 'n', 'k', 'e', 't', 't', 'e', '\0' };

// Es ist jedoch bequemer einen literalen String zu verwenden
char string[10] = "Zeichenkette";

// Hier wird automatisch ein Array mit der richtigen Groesse erzeugt
char string[] = "Zeichenkette";
```

Stringspeicher zur Kompilierzeit festlegen

- Ist die nötige Größe für ein String im Vorfeld bekannt, so kann man die Arrays in gewohnter Weise deklarieren/definieren:
  - `char botschaft[20]` erzeugt einen String mit fester Anzahl an Elementen
  - `char *botschaft = "Veni, vidi, vici"` erzeugt einen String mit fester Anzahl an Elementen und initialisiert diesen
- In Situationen wo vorher unbekannt ist wie lang ein String sein muss, weil er beispielsweise von einer Nutzereingabe abhängig ist, ist diese Vorgehensweise ungeeignet
- C bietet Möglichkeiten zur Reservierung von Speicher zur Laufzeit des Programms

Inhaltsverzeichnis

Stack und Heap

- Stack
  - Lokale Variablen von Funktionen und Übergabeparameter werden auf dem Stack abgespeichert
  - Vorteil des Stack ist, dass er vom Programm selbst verwaltet wird und durch die einfache Struktur keinerlei zusätzlicher Verwaltungsaufwand nötig ist
  - Datenfelder, die auf dem Stack angelegt werden, sind, sobald sie einmal reserviert sind, fest
  - Der Zugriff auf Daten im Stack muss auf einen zulässigen Stackframe erfolgen
- Heap
  - Der Heap ist eine Ergänzung zum Stack um einen dynamischen Speicher zur Verfügung zu haben
  - Der Heap ist ein Speicherbereich aus dem zur Laufzeit zusammenhängende Speicherabschnitte angefordert und in beliebiger Reihenfolge wieder freigegeben werden können
  - Für den Heap ist eine Instanz für die Verwaltung von Speicheranfragen und Speicherfreigaben notwendig
  - Diese Verwaltungsinstanz ist Teil des Betriebssystems

C-Funktionen zur Speicherreservierung

- C stellt vier Funktionen für die Speicherreservierung auf dem Heap bereit:
  - void *malloc(size_t groesse)
    - reserviert einen Speicherblock, der die in *groesse* angegebene Anzahl von Bytes umfasst
    - bei Erfolg wird ein Zeiger auf den Speicherblock zurückgegeben, andernfalls NULL
    - der Speicher wird nicht initialisiert
  - void *calloc (size_t num, size_t size)
    - reserviert ebenfalls Speicher
    - *num* ist die Anzahl der gewünschten Elemente und *size* die Größe eines Elements
    - der komplette reservierte Speicherbereich wird mit Nullen initialisiert
- Für alle ist das einbinden der *stdlib.h* notwendig

C-Funktionen zur Speicherreservierung

- ▷ `void *realloc(void *ptr, size_t size)`
  - ändert die Größe eines Speicherblocks, der vorher mit *malloc* oder *calloc* reserviert wurde
  - muss wegen Platzmangel ein neuer Speicherblock angelegt werden, so wird der Inhalt des alten Speicherblocks in den neuen kopiert und die Adresse zu dem neuen Block zurückgegeben
  - der Speicher wird nicht initialisiert
- ▷ `void *free(void *ptr)`
  - gibt Speicherbereiche frei, die zuvor mit malloc, calloc oder realloc reserviert wurden

- Für alle ist das einbinden der *stdlib.h* notwendig

Beispiel: Dynamische Stringspeicher-Reservierung

Listing 2: src/stringmalloc.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  char *string;
4
5  int main(void) {
6    string = malloc(27);     // Speicher fuer 26 Zeichen
7    if( string == NULL ) {
8      puts("Fehler bei der Speicherzuweisung.");
9      return 1;
10   }
11   for(int i = 0; i < 26; i++) // String beschreiben
12     *(string+i) = 65+i;
13
14   *(string+26) = '\0';     // Stringende-Zeichen
15   puts(string);      // Ausgabe
16   return 0;
17 }
```

Inhaltsverzeichnis

Modularisierung

- Die Lösung großer Probleme ist oft mit einer hohen Komplexität verbunden
- Um die Komplexität in den Griff zu bekommen ist eine Aufspaltung des Problems in mehrere Teilprobleme nötig
- Jedes Teilproblem wird dann von einem eigenen Modul bearbeitet
- Hiding-Prinzip:
  - Die Funktionalität eines Moduls wird durch seine Schnittstelle spezifiziert (bei C: *.h-Datei)
  - Programme, welche das Modul nutzen können die in der Schnittstelle definierten Funktionen und Datentypen verwenden
  - Wie jedoch eine Funktionalität im Detail implementiert ist geht aus der Schnittstelle nicht hervor und bleibt verborgen

Inhaltsverzeichnis

Pfeiloperator

- Verfügt man über einen Zeiger, der auf eine Struct-Variable zeigt, so kann man mit dem Pfeiloperator $->$, direkt von dem Zeiger aus, auf die Unterelemente des Structs zugreifen
- Der Ausdruck p$->$whatever ist dabei äquivalent zu $(*$p$)$.whatever

Pfeiloperator - Beispiel

Listing 3: src/arrow.c

```c
struct type {
  int alpha;
  double beta;
};

int main()
{
  struct type y;
  struct type *p;

  p = &y;
  y.alpha = 33;
  p->beta = 2.25; // das gleiche wie y.beta = 2.25

  return 0;
}
```