Übersicht

Lehrinhalt: Programmierung in C

- Überblick über Programmiersprachen, Allgemeines
- C-Basisdatentypen, Zahlendarstellung, Variablen, Konstanten
- Operatoren und Ausdrücke
- Anweisungen
- Kontrollstrukturen
- Funktionen
- Zeiger und Felder
- Zeichenketten (Strings)
- Benutzerdefinierte Datentypen
- **Dynamischer Speicher**
- Dateiarbeit
- Funktionspointer, Rekursion
- Preprozessor

Dynamischer Speicher

In diesem Abschnitt werden

(1) Felder fester Länge,

(2) Felder mit variabler Länge, die sich erst zu Laufzeit des Programms ergibt, dann aber unverändert bleibt

und

(**3) Dynamische Felder**, deren Länge zur Laufzeit wachsen und schrumpfen kann

gegenübergestellt

Für (2) und (3) benötigt man Funktionen zum Reservierung (Allokation), zur Änderung (Reallokation) und zur Freigabe dynamischer Speicher-bereiche. Datenstrukturen, die über Felder hinausgehen, werden in einem gesonderten Abschnitt zu Datenstrukturen, Listen und Bäumen behandelt.

Felder fester Länge (1)

Felder:
Aneinanderreihung von mehreren Variablen gleichen Typs

Ursprünglich als 1-dimensionale Organisationsform gedacht, aber auch zwei- und höherdimensional, beispielsweise Matrizen

Benutzung Felder:

- für Vektoren und Matrizen (im mathematischen Sinne)

- zur Speicherung einer Menge gleichartiger Daten,
  zum Beispiel
  - mehrere aufgenommene Messwerte
  - mehrere Buchungen, Einkäufe, Rechnungen, usw.

Felder fester Länge (2)

Vorgehensweise:

- Deklaration einer oder mehrerer Feldvariablen mit einer festgelegten Elementanzahl

- Elementanzahl muss konstant sein und die maximale Anzahl benutzter Feldelemente abdecken

- Die tatsächlich benutzte Anzahl von Elementen muss in einer weiteren int-Variable (auch unsigned int) gespeichert werden, oder aus den Elementen ablesbar sein.

Beispiel:

```
#define MAX_ANZ_BUCHST 26
char Buchstaben[MAX_ANZ_BUCHST];
int anz_buchstaben = 0;
```

Felder fester Länge (3)

Das Feld wird mit aufsteigendem Index mit Elementen gefüllt.
Die Elemente sollen hier nicht sortiert werden.

Operationen mit Feldern:

- Einfügen eines Elements
- Finden eines Elements und ggf. Zugriff auf weitere Informationen
- Löschen eines Elements
- Rückgabe der Anzahl aller Elemente

Felder fester Länge (4)

Einfügen eines Elements:

- die aktuelle Elementanzahl kann als Index auf das erste freie Element benutz werden
- wenn dieser Index noch innerhalb des deklarierten Feldes liegt, dann kann noch weiter eingefügt werden, sonst nicht.
- erstes freie Element mit einzufügendem Inhalt beschreiben (Zuweisung oder Kopieren)
- Anzahl tatsächlich benutzter Elemente um Eins erhöhen

Felder fester Länge (5)

Einfügen eines Elements:

```
// Buchstabe b soll eingefügt werden

if ( anz_buchstaben < MAX_ANZ_BUCHST)
{ buchstaben[anz_buchstaben]=b;
  anz_buchstaben =  anz_buchstaben + 1;
}
else
{
  // nicht einfügen
}
```

Bislang können gleiche Buchstaben auch mehrfach eingefügt werden. Falls jeder Buchstabe nur einmal eingefügt werden soll, so ist das Feld vorab zu durchsuchen.

Felder fester Länge (6)

Finden eines Elements im Feld:

```
// Buchstabe b soll gefunden werden
   char gefunden = 0;
   int i;
   int position = -1; // -1 als Wert für
                      // eine uninitialisierte Position
   …
   for (i=0; i< anz_buchstaben && !gefunden; i=i+1)
          if (buchstaben[i]==b)
        {  gefunden = 1;
           position = i;
        }
// weitere Anweisungen in Abhängigkeit, ob b gefunden wurde
```

Wenn die Elemente Strukturen sind, dann können hier weitere Daten bereitgestellt werden.

Felder fester Länge (8)

Löschen eines Elements im Feld:

```
// Buchstabe b soll gelöscht werden,  zuerst Finden von b
char gefunden = 0;
int i, position = -1;
…
for (i=0; i< anz_buchstaben && !gefunden; i=i+1)
   if (buchstaben[i]==b)
   {  gefunden = 1;
      position = i;
   }
if (gefunden) // Feldelemente um eine Pos. nach links setzen
{  for (i=position+1; i< anz_buchstaben; i=i+1)
         buchstaben[i-1] = buchstaben[i];
   anz_buchstaben = anz_buchstaben -1;
}
```

Felder fester Länge (9)

Vor dem Programmieren der Feld-Operationen sollte immer entschieden werden:

- **Sollen Elemente nur einmal im Feld vorhanden sein oder auch mehrfach?**
  Auswirkung auf Operationen Einfügen, Finden und Löschen

- **Sollen Elemente in der Reihenfolge des Einfügens gespeichert werden, oder soll eine Sortierung stattfinden?**
  Bei sortierter Speicherung kann sich die Zeit für das Finden und das Löschen von Elementen verkürzen, was bei großen Datenmengen nützlich ist.

Neben (linearen) Feldern (sortiert oder unsortiert) gibt es noch eine Reihe weiterer und besserer Datenstrukturen.
Näheres dazu in einem noch folgenden Abschnitt.

Felder fester Länge (10)

Nachteile:

Durch Deklaration mit einer maximal möglichen Elementanzahl wird möglicherweise Speicherplatz vergeudet.

Wenn die durch den Programmierer festgelegte Elementanzahl erreicht wurde, kann das Programm keine weiteren Daten mehr aufnehmen und verarbeiten.

Dynamische Speichertechnik (1)

Unter ***dynamischer Speichertechnik*** versteht man die Bildung neuer Speicherplätze auf der Basis von Standarddatentypen und deklarierten Strukturen zur Laufzeit. Es wird Speicherplatz zur Laufzeit bereitgestellt und darauf ein oder mehrere neue Elemente einer Datenstruktur platziert.

Ein neues Element wird mit bereits vorhandenen Elementen in der Datenstruktur in Beziehung gesetzt. Ebenso können auch zur Laufzeit Elemente aus der Datenstruktur herausgelöst werden und deren Speicherplatz wieder freigegeben werden.

Im Gegensatz werden die bisher benutzen Variablen und Felder als ***statische Datenstrukturen*** angesehen.

Dynamische Speichertechnik (2)

Zur Realisierung solcher dynamischer Strukturen werden zwei Techniken benötigt:

1. Die dynamische Speicherallokation und –freigabe

2. Die Verwendung der Zeigertechnik zum dynamischen Herstellen und Lösen von Verbindungen von Elementen innerhalb der Datenstruktur

Die vom Programmierer geschaffene Organisation der Daten (also wo welches Element im Speicher steht und wie es mit den anderen Elementen in Beziehung gesetzt wird) wird durch ***Datenstrukturen*** beschrieben.

Dynamische Speichertechnik (3)

Die dynamische Speicherallokation und –freigabe wird in C mit den Funktionen

```
void *malloc(size_t size); // gibt die Anfangsadresse
                          // des allokierten Bereichs zurück

void free(void *addresse);
```

durchgeführt, die in `<stdlib.h>` zu finden sind.
Es werden *size* Bytes allokiert und die Adresse dieses Speicherbereiches zurückgegeben. Ist der Rückkehrwert NULL, ist nicht genügend Speicher vorhanden. Ist die Größe in Bytes bekannt, kann diese direkt als Argument übergeben werden. Bei Struktur-Typen empfiehlt sich die Verwendung von sizeof(), welche die notwendige Größe in Bytes für ein Element ermittelt.

Bei der Speicherfreigabe mit `free` ist nur die Adresse des Speicherbereichs anzugeben.

Lineare Datenstrukturen (1)

Feld fester Länge:

- Längenfestlegung durch Programmierer (vor Laufzeit)
- Elemente werden durch Zahlenindex angesprochen

Feld variabler Länge:

- Wie Feld fester Länge aber mit Längenfestlegung zur Laufzeit
- Elemente werden wie gewohnt durch Zahlenindex angesprochen

Dynamisches Feld:

- Wie Feld variabler Länge
- Feld kann wachsen und schrumpfen – durch dynamische Speichertechnik
- Elemente werden wie gewohnt durch Zahlenindex angesprochen

Lineare Datenstrukturen (2)

Lineare Liste:

- Einzelne Elemente durch Zeigerverweise gekoppelt
- Liste kann wachsen und schrumpfen
- Effizientes Einfügen neuer Elemente und Löschen
- (ohne weiteres) kein Zahlenindex zum Zugriff mehr möglich

Weitere lineare Datenstrukturen: Warteschlange und Stapel

Feld variabler Länge

Bei manchen Anwendungen ist die Anzahl der zu verarbeitenden Elemente nicht bekannt und kann je nach Einsatzzweck des Programms stark variieren.

Felder mit einer maximalen Anzahl von Elementen sind dann ungünstig.

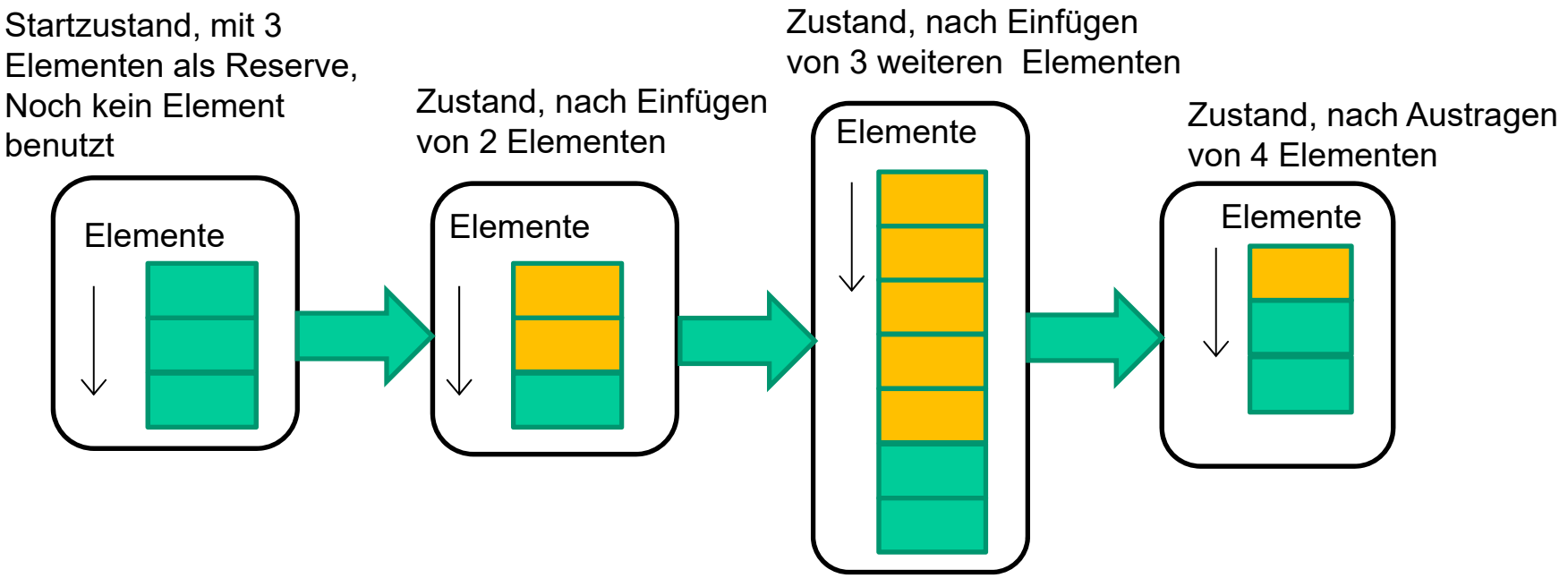
Lösung:
Ein Feld, das eine zur Laufzeit ermittelte Anzahl an Elementen besitzt

```
int n_max;

/* n_max: Wert wird durch das Programm berechnet,
   oder ergibt sich aus den Eingabedaten */

elem_typ* feld = (elem_typ*) malloc(sizeof(elem_typ) * n_max);

/* feld[0] bis feld[n_max-1] benutzbar … */


free(feld);
```

Dynamisches Feld (1)

Bei manchen Anwendungen variiert die Anzahl genutzter Elemente stark über die Laufzeit und ist auch zum Programmstart noch nicht bekannt.

Lösung:
Ein dynamisches Feld, das sich mit Mitteln der dynamischen Speicherverwaltung (malloc, realloc, free) der benötigten Anzahl von Elementen anpasst.



Dynamisches Feld (2)

Konstanten und Variablen zur Verwaltung:

```
#define N_START 3
#define N_DELTA 5
unsigned int n_allocated=0, n_elements=0;
elem_typ* feld;
```

Anfang:

- Allokieren von N_START Elementen
- Setzen von n_allocated = N_START

```
feld = (elem_typ*) malloc(sizeof(elem_typ)* N_START);
if (feld==NULL) { /* Fehlerbehandlung */}
n_allocated = N_START;
```

Dynamisches Feld (3)

Einfügen eines neuen Elements am Ende:

- Prüfen, ob ausreichend Speicher allokiert ist
- ggf. dynamisches Vergrößern des Feldes und Anpassen der Variable `n_allocated`
- Einfügen des Elements in `feld[n_elements]`
- Erhöhen von `n_elements` um 1

```
if (n_elements+1 > n_allocated)
{
  feld = realloc(feld, sizeof(elem_typ) * (n_allocated + N_DELTA));
  if (feld==NULL) { /* Fehlerbehandlung */ }
  n_allocated = n_allocated + N_DELTA;
}
feld[n_elements] = neues_element;
n_elements = n_elements+1;
```

Dynamisches Feld (4)

Ausgliedern eines Elements an Position *p* *(0≤p<n_elements)*:

- Verschieben der Feldelemente ab Position p+1 um eine Stelle nach links (kleinerer Index)
- Verringern von `n_elements` um 1
- Prüfen, ob `n_elements < n_allocated - N_DELTA`, wenn ja, dann Verkleinern des Feldes mittels `realloc()`

```
for (i=p+1;i<n_elements; i++)
  feld[i-1] = feld[i];
n_elements = n_elements - 1;

if (n_elements < n_allocated - N_DELTA)
{
  n_allocated = n_allocated - N_DELTA;
  feld = realloc(feld, sizeof(elem_typ) * n_allocated);
}
```

Dynamisches Feld (5)

Ein vollständiges Programmbeispiel wird als *dynamisches_array.c* bereitgestellt.