

Kurze Einführung in Pointer

Sonderübung

April 12, 2022

1. Motivation
2. Erstes Pointer-Programm
3. Übungsaufgabe

Motivation

Bisher: Arbeit mit Feldern

Nachteil: Wir können nachträglich keine Feldelemente hinzufügen. Zum Zeitpunkt des Allokierens muss die Anzahl von Feldeinträgen bekannt sein.

Ausweg: Pointer (Zeiger)

Typische Pointerstrukturen:

- einfach oder doppelt verkettete Listen
- zyklische Listen
- (Binär-) Bäume

An eine Liste können beliebig viele Elemente angehängt werden.

Anwendungsbeispiel: Einkaufsliste (Wenn wir eine Einkaufsliste schreiben, wissen wir in der Regel zunächst nicht, wie lang die Liste wird.)

Erstes Handwerkszeug

Deklaration:

... das **POINTER-Attribut**:

```
INTEGER, POINTER :: p, q
```

... das **TARGET-Attribut**:

```
INTEGER, TARGET :: a, b
```

```
a = 3
```

```
p => a
```

Pointerzuweisung mit =>

Pointer können auf folgende Objekte zeigen:

- andere Pointer
- allokierte Objekte
- Variablen mit Target-Attribut

NULLIFY und ASSOCIATED

NULLIFY und NULL:

```
INTEGER, POINTER :: p => NULL()
```

```
INTEGER, POINTER :: p  
NULLIFY(p)
```

ASSOCIATED:

```
IF (ASSOCIATED(p)) THEN  
do something ...
```

Seien p1,p2 Pointer und t eine Variable mit Targetattribut.

- ASSOCIATED(p1) ist .TRUE., wenn p auf ein Objekt zeigt, es ist .FALSE., falls das Pointerargument NULL ist
- ASSOCIATED(p1,p2) ist .TRUE., wenn beide Pointer auf dasselbe Objekt zeigen, .FALSE. sonst
- ASSOCIATED(p1,t) ist .TRUE., wenn p1 auf t zeigt, .FALSE. sonst

! Sind p1 oder p2 in einem nicht definierten Zustand, ist der Rückgabewert von ASSOCIATED zufällig.

Unser erstes Pointer-Programm

Einkaufsliste

Definiere einen abstrakten Datentypen *Produkt* mit einer CHARACTER-Komponente für den Namen des Produkts (maximale Länge 15) und einer Komponente vom Typ INTEGER für die Menge. Außerdem soll der Datentyp eine Pointer-Komponente vom Typ *Produkt* besitzen, die auf das nachfolgende Produkt in der Liste zeigt. (Am besten setzt du diese gleich bei der Deklaration auf **NULL()**).

Damit wir anschließend mit der Liste arbeiten können, sollte ein sogenannter *HEAD*-Pointer den Kopf der Liste festhalten. Definiere daher einen abstrakten Datentypen *Liste* mit einer INTEGER-Komponente für die Länge der Liste und einer Pointer-Komponente vom Typ *Produkt*.

Programmablauf: Lies zunächst die Länge n der Liste ein. Allokieren anschließend Speicherplatz für das erste Listenelement. Lies den Namen des ersten einzukaufenden Produkts und die benötigte Menge von der Tastatur ein, z. B. 'Banane', 5. In einer Schleife (noch $n-1$ Iterationen) sollen nacheinander weitere Listenelemente eingelesen werden.

Hinweis: Damit die Liste am Ende erweitert werden kann, sollte ein Hilfspointer auf das Ende der Liste zeigen. Achte darauf, diesen weiterzusetzen, wenn ein neues Produkt zur Liste hinzugefügt wird.

Nachdem die gesamte Liste eingelesen wurde, sollen alle Lebensmittel, die auf der Liste stehen, von vorn beginnend ausgegeben werden.

Einkaufsliste

ZUSATZ 1: (Ausgabe der Liste)

Wir befinden uns nun im Supermarkt und arbeiten die Liste der Reihe nach ab. Schreibe dafür eine Subroutine: In einer Schleife wird in jeder Iteration ein weiteres Produkt in den Einkaufswagen gelegt. Der Name und die Menge sind auszugeben.

ZUSATZ 2: Nun soll die Länge der Liste beliebig sein. Beim Einlesen soll zu Beginn jedes Schleifendurchlaufs abgefragt werden, ob die Liste der Einkäufe noch weitergeht. Falls nicht, ist die Schleife zu verlassen. Auch das Abarbeiten der Liste im Supermarkt (Z1) muss nun angepasst werden. In jedem Schleifendurchlauf muss überprüft werden, ob wir das Ende der Liste erreicht haben, d. h. ob der Nachfolger-Pointer nicht assoziiert ist.

ZUSATZ 3: (Löschen der Liste)

Zu Hause wird ausgepackt! Schreibe eine Subroutine, in welcher iterativ (von vorn beginnend) ein Produkt aus dem Beutel genommen und im Schrank verstaut wird. Ein entnommenes Produkt ist aus der Liste zu löschen.