Programming Pearls

Data structures programs

(Daten strukturieren Programme)

1.0 Umfragen auswerten...

Beispiel

Für die Auswertung einer Umfrage sollen die Ergebnisse verarbeitet werden...
Es gibt folgende Felder:

-Universität (Werte: 0,1,2,3)

-Geschlecht (Werte: 0,1)

-Fakultät (Werte: 0,1,2,3,...19)

-...

1.1 Umfragen auswerten..

Eine Möglichkeit wäre es, sehr viele einzelne Variablen zu haben, in denen man entsprechend die Anzahl der Antworten zählt und dazu jeweils noch eine Reihe von Variablen für Enthaltungen.

Nachteil: sehr langes Programm, kompliziert, schwer zu warten

Eine erste Vereinfachung besteht hier natürlich daraus Arrays zu benutzen, allerdings sähe das dann erst mal so aus:

eine mehrdimensionale Tabelle mit dem Aufbau:

Count[Uni][Frage][Antwort]=Anzahl der Antworten zu der Frage

Nachteil: unnötig hoher Speicherplatzverbrauch, da die letzte Dimension von der Frage mit der höchsten Antwortanzahl abhängt!

Declined[Uni][Frage]=Anzahl der Bögen mit der Frage ohne Antwort

1.2 Umfragen auswerten...

Das ist schon besser, aber noch nicht optimal, weil der erforderliche Code noch recht umfangreich wäre! Eine elegante Möglichkeit besteht darin, eine Tabelle für die Antworten zu benutzen, eine für den Offset und eine für die ausgelassenen Fragen.

1.3 Umfragen auswerten...

Aufbau Zähltabelle: Count[Uni][Antworten]

...	Fr1A1	Fr1A2	Fr2A1	Fr2A2	Fr2A3	Fr2A4	...

Aufbau Offsettabelle: Offset[Index]

0	0	2	22	Sa+#nF	...		

1.4 Umfragen auswerten...

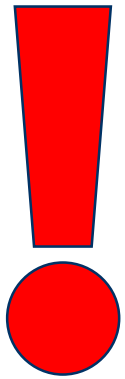
```
For i=1 to Endidx do
        If Entry[i]=refused
                declined[Entry[0]][i]++
        Else count[Entry[0]][offset[i]+Entry[i]]++
```

2.1 Datenstrukturen, dein Freund und Helfer

Der naive Ansatz

```
If (k==1) c01++;
If (k==2) c02++;
...
If (k==42) c42++;
...
```

Nicht sehr elegant und äußerst platz- und zeitintensiv



2.2 Datenstrukturen, dein Freund und Helfer

Besser:

```
c[k]++;
```

Vorteile:

- Kurz (daher auch besser zu warten)
- Flexibel (simples Erweitern durch Vergrößern des Arrays)

Nachteile:

- Man ist auf einen Datentyp festgelegt
- Ohne Überprüfung der Array Grenzen können Fehler auftreten

3.1 Serienbriefe, nicht einfach! Oder?

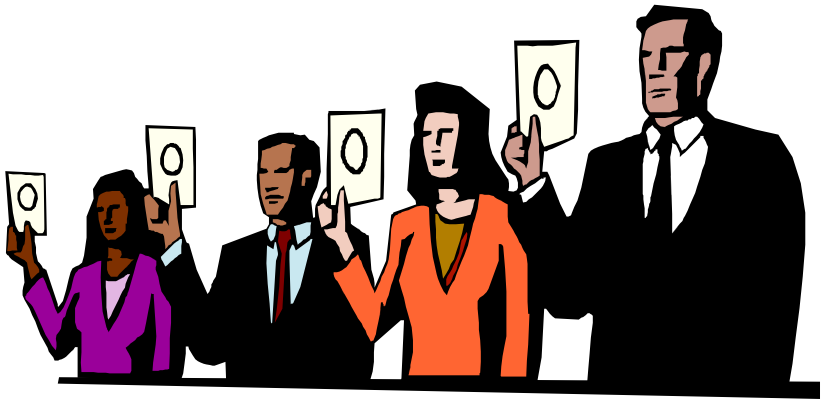
Hallo **vorname nachname**,
Du hast gewonnen! Wir senden dir deinen Gewinn in die **strasse** in **PLZ stadt**, wenn du jetzt bestellst...

Read nachname, vorname, strasse, stadt, PLZ
print „Hallo „ + vorname + “ „ + nachname
print „Du hast gewonnen! Wir senden dir deinen Gewinn in die“
print strasse+“ in „+ PLZ + „ „ + stadt + „, wenn du jetzt bestellst...“

3.2 Serienbriefe, nicht einfach! Oder?

Anforderungen

- Parametrisierbar oder Datenbankanbindung
- Brieftext soll editierbar bleiben



3.3 Serienbriefe, nicht einfach! Oder?

```
Read fields from database
Loop from start to end of schema
        c = next char in schema
        if c != ‚$‘ printchar c
        else
                c = next char in scheme
                case c of
                        ‚$‘: printchar c
                        ‚0‘..‘9‘:printstring field[c]
                        default: error(‚Fehler in Vorlage‘)
```

4.1.1 Arrays sinnvoll nutzen

Beispiel:

```
Sub menuitem0_click()
        Menuitem0.checked = 1
        Menuitem1.checked = 0
        Menuitem2.checked = 0
        Menuitem3.checked = 0
        Menuitem4.checked = 0
        Menuitem5.checked = 0
        Menuitem6.checked = 0
```

4.1.2 Arrays sinnvoll nutzen

Beispiel:

```
Sub menuitem1_click()
        for i = 0 to numchoices
                Menuitem[i].checked = 0
        Menuitem1.checked = 1
```

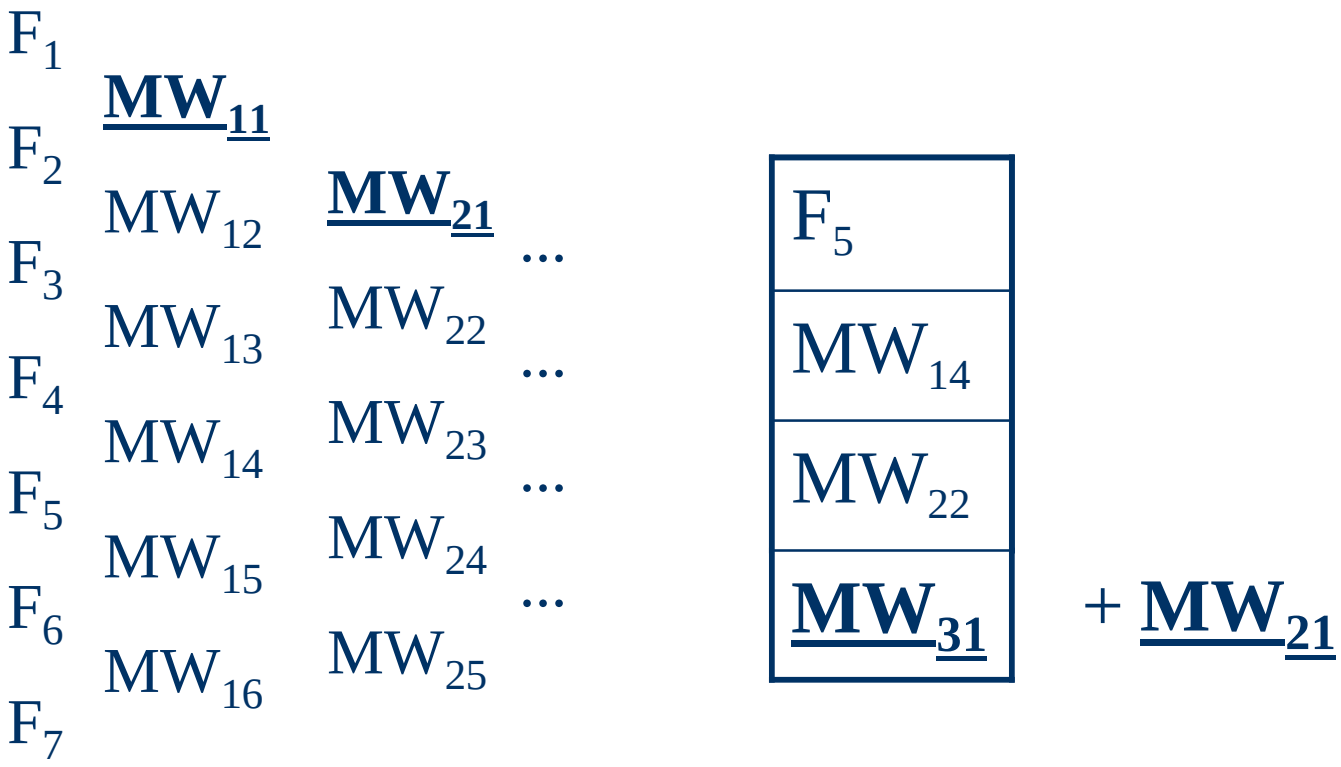
4.2.1 Arrays sinnvoll nutzen

Aufgabe:

Per sukzessiver Mittelwertbildung sollen Funktionswerte approximiert werden.

Meistgewählte Lösung:
ein mehr als ausreichend großes Array deklarieren und entsprechend mit Werten füllen, wenn Abbruchbedingung erfüllt, letzten Wert ausgeben

4.2.2 Arrays sinnvoll nutzen



5.1 OOP

- Objektorientierte Programmierung ist eine „Programmier-Philosophie“
- Sie basiert darauf, Daten und den dazugehörigen Code zu vereinen
- Es werden i.d.R. Prozesse oder Objekte der Umwelt abgebildet
- Objekte können von anderen Objekten abstammen, sie „können“ dann alles, was ihre Vorfahren können und man hat die Möglichkeit sie entsprechend zu erweitern

5.2 OOP

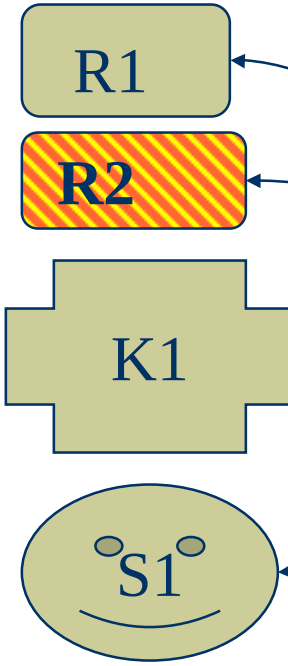
- Diese Vererbung bietet viele Möglichkeiten, elegant auf viele ähnliche Objekte zuzugreifen (dazu in 5.3.0 mehr)
- Java ist z.B. eine konsequent objektorientierte Sprache
- KDE, ein Fenstermanager unter Linux, basiert auf QT, einer ebenfalls objektorientierten Sammlung von Fensterelementen (Knöpfe, Listboxen, ...)

5.3.0 OOP am Beispiel

Geom. Objekt
-Position
-Farbe
-Breite,Höhe
-...
-setFarbe
-getFarbe
-...

Dieses Objekt hat Zustände/Eigenschaften, wie Position, Farbe,... Diese sollten nicht direkt verändert oder ausgelesen werden, sondern über vom Objekt zur Verfügung gestellte Funktionen und Methoden. Damit wird sichergestellt, dass bei abgeleiteten Objekten Funktionalität hinzugefügt, bzw. die vorhandene verändert werden kann.

5.3.1 OOP am Beispiel



Geom. Objekt
-Position
-Farbe
-Breite,Höhe
-...

Bei Interesse kann ich gleich noch auf OOP eingehen...

Diese neuen Objekte sind geometrische Objekte, aber sie haben weitere Fähigkeiten/Eigenschaften. Der Vorteil ist, dass man sie in einer Liste vom Typ Geom. Obj. abspeichern kann und dort alle gemeinsamen Funktionalitäten nutzen kann.

6. Tools benutzen ...

Es gibt schon vieles, was Programmierern viel Arbeit abnimmt:

- IDE‘s (Delphi, CBuilder, JBuilder, VisualC, Forte, ...) benutzen
- Spezialisierte Tools nutzen
  - HTML für Dokumentationen oder Menüstrukturen
  - Tabellenkalkulationen nutzen, sofern sinnvoll
  - Datenbanken
  - ...

7. Worum es eigentlich geht...

- Programme (bzw. Funktionen) klein und übersichtlich halten
- Manche Probleme lassen sich besser erst allgemein lösen und dann auf den Spezialfall anwenden
- Kapseln von komplexen Strukturen (OOP)
- Durch geänderte Randbedingungen (oder Hauptaufgaben) ist ein ReDesign eigentlich immer angebracht
- Das Programm nach den zu verarbeitenden Daten planen und nicht umgekehrt

8. Weiterführende Literatur

- **Code Complete**
  Steve McConnell
  Microsoft Press 1993
- **Rapid Development**
  S. McConnell
  Microsoft Press 1996
- **Software Projekt Survival Guide**
  S. McConnell
  Microsoft Press 1998

Persönliche Empfehlungen

- **Writing Solid Code**
  Steve Maguire
  Microsoft Press 1993