Seminar zum Buch

Design – Menschenwerk

Sichten auf ein vielseitiges Phänomen

Prof. Dr. Wolf Knüpffer

mailto: wolf.knuepffer@hs-ansbach.de

HOCHSCHULE
ANSBACH

Der rote Faden

1. Vom Algorithmus zum Programm
2. Vom Programm zum Bild
3. Von der computergenerierten Grafik zur computergestützten Bildbearbeitung
4. Von der strukturierten zur objektorientierten Programmierung (OOP)
5. Von der OOP zum objektorientierten (Software-) Design (OOD)

Was ist ein Algorithmus?

Ein **Algorithmus** ist eine eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen. Algorithmen bestehen aus endlich vielen, wohldefinierten Einzelschritten. Somit können sie zur Ausführung in einem Computerprogramm implementiert, aber auch in menschlicher Sprache formuliert werden. Bei der Problemlösung wird eine bestimmte Eingabe in eine bestimmte Ausgabe überführt.

Beispiele:

- Page-Algorithmus von Google
- Verschlüsselungsalgorithmen (z. B. RSA-Verfahren)
- Such- und Sortieralgorithmen (z. B. Bubble Sort, Quick Sort, Binäre Suche)
- Algorithmen zur Bildbearbeitung (z. B. Filter, Transformationen)

Was ist eine Programmiersprache?

- Formale Sprache zur „computerverträglichen“ Formulierung von Algorithmen.

- Enthält **Kontrollstrukturen** zur Steuerung des Ablaufs.

Beispiele:

COBOL, FORTRAN

Strukturierte Programmierung: Pascal

Objektorientierte Programmierung: Smalltalk, C#, C++, Java, Objective C, Swift

Kontrollstrukturen

Grundstruktur ist die Befehlssequenz

```
betrag = 7
index = 0
read (gesuchterBetrag)
```

Arten von Kontrollstrukturen

- <u>Bedingung</u>

```
If (betrag = gesuchterBetrag) Then
        Write ("Der Betrag " + betrag + "ist vorhanden.")
EndIf
```

- <u>Schleife</u>

```
While (index <= anzahlDatenfelder)
        read (Datenfeld [index])
        index = index +1
EndWhile
```

Übungsbeispiel

Aus einer Menge an Menschen ist der jüngste zu ermitteln.

- Beschreiben Sie mit eigenen Worten einen Algorithmus zur Lösung dieser Aufgabe.
- Formulieren Sie diesen Algorithmus anhand der Strukturen aus Folie 4 als Programm.

Lösung in Small Basic

```
anzahl = 3 'Anzahl der Menschen
index = 1 'index auf 1

'Einlesen der Daten
While index <= anzahl
  TextWindow.Write(index + ". Name:")
  name[index] = TextWindow.Read()
  TextWindow.Write(index +". Alter:")
  alter[index] = TextWindow.Read()
  index = index + 1
EndWhile

indexJuengster = 1

index = 1
While index <= anzahl
  If (alter[index] < alter[indexJuengster]) Then
    indexJuengster = index
  EndIf
  index = index + 1
EndWhile
'Ausgabe
TextWindow.WriteLine("Der jüngste ist " + name[indexJuengster])
```

Vom Programm zum Bild

Beispiel: Das Verhulst-Fraktal (S. 113)

```
Y = 0.63761
x = 3.5
'So lange x < 3.8, tue:
While x < 3.8
    'Initialisierung des Index der jeweils zu berechnenden
    Zahlenfolge.
    i = 0
    'So lange Index < 160, tue:
    While i < 160
      'Berechnung der nächsten Ordinate
      '(zentrale Verhulst-Formel)
      Y = x*y-x*y*y
      y ansformierung von Abszisse und Ordinate in
      Bildschirmkoordinaten
      xx = 20.0+20.0*(x*100.0-350.0)
      yy = -50+Y*500.0
      'Aufruf des Unterprogramms zur Pixeldarstellung
      SetPixel()
      'Erhöhung des Index der zu berechnenden Zahlenfolge
      i = i+1
      'Ende der inneren Programmschleife
    EndWhile
    'Berechnung der nächsten Abszisse
  x = x + 0.0003
    'Ende der äußeren Programmschleife
    EndWhile
```

Variiere!

```
'Unterprogramm zur Pixeldarstellung
Sub SetPixel
  GraphicsWindow.SetPixel(xx,yy,"black")
EndSub
```

Fragen:

- Wie viele Punkte zeichnet das Programm?
- Sind alle diese Punkte sichtbar?

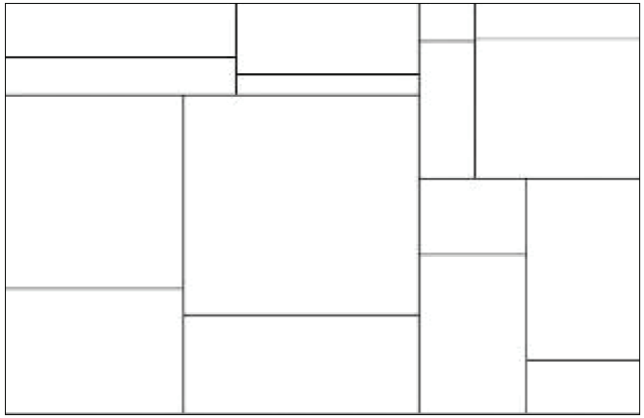
Betrachtung der Verhulst-Formel:

$y_{neu} = x*y_{alt} - x*y_{alt}^2$

- $x*y$ wächst linear.
- $x*y^2$ wächst exponentiell.
- y geht in beide Ausdrücke ein.
  -> y_{neu} „springt“.
  -> beide Verläufe sind parabelförmig.
- Die Differenz der Steigungen ist ein Maß für die Konvergenz von y_{neu}.

Rekursion: Ein Algorithmus benutzt sich selbst

```
//Aufruf der Methode picsub (der Schalter reguliert die Schneiderichtung).
    private void picsub(int u, int v, int p, int q, int dir, int Depth)
    {
      //wenn rekursive Tiefe nicht erreicht
      if(Depth<15)                                                    (5)
      {
        Depth=Depth+1;
        //Wenn Schneiderichtung vertikal (weil dir gleich 0)
        if(dir==0)
        {
          //Aufbewahrung der Abszisse u.
          int uu;
          uu=u;
          //Ermittlung einer Abszisse für den senkrechten Schnitt.
          //Die Zufallszahl r.Next(p-u) liegt zwischen 0 und (p-u).
          //Die Schnittabszisse liegt nicht zu weit links und nicht
          //zu weit rechts (zur Vermeidung von
          //Extremlösungen, die unmöglich für die Schere sind).
          u = u +(int)(0.2*(p-u)+0.6*r.Next(p-u));

          //Zeichnen der vertikalen Schnittlinie.
          verlin(u,v,q-v);

          //Bearbeiten des linken Resultats aus dem vertikalen Schnitt
          //(die 1. Schnitthälfte fällt links von der Schere herunter)
          //Umschaltung zum horizontalen Schnitt durch 1-dir.
          //Zwischen den beiden Aufrufen von picsub
          //herrscht viel Auf und Ab.

          picsub(uu, v, u, q, 1-dir,Depth);
          //Bearbeitung des 2. Resultats aus dem vertikalen Schnitt.
          //(die 2. Schnitthälfte fällt rechts von der Schere herunter)
          picsub(u, v, p, q, 1-dir,Depth);
        }
        else
        //Der Kommentar zur waagerechten Zerschneidung ist analog zur
        senkrechten.
        {
          int vv; vv = v;

          v = v+(int)(0.2*(q-v)+0.6*r.Next(q-v));

          horlin(u, v, p-u);
          picsub(u, vv, p, v, 1-dir,Depth);
          picsub(u, v, p, q, 1-dir,Depth);
          }
        }
        //Ende von picsub.
      }
```



Gegeben sind: Ein Rechteck,
eine Variable Depth = 0.

Algorithmus:

Falls (Depth <15)

- Erhöhe Depth um 1.
- Zerschneide das Rechteck (zuerst vertikal).
- Führe den Algorithmus innerhalb der neuen Rechtecke mit anderer Schnittrichtung erneut aus.

Ende

Zwischenergebnis

- Ursprünglich bildete ein Programm je einen Algorithmus ab.
- Mit steigender Komplexität der Programme wurde es sinnvoll, die Programme in ein Hauptprogramm und Unterprogramme zu gliedern.

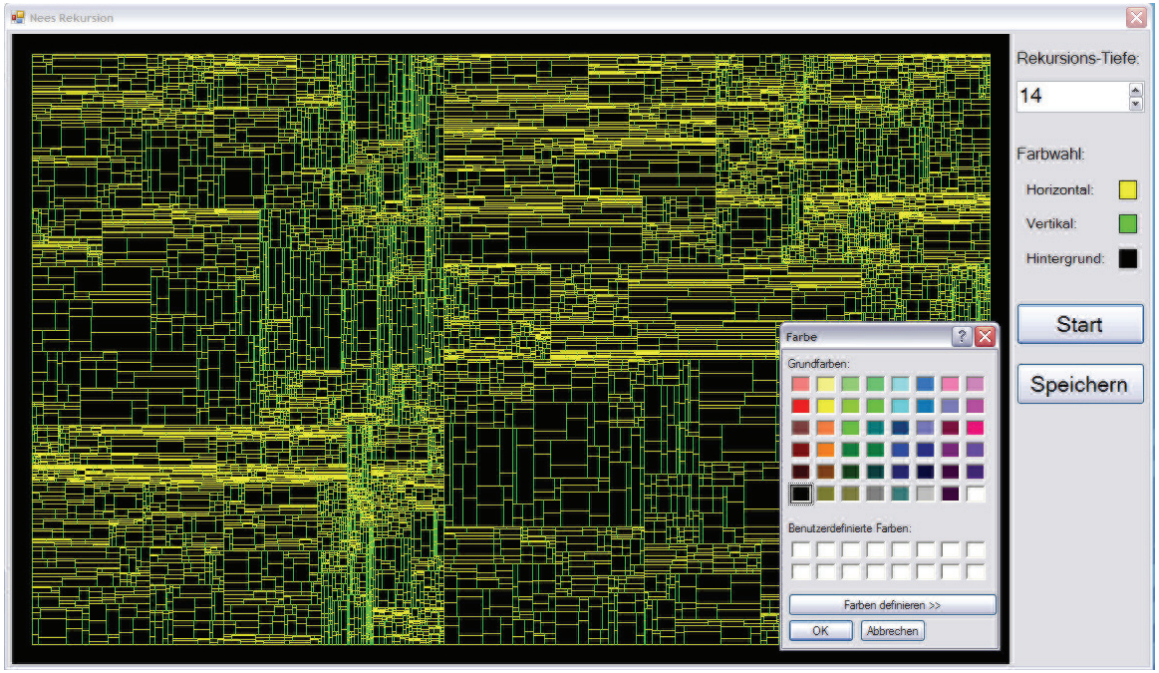
➢ Es entsteht die „Strukturierte Programmierung“ (auch „GOTO-freie“ Programmierung).

➢ Häufig benötigte Unterprogramme können in Programmbibliotheken bereitgestellt werden.

➢ Strukturierung gewinnt an Bedeutung.

Von der computergenerierten Grafik zur Bildbearbeitung

Schritt 1: Parametrisierung des Algorithmus



Ein Programm wird so allgemein wie möglich formuliert und Parameter definiert, die der Nutzer vorgeben kann.

- Es entsteht ein Anwender-Werkzeug.
- Durch Parametrisierung entsteht Standardsoftware (z. B. SAP/R3)

Frage: Wie viele Rechtecke erzeugt der Algorithmus im obigen Bild?

Von der computergenerierten Grafik zur Bildbearbeitung

Schritt 2: Interaktive Nutzung von Algorithmenbibliotheken

Moderne, interaktive Softwarelösungen ermöglichen die flexible Nutzung von Algorithmen-bibliotheken zur Bild-bearbeitung durch den Nutzer.

Interaktive, grafische Benutzeroberflächen begünstigen das Arbeiten mit „Objekten“.

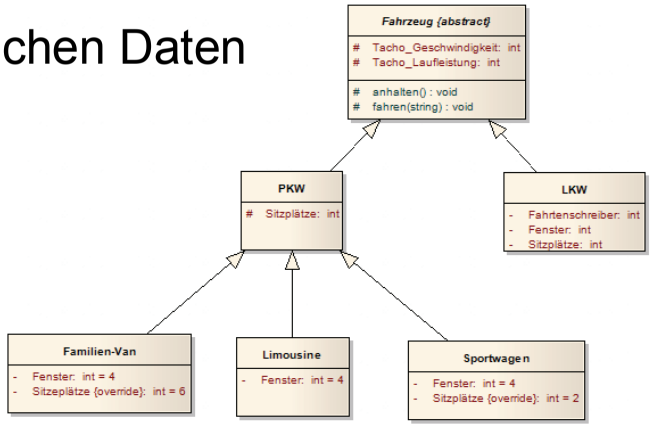
Von der strukturierten zur objektorientierten Programmierung

- **Erkenntnis:** Auch die strukturierte Programmierung ist für komplexe Softwarelösungen wenig geeignet, u. a. da alle Unterprogramme auf gemeinsamen Daten arbeiten.
- **Ziel:** Software sollte aus möglichst unabhängigen „Bauteilen“ (=Objekten) zusammen gesetzt werden können.

- **Wichtige Eigenschaften eines (IT-)Objektes**
  - Besitz eigener Daten (Zustände) und Methoden (Fähigkeiten/Algorithmen).
  - Kapselung aller Daten und Methoden. Nur das Objekt selbst kann diese ausführen. Objekte schicken sich gegenseitig Botschaften.
  - Vererbung: Das Objekt erbt alle seine Methoden und Datenstrukturen von der Objekt-Klasse, die es erzeugt.
  - Objekt-Klassen können Oberklassen erben.

Von der OOP zum objektorientierten (Software-) Design (OOD)

- Es ist festzulegen, welche Objekt-Klassen mit welchen Daten und Methoden es geben soll.
- ➢ Gute Klassendefinitionen schaffen Klarheit!



- Abstraktion und Vererbung sind sehr gut zur Beschreibung komplexer Systeme geeignet.
- ➢ Gute Definition der Klassenhierarchie ist wichtig!

- ➢ Das Design des Objektmodells (OOD) ist entscheidend! (60 -70 % des Aufwands). Darstellungsmittel im OOD sind Diagramme der Unified Modeling Language (UML).