

Algorithm Engineering

May 4, 2015

1 Datentypen

Getränkeautomat

Automat akzeptiert 1E, ein Getränk kostet 3E

Operatoren:

1. Init(Reset)
2. Akzeptiere1E

Init \rightarrow Zustand

Semantik: Automat geht in Zustand 0

Akzeptiere1E: ZustandX $\{0,1\} \rightarrow$ ZustandX $\{\text{tue nichts, gib Getränk}\}$

Semantik: Beschreibung durch einen endlichen Automaten.

Stadtplan

Übung 1

1.1 Bemerkungen

- Operatoren können partiell Definiert sein. Man gibt Definitionsbereich oft in einer Vorbedingung an.
- Operatoren, bei denen der Datentyp selbst auf der linken Seite nicht vorkommt, heißen Konstruktoren. Sie erzeugen ein neues Objekt (bzw. versetzen den Typ in einem bestimmten Zustand).
 - Create: $\rightarrow \text{stack}<T>$
 - Create: $\text{int} \rightarrow \text{vector}$ (Vektor bestimmter Dimension)
- Objekt- und Zustandssicht sind beide nützlich. Stack/Getränkeautomat haben internen Zustand, Operatoren können ihn verändern.
 - Integer: Objektsicht besser, Operatoren erzeugen neue Objekte, existierende werden nicht geändert.
- $\text{stack}<T>$ ist ein parametrisierbarer Datentyp: Stack mit Elementen vom Typ T. Hat eventuell besondere Anforderungen an Typ T, z.B. $x \leq y$ in Dictionaries.
- Man kann nun eigentlich schon programmieren, obwohl über die Interpretierung noch nichts bekannt ist.

Anwendung von $\text{stack}<T>$

Auswertung von Postfix-Ausdrücken

Vereinfachungen: alle Operatoren binär (+-*/), Eingabe nur Zahlen 0-9

Bsp.: $(7 - 5) * (3 + 1) \rightarrow 75 - 31 + *$

1.2 Definition eines Datentyps

(In einer Objekt-Orientierten Programmiersprache)

```
class Typname {  
    //Definition der Menge der Objekte bzw Zustaende  
    private: //Deklaration von Variablen zur Darstellung der Objekte/Zustaende  
    public: //Operatoren  
    //Kommentare z.B. ueber Effizienz  
};
```

Operatoren

Methoden/Memberfunktionen

Syntax: Ergebnistyp Name(Argumente...);

Spezielle Methoden:

- Kein Ergebnistyp: stack(); stack(size);
- Destruktor: ~ Typname();

1.3 Beispiel

int_stack \rightarrow stack< T >

```
class int_stack {  
    /* Eine Instanz vom Typ int_stack ist eine Folge von ganzen Zahlen (int). Eine Fol  
    private: //Implementierung  
    public: stack(int sz); //Konstruktor  
    //Erzeugt einen Stack mit maximaler Groesse sz  
    ~stack() //Destruktor  
    void push (int x);  
    //fuegt x als letztes Element (top) an die Folge an.  
    int top() const;  
    //liefert das letzte (top) Element  
    //Precondition: Stack nicht leer  
    int pop();  
    //entfernt letztes (top) Element der Folge und gibt es zurueck  
    //Precondition: Stack nicht leer  
    bool empty() const;  
    //true, wenn Stack leer, false sonst.
```

In c++ Spezielle Header Datei, die die Deklarationen ohne Rumpf enthält. Implementierung in .cpp

Implementierung der Klasse int_stack

Mehrere Möglichkeiten: Array, Liste Array Implementierung: *int_stack.h*

```
class int_stack {  
    private  
        int* A; //Feld  
        int sz; //Laenge von A  
        int t;  
};
```

int_stack.cpp

```

#include "int_stack.h"
int_stack::int_stack(int n) {
    sz=2;
    A = new int[sz];
    t = -1; //leer
}
int_stack::~~int_stack(){
    delete[] A;
}
void int_stack::push(int x){
    if (t == sz-1){
        //stack voll
        int* B=new int[2*sz];
        sz ← 2*sz;
        for(int i=0; i ≤ i++){
            B[i] ←
        }
        delete[] A;
    }
    A[++t] ← x; //eigentlich push
}
int int_stack::pop(){
    if (t==0){
        EXCEPTION(" Leerer Stack")
    }
    return A[t--];
}

```

Einschub: Variablen, Konstruktoren, Wertzuweisung

ablen Deklaration c++: Aufruf des Konstruktors generiert ein Objekt.

Java: Erst eine Referenz erstellen, dann ein Objekt generieren und auf dieses verweisen.

Wertzuweisung c++: $int_stack\ s1, s2; s1 = s2;$ Objekt wird kopiert, es gibt 2 Objekte.

Java: $int_stack\ s1, s1; s1 = s2;$ Referenzen zeigen auf ein einziges Objekt.

semantik in c++: Verwendet Pointer auf ein Objekt.

Test auf Gleichheit (==) Operator

Parameterübergabe sind Pointer

semantik in Java: Parameter by Value, gesamtes Objekt kopiert und dann übergeben.

Korrektheit einer Implementierung

(Hier der Array-Implementierung von int_stack)

Eigentlich 2 Datentypen:

1. der abstrakte Datentyp int_stack
2. der konkrete Datentyp Array

Abstrakter Zustand: Folge von int's

Konkreter Zustand: Werte der Variable A,t,sz

Wir garantieren (Invariante), dass nicht die Kombination von A,t,sz möglich sind, sondern nur gültige Zustände mit:

1. A ist ein Feld der Länge sz
2. $-1 \leq t \leq sz - 1$

Sei Z =Menge der konkreten Zustände und S = Menge der abstrakten Zustände

Um die Korrektheit zu zeigen, definieren wir eine Abbildung $F : Z \rightarrow S$

$(A, sz, t) \rightarrow \begin{cases} Folge\ A[0], \dots, A[t] \text{ falls } t \geq 0 \\ Leere\ Folge, t = -1 \end{cases}$ Und zeigen:

1. Konstruktoren erzeugen gültige konkrete Zustände
2. Für jede abstrakte Operation und die dazugehörige konkrete Operation f_{op} zeige $F(f_{op}(Z)) = op(F(Z))$
Bsp.: push: $S \times int \rightarrow S$
 $f_{push} : Z \times int \rightarrow Z$

Kommutatives Diagramm:

$$\begin{array}{ccc} z & \xrightarrow{F} & s \\ \downarrow f_{op} \uparrow op & & \\ z' & \xrightarrow{F} & s' \end{array}$$

Vererbung/Generische Datentypen

Templates/Wiederverwendung von Code

Situation

Man braucht einen Datentyp A, der sehr ähnlich zu einem bereits vorhandenen definierten Typ B ist.

A soll:

- einen Teil der Daten/Operationen verwenden
- andere Daten/Operationen anfügen
- einige verändern (auf andere Weise implementieren)

Beispiel

Es existiert die Klasse Polygon (B), implementiert werden soll eine Klasse Rechteck (A).

A ist eine Spezialisierung von B

Rechteck könnte alle Polygon-Operationen (draw(), translate etc)

Einige Operationen können effizienter implementiert werden (Flächeninhalt etc.)

Andere sind nur für Rechtecke definiert.

Jedes Rechteck ist ein spezielles Polygon.

Allgemein

Wir leiten die Klasse A von der Klasse B ab.

Syntax in c++: A: public B

```
class Rechteck: public Polygon{
    //Konstruktoren -> uebung :(
private:
    //Ueberschreiben
    double b,h;
    double area() {return b*h;}
    double umfang() {return 2*(b+h);}
    //Hinzufuegen
    double ratio() {return b/h;}
};
```

Ableitungen auch als Baum darstellbar (Shape mit Kindern Polygon/Ellipse/Punkt etc)

Typverträglichkeit

Einer Variable vom Typ B* oder B& (alle Variablen in JAVA vom Typ B) kann ein Objekt (Pointer/Referenz) vom Typ A zugewiesen werden.

Alle im Ableitungsbaum erreichbaren Typen können zugewiesen werden (Kinder).

Eine Variable hat 2 Typen: Einen Statischen Typ, der zur Compilezeit bekannt ist. Dynamischer Typ, der zur Laufzeit bekannt ist.

Polymorphe Datenstrukturen

Bsp.: Feld von Polygonen

c++: Polygon** (Ein Pointer auf ein Feld von Pointern) A = new Polygon*[100];

A[0] = new Polygon(...);

A[1] = new Rechteck(...);