Mathias Hörtnagl

TURING

Betreuer: Simon Legner, BSc

GESCHICHTE
IMPLEMENTIERUNGEN
ANWENDUNGSBEREICHE DAMALS

SYNTAX GRUNDLAGEN
SUBROUTINEN
OBJEKT-ORIENTIERUNG

ANWENDUNGSBEREICHE HEUTE

GESCHICHTE

Original Turing

- Entwickelt 1982 an der Universität Toronto
- Basiert auf Pascal und Euclid
- Ersetzte PL/1 in der Lehre an der Universität

Original Turing

- Sollte leicht zu erlernen sein
- „life cycle language“
- Vollständige und sinnvolle Fehlermeldungen
- Konzisere Syntax als Euclid oder Pascal
- Unterstützt dynamische Strings und Arrays

System Turing

- Erweiterung zur systemnahen Sprache
- Zeiger, Zeigerarithmetik
- Explizite Typkonvertierungen
- Prozesse, Monitore, Interruptbehandlung
- Binäre Datei I/O

Object-Oriented Turing

- Erweiterung um objekt-orientierte Aspekte

- Kein Garbage-Collector
- Keine Methodenüberladungen

IMPLEMENTIERUNGEN

Holtsoft Turing

- Proprietäre Implementierung
- Vertrieb des Compilers u.a. an die Bildungseinrichtungen Ontarios

- Weiterentwicklung und Unterstützung 2007 beendet

TPlus

- Open-Source Implementierung von System Turing
- Entwickelt an der Universität Toronto
- Verschwand mit Object-Oriented Turing

OpenT

- Erste Open-Source Implementierung von Object-Oriented Turing
- Letzte Änderung am 01. August 2009

Open Turing

- Weiterentwicklung des 2007 freigegeben Holtsoft Compilers
- Unterstützung von OpenGL 3D
- HashMap Modul verfügbar
- Bis zu 50% schneller

- Nur für Windows verfügbar

ANWENDUNGSBEREICHE DAMALS

Anwendungsbereiche Damals

- Unterrichtssprache in Einführungskursen an der
  - York University (Toronto)
  - University of the Pacific (Stockton, California)
  - Columbia University (New York)
  - Concordia University (Montreal)
  - Queen's University (Kingston, Ontario)

- Mit System Turing auch in Kursen zu den Themen
  - Systemkomponenten
  - Betriebssysteme
  - Übersetzerbau

Anwendungsbereiche Damals

- Unterrichtssprache an vielen high schools in Ontario, Canada

SYNTAX GRUNDLAGEN

99 Bottles

```
for decreasing bottles : 99 .. 0
  if bottles >= 2 then
    put bottles, " bottles of beer on the wall, ", bottles, " bottles of beer."
    if bottles not= 2 then
      put "Take one down and pass it around, ", (bottles - 1),
          " bottles of beer on the wall.\n"
    else
      put "Take one down and pass it around, 1 bottle of beer on the wall.\n"
    end if
  elsif bottles = 1 then
    put "1 bottle of beer on the wall, 1 bottle of beer."
    put "Take one down and pass it around, no more bottles of beer on the wall.\n"
  else
    put "No more bottles of beer on the wall, no more bottles of beer."
    put "Go to the store and buy some more, 99 bottles of beer on the wall."
  end if
end for
```

Strings

- Eigener primitiver Datentyp `string`
- Strings mit zur Laufzeit unbekannter Länge
- Konkatenation mit dem + Operator
- Syntaktische Struktur zur Substringextraktion

- Länge dynamischer Strings auf **255** begrenzt

Strings Beispiel

```
var strCon : string(3) := "con"
var str : string
str := strCon + "CAT"
put str
put str(1 .. 3)
put str(4 .. *)
put length(str)
```

Ausgabe

```
conCAT
con
CAT
6
```

Benutzerdefinierte Typen

- Enumerationen
- Teilbereiche aus Integern oder ASCII Zeichen
- Records
- Unions
- Sets / Mengen

Benutzerdefinierte Typen

```
type state : enum(IDLE, START, DATA, STOP, END)

type upperLetters : 'A' .. 'Z'      % ASCII values.

type numPad : 0 .. 9

type node : record
  val  : string
  next : nat
end record



var s : state := state.IDLE

var letter : upperLetters := 'X'

var startNode : node                % := init("Mauern", 31)
startNode.val := "Mauern"
startNode.next := 31
```

Arrays

```
var sums : array 1 .. 10 of int

var charCounts : array char of nat

var index : array 'a' .. 'z' of string

var table : array boolean of array boolean of boolean



charCount(‘{') := 2

table(false)(false) := false
```

Flexible Arrays

```
var sums : flexible array 1 .. 10 of int

sums(1) := 13
new sums, 20
sums(20) := 382
put sums(1)
put sums(20)
```

Ausgabe

```
13
382
```

SUBROUTINEN

Funktionen

```
function myFunction(a : boolean, b : boolean) : boolean
  result a => b   % Logical implication.
end myFunction

put myFunction(false, false)
put myFunction(false, true)
put myFunction(true, false)
put myFunction(true, true)
```

Ausgabe

```
true
true
false
true
```

Prozeduren

```
procedure myProcedure(x : char)
  if x = 'x' then
    return
  end if
  put x, " is not x!"
end myProcedure

myProcedure('x')
myProcedure('y')
```

Ausgabe

```
y is not x!
```

Aufruf vor Implementierung

```
forward procedure myProcedure(x : char)

myProcedure('x')
myProcedure('y')

body procedure myProcedure(x : char)
  if x = 'x' then
    return
  end if
  put x, " is not x!"
end myProcedure
```

Ausgabe

```
y is not x!
```

Parameterlose Routinen

```
function a : real
  result 0
end a

procedure b()
  put 'x'
end b

put a()
b
```

Ausgabe

```
0
x
```

OBJEKT-ORIENTIERUNG

Klassen

- Keine Konstruktoren
- Nur 2 Zugriffsmodifikatoren
- Keine Unterscheidung zwischen Interfaces, abstrakten Klassen und Klassen
- Nur einfache Vererbung

Klassen

```
class ClassA
  export procedureA, var varA

  var varA : nat4 := 666

  procedure procedureA()
    put "Called ClassA.procedureA()"
  end procedureA

end ClassA
```

Objekterzeugung

```
var a0 : ^ClassA    % var a0 : pointer of ClassA
new a0              % new ClassA, a0
a0->procedureA()    % ClassA(a0).procedureA()
put a0->varA
```

```
var a1 : ^anyclass  % Turing equivalent to Java's class Object.
new ClassA, a1
ClassA(a1).procedureA()
```

Ausgabe

```
Called ClassA.procedureA()
666
Called ClassA.procedureA()
```

Vererbung

```
class ClassB
  inherit ClassA
  export procedureB

  deferred procedure procedureB()

end ClassB
```

```
class ClassC
  inherit ClassB

  body procedure procedureA()
    ClassA.procedureA()                % Call superclass implementation.
    put "Called ClassC.procedureA()"
  end procedureA

  body procedure procedureB()
    put "Called ClassC.procedureB()"
  end procedureB

end ClassC
```

Objektzerstörung

- Turing besitzt **keinen** Garbage-Collector

```
free a0
```

ANWENDUNGSBEREICHE HEUTE ?

ENDE