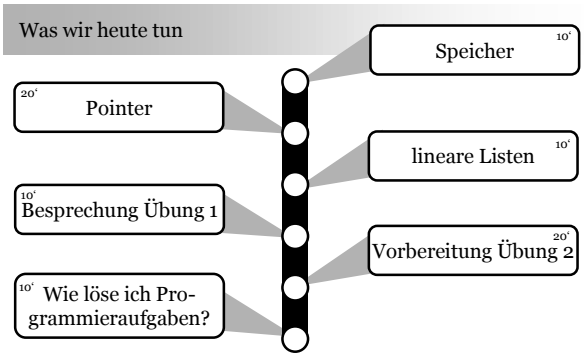
Informatik I WS 02/03

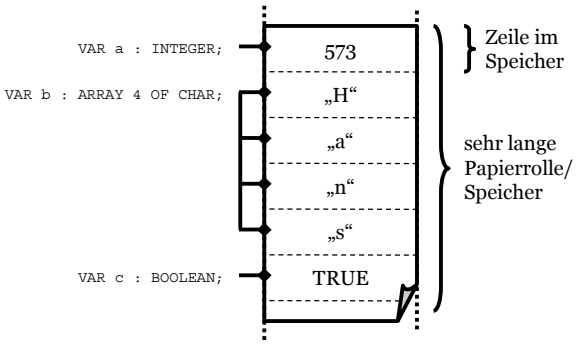
Übungsserie 2 jetzt abgeben, **Übung 3 per Email**!

Was wir heute tun



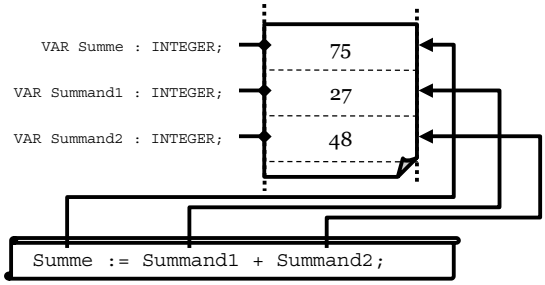
Der Speicher

Wie merkt sich eigentlich der Computer Dinge?



Speicherzugriffe – lesen und schreiben

Variablen beim Lesen & Schreiben



1. lies `Summand1`; 2. lies `Summand2`; 3. schreib `Summe`

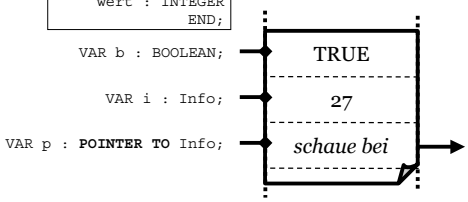
Pointer

Variablen haben fixe Zeile

dynamisch neue andere Zeile? ja, Pointer!

Pointer zeigen

```
TYPE Info = RECORD
    wert : INTEGER
           END;
```



Speicher alloziieren

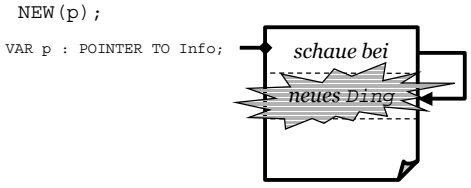
Pointer können nur auf Record & Array zeigen!

Deklaration: `VAR p : POINTER TO Info;`

jetzt zeigt `p` erst ins Speichernirvana!

das `Info` muss erst noch erstellt werden (Alloziieren):

```
NEW(p);
```

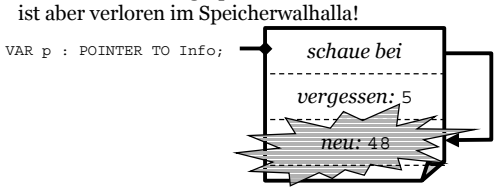


Vergessenes im Speicher & Garbage Collection

```
NEW(p);
p.wert := 5;
NEW(p);
p.wert := 48;
```

} was passiert jetzt?!

die Zeile hat etwas gespeichert,
ist aber verloren im Speicherwalhalla!



was geschieht mit dem Vergessenen?
Garbage Collector sammelt Abfall ein

Der NIL Pointer

```
Var p : POINTER TO Info;
```

wo zeigt `p` jetzt hin? 

eben ins Speichernirvana!

wie heisst das in Oberon?

```
NIL
```

gefährlich beim Programmablauf, deshalb immer testen:

```
IF p = NIL THEN
```

Arrays versus Listen

Array:

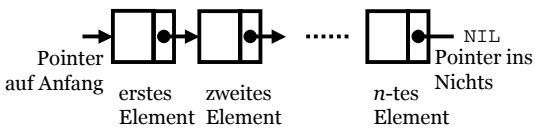
- fixe Länge
- beschränkte Länge
+ direkter Zugriff

„Listen“:

+ variable Länge, Einfügen & Löschen möglich
+ annähernd unlimitierte Länge
- kein direkter Zugriff

Listen

Liste besteht aus Anfang, Elementen & Ende



ein Element:

Teil mit Nutzdaten

Pointer auf Nachfolger

Ein Listenelement in Oberon

Typdeklaration in Oberon:

```
TYPE Element = POINTER TO RECORD
     wert : INTEGER;
     naechster : Element;
END;
```

Wieder ein Record das selben Typ hat, weil Nächster auch ein Element ist

Element ist
Pointer auf ein Record mit
einem Teil mit Nutzdaten (`wert`) und
einem Pointer auf Nachfolger
(`naechster` mit Typ `Element`!)

Operationen auf Listen

Elemente einfügen in Listen?

Elemente aus Listen entfernen?

Nachbesprechung Übung 1 Aufgabe 1

formale syntaktische Definition der UNIX Pfadnamen:

```
Unixpfad = [[Punkt] Separator] {Name Separator} Name.
Name = Buchstabe {Buchstabe}.
Buchstabe = „a“ | „b“ | „c“.
Separator = „\“.
Punkt = „.“.
```

zusätzlich Wildcards:

```
Unixpfad = [[Punkt] Separator] {Name Separator} Name.
Name = („[„ Name „]“) | (Buchstabe {Buchstabe}).
Buchstabe = „a“ | „b“ | „c“ | Wildcard.
Wildcard = „*“ | „?“.
Separator = „\“.
Punkt = „.“.
```

Problem: die Zeichen „[“ und „]“ können im Dateinamen sein, wie auch als Wildcard interpretiert werden. EBNF nicht mehr **eindeutig**!

Nachbesprechung Übung 1 Aufgabe 2

HTML Tabellensyntax:

```
Table = „<TABLE“ {Attribute} „>“ Caption {Row} „</TABLE>“.
Attribute = Key „=“ Value.
Caption = „<CAPTION>“ {AllowedChar} „</CAPTION>“.
Row = „<TR“ {Attribute} „>“ {Column} [„</TR>“].
Column = „<TH“ {Attribute} „>“ {Data} [„</TH>“].
Data = „<TD“ {Attribute} „>“ {AllowedChar} [„</TD>“].
Key = „ROWSPAN“ | „ALIGN“.
```

Nachbesprechung Übung 1 Aufgabe 3

repeat führt min. 1-mal aus

```
REPEAT S UNTIL B;

S;
WHILE ~B DO S END;
```

repeat macht bis B wahr ist

while führt nur, wenn B wahr ist, aus

```
WHILE B DO S END;

IF B THEN
     REPEAT S UNTIL ~B
END;
```

while macht bis B falsch ist

```
IF B THEN S
ELSE IF C THEN T
     ELSE IF D THEN U
          ELSE V END
     END
END;
```

verschachtelte If-Anweisungen

Nachbesprechung Übung 1 Aufgabe 4a

a {B} x := y; {A} y := x {x = a, y = b}

y := x, also A: {x = y = a = b}

x := y, also wurde x überschrieben,
man weiss also nichts über x, nur über y:
B: **{y = a = b}**
->x erhält Wert von y, y bleibt.

b {C} x:=x+y; {B} y:=x-y; {A} x:=x-y {x = a, y = b}

a = x-b, also A: {x = a+b, y = b}

b = (a+b)-y, also B: {y = a, x = a+b}

a+b = x + a, also C: **{x = b, y = a}**

->werte von x und y werden vertauscht.

Nachbesprechung Übung 1 Aufgabe 4b

was gilt nach der While-Anweisung?

```
WHILE B & C DO S END {?}
```

while wird solange ausgeführt bis
`~(B & C) = ~B OR ~C` ist,
so ist die While-Anweisung definiert

```
WHILE B OR C DO S END {?}
```

while wird solange ausgeführt bis
`~(B OR C) = ~B & ~C` ist

Nachbesprechung Übung 1 Aufgabe 4c

Fallunterscheidung
`IF n MOD 2 = 0 THEN`, also n ist gerade

```
a := a*a; n := n DIV 2;
```

`ELSE`, also n ist ungerade

```
x := x*a; n := n - 1;
```

Vorbesprechung Übung 3

andere Module benutzen, z.B. `RandomNumbers`

```
IMPORT RandomNumbers;

zufall := RandomNumbers.Uniform();
```

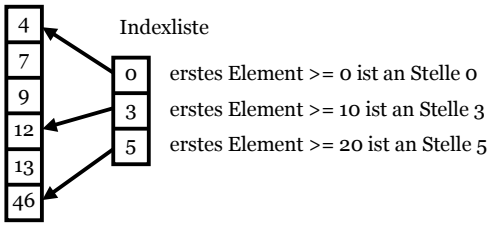
```
IMPORT Out;

Out.String("Hallo da draussen!");
Out.Int(zahl, 5);
Out.Ln();
```

Vorbesprechung Übung 3

Datenstrukturen:

immer durch den ganzen Array aufwändig!

Hilfsmittel Index



Wie löse ich Programmieraufgaben?

1. Aufgabenstellung lesen
2. Abstrakte Lösung überlegen
   welche Module, welche Prozeduren brauche ich?
3. Deklarationen überlegen
   welche Variablen & Datenstrukturen brauche ich?
4. Algorithmus in Pseudocode
5. Oberon starten
6. Modul- & Prozedurdefinitionen & Deklarationen
   ohne Quellcode schreiben
7. Reicht das bis hier hin? Kann ich compilieren?
   Fehlen Prozeduren?
8. Module & Prozeduren mit Quellcode gemäss
   Pseudocode füllen
9. Kompilieren
10. Compilerfehler korrigieren
11. Eigene Lösung mit Eingaben füttern & testen