UNIVERSITETET I OSLO
DET MATEMATISK-NATURVITENSKAPELIGE FAKULTET

Noen andre språkmekanismer (kap 6 og 8)

- Array-er
- Unntak

Noe helt annet

- Dokumentasjon

INF3110/4110

Array-er

Array-er er vanligvis et sammenhengende område i minnet:

0xa0000	122
0xa0004	3
0xa0008	1023
0xa000c	77

Så godt som alle språk har array-er. De kan ha

statisk adresse og størrelse kjent av kompilatoren (Fortran)

statisk størrelse der bare størrelsen er kjent av kompilatoren (C)

variabel størrelse der størrelsen bestemmes ved opprettelsen (C, Java)

dynamisk størrelse der størrelsen kan variere under eksekveringen (Perl)

Fordeler

- Enkle å implementere.
- Raske å slå opp i (og raskere jo mindre dynamisk array-en er)

Ulemper

- Kan av og til være for statiske.
- Noen språk krever indeksering fra 0 eller 1.

Array-er i ML

De finnes ikke, men vi kan lage erstatninger.

1. Lister

Lister ligner litt på array-er — det kan vi utnytte.

```
exception ArrayList;

fun get (nil, i) = raise ArrayList
  | get (a::r, 1) = a
  | get (a::r, i) = get(r,i-1);

fun put (nil, 1, v) = [v]
  | put (nil, i, v) = 0::put(nil,i-1,v)
  | put (a::r, 1, v) = v::r
  | put (a::r, i, v) = a::put(r,i-1,v);

val a = put(put(put(nil, 3, 17), 2, 15), 5, 13);
get(a, 3);
```

Vurdering

+ Enkelt konsept.

– Ubrukte elementer lagres også.

2. Liste med nøkler

Vi kan også lagre indeksen i listen:

```
exception ArrayList;
datatype arrayelem = KeyVal of int * string;

fun get (nil, i) = raise ArrayList
  | get (KeyVal(k,v)::r, i) = if k=i then v else get(r,i);

fun put (a, i, v) = KeyVal(i,v)::a;

val a = put(put(put(nil, 3, "that"), 1, "i"), 5, "shall");
get(a, 3);
val b = put(a, 1, "l");
get(b, 1);
```

Vurdering

+ Raskt å lagre verdier.

– Lagrer «gamle» verdier.

3. Funksjoner

Vi kan lagre array-en som funksjoner:

```
- exception ArrayFun;
exception ArrayFun

- fun put (a, i, v) = (fn(k) => if i=k then v else a(k));
val put = fn : ('a -> 'b) * 'a * 'b -> 'a -> 'b

- fun arr (i) = raise ArrayFun;
val arr = fn : 'a -> 'b
- val arr = put(arr, 3, "that");
val arr = fn : int -> string
- val arr = put(put(arr, 1, "i"), 6, "never");
val arr = fn : int -> string
- val arr = put(put(arr, 2, "think"), 1, "I");
val arr = fn : int -> string

- arr(1);  arr(2);  arr(3);
val it = ``I'': string
val it = ``think'': string
val it = ``that'': string
```

Unntak

Hva gjør man når noe går galt?

De tidligste språkene hadde ingen mekanismer for dette og måtte bruke

Feilverdier er spesielle verdier som ble gitt en spesiell betydning.

- Vanskelig å finne passende verdier.
- Forkludrer koden

Statusverdier er ekstra verdier med spesiell informasjon

- Vanskelig å bruke med funksjoner
- Forkludrer koden

goto-setninger hopper dit feilen kan behandles

- Vanskelig å implementere i blokkstrukturerte språk

Et tidlige forsøk

UCSD Pascal hadde en enkel variant:

```
exit(p)
```

(der p er en prosedyre) hopper ut av prosedyrekall etter prosedyrekall til den har hoppet ut av p.

+ Enkel å implementere

– Kan ikke sende med data

– Kun for prosedyrekall

Dagens unntak bygger på denne ideen med følgende tillegg:

- Unntak er en datatype.
- Brukeren definerer en «handler» der man tar hånd om unntaket.

Hovedtanken med unntak er at *funksjonen* oppdager feil mens *kalleren* vet hvorledes de bør håndteres.

Unntak i ML

ML har en unntaksmekanisme typisk for moderne språk:

```
exception Determinant;  (* Deklarasjon *)

fun invert (aMatrix) =
   if ... then raise Determinant (* Unntaket inntreffer *)
          else ... ;

invert(myMatrix)
   handle Determinant => ...   (* Unntaket tas hånd om *)
```

Eksempel

Det er kalleren som vet best hvorledes feilen bør håndteres:

```
- val Min = 0.001;
val Min = 0.001 : real
- exception Overflow;
exception Overflow
- fun f(x) = if x<Min then raise Overflow else 1.0/x;
val f = fn : real -> real
- (f(1.0) handle Overflow=>0.0)/(f(2.0) handle Overflow=>1.0);
val it = 2.0 : real
- (f(0.0) handle Overflow=>0.0)/(f(0.0) handle Overflow=>1.0);
val it = 0.0 : real
```

Ett eksempel til

Denne funksjonen finner n'te element i en liste:

```
- exception Nth;
exception Nth
- fun nth (x::_, 1) = x | nth(_::r,n) = nth(r,n-1) | nth(_) = raise Nth;
val nth = fn : 'a list * int -> 'a
- val lst = ["A", "B", "C", "D", "E"];
val lst = ["A","B","C","D","E"] : string list
- nth(lst,3);
val it = "C" : string
- nth(lst,0) handle Nth=>"ERROR";
val it = "ERROR" : string
```

(Notasjonen «_» betegner en variabel vi ikke er interessert i.)

Hva brukes unntak til?

Feilsituasjoner

En funksjon kan oppdage at den kalles ulovlig:

```
- datatype 'a tree = LEAF of 'a | NODE of 'a tree * 'a tree;
datatype 'a tree = LEAF of 'a | NODE of 'a tree * 'a tree
- exception NoSubTree;
exception NoSubTre
- fun lsub (LEAF(x)) = raise NoSubTree  |  lsub (NODE(x,y)) = x;
val lsub = fn : 'a tree -> 'a tree

- lsub(NODE(LEAF("A"), NODE(LEAF("B"),LEAF("C"))));
val it = LEAF "A" : string tree
- lsub(LEAF("X"));
/store/opt/smlnj/bin/sml:
   Fatal error -- Uncaught exception ExnDuringExecution
      with <unknown> raised at
      ../compiler/Execution/main/execute.sml:49.48-49.60
```

Optimalisering

Denne funksjonen multipliserer alle verdiene i treet:

```
- datatype 'a tree = LEAF of 'a | NODE of 'a tree * 'a tree;
datatype 'a tree = LEAF of 'a | NODE of 'a tree * 'a tree
- fun prod (LEAF(x)) = x  |  prod (NODE(x,y)) = prod(x)*prod(y);
val prod = fn : int tree -> int
- prod(NODE(LEAF(3), NODE(LEAF(4),LEAF(5))));
val it = 60 : int
```

Av og til kan man bryte av utregningen litt tidlig og spare tid:

```
- datatype 'a tree = LEAF of 'a | NODE of 'a tree * 'a tree;
datatype 'a tree = LEAF of 'a | NODE of 'a tree * 'a tree

- exception Zero;
exception Zero
- fun prodZ (t) =
=     let exception Zero
=         fun p (LEAF (x)) = if x=0 then raise Zero else x
=           | p (NODE(x,y)) = p(x)*p(y)
=     in
=         p(t) handle Zero => 100 (* Should be 0 *)
=     end;
val prodZ = fn : int tree -> int

- prodZ(NODE(LEAF(3), NODE(LEAF(4),LEAF(5))));
val it = 60 : int
- prodZ(NODE(LEAF(0), NODE(LEAF(4),LEAF(5))));
val it = 100 : int
```

Problem med unntak

Statisk og dynamisk binding

ML er et språk med *statisk* binding:

```
val x = 6;
let fun f(y) = x
    and g(h) = let val x = 2 in h(1) end
in
    let val x = 4 in g(f) end
end;
```

gir svaret

```
    val x = 6 : int
    val it = 6 : int
```

INF3110/4110

Unntak har *dynamisk* binding:

```
exception X;

( let fun f(y) = raise X
      and g(h) = h(1) handle X=>2
  in
      g(f) handle X=>4
  end
) handle X=>6;
```

gir svaret

```
exception X
val it = 2 : int
```

INF3110/4110

Frigjøring av data

Anta at vi har følgende kode:

```
exception X;

( let
      val y = ref [1, 2, 3];
  in
      y := 99::(!y);
      raise X
  end
) handle X=>0;
```

Ved unntaket blir listen [99,1,2,3] liggende igjen.

ML har *automatisk søppeltømming* som tar seg av slikt.

I språk som C++ må programmereren legge inn ekstra kode for å unngå minnelekkasje.

Dokumentasjon

- Hvorfor skrive dokumentasjon? For hvem?
- Javas dokumentasjon
- *«Lesbar programmering»* («Literate programming»)

Med hvert skikkelig program bør det komme følgende dokumentasjon:

- innføring for nye brukere
- oppslagsverk for erfarne brukere
- guide om installasjon og tilpasning for systemansvarlige
- dokumentasjon for de som senere skal vedlikeholde og oppdatere programmet.

I dette kurset er det siste punkt som opptar oss.

Dokumentasjon og programkode

Alle programmeringsspråk har muligheten for å legge inn kommentarer i programkoden. Uten slik dokumentasjon er det ofte vanskelig å forstå ideene bak programmet.

```
/* Sort array a with n elements using 'bubble sort'. */

void bubble(int a[], int n)
{
  int i, temp, n_swaps;

  do {
    n_swaps = 0;
    for (i=0; i<n-1; ++i)
      if (a[i]>a[i+1]) {
        temp = a[i];  a[i] = a[i+1];  a[i+1] = temp;
        ++n_swaps; }
  } while (n_swaps > 0);
}
```

Problemet med slike kommentarer er imidlertid at programkoden ofte «drukner» i kommentarene.

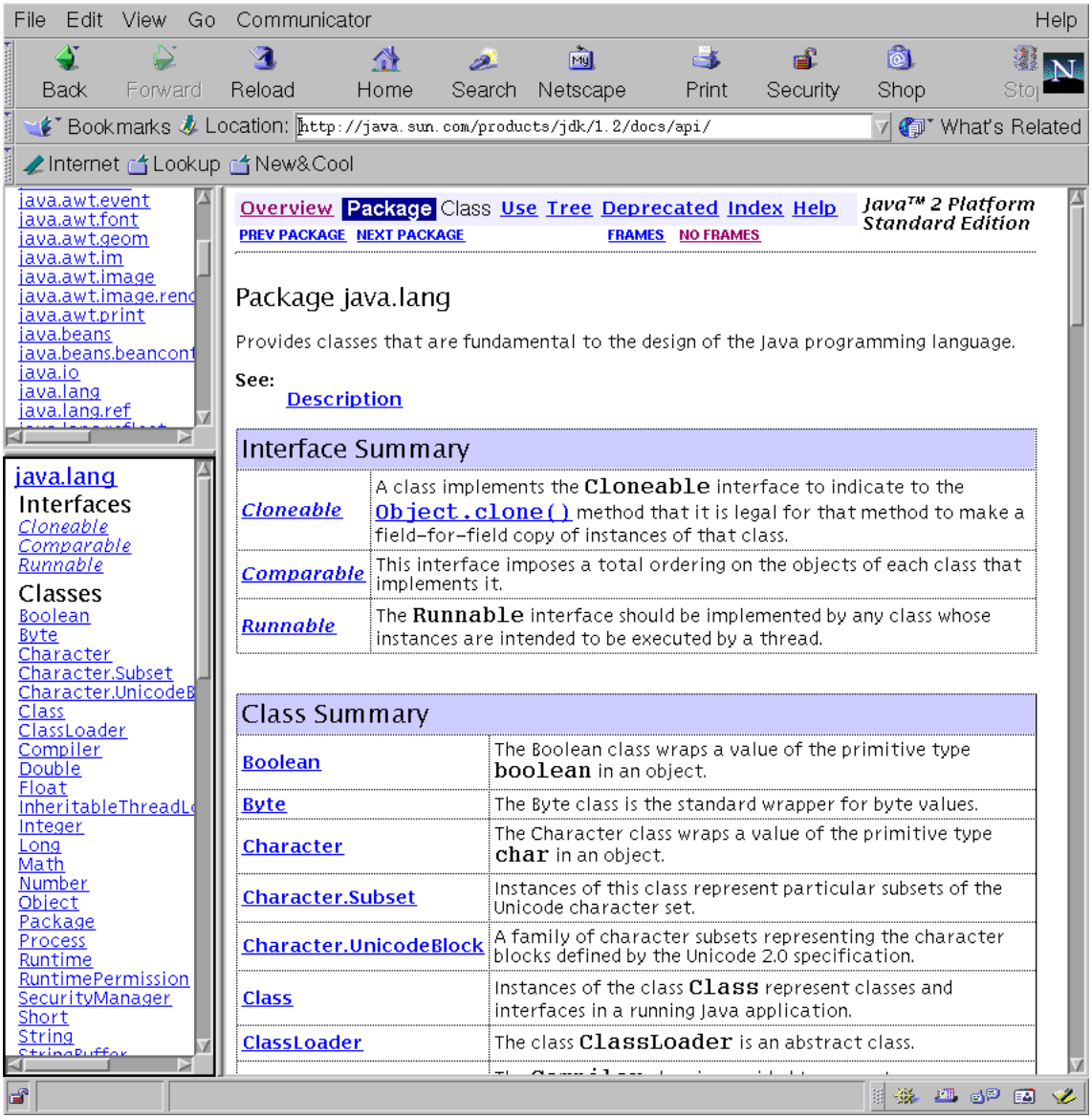
```
/* Sort array 'a' with 'n' elements using
   the technique called 'bubble sort'. */

void bubble(int a[], int n)
{
  /* Variables:
       i:       loop index
       temp:    temporary variable used during swapping
       n_swaps: count the number of swaps done */
  int i, temp, n_swaps;

  /* Loop until no swaps were performed, as this implies
     that the array is completely sorted. */
  do {
    /* Initialize the variables used in the loop. */
    n_swaps = 0;
    for (i=0; i<n-1; ++i)
      /* In a bubble sort, all adjacent array elements are
         compared.  If they are out of order, they are
         swapped*/
      if (a[i]>a[i+1]) {
        /* Swapping uses a temporary variable. */
        temp = a[i];  a[i] = a[i+1];  a[i+1] = temp;
        /* Remember to update the swap counter. */
        ++n_swaps; }
  } while (n_swaps > 0);
  /* The array should now be completely sorted. */
}
```

Javas dokumentasjon

I Java har man prøvd å gjøre dette bedre. Der kan man legge inn dokumentasjon i tillegg til vanlige kommentarer:

```
/*
  Copyright (c) 1996, 2000.
        :
 */
package java.lang;
/**Provides classes that are fundamental to the design of
 * the Java programming language.
 * @see Description
 */
public class Object {
/** Class Object is the root of the class hierarchy.
  * Every class has Object as a superclass. All objects,
  * including arrays, implement the methods of this class.
  * @see Class
       :
  */
```

Utifra dette genereres automatisk dokumentasjonen:



Lesbar programmering

Donald Knuth var én av de første med et videre syn på dokumentasjon:

> *Programmer bør skrives for mennesker og ikke for maskiner.*

Dette medfører

- Programmet må deles opp i biter av passe størrelse.
- Rekkefølgen må tilpasses leseren.
- Man bør utnytte det rike utvalget av typografiske teknikker som har oppstått de siste fem hundre år:
  - Angivelse av dokumentets struktur (kapittel, hovedavsnitt, underavsnitt, ...)
  - Kryssreferanser, fotnoter, stikkordlister, ...
  - Matematiske formler, figurer, tabeller, ...
  - Ulike typesnitt som *kursiv*, **fete typer** og `skrivemaskinskrift`.

Bubble sort

Dag Langmyhr
Department of Informatics
University of Oslo
dag@ifi.uio.no

November 6, 2000

This short article describes *bubble sort*, which quite probably is the easiest sorting method to understand and implement. Although far from being the most efficient one, it is useful as an example when teaching sorting algorithms.

Let us write a function `bubble` in C which sorts an array `a` with `n` elements. In other words, the array `a` should satisfy the following condition when `bubble` exits:

$$\forall i, j \in \mathbb{N} : 0 \le i < j < \mathtt{n} \Rightarrow \mathtt{a}[i] \le \mathtt{a}[j]$$

#1 ⟨*bubble sort*⟩ ≡

```
1  void bubble(int a[], int n)
2  {
3      ⟨local variables #4 (p.1)⟩
4
5      ⟨use bubble sort #2 (p.1)⟩
6  }
```

(This code is not used.)

Bubble sorting is done by making several passes through the array, each time letting the larger elements "bubble" up. This is repeated until the array is completely sorted.

#2 ⟨*use bubble sort*⟩ ≡

```
7  do {
8      ⟨perform bubbling #3 (p.1)⟩
9  } while (⟨not sorted #7 (p.2)⟩);
```

(This code is used in #1 (p.1).)

Each pass through the array consists of looking at every pair of adjacent elements;[1] if the two are in the wrong sorting order, they are swapped:

#3 ⟨*perform bubbling*⟩ ≡

```
10 ⟨initialize #6 (p.2)⟩
11 for (i=0; i<n-1; ++i)
12   if (a[i]>a[i+1]) { ⟨swap a[i] and a[i+1] #5 (p.2)⟩ }
```

(This code is used in #2 (p.1).)

The `for`-loop needs an index variable `i`:

#4 ⟨*local variables*⟩ ≡

```
13 int i;
```

(This code is extended in #4$_a$ (p.2). It is used in #1 (p.1).)

[1] We could, on the average, double the execution speed of `bubble` by reducing the range of the `for`-loop by 1 each time. Since a simple implementation is the main issue, however, this improvement was omitted.

Swapping two array elements is done in the standard way using an auxiliary variable `temp`. We also increment a swap counter named `n_swaps`.

#5 ⟨*swap a[i] and a[i+1]*⟩ ≡

```
14  temp = a[i];  a[i] = a[i+1];  a[i+1] = temp;
15  ++n_swaps;
```

(This code is used in #3 (p.1).)

The variables `temp` and `n_swaps` must also be declared:

#4a ⟨*local variables* #4 (p.1)⟩ +≡

```
16  int temp, n_swaps;
```

The variable `n_swaps` counts the number of swaps performed during one "bubbling" pass. It must be initialized prior to each pass.

#6 ⟨*initialize*⟩ ≡

```
17  n_swaps = 0;
```

(This code is used in #3 (p.1).)

If no swaps were made during the "bubbling" pass, the array is sorted.

#7 ⟨*not sorted*⟩ ≡

```
18  n_swaps > 0
```

(This code is used in #2 (p.1).)

Ulike utgaver av *lesbar programmering*

Det finnes mange implementasjoner av *lesbar programmering*: web, cweb, noweb, web_0, ...

Hvordan skriver man *lesbar programmering*?

Dokumentasjonen skrives som annen dokumentasjon for det valgte verktøyet (f. eks. LaTeX).

Programkoden flettes inn i dokumentasjonen:

- Man definerer diverse metasymboler.
- Hvert metasymbol defineres som én eller flere linjer programkode.
- Definisjonen av et metasymbol kan inneholde referanser til andre metasymboler.

```
\documentclass[12pt,a4paper]{webzero}
\usepackage[latin1]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{amssymb,palatino,mathpple}

\title{Bubble sort}
\author{Dag Langmyhr\\ Department of Informatics\\
  University of Oslo\\[5pt] \texttt{dag@ifi.uio.no}}

\begin{document}
\maketitle

\noindent This short article describes \emph{bubble
  sort}, which quite probably is the easiest sorting
method to understand and implement.
Although far from being the most efficient one, it is
useful as an example when teaching sorting algorithms.

Let us write a function \texttt{bubble} in C which sorts
an array \texttt{a} with \texttt{n} elements. In other
words, the array \texttt{a} should satisfy the following
condition when \texttt{bubble} exits:
\[
  \forall i, j \in \mathbb{N}: 0 \leq i < j < \mathtt{n}
  \Rightarrow \mathtt{a}[i] \leq \mathtt{a}[j]
\]


<<bubble sort>>=
void bubble(int a[], int n)
{
  <<local variables>>

  <<use bubble sort>>
}
@
Bubble sorting is done by making several passes through the
array, each time letting the larger elements
``bubble'' up. This is repeated until the array is
completely sorted.

<<use bubble sort>>=
do {
  <<perform bubbling>>
} while (<<not sorted>>);
@
```

INF3110/4110

```
Each pass through the array consists of looking at
every pair of adjacent elements;\footnote{We could, on the
  average, double the execution speed of \texttt{bubble} by
  reducing the range of the \texttt{for}-loop by~1 each time.
  Since a simple implementation is the main issue, however,
  this improvement was omitted.} if the two are in
the wrong sorting order, they are swapped:
<<perform bubbling>>=
<<initialize>>
for (i=0; i<n-1; ++i)
  if (a[i]>a[i+1]) { <<swap a[i] and a[i+1]>> }
@
The \texttt{for}-loop needs an index variable
\texttt{i}:

<<local var...>>=
int i;
@
Swapping two array elements is done in the standard way
using an auxiliary variable \texttt{temp}. We also
increment a swap counter named \texttt{n\_swaps}.

<<swap ...>>=
temp = a[i];  a[i] = a[i+1];  a[i+1] = temp;
++n_swaps;
@
The variables \texttt{temp} and \texttt{n\_swaps}
must also be declared:

<<local var...>>=
int temp, n_swaps;
@
The variable \texttt{n\_swaps} counts the number of
swaps performed during one ‘‘bubbling’’ pass.
It must be initialized prior to each pass.

<<initialize>>=
n_swaps = 0;
@
If no swaps were made during the ‘‘bubbling’’ pass,
the array is sorted.

<<not sorted>>=
n_swaps > 0
@

\wzvarindex  \wzmetaindex
\end{document}
```

Eksempler

De mest kjente eksemplene på bruk av *lesbar programmering* er Donald Knuths bøker «TEX: the program» og «METAFONT: the program».

Ellers finnes flere eksempler på /store/opt/doc/programs/.

Hvor finnes mer informasjon?

- /store/opt/doc/programs/web0.pdf, spesielt de første 13 sidene.
- Donald Knuth og Silvio Levy: *The CWEB system of structured documentation*; Addison-Wesley 1994.

  En tidlig versjon finnes som /local/doc/tex/cwebman.dvi.