Tampereen yliopisto

From pseudocode to implementation

COMP.CS.300 Data structures and algorithms 1

Matti Rintala (matti.rintala@tuni.fi)

Implementing pseudocode

- Adapting to the application
- Sanity check of inputs etc.
- Error handling
- Limitations from programming language
- Speed and practicality issues arising from hardware and language
- Maintainability ⇒ modularity etc.

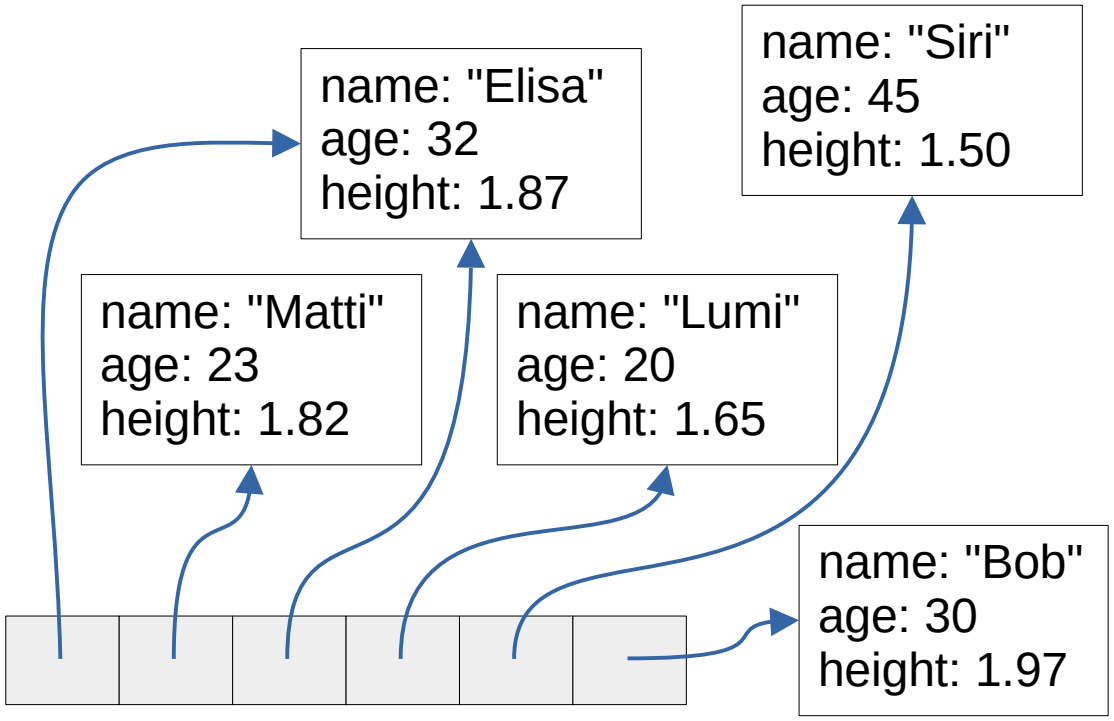
Pseudocode

Insertion-Sort(*A*)

```
1 for next_elem := 2 to A.length do
2     key := A[next_elem]
3     place := next_elem − 1
4     while place > 0 and A[place] > key do
5         A[place + 1] := A[place]
6         place := place − 1
7     A[place + 1] := key
```

Effects of the programming language

- Indexing starts from 0 (in pseudocode often from 1)
- Is indexing even used (or arrays, or...)
- Is data copied, or referred to indirectly
- If outside data is referred to **indirectly**, does it happen with: *pointer*, *smart pointer* (shared_ptr...), *iterator* (if data in a data structure), *index* (if data in vector etc.), *search key* (if data in a structure with fast search)
- Are algorithm's "parameters" real parameters, or just variables etc.


name: "Elisa"
age: 32
height: 1.87
name: "Siri"
age: 45
height: 1.50
name: "Matti"
age: 23
height: 1.82
name: "Lumi"
age: 20
height: 1.65
name: "Bob"
age: 30
height: 1.97

Implementation

```
1  #include <vector>
2  struct Data { string name; int age; float height; };
3  using Taulukko = std::vector<Data*>;
4  // Täytetään taulukko osoittimilla dataan
5  void insertion_sort(Taulukko& A) {
6      Data* keyp = nullptr; int place = 0;
7      for (Taulukko::size_type next_elem = 1;
               next_elem < A.size(); ++next_elem) {
8          keyp = Taulukko.at(next_elem);
9          place = next_elem-1;
10          while (place >= 0) {
11            elemp = Taulukko.at(place);
12             assert(elem != nullptr);
13             if (keyp->name < elemp->name) { break; }
14            Taulukko.at(place+1) = elemp; --place;
15         }
16        Taulukko.at(place+1) = keyp;
17     }
18 }
```

Insertion-Sort(*A*)

```
1 for next_elem := 2 to A.length do
2     key := A[next_elem]
3     place := next_elem − 1
4     while place > 0 and A[place] > key do
5         A[place + 1] := A[place]
6         place := place − 1
7     A[place + 1] := key
```