Mappings

• Types of mappings:
  – Arrays
  – Functions

• Arrays map integers to the declared type of the variables in the array.

• Functions map an arbitrary domain type to an arbitrary range type.
Mappings in general

• $S = \{0,1\}$
• $T = \{a,b,c\}$
• How many mappings $S \rightarrow T$ are there?
• $\#(S \rightarrow T) = \#(T)^{\#(S)}$
• The possible $S \rightarrow T$ mappings are:
  
  \[
  \{0 \rightarrow a, 1 \rightarrow a\}, \{0 \rightarrow a, 1 \rightarrow b\}, \{0 \rightarrow a, 1 \rightarrow c\}, \\
  \{0 \rightarrow b, 1 \rightarrow a\}, \{0 \rightarrow b, 1 \rightarrow b\}, \{0 \rightarrow b, 1 \rightarrow c\}, \\
  \{0 \rightarrow c, 1 \rightarrow a\}, \{0 \rightarrow c, 1 \rightarrow b\}, \{0 \rightarrow c, 1 \rightarrow c\}
  \]
Implementing a finite mapping as an array

T[] map = {a,c};
defines map to be the third mapping given on previous slide:

map[0] = a
map[1] = c
Implementing a finite mapping as a function

We can also define the mapping as a function (ML example):

```ml
fun map 0 = a
   | map 1 = c;
```

or (in Pascal):

```pascal
function map(s:S):T;
begin
  case (s) of
    0: map:=a;
    1: map:=c;
  end;
end;
```
More discussion

We can also define the mapping as a function (ML example):

```ml
fun map x =
    case x of
        0 => a
        1 => c
```

(Notice function syntax in case expression!)

In C:

```c
T map(int x) {
    switch (x) {
        case 0: return a;
        case 1: return c;
    }
}
```
More general finite mappings

Hash tables let us define more general finite mappings (domain type does not need to be integral).
Memoization

• Combination of computation and storage!
Array-based vs. function-based mappings

• An array-based mapping is explicit:
  – it gives domain-range pairs explicitly

• A function-based mapping is implicit:
  – for each domain element it gives a method for computing the corresponding range element.