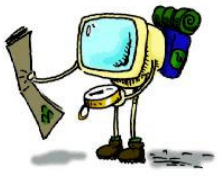
Programming Languages

Tevfik Koşar

Lecture - XV
March 14th, 2006

Roadmap

- Data Types
- Type Checking
- Polymorphism
- Type Equivalence
- Polymorphism



Data Types

Types provide implicit context for many operations

- a+b
  - Types of a and b determine which addition function to use
  - eg. integer addition, or floating point addition
- new p
  - will allocate storage from the heap right size to hold the object pointed by p

3

Data Types

Types limit set of operations

- prevents illegal or meaningless operations:
  - adding a record and a char
  - passing a file as parameter to a functions which expects an int
  - taking sinus of a pointer

4

Type Checking

- Process of ensuring a program obeys the language's type compatibility rules
- Strongly typed language: prohibits application of any operation to any object that is not intended to support that operation.
- Statically typed language: if it is strongly typed and type checking can be performed at compile time.
- In practice, most type checking is performed at compile time, and the rest at runtime.

5

Polymorphism

- Allows same code to work with objects of multiple types.
- p1:= p2 + p3; // p1 can be int, char, pointer, // array, list, record ...
- the compiler does not need to know whether "addition" function is implemented for all of these types.

6

Type Checking

- Every definition of an object must specify the object's type *(statically types lang.)*
- In type checking, three important notions:
  - type equivalence
  - type compatibility
  - type inference

7

Type Equivalence

Two principal ways of defining type equivalence:

- Structural equivalence: Two types are equal if they consist of the same components

```
1) type foo1 = record a, b: integer end;

2) type foo2 = record
        a , b: integer
   end;

3) type foo3 = record
        a: integer;
        b: integer;
   end;

4) type foo4 = record
        b: integer;
        a: integer;
  end;
```

- The last one is equal to the rest in most languages, but not in ML!

8

Type Equivalence

- Structural equivalence:

Example 2:

1) type str1 = array [1..10] of char;

2) type str2 = array [1..2*5] of char;

3) type str3 = array [0..9] of char;

- The second one is equal to the first one, but not the third one!
  (length of the array is same, but index values are different)

9

Type Equivalence

- Name equivalence: Each definition introduces a new type.

1) type foo1 = record a, b: integer end;
2) type foo2 = record a, b: integer end;

foo1 and foo2 are considered as different types!

10

Type Compatibility

- Most languages require type compatibility instead of type equivalence (depending on the context).
- Eg. Assignment statement (a:=b;)
  - The type of the right hand side must be compatible with that of the left hand side.
- Eg. Addition (a+b)
  - The types of both operators must be compatible with either integer or with floating point type.
- Coersion: Automatic, implicit conversion of types.
- Eg. int a; float b; float c;
  - b = a;
  - c = a + b;

11