CHAPTER 7: DATA TYPES – PART 1

CMPSC 460 – Principles of Programming Languages

Types

- Meaning of types:
  1. Denotational
  2. Constructive
  3. Abstraction-based

Type Systems

- A type system consists of:
  - Mechanism to define types and associate values to them
  - Set of rules for:
    - Type equivalence
    - Type compatibility
    - Type inference

Data Types

- Type declaration can be:

1. Following rules
2. Implicit
   a) Dynamic Binding
   b) Static Binding
3. Explicit

Data Types

```
dynamic x;

x = "Hello world";
Console.WriteLine(x.Length);

x = 1;
Console.WriteLine(x);
```

Data Types

```
include <iostream>
#include <list>
using namespace std;

int main()
{
    list<int> lst1;
    for (int j = 1; j <= 5; j++)
        lst1.push_back(j);


    for(auto i = lst1.begin(); i != lst1.end(); ++i)
        cout << *i << endl;

    return 0;
}
```

Subtypes

- Subtype:
  - Part of a larger class (Supertype)
- Example:
  - type

```
type
    smallInteger = 1..20;
    weekday = (sun, mon, tue, wed, thu, fri, sat);
    workday = mon..fri;
```

Type Checking

```
X = A + B * C(23, 34.4)
```

Type Checking

- The activity of ensuring that the operands of an operator are of compatible types.

Type Checking

- When do we perform type checking:
  - Run-Time Checking (Dynamically Typed Languages)
  - Compile-Time Checking (Statically Typed Languages)
  - Compile-Time & Run-Time Checking

Static Typing vs Dynamic Typing

- Advantages of Static Typing
  - Efficiency
  - Choice of storage management & representation
  - Static type checking
  - Readability

- Disadvantages of Static Typing
  - Lack of flexibility
  - Compilers are harder to implement

Data Types

- ***Strongly Typed Languages***
  - Each name has a single type
  - Type errors are always detected
- ***Weakly Typed Languages***
  - Allows a value of one type to be treated as another

Data Types

- Examples
  - Common Lisp, Scheme
  - ML
  - Ada
  - Pascal
  - Java
  - C, C++

Standard ML

ML

- No side effects
- Functions are first-class values
- all functions take exactly one parameter and return exactly one result
- **<u>ML is strongly typed</u>**
- **<u>polymorphic type mechanism</u>**

ML - Types

- int
- real
- string
- tuple
- list

ML - Types

- 2;
- 1.2;
- “hello“;
- (3,4);
- [5, 10, 15];

ML - Identifiers

- val a:int = 4;
- val a = 4;
- val a = 3 and

  b = 2.5 and

  c = 2;

ML - Lists

- val lst = [(1,2),(3,4),(5,6)];
- val lst = [[1,2],[3,4]];
- val lst = [[1,2],[3,4,5]];
- val lst = [(1,1.2),(2.2,2)];
- val lst = [(1,2),(3,4,5)];

ML - Lists

nil	[]
1 :: [2,3]	[1,2,3]
1 :: 2 :: 3 :: nil	[1,2,3]
null []	true
null [1,2,3]	false
hd [1,2,3]	1
tl [1,2,3]	[2,3]
[1,2] @ []	[1,2]
[] @ [3,4]	[3,4]
[1,2] @ [3,4]	[1,2,3,4]

ML - Functions

- all functions take exactly one parameter and return exactly one result
- Parameters to functions do not generally need to be parenthesized
- A function that expects a pair of elements can be converted to an infix operator
- ML programs seldom need to use the *if* expression when pattern alternatives are used.

ML - Functions

```
val fourthroot : real -> real =
        fn x : real => Math.sqrt x;
```

ML - Functions

- **val** f = **fn** x => x + 1;
- **val** g = **fn** (a,b) => (a + b) **div** 2;

- f 3 * 4
- (f, f 3);
- g(f 2, f 3);
- **val** h = g;

ML - Functions

```
val rec summation =
        fn nil => 0
        | (head :: tail) => head + summation tail;
```

ML - Functions

Exercise:

- Write the factorial function
- Write the power function

ML - Functions

```
val rec ** = fn (a,0) => 1 | (a,b) => a * **(a, b-1);

infix **;
4**5;
```

Type Inference

- The process of determining the types of expressions based on the known types of its sub-parts

ML - Type Inference

- [1,2];
- [(3,4),(5,6)];
- 3.14 + 2;
- [(3,4),(5,6)] : (int * int) list;
- val nl = fn s => s ^ "\n“;
- val Identity = **fn** x =>x;
- if (2>3) then 5 else 14.7;

ML - Type Inference

1. ML assigns a <u>type identifier</u> to each expression whose type is unknown
2. Solve for the type identifiers using the following inference constraints:
   a. All occurrences of the same identifier <u>have the same type</u>
   b. Conditional expression: Cond must have type bool, and branch1, branch2, and the total expression have the same type.
   c. Function application: ('a -> 'b) 'a : 'b
   d. Function definition: **fn** 'a => 'b : 'a -> 'b

ML - Type Inference

- Example 1

```
val rec length = fn AList =>
        if null AList then 0
        else
                1 + length(tl AList);
```

ML - Type Inference

- Example 2

```
val f1 = fn x => (x 3, x true);
```

ML - Type Inference

Example 3

val plus = **fn** (a,b) => a + b;

- **val** f = **fn** (a : int, b : int) => a+b;
- **val** f = **fn** (a : int, b) => a+b;
- **val** f = **fn** ((a,b) : int * int) => (a + b) : int;
- **val** f = **fn** (a,b) => (a + b) : int;

ML - Type Inference

Exercise:

```
fun square x = x * x;
```

ML - Polymorphic Types

A function is **polymorphic** when it can work uniformly over parameters of different data types.

ML - Polymorphic Types

```
val f = fn [x,y,z] => (x, y, z);
val (a,b,c) = f[1,2,3];
```

ML - Polymorphic Types

nil	`'a list`
::	`('a * 'a list) -> 'a list`
null	`('a list) -> bool`
hd	`('a list) -> 'a`
tl	`('a list) -> ('a list)`
@	`('a list * 'a list) -> ('a list)`

ML - Polymorphic Types

```
val rec length =
    fn nil => 0
    | (h :: tail) => 1 + length tail;
```

ML - Polymorphic Types

Exercise

- Write an ML function that swaps a tuple (pair) of two values.
- What is the type of this function?

ML - Currying

1. fun curry a = fn b => a+b;
2. fun add x y z = x + y + z;

Type Equivalence

- Two major approaches to define type equivalence:
  - Structural equivalence
  - Name equivalence

Structural Type Equivalence

- Two variables have equivalent types:
  - their types have identical structures

- Low level implementation of equivalence

Structural Type Equivalence

```
#include <iostream>
using namespace std;

typedef  int  A;
typedef  int  B;

int main()
{
   A x;
   B y = 7;
   x = y;
   cout << x << endl;

   return 0;
}
```

Structural Type Equivalence

```
Program HelloWorld(output);
type
        weekday = (sun, mon, tue, wed, thu, fri, sat);
        test_score = 0..100;
var
        i: integer;
        score: test_score;
        today : weekday;
begin
    i := 299;
    score := i;
    today := mon;

    writeln('i = ', i);
    writeln('score = ', score);
    writeln('today = ', today);
end.
```

Structural Type Equivalence

- What about the format of the declaration?

```
struct { int a, b; }
```

```
struct {
    int a, b;
}
```

```
struct {
    int a;
    int b;
}
```

Structural Type Equivalence

- What about the order of the declaration?

```
type R1 = record
    a : integer;
    b : integer;
end;
```

```
type R2 = record
    b : integer;
    a : integer;
end;
```

Structural Type Equivalence

- What about array subscripts?

```
type str = array [1..10] of char;
```

```
type str = array [0..9] of char;
```

Structural Type Equivalence

```
type student = record
         name, address : string;
         age : integer;
     end;
```

```
type university = record
         name, address : string;
         age : integer;
     end;
```

Name Type Equivalence

- Two variables have equivalent types:
  - They are in the same declaration
  - Declared with the same type name

Name Type Equivalence

```
TYPE celsius_temp = REAL;
fahrenheit_temp = REAL;
VAR c : celsius_temp;
    f : fahrenheit_temp;

    c := 100.0;
    f := c;
```

Name Type Equivalence

```
#include <iostream>
using namespace std;

typedef struct {int m;} A;
typedef struct {int m;} B;

int main()
{
    A x;
    B y;
    x = y;

   return 0;
}
```

Name Type Equivalence

- Two common variants on name equivalence
  - <u>Loose name equivalence</u>: aliased types are considered the same
  - <u>Strict name equivalence</u>: aliased types are considered different

Name Type Equivalence

Ada allows both:

```
With Ada.Text_IO; Use Ada.Text_IO;

procedure Program is
        type weekday is (sun, mon, tue, wed, thu, fri, sat);
        subtype test_score is integer range 0..100;
        subtype workday is weekday range mon..fri;
        score : test_score;
        day1 : weekday;
        day2 : workday;
        x : Integer := 1;

begin
    day1 := mon;
    day2 := day1;
    score := x;
        put(natural'image(score));
end Program;
```

Type Checking

- What is a type mismatch:
  - Two types are not equivalent

  or

  - Two types are not compatible

Type Conversion

```
#include <iostream>
using namespace std;

int main()
{
    int x =55;
    char c;

    c= x;
    cout << c << endl;

    c= -10;
    cout << c<< endl;

   return 0;
}
```

References

- Michael L. Scott, Programming Language Pragmatics, Morgan Kaufmann, 3rd edition, 2009.
- Robert W. Sebesta, Concepts of Programming Languages, Addison Wesley, 10th edition, 2012.
- John C. Mitchell, Concepts in Programming Languages, Cambridge University Press, 2002.
- Terrence W. Pratt and Marvin V. Zelkowitz, Programming Languages: Design and Implementations, Prentice Hall, 4th edition, 2001.