COEN 175

Lecture 11: Type Expressions

Type Expressions

- A **type expression** or **type signature** denotes the type of an expression.
- Any built-in or “atomic” type in the language is a legal type expression.
- If *S* and *T* are type expressions, then:
  - $S \rightarrow T$ denotes a mapping from type *S* to type *T*
  - $S \times T$ denotes a (Cartesian) product of type *S* and type *T*
  - pointer(*T*) denotes a pointer to type *T*
  - array(*T*, length) denotes an array of type *T*

Examples

- Give type expressions for the following declarations:

```
int x;            // int
double *p;        // pointer(double)
char a[10];       // array(char, 10)
int **q;          // pointer(pointer(int))
double *b[10];    // array(pointer(double), 10)
char (*c)[10];    // pointer(array(char, 10))
int **d[10];      // array(pointer(pointer(int)), 10)
long *(*e)[10];   // pointer(array(pointer(long), 10))
```

More Examples

- Give type expressions for the following declarations:

```
void f(int);              // int -> void
double g(int *);          // pointer(int) -> double
char h(int, int);         // int x int -> char
int (*p)(int);            // pointer(int -> int)
double **x(int);          // int -> pointer(pointer(double))
char *(*q)(void);         // pointer(void -> pointer(char))
int (*a[4])(double);      // array(pointer(double -> int), 4)
char y(int (*)(long));    // pointer(long -> int) -> char
```

Operators vs. Functions

- An operator is just a special way of writing a function with a different syntax and name.
- In languages such as Scheme, operators are in fact functions with the same syntax.
  - We don't write `a + b` but rather `(+ a b)` just as we would write `(cons a b)`.
- An operator such as + is just a binary function.
- Thus, we can write type expressions for operators.

Example: Division

- What are the type expressions for `/` in Simple C?
  - `int` × `int` → `int`
  - `double` × `double` → `double`
- Thus, the `/` operator is overloaded.
  - Either integer or floating-point division is performed.
- Two questions should come to mind:
  - What about characters?
  - What about division with mixed-type operands?

Type Promotions

- In C and Simple C, all characters are coerced to integers before any operation.
- This coercion is called a **promotion**.
- There are two automatic promotions in Simple C:
  - A character is promoted to an integer.
  - An array is promoted to a pointer.
- Promotions help reduce the total number of cases.
  - In the original C standard, a `float` was always promoted to a `double`, but a more recent standard eliminated it.

Type Coercions

- Type coercions also reduce the number of cases.
- Whereas a processor will have both integer and floating-point division, it likely won't have a mixed-type operation.
- An `int` can be converted to a `double` with little or no loss in precision, so it can be coerced.
- C includes coercions for all built-in types, both signed and unsigned, and integral and floating-point.

Type Coercions

- Type coercions are just operations or functions that are implicitly performed for us.
- Therefore, we can write type expressions for them:
  - `char` → `int`
  - `int` → `double`
  - array(α, *n*) → pointer(α)
- In this last expression, we use α to denote any type.
  - Thus, this last coercion is itself polymorphic.

Another Example

- What are the type expressions for * in Simple C?
  - `int` × `int` → `int`
  - `double` × `double` → `double`
  - pointer(α) → α
- The first two expressions are for the binary case of multiplication.
- The last expression is for the unary case of pointer dereference.
  - Given a pointer to an object of some type, the result is an object of that type.