Chapter 7
(Selected Topics)

Data Types, Compatibility, Arrays

Type Checking

The process of ensuring that the operands of an operator are of compatible types

Compatible type: one that is either legal for the operator, or is allowed under language rules to be implicitly converted to a legal type

Type error: attempting an operation on a value for which the operation is not defined

Strong Typing

A programming language is *strongly typed* if type compatibility rules are always enforced (prevents applying an operation to data for which it is not appropriate)

Strong typing can be an effective tool for improving the reliability and security of programs

- *Never trust user supplied data*

Type Binding

Static: Done at compile time (e.g. Ada, C)

Dynamic: Done at run time (e.g. JavaScript, Ruby)

JavaScript:

```
list = [2, 4.33, 6, 8];
list = 17.3;
```

Java has aspects of both

Coercion

Weakens strong typing

Implicit type conversion

Used extensively in C++ (operator overloading)

Example:

```
int i, j = 3;
i = j + 2.718;
```

1) Convert `j` to `double`
2) Add `j` and `2.718` to get a `double`
3) Truncate the result to `int` and assign to `i`

Coercion

Not allowed in Ada

Example:

```
i, j : Integer;
. . .
j := 3;
i := Integer( (Float(j) + 2.718) - 0.5 );
```

Type conversions (explicit)

Array Types

An array is an aggregate of (usually) homogeneous data where an individual element is identified by its position relative to the first element.

A *heterogeneous array* is one in which the elements need not be of the same type
(e.g. Perl, Python, JavaScript, and Ruby)

Array Design Issues

- What types are legal for subscripts?
- Are subscripting expressions in element references range checked?
- When are subscript ranges bound?
- When does allocation take place?
- What is the maximum number of subscripts?
- Can array objects be initialized?
- Are slices supported?

Array Indexing

Indexing (or subscripting) is a mapping from indices to elements

`array_name(index_value_list)` → `an element`

Index Syntax

- FORTRAN, Ada use parentheses
  Ada uses parentheses to show uniformity between array references and function calls because both are *mappings*
- Most other languages use brackets

Arrays Index Types

FORTRAN, C, Java: integer only

Ada: integer or enumeration (includes Boolean and char)

Java: integer only, starting at 0

C#: integer or *string* (actually a hash)

Index range checking

C, C++, Perl, and Fortran do not use range checking

Ada, Java, C# enforce range checking

Subscript Binding and Storage Allocation

Static: subscript ranges are statically bound and storage allocation is static (before run-time)

- Advantage: efficiency (no dynamic allocation)

Stack-dynamic: subscript ranges are dynamically bound and the storage allocation is dynamic (done at run-time)

- Advantage: flexibility (the size of an array need not be known until the array is to be used)

Subscript Binding and Storage Allocation

Fixed heap-dynamic: storage binding is dynamic but fixed after allocation (storage is allocated from heap, not stack)

- Advantage: space efficiency

Heap-dynamic: binding of subscript ranges and storage allocation is dynamic and can change any number of times

- Advantage: flexibility (arrays can grow or shrink during program execution)

Subscript Binding and Storage Allocation

C examples:

```
int foo (int v) {
   int y[7]; /* fixed stack dyanmic array */
   int w[v]; /* stack dynamic array */
   int f[] = (int[])malloc(30);
                  /* fixed heap dynamic array */
}
```

C# `ArrayList` class provides fixed heap-dynamic

Perl, JavaScript, Python, and Ruby support heap-dynamic arrays

Array Implementations

Access function maps subscript expressions to an address in the array

Access function for single-dimensioned arrays:

```
address(list[k]) =
            address(list[lower_bound])
          + ((k-lower_bound) * element_size)
```

Array Implementations

Two layout strategies:

- Contiguous elements
- Row pointers

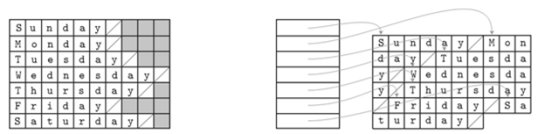
Row pointers

- an option in C
- allows rows to be put anywhere - nice for big arrays
- nice for arrays with variable length rows
  e.g. an array of strings
- requires extra space for the pointers

Array Implementations

```
char days[][10] = {
    "Sunday", "Monday", "Tuesday",
    "Wednesday", "Thursday",
    "Friday", "Saturday"
};
...
days[2][3] == 's';  /* in Tuesday */
```

```
char *days[] = {
    "Sunday", "Monday", "Tuesday",
    "Wednesday", "Thursday",
    "Friday", "Saturday"
};
...
days[2][3] == 's';  /* in Tuesday */
```



Accessing Multi-dimensioned Arrays

Two ways:

- Row major order
- Column major order – just Fortran