CS 350: Programming Language Design

Lecture 11

Array Types

Arrays are homogeneous aggregate of data elements in which an individual element is identified by its position in the aggregate, relative to the first element.

Design Issues:

What types are legal for subscripts

Are subscripting expressions in element references range checked?

When are subscript ranges bound?

When does allocation occur?

Are ragged or rectangular multidimensional arrays allowed, or both?

What is the maximum number of subscripts?

Are any kind of slices supported?

Array Indexing

Indexing is mapping from indices to elements

Array\_name

Index Syntax

* Fortran and Ada use parentheses
  + Ada explicitly uses parentheses to show uniformity between array references and function calls because both are mappings
* Most languages use brackets
* In some languages, the lower bound of the subscript range is implicit
* Perl Allows negative subscripts
  + Offset from the end of the array

Array Categories

* When are the type/ranges bound?
* Static: subscript ranges are statically bound and storage allocation is static (before run-time)
  + Advantage: efficiency (no dyn. Alloc.)
  + C and C++ static arrays
* Fixed stack-dynamic: subscript ranges are statically bound, but the allocation is done at declaration/elaboration time during execution
  + Advantage: space efficiency
  + C and C++ local arrays declared in functions
* Fixed heap-dynamic: subscript ranges are statically bound, storage binding is dynamic but fixed after allocation
  + Binding is done when requested and storage is allocated from heap, not stack
  + Advantage: flexibility, allocated space fits the problem
* Heap-dynamic: binding of subscript ranges and storage allocation is dynamic and can change any number of times
  + Flexibility: arrays can grow or shrink during program execution
  + Java’s ArrayList
* Heterogeneous arrays have a variety of types as elements but are only supported in JS and Python among others.
* Slices
  + A slice is a substructure of an array; nothing more than a referencing mechanism
  + Only useful for languages with array operations
* Python and Ruby offer slices
  + Python uses ranges within subscripts
  + Ruby supports them with slice method
* Rectangular and Jagged Arrays
  + Rectangular arrays are m-D arrays with the same number of columns for each row.
  + Jagged arrays or matrices are arrays with differing number of columns for each row.
* Associative arrays
  + An array which is an unordered collection of data elements that are indexed by an equal number of values called keys (think maps or sets)
  + User-defined keys must be stored
* Design Issues:
  + What is the form of references to elements?
  + Is the size static or dynamic?
  + Built-in Perl, Python, Ruby, and Swift.
* Records
  + A record is a possibly heterogeneous aggregate of data elements in which the individual elements are identified by names
  + Issues:
    - What is the syntactic form of references to the field?
    - Can records be nested?
  + In comparison with arrays, access to array elements is slower than access to record fields
    - Subscripts are dynamic, field names are static
      * Dynamic subscripts could be used with record field access, but it would disallow type checking and it would be much slower
  + Tuples
    - A tuple is a data type that is similar to a record, except that the elements are not named.
    - Used in Python, ML, and F# to allow functions to return multiple values
  + Python
    - Closely related to its lists, but immutable
    - Create with a tuple literal
    - myTuple = (3, 5.8, ‘apple’)
    - Referenced with subscripts (start at 1)
    - Concatenation with + and deleted with “del”
* Pointer and Reference Types
  + A pointer type variable has a range of values that consists of memory addresses and a special value, nil.
  + Provide the power of indirect addressing
  + Provide a way to manage dynamic memory
    - Storage is allocated from the heap
  + Design issues:
    - What are the scope of a pointer variable?
    - What is the lifetime of a heap-dynamic variable?
    - Are pointers restricted as to the type of value to which they can point?
    - Are pointers used for dynamic storage management, indirect addressing, or both?
    - Should the language support pointer types, reference types
* Pointer Operations
  + Two fundamental operations
  + Assignment is used to set a pointer variable’s value to some useful address
  + Dereferencing yields the value storage at the location represented by the pointer’s value
  + Dereferencing can be explicit or implicit
  + C++ uses an explicit operation via \*
* Pointers are dangerous
  + Dangling pointers
    - A pointer points to a heap-dynamic variable that has been deallocated
  + Lost heap-dynamic variable
    - An allocated heap-dynamic variable that is no longer accessible to the user program (called garbage)
    - Pointer p1 is set to point to a newly created heap-dynamic variables
    - Pointer p1 is later set to point to another newly created heap-dynamic variable
    - The process of losing heap-dynamic variables is called memory leakage
  + Aliasing
    - Using a two identifiers to reference a single data entity.
* Pointers in C and C++
  + Handle with Care
  + Pointers can point at any variable regardless of when or where it was allocated
  + Used for dynamic storage management and addressing
  + Pointer arithmetic is possible
  + Explicit dereferencing and address-of operators
  + Domain type need not be fixed (void \*)
  + Void ( can point to any type and can be type checked (can’t be dereferenced)
* Reference Types
  + C++ includes a special kind of pointer type called a reference type that is used primarily for formal parameters
    - Advantages of both pass-by-ref and by-val
  + Java extends C++’s reference variables and allows them to replace pointers entirely
    - References are references to objects, rather than being addresses
  + C# includes both the references of Java and the pointers of C++
  + What of ruby?