

CSE 3302 Programming Languages


Data Types (cont.)

Chengkai Li
Fall 2007

Lecture 8 – Data Types, Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

1

Subset




- $U = \{v \mid v \text{ satisfies certain conditions and } v \in V\}$
- Ada subtype
- Example 1
 - type Digit_Type is range 0..9;
 - subtype IntDigit_Type is integer range 0..9;

Lecture 8 – Data Types, Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

2

subtype in Ada



- Example 2

```

type Disc is (IsInt, IsReal);
type IntOrReal (which: Disc) is
  record
    case which is
      when IsInt => i: integer;
      when IsReal => r: float;
    end case;
  end record;


subtype IRInt is IntOrReal(IsInt);
subtype IRReal is IntOrReal(IsReal);

x: IRReal := 2.3;
  
```

Lecture 8 – Data Types, Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

3

Powerset




- $P(U) = \{U' \mid U' \subseteq U\}$
- Example: Pascal
 - set of** <ordinal type>
 - var S: **set of** 1.. 10;
 - var S: 1.. 10;
 - What's the difference?
- The order isn't significant though
unordered, values are distinct

Lecture 8 – Data Types, Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

4

set of in Pascal



```

var S, T: set of 1 .. 10;
S := [1, 2, 3, 5, 7];
T := [1 .. 6];
  
```

Set operations can be performed on the variables.
What are these?


- $T := S * T;$
- If $T = [1, 3, 5]$ then ...;
- $x = 3;$ if x in S then ...;
- if $S \leq T$ then ...;

- $\cap (*)$ $\cup (+)$ $- (-)$ $= (=)$ $\neq (< >)$
- $\supset (>)$ $\supseteq (>=)$ $\subset (<)$ $\subseteq (<=)$ \in (in)

Lecture 8 – Data Types, Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

5

implementations



```

type
  students = (Alice, Bob, Charlie, Dave, Eve, ...);
var
  CSE3302, CSE3310, whotookboth: set of students;
  CSE3302 := [ Alice, Bob, Dave ];
  CSE3310 := [ Alice, Charlie, Dave, Eve ];
  whotookboth := CSE3302 * CSE3310;
  
```

	Alice	Bob	Charlie	Dave	Eve	...
CSE3302	1	1	0	1	0	...
CSE3310	1	0	1	1	1	...
Whotookboth	1	0	0	1	0	...

How about other operations? (membership, intersection, union, count, subsumption, ...)

Lecture 8 – Data Types, Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

6

bit operations for set

Implementation can be very efficient

- bit-and (&), bit-or (|), bit-negation (~), bit-xor
- Hardware support
- Encoding schemes

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

7

Alternatives?

- Having Set type is convenient and efficient.
- If not present, can we use other types as alternatives?

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

8

Array and Functions

$f: U \rightarrow V$

index type **component type**

- [0, ...] (C/C++/Java)
- Ordinal type (Ada/Pascal)

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

9

C/C++

Allocated on stack, size statically specified.

```
typedef int TenIntArray [10];
typedef int IntArray [];
```

```
TenIntArray x;
int y[5];
int z[]={1,2,3,4};
IntArray w={1,2};
IntArray w; //illegal
int n = ... //from user input
int a[n]; //illegal
```

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

10

Java

Allocated on heap, size dynamically specified;
Size can be obtained by .length

```
int n = ... //from user input
int [] x = new int [n];
System.out.println(x.length);
```

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

11

Ada

size dynamically specified;
Set of subscripts

```
type IntToInt is array(integer range <>) of integer;
```

```
get(n); //from user input
x: IntToInt(1..n);
for i in x'range loop
    put(x(i));
end loop;
```

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

12

Four Categories of Arrays

Based on **subscript binding** and **location binding time**

- *Static* - range of subscripts (static), location (static)
 - e.g. FORTRAN 77
 - *Advantage*: execution efficiency
- *Fixed stack dynamic* - range of subscripts (static), location (dynamic).
 - e.g. Pascal, C, C++
 - *Advantage*: space efficiency

Courtesy of Charles Nicholas at UMBC

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

13

Four Categories of Arrays

- *Stack-dynamic* - range of subscripts (dynamic), location (dynamic), immutable once bound
 - e.g. Ada
 - *Advantage*: flexibility
- *Heap-dynamic* – range of subscript (dynamic), location (dynamic), can be changed
 - e.g. FORTRAN 90
 - *Advantage*: maximum flexibility

Courtesy of Charles Nicholas at UMBC

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

14

Multi-dimensional arrays

- C/C++

```
int x[10][20];
```

- Java

```
int [][] x = new int [10][20];
```

- Ada

These two are different

```
type Matrix_Type is array(1..10, -10..10) of integer;
x(i,j);
```

```
type Matrix_Type is array(1..10) of array (-10..10) of integer;
x(i)(j);
```

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

15

Storage

- row-major form

```
x[1,-10],x[1,-9],...,x[1,10],x[2,-10],...,x[2,10],x[3,-10],...
```

- column-major form

```
x[1,-10],x[2,-10],...,x[10,-10],x[1,-9],...,x[10,-9],x[1,-8],...
```

- C/C++

```
int array_max(int a[][20][10], int size)
```

- Java

```
int array_max(int [][][]a)
```

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

16

FORTRAN 90

```
INTEGER, ALLOCATABLE, ARRAY (:,:) :: MAT
(Declares MAT to be a dynamic 2-dim array)
```

```
ALLOCATE (MAT (10, NUMBER_OF_COLS))
(Allocates MAT to have 10 rows and NUMBER_OF_COLS columns)
```

```
DEALLOCATE MAT
(Deallocates MAT's storage)
```

Courtesy of Charles Nicholas at UMBC

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

17

Array Operations

- Access individual elements
- The whole array
 - Pascal: A := B
 - FORTRAN: A+B
 - Ada: Array concatenation A&B
 - FORTRAN 90: matrix multiplication, transpose

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

18

Functions



- C:


```
int incfn(int x) { return x+1; }
typedef int (*IntFunc)(int); // why *?
IntFunc inc = incfn;
```
- ML:


```
type IntFunc = int -> int;
val inc: IntFunc = fn x => x+1;
```

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

19

Vector, List, Sequence



Functional languages:

- Vectors: like arrays, more flexibility, especially dynamic resizability.
- Lists: like vectors, can only be accessed by counting down from the first element.
- Sequences: typically (potentially) infinite.

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

20

Pointer



- A *pointer type* is a type in which the range of values consists of memory addresses and a special value, nil (or null)
- *Advantages:*
 - Addressing flexibility (address arithmetic, explicit dereferencing and address-of, domain type not fixed (void*))
 - Dynamic storage management
 - Recursive data structures
 - E.g., linked list


```
struct CharListNode
{ char data;
  struct CharListNode* next;
};
typedef struct CharListNode* CharList;
```

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

21

Problems with Pointers



- **Alias (with side-effect)**

```
int *a, *b;
a=(int *) malloc(sizeof(int));
*a=2;
b=(int *) malloc(sizeof(int));
*b=3;
b=a;
*b=4;
printf("%d\n", *a);
```

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

22

Problems with Pointers



- **Dangling pointers (dangerous)**

```
int *a, *b;
a= (int *) malloc(sizeof(int));
b=a;
free(a);
*b=2;
printf("%d\n", *b);
```

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

23

Problems with Pointers



- **Garbages (waste of memory)**

memory leakage

```
int *a;
a = (int *) malloc(sizeof(int));
*a=2;
a = (int *) malloc(sizeof(int));
```

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

24

Type System



- **Type Constructors:**
 - Build new data types upon simple data types
 - **Type Checking:** The translator checks if data types are used correctly.
 - **Type Inference:** Infer the type of an expression, whose data type is not given explicitly.
e.g., x/y
 - **Type Equivalence:** Compare two types, decide if they are the same.
e.g., x/y and z
 - **Type Compatibility:** Can we use a value of type A in a place that expects type B?
- Nontrivial with user-defined types and anonymous types

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

25

Strongly-Typed Languages



- **Strongly-typed:** (Ada, ML, Haskell, Java, Pascal)
 - Most data type errors detected at translation time
 - A few checked during execution and runtime error reported (e.g., subscript out of array bounds).
- **Pros:**
 - No data-corrupting errors can occur during execution. (I.e., no unsafe program can cause data errors.)
 - Efficiency (in translation and execution.)
 - Security/reliability
- **Cons:**
 - May reject safe programs (i.e., legal programs is a subset of safe programs.)
 - Burden on programs, may often need to provide explicit type information.

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

26

Weakly-typed and untyped languages



- **Weakly-typed:** C/C++
 - e.g., interoperability of integers, pointers, arrays.
- **Untyped (dynamically typed) languages:** scheme, smalltalk, perl
 - Doesn't necessarily result in data errors.
 - All type checking performed at execution time.
 - May produce runtime errors too frequently.

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

27

Security vs. flexibility



- **Strongly-typed :**
 - No data errors caused by unsafe programs.
 - Maximum restrictiveness, static type checking, illegal safe programs, large amount of type information supplied by programmers.
- **Untyped:**
 - Runtime errors, no data-corruptions. Legal unsafe programs.
 - reduce the amount of type information the programmer must supply.

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

28

Security vs. flexibility



- **Strongly-typed :**
- A type system tries to maximize both flexibility *and* security, where flexibility means: reduce the number of safe illegal programs & reduce the amount of type information the programmer must supply.
- Flexibility, no explicit typing or static type checking
vs.
- Maximum restrictiveness, static type checking

Lecture 8 – Data Types, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

29