Data Objects

Data objects

Scalar data objects:

- Numeric (Integers, Real)
- Booleans
- Characters
- Enumerations

Composite objects:

- String
- Pointer

Structured objects:

- Arrays
- Records
- Lists
- Sets

Abstract data types:

- Classes

Active Objects:

- Tasks
- Processes

Copyright©Prentice Hall, 2000

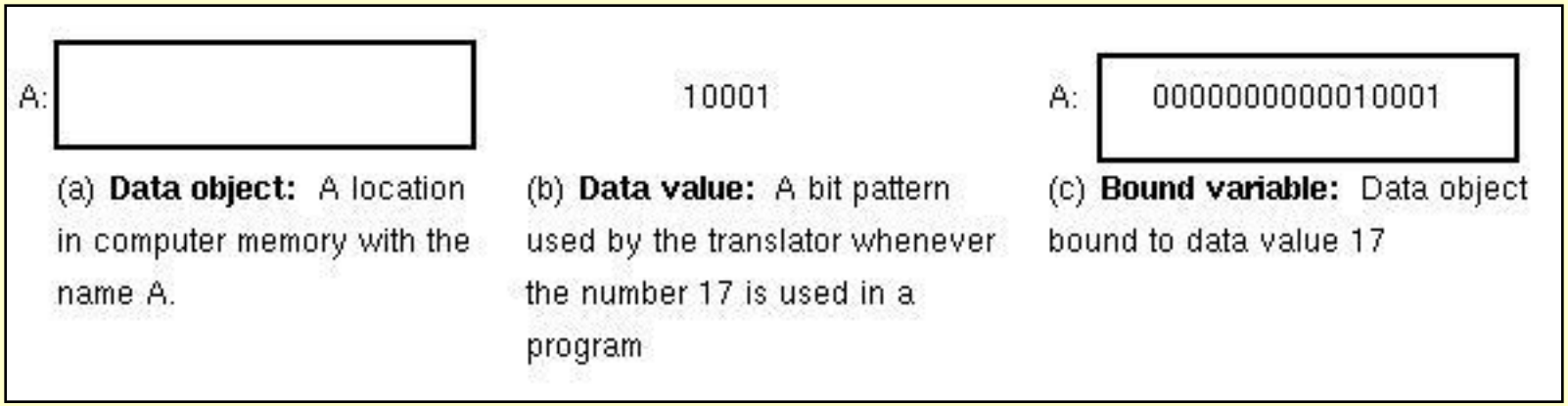
Binding of data objects

A compiler creates two classes of objects:

- Memory locations
- Numeric values

A variable is a binding of a name to a memory location:

- Contents of the location may change



(a) **Data object:** A location in computer memory with the name A.

(b) **Data value:** A bit pattern used by the translator whenever the number 17 is used in a program

(c) **Bound variable:** Data object bound to data value 17

Copyright©Prentice Hall, 2000

Data types

Each data object has a type:

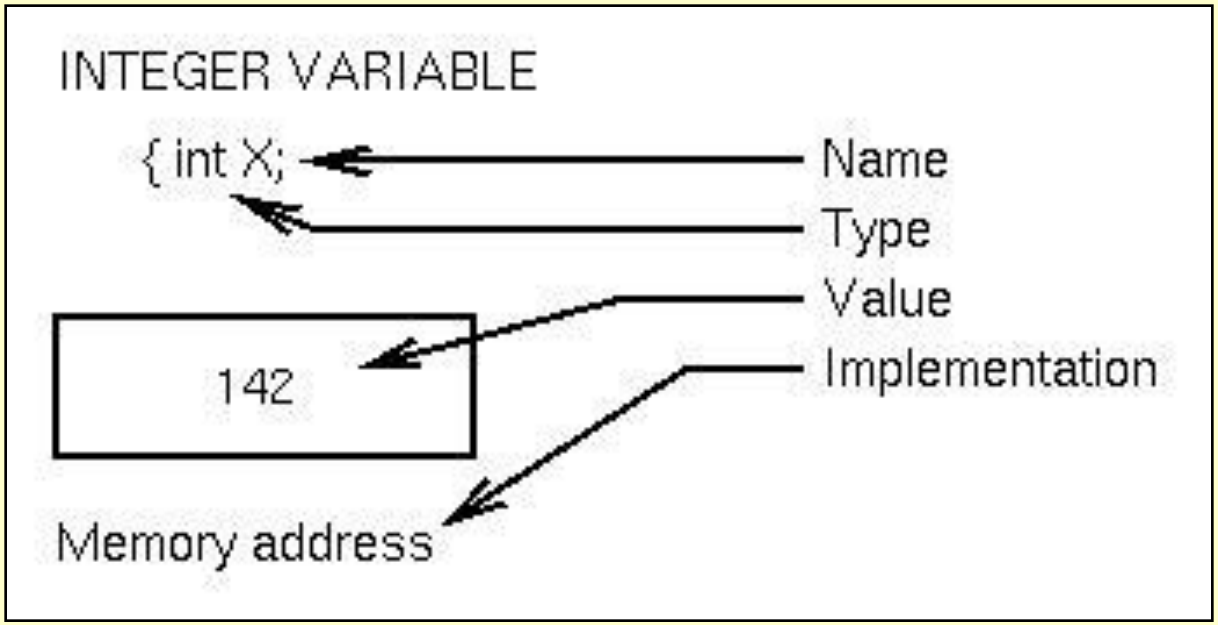
Values: for objects of that type

Operations: for objects of that type

Implementation: (Storage representation) for objects of that type

Attributes: (e.g., name) for objects of that type

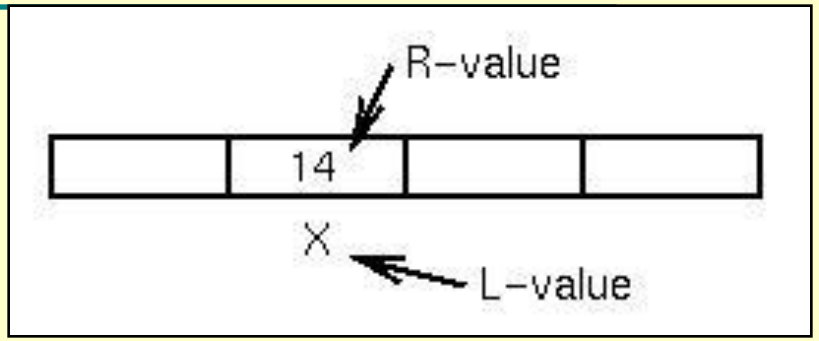
Signature: (of operation f): f: type x type → type



Copyright©Prentice Hall, 2000

L-value and R-value

Location for an object is its L-value. Contents of that location is its R-value.



Where did names L-value and R-value come from?

Consider executing: A = B + C;

1. Pick up contents of location B
2. Add contents of location C
3. Store result into address A.

For each named object, its position on the right-hand-side of the assignment operator (=) is a *content-of* access, and its position on the left-hand-side of the assignment operator is an *address-of* access.

- address-of then is an L-value
- contents-of then is an R-value
- Value, by itself, generally means R-value

Copyright©Prentice Hall, 2000

Subtypes

A is a subtype of B if every value of A is a value of B.

Note: In C almost everything is a subtype of integer.

Conversion between types:

Given 2 variables A and B, when is A:=B legal?

Explicit: All conversion between different types must be specified

Implicit: Some conversions between different types implied by language definition

Copyright©Prentice Hall, 2000

Coersion examples

Examples in Pascal:

```
   var A: real;
   B: integer;
```

A := B - Implicit, called a coersion - an automatic conversion from one type to another

A := B is called a widening since the type of A has more values than B.

B := A (if it were allowed) would be called a narrowing since B has fewer values than A. Information could be lost in this case.

In most languages widening coersions are usually allowed;

narrowing coersions must be explicit:

```
   B := round(A); Go to integer nearest A
   B := trunc(A); Delete fractional part of A
```


Copyright©Prentice Hall, 2000

Integer numeric data

Integers:

Binary representation in 2's complement arithmetic

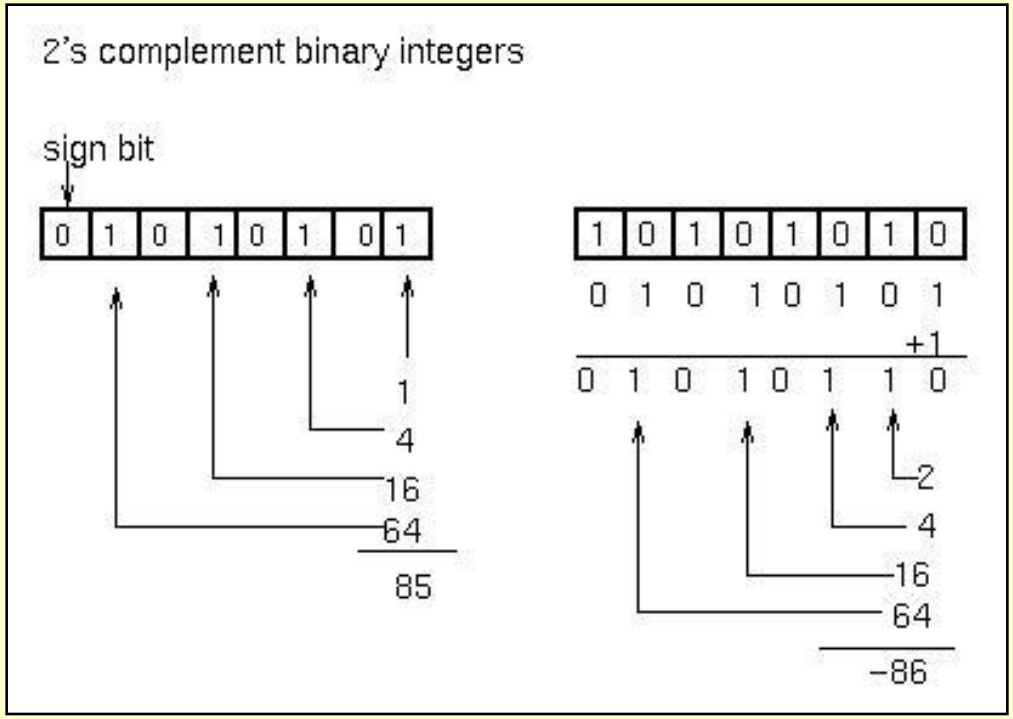
For 32-bit words:

Maximum value:

$2^{31}-1$

Minimum value:

-2^{31}

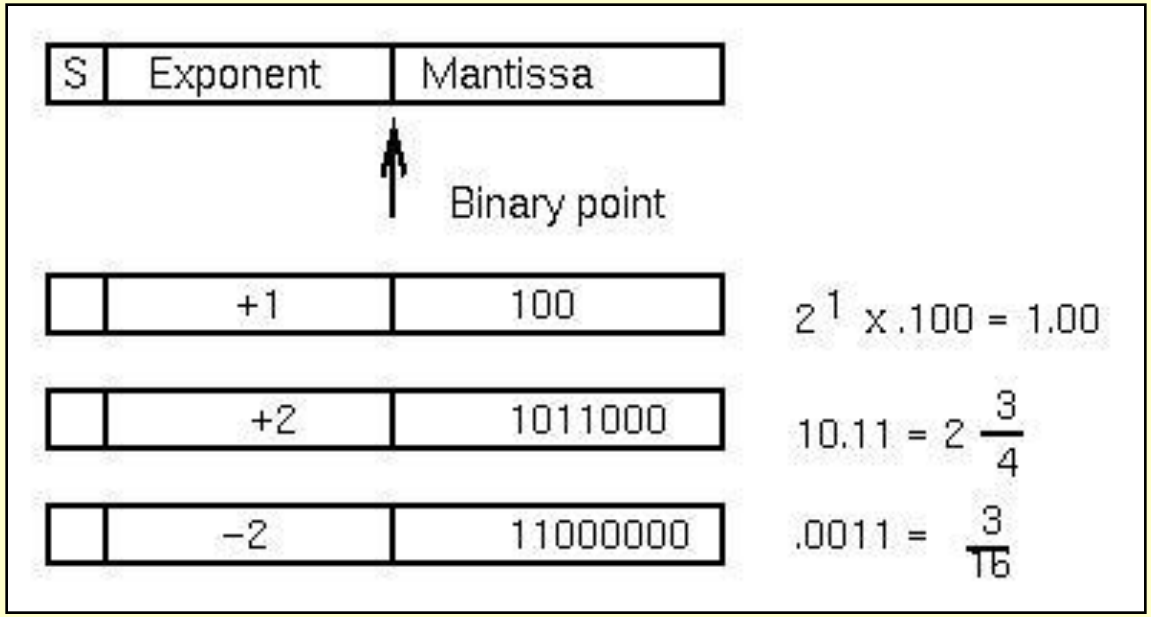


Positive values Negative values

Copyright©Prentice Hall, 2000

Real numeric data

Float (real): hardware representations



Exponents usually *biased*

e.g., if 8 bits (256 values) +128 added to exponent

- so exponent of 128 = 128-128 = 0 is true exponent
- so exponent of 129 = 129-128 = 1 is true exponent
- so exponent of 120 = 120-128 = -8 is true exponent

Copyright©Prentice Hall, 2000

IEEE floating point format

IEEE standard 754 specifies both a 32- and 64-bit standard.

Numbers consist of three fields:

S: a one-bit sign field. 0 is positive.

E: an exponent in excess-127 notation. Values (8 bits) range from 0 to 255, corresponding to exponents of 2 that range from -127 to 128.

M: a mantissa of 23 bits. Since the first bit of the mantissa in a normalized number is always 1, it can be omitted and inserted automatically by the hardware, yielding an extra 24th bit of precision.

Copyright©Prentice Hall, 2000

Decoding IEEE format

Given E, and M, the value of the representation is:

Parameters	Value
E=255 and M ≠ 0	An invalid number
E=255 and M = 0	∞
0<E<255	$2^{\{E-127\}}(1.M)$
E=0 and M ≠ 0	$2^{\{-126\}}.M$
E=0 and M=0	0

Copyright©Prentice Hall, 2000

Example floating point numbers

$+1 = 2^0 * 1 = 2^{\{127-127\}} * (1).0$ (binary) 0 01111111 000000...

$+1.5 = 2^0 * 1.5 = 2^{\{127-127\}} * (1).1$ (binary) 0 01111111 100000...

$-5 = -2^2 * 1.25 = 2^{\{129-127\}} * (1).01$ (binary) 1 10000001 010000...

- This gives a range from 10^{-38} to 10^{38}.
- In 64-bit format,the exponent is extended to 11 bits giving a range from -1022 to +1023, yielding numbers in the range 10^{-308} to 10^{308}.

Copyright©Prentice Hall, 2000

Other numeric data

Short integers (C) - 16 bit, 8 bit

Long integers (C) - 64 bit

Boolean or logical - 1 bit with value true or false

Byte - 8 bits

Character - Single 8-bit byte - 256 characters

- ASCII is a 7 bit 128 character code

In C, a char variable is simply 8-bit integer numeric data

Copyright©Prentice Hall, 2000

Enumerations

```
typedef enum thing {A, B, C, D } NewType;
```

- Implemented as small integers with values:

```
    A = 0, B = 1, C = 2, D = 3
```

- NewType X, Y, Z;

```
  X = A
```

Why not simply write: X=0 instead of X=A?

- Readability
- Error detection

Example:

```
enum { fresh, soph, junior, senior} ClassLevel;
enum { old, new } BreadStatus;

BreadStatus = fresh;  An error which can be detected
```


Copyright©Prentice Hall, 2000

Declaring decimal data

Fixed decimal in PL/I and COBOL (For financial applications)

```
DECLARE X FIXED DECIMAL(p,q);
    p = number of decimal digits
    q = number of fractional digits
```

Example of PL/I fixed decimal:

```
DECLARE X FIXED DECIMAL (5,3),
Y FIXED DECIMAL (6,2),
Z FIXED DECIMAL (6,1);
X = 12.345;
Y = 9876.54;
```


Copyright©Prentice Hall, 2000

Using decimal data

What is Z=X+Y?:

By hand you would line up decimal points and add:

```
        0012.345
        9876.540
        9888.885 = FIXED DECIMAL(8,3)
```

p=8 since adding two 4 digit numbers can give 5 digit result and need 3 places for fractional part.

p=8 and q=3 is known before addition

- Known during compilation - No runtime testing needed.

Copyright©Prentice Hall, 2000

Implementing decimal data

Algorithm:

1. Store each number as an integer (12345, 987654)

Compiler knows scale factor (S=3 for X, S=2 for Y).

True value printed by dividing stored integer by 10^S

2. To add, align decimal point. Adjust S by 1 by multiplying by 10.

3. 10*Y+X = 9876540 + 12345 = 9888885, Compiler knows S=3

4. S=1 for Z, so need to adjust S of addition by 2; divide by 10^2 (98888)

5. Store 98888 into Z. Compiler knows S=1

Note: S never appears in memory, and there is no loss of accuracy by storing data as integers.

Copyright©Prentice Hall, 2000

Composite data

Character Strings: Primitive object made up of more primitive character data.

Fixed length:

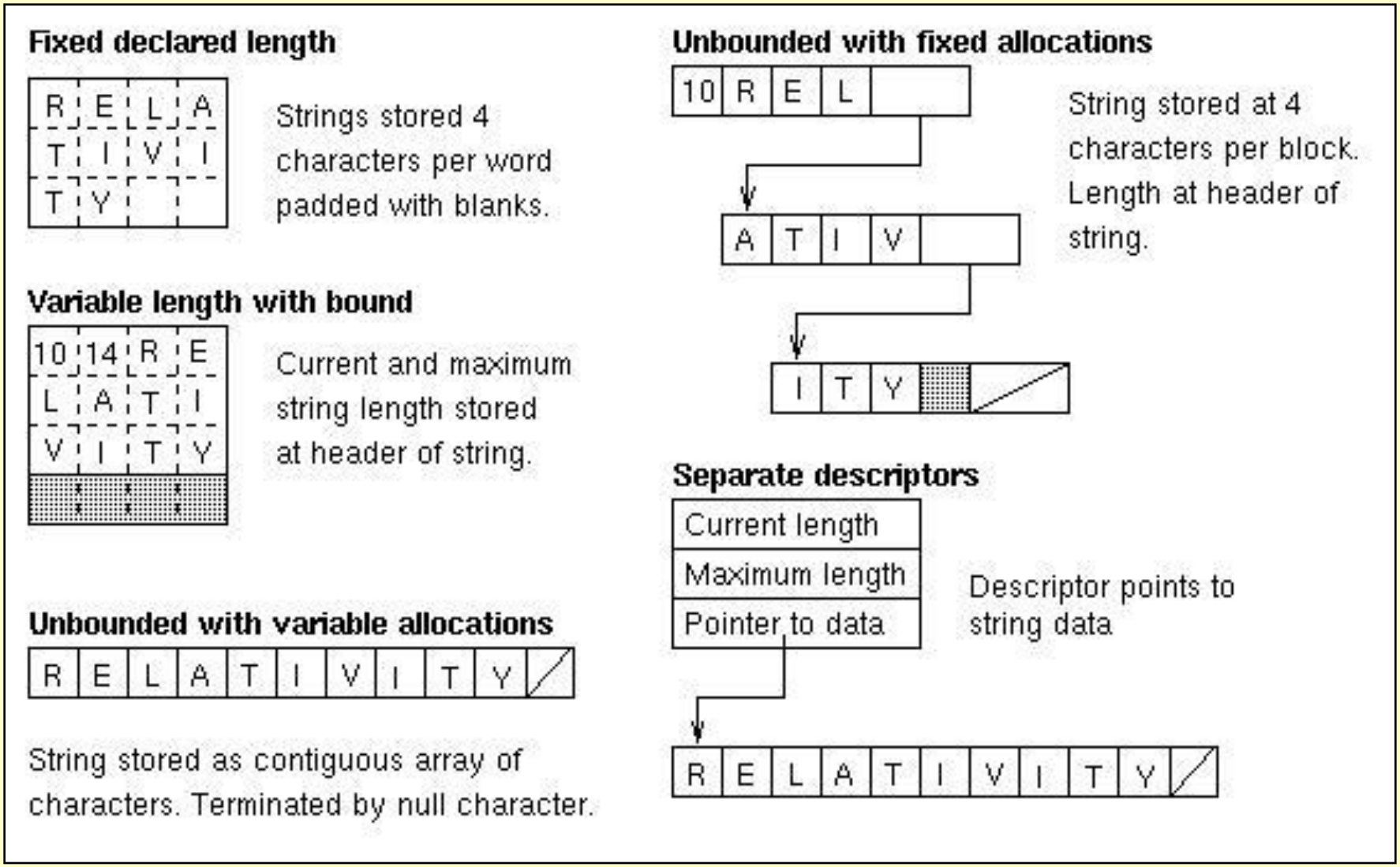
- char A(10) - C
- DCL B CHAR(10) - PL/I
- var C packed array [1..10] of char - Pascal

Variable length:

- DCL D CHAR(20) VARYING - PL/I - 0 to 20 characters
- E = “ABC” - SNOBOL4 - any size, dynamic
- F = `ABCDEFG\0' - C - any size, programmer defined

Copyright©Prentice Hall, 2000

String implementations



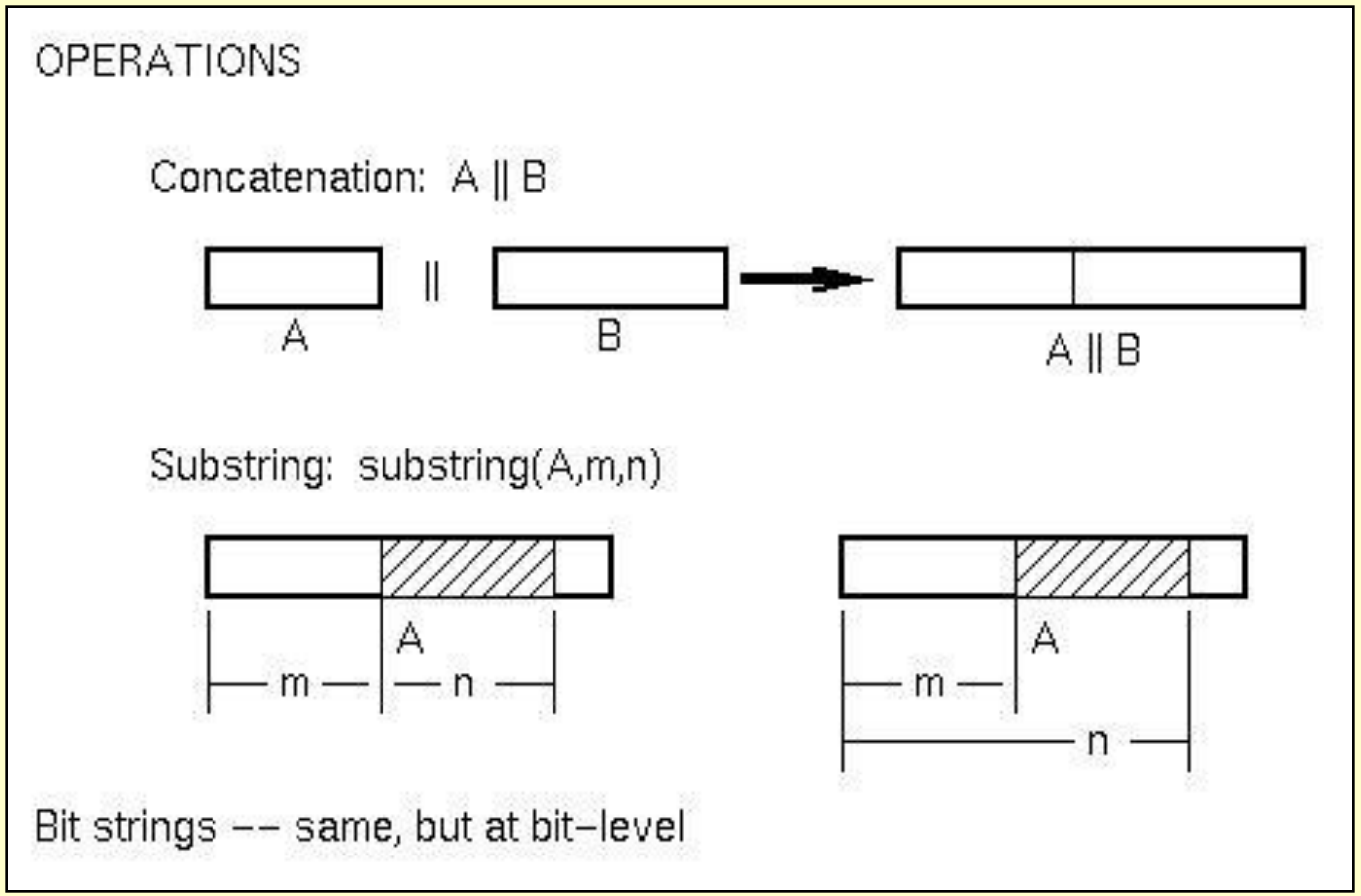
Copyright©Prentice Hall, 2000

String operations

In C, arrays and character strings are the same.

Implementation:

L-value(A[I]) = L-value(A[0]) + I



Copyright©Prentice Hall, 2000

Pointer data

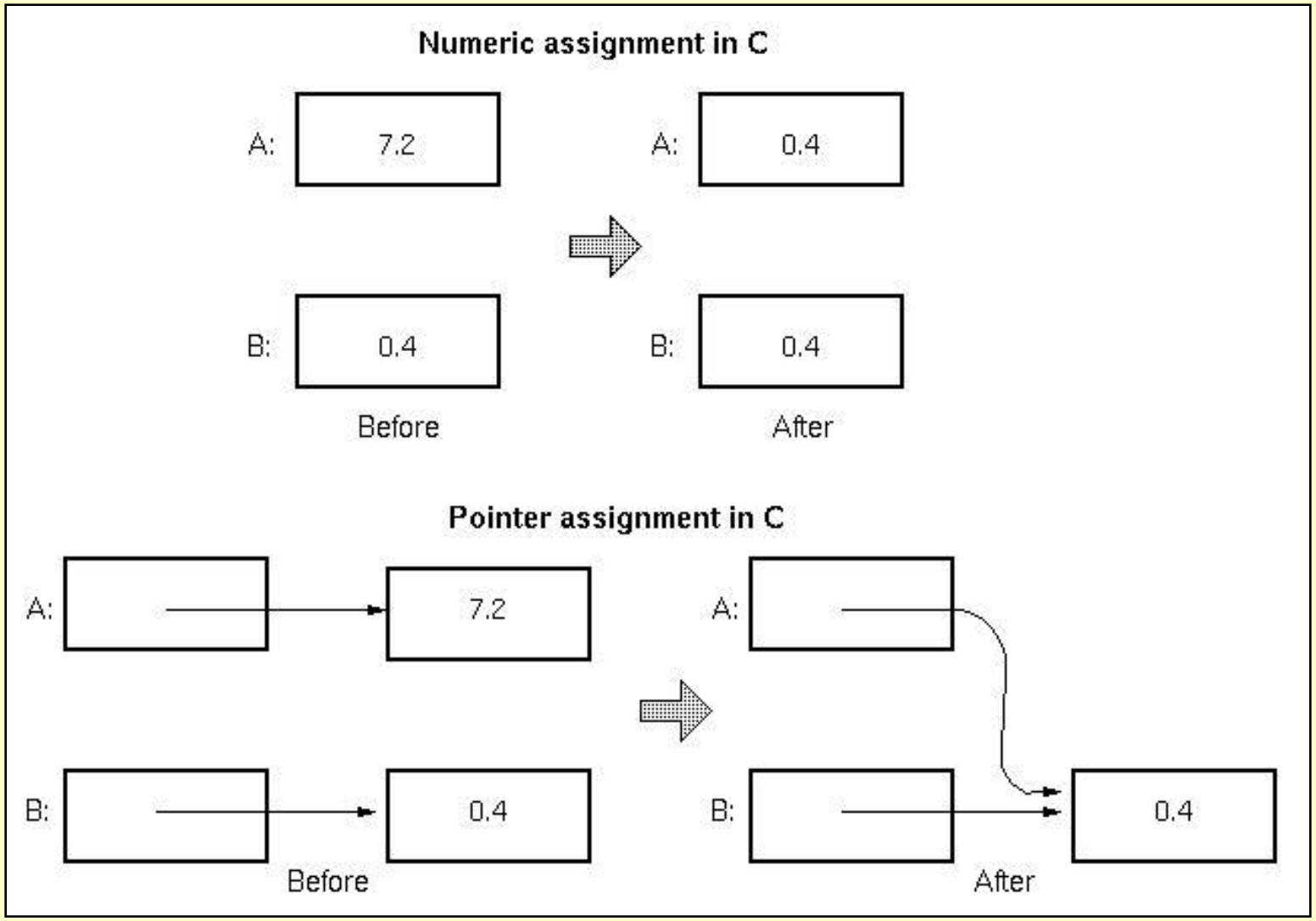
Use of pointers to create arbitrary data structures

Each pointer can point to an object of another data structure

In general a very error prone construct and should be avoided

Copyright©Prentice Hall, 2000

Pointer aliasing



Copyright©Prentice Hall, 2000