Binding and Data Types

Compiled from
"Programming Languages: Design and Implementation"
Author: T. W. Pratt
Chapter: Elementary Data Types

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

Binding

- Binding = establishing a relationship (between two things)
  - Also “making a choice from a set of available choices”
- X = 5
  - Binding the value 5 to the variable X

Various possible bindings

- X = X + 10
  - Set of available types for X
  - The particular data type chosen for X
  - Set of possible values for X
  - The particular value for X
  - Representation of “10” (how many bits, order of bits, etc)
  - Meaning of + , =
  - …

Various possible binding times

- Language Design time
  - Some things are decided by language designers, for example the syntax

- Language Implementation time
  - Some things are decided while writing the compiler or interpreter, for example, how to evaluate expressions is decided by compiler

Various possible binding times

- Translation time (Compilation or Interpretation)
  - By the programmer
    - Names of variables, …
  - By the compiler
    - Actual in memory order of various variables
- Linking time
  - Code of printf() with the call to printf()
- Loading Time:
  - Actual memory location of the program and the data objects
- Run time
  - Value of a variable

Various possible bindings in C and Python

- X = X + 10
  - Set of available types for X: lang definition time, lang definition time
  - The particular data type chosen for X: translation time, run time
  - Set of possible values for X: translation time, lang. Definition time
  - The particular value for X: run time, run time
  - Representation of “10” (how many bits, order of bits, etc): translation time, translation time,
  - Meaning of + , = translation time, run time

Importance of Binding Times

- Depending on “when” the decision of binding is taken, the languages differ from each radically
- Languages in which most bindings happen during or before translation are called “early binding” languages
- Languages in which most bindings happen after translation are called “late binding” languages
- “Early Binding” languages are normally compiled and “Late binding” languages are normally interpreted

Importance of Binding Times

- In python the type of a variable is determined at Run time, in C at translation time.
  - So python is “late binding” while C is “early binding” language.
- Python is interpreted, while C is compiled.
- Can we write a compiler for python? If yes, then why it is not written ?

Cost of “late binding” with compilation

- Need to generate m/c code that will remember the type of each variable
- Need to generate m/c code which will check if an operation is supported on that type
- Type determination code itself will have some “cost” , slowing the program
- The generated object code will be bulky also (more disk space)
- Writing the compiler, which generated complex code like this, will also be a difficult task

Cost of "early binding" with interpretation

- The interpreter itself is a program which is always running
  - So your program and the interpreter are both running
  - So interpretation is inherently slow
- Early binding means, most of the meanings are known before translation
  - So why not take care of those during translation and just run the program's code while execution
  - Interpretation will be an additional cost now

What does the compiler do with the data type ?

Some definitions from the literature

Various data types that languages provide

What is done by the hardware and what is done by the compiler

Let's start with some Definitions

- A data object is a container for data.
  - programmer defined
    - variables, constants, arrays, files
    - can be defined and manipulated explicitly
  - system defined
    - stack, activation records, file buffers. free-space lists
    - are not directly accessible to the programmer
    - THERE IS SOME ADDITIONAL CODE GENERATED BY THE COMPILER !!
- Elementary data objects are manipulated as a unit.
- Data structures are aggregates made up of other data objects.

Data Object Properties

- Attributes are generally fixed for the lifetime of the object, bindings may change.
  - E.g. student has attributes age, year. Binding of age to “21” and year to “sy” or “ty” keeps changing.
  - Some attributes are used only by the translator; some are stored in a descriptor (dope vector) for use during execution.

Data Object Properties

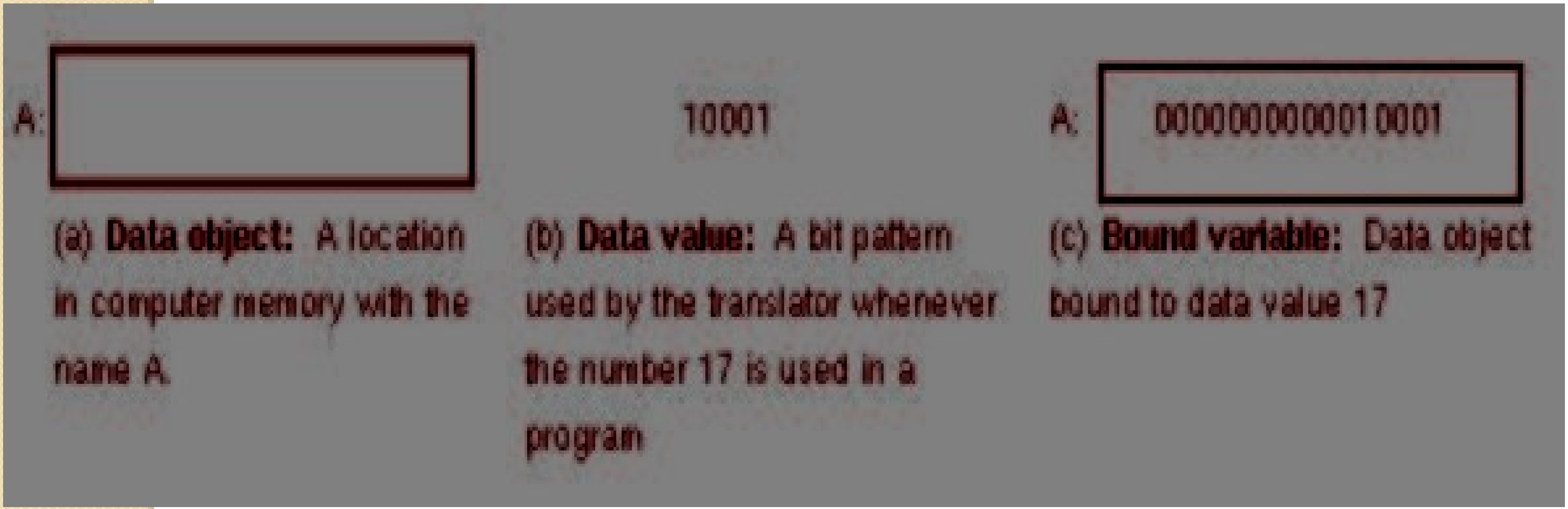
- lifetime (extent) – the period of time during program execution during which the data object exists
- type – determines the set of values the object can have
- storage location
- value
- name – used to reference the object, an object may have multiple names
- component – binding a data object to another object of which it is a part

Data objects

- Scalar data objects:
  - Numeric (Integers, Real)
  - Arrays
  - Booleans
  - Characters
  - Enumerations
  - Pointer (what it points to is often composite)
- Composite objects
  - String
  - File
- Structured objects
  - Records
  - Lists
  - Sets
- Abstract data types
  - Classes
- Active Objects
  - Tasks
  - Processes

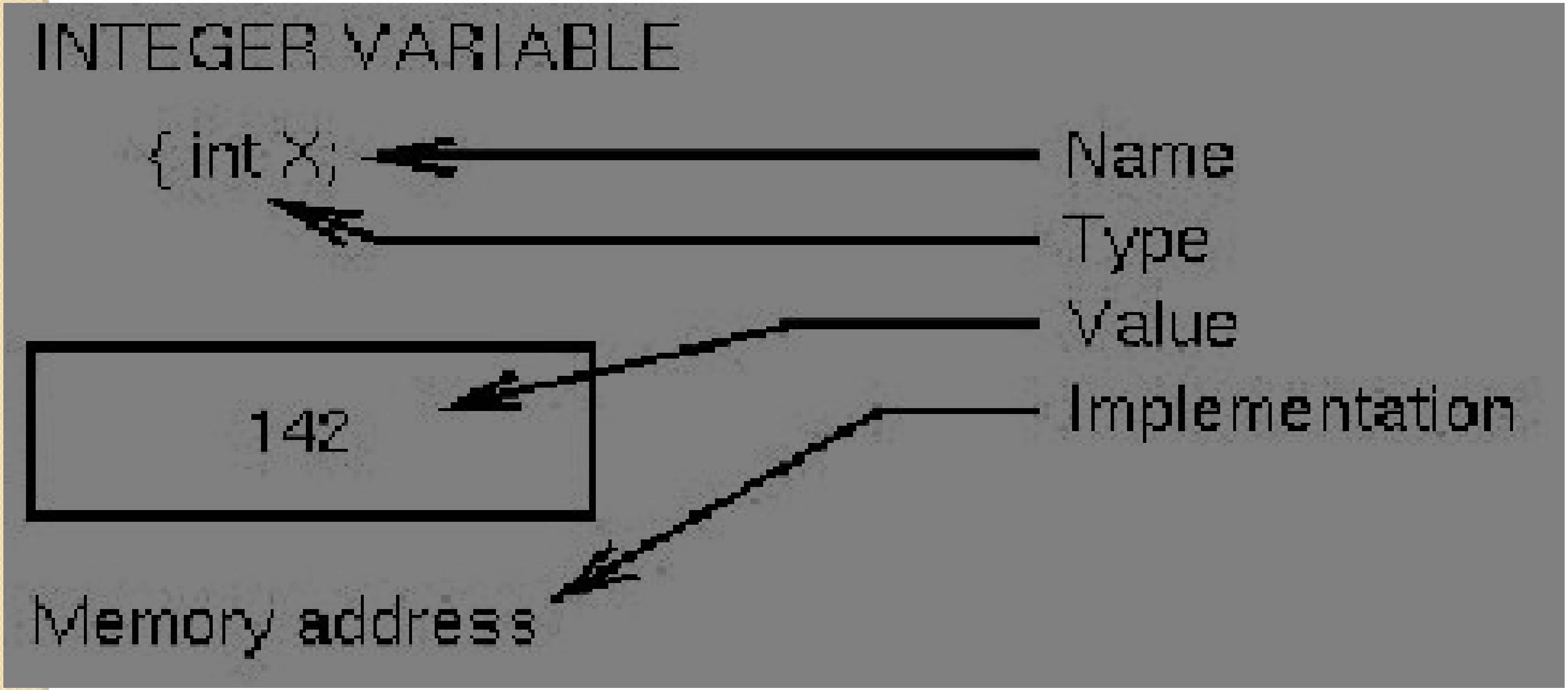
What is a Variable ?

- A variable is a binding of a name to a memory location
  - Contents of the location may change



Variables

- A variable is a data object that is defined and named by the



Constants

- A constant is a name bound to a value
  - a literal constant has a name which is its value
  - a manifest constant is named by the programmer
  - #define MAX 10
  - const int MAX = 10
- Since a constant can't change, its value is known to and can be used by the compiler
  - A constant could be bound at compile time

Data Types

- A data type is a class of data objects along with the operations that can be used to created and manipulate them.
- Primitive types are defined as part of the language specification and implemented with the language. e.g. int, float, etc. in C
- Most modern languages allow the programmer to define new types, e.g. typedef struct stack { .. } stack;
  - Object-oriented languages allow the programmer to define the operations as well.

Data types

- Each data type is defined by the following information:
  - Attributes: (e.g., name, data type)
  - Values: range of allowed values
  - Operations: possible manipulations for the data type
    - Textbook Notation for Signatures of operations: $f : type \times type -> type$
      pow: int x int → float
- A data type is implemented by choosing
  - storage representation (data structure)
  - algorithms for implementing the operations

Specifying Operations

- Operations can be built-in or programmer specified.
- An operation should be a mathematical function
  - should have a unique output for every possible input.

- **Difficulties in specifying operations as mathematical functions**
  1. The result may be undefined for some inputs – x / 0 e.g.
  2. There may be implicit arguments
  3. int x; int f(int y) { return y * x;}
  4. There may be side effects – modification of parameters e.g.
  5. int i; int f(int x) { i = x*x; return i; }
  6. Self Modification: The result may depend on previous calls to the same operation, use of static in C

Implementation of Elementary data types

- Storage
  - Depends on the hardware
  - Usually using hardware primitives
- Attributes
  - Many times determined by compiler, e.g. C
  - Stored as run time descriptor, LISP
- Operations
  - Next slide

Data type implementation

- Implementation approaches for operations
  - use underlying hardware operations
  - `c = a + b (uses hardware add instruction)`
  - write function or procedure
  - `MyInteger add(MyInteger a, MyInteger b) {...}`
  - inline code sequence - replacement of function call with the body of function
  - C++
  - `inline MyInteger add(MyInteger a, MyInteger b)`

Assignment operation

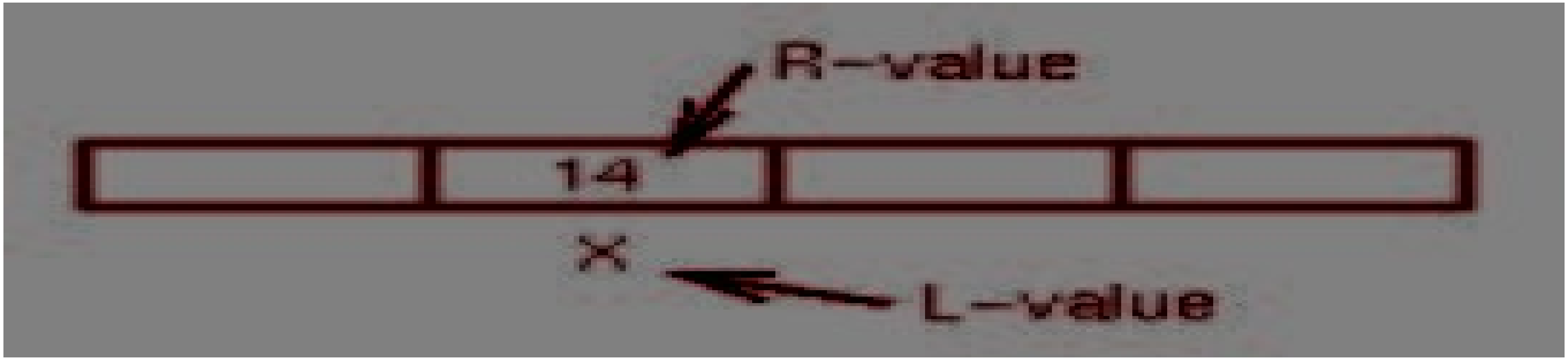
- Assignment is a special type of operation
  - its primary effect is to change the binding of a value to a data object
- There are two common signatures for this operation:
- 1. Pascal: Integer × Integer − > void
- 2. C/C++: Integer1 × Integer2 − > Integer3
- Thus a C assignment operation can be reused
- (c = getchar()) != '\n')

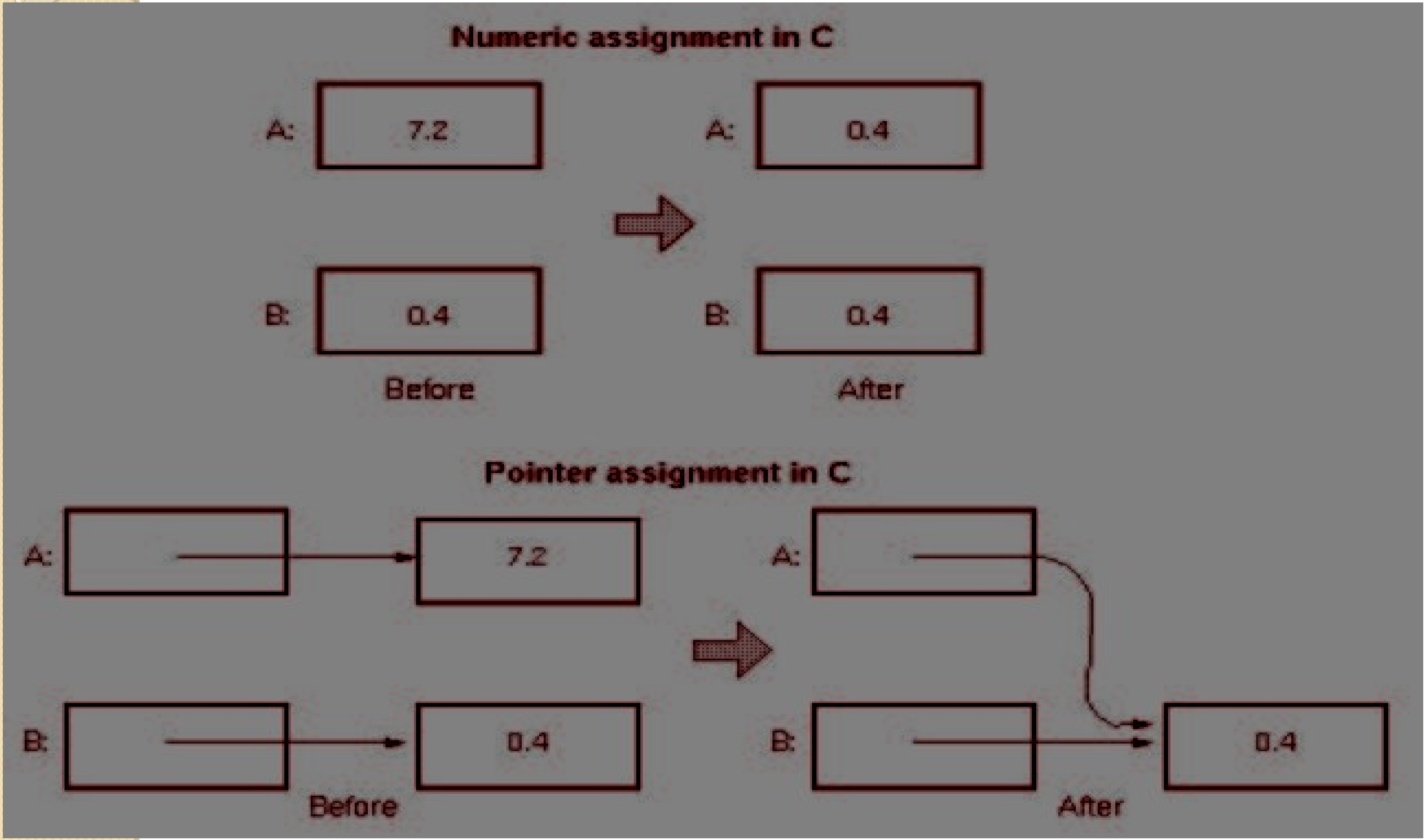
Where did names L-value and R-value come from?

- 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 (=) means content-of location,
- and its position on the left-hand-side of the assignment operator means an address-of location.

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

- address-of means L-value
- Contents-of means R-value
- Value, by itself, generally means R-value




Numeric assignment in C
A:
7.2
A:
0.4
B:
0.4
B:
0.4
Before
After
Pointer assignment in C
A:
7.2
A:
B:
0.4
B:
0.4
Before
After

Type Checking

- Checking each operation received the proper number of arguments of proper type
  - X = A + B * C
  - What is type of A , B, C ?
  - Can they be operated upon by + * ?
  - Can the result be assigned to X ?
  - If A is array and B is integer → then error in C
- One of the most important issues that distinguishes between language types

Type Checking

- Static Type Checking
  - Type checking done at compile time
  - Compiler should know the types for all variables
  - C, Pascal, Java
- Dynamic Type checking
  - Type checking done at run time
  - The “run time system” – interpreter – which may be a software of hardware (i.e. processor) should know about type
  - Store information about type with the variable, usually in a “tag”
  - Perl, python, prolog,…

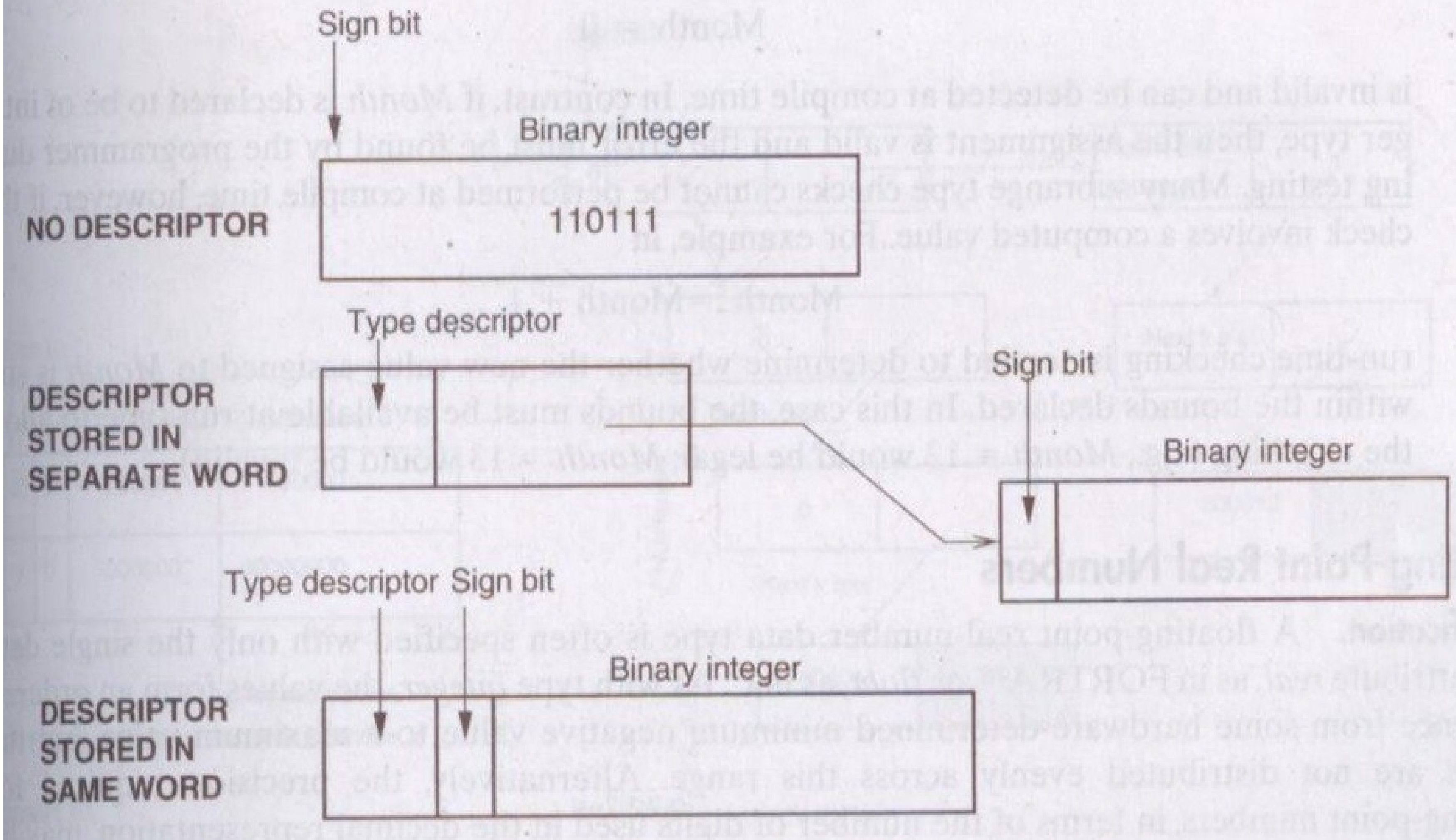


Figure 5.3. Three storage representations for integers.

Dynamic Type Checking

- + Flexibility on programming
- + No declarations needed
- - Difficult to debug
  - Unexecuted paths never type checked
- - Type info to be stored at run time
- - Normally software interpreted, so slow

Static Type Checking

- Compiler needs info: about type of each variable, for each operation the signature, type of each constant
- During syntax analysis collects type info in Symbol Table
- + More efficiency – no run time additional code
- - Less flexibility
- - Not possible for some constructs of languages

Strong Typing

- If all type errors can be checked statically
- F: S $\rightarrow$ R is *type safe* if all results are in the domain of R
  - short X, Y; X+ Y may overflow , so not type safe
- Difficult to achieve 100% strong typing

Type Inference

- Compiler “infers” the type of a variable
- Language ML has this feature
  - Infers missing type from other variable's types
- fun area(length, width):int = length * width
- fun area(length:int, width) = length * width
- fun area(length,width:int) = length * width
- fun area(length,width) = length * width → INVALID

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

Conversion examples

- Examples in Pascal:
- `  var A: real;`
- `      B: integer;`
- `A := B` – Implicit, called a coercion - an automatic conversion from one type to another
- `A := B` is called a widening conversion since the type of A has more values than B.
- `B := A` (if allowed) is a narrowing conversion since B has fewer values than A. Information could be lost in this case.
- `B := (int)A` – Explicit, called a cast

Coercion

- In most languages widening coercions are allowed; narrowing coercions usually must be explicit.
  - This is true in Java
- In addition to casting, there are often functions that can be used for conversion.

  B := round(A); Go to integer nearest A

  B := trunc(A); Delete fractional part of A

- Review and Self Study

Integer numeric data

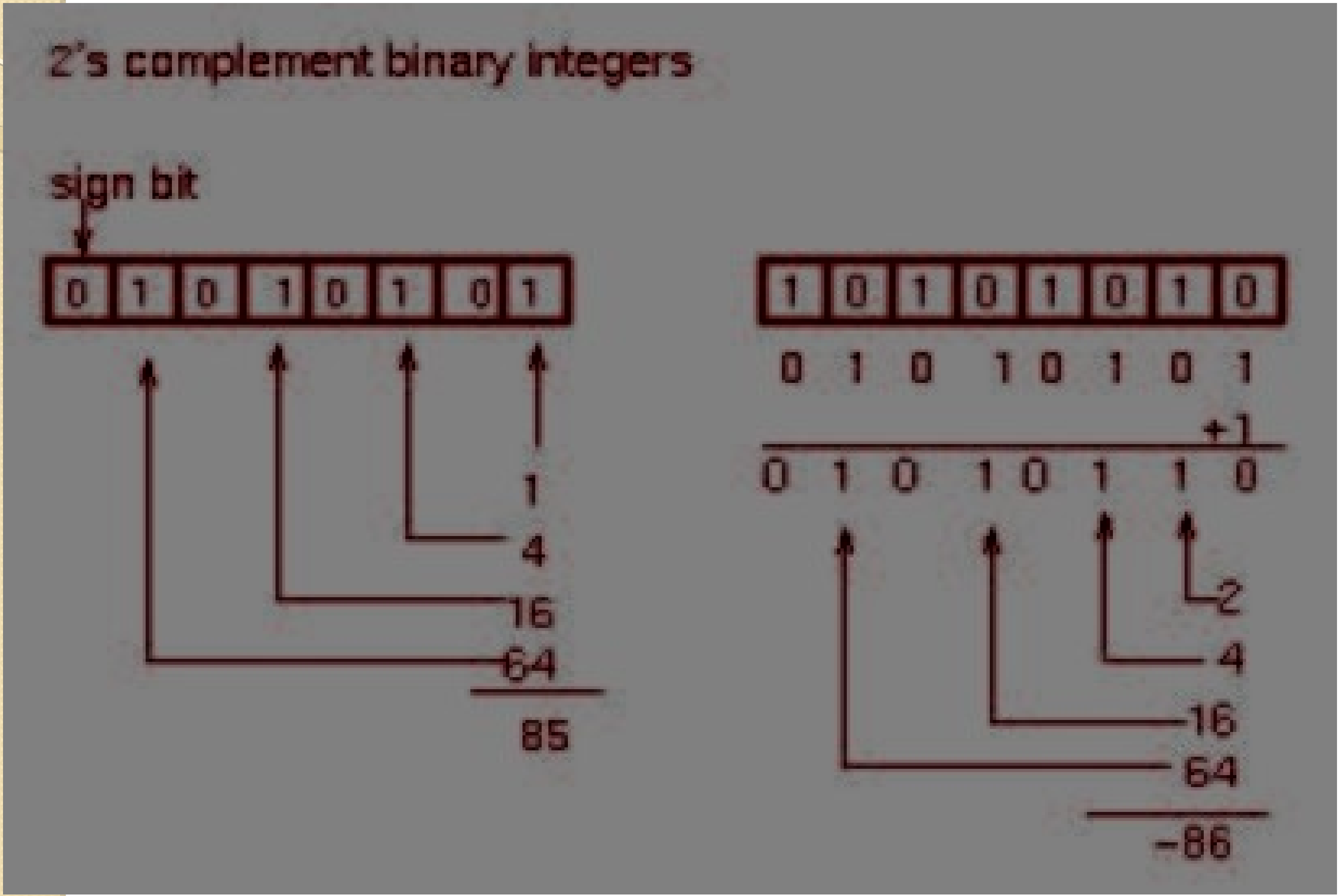
- Integers: Binary representation in 2’s complement arithmetic
- For 32-bit words:
  - Maximum value:

    $2^{31} - 1$

  - Minimum value:

    -2^{31}

Integer numeric data



Converting integers to binary

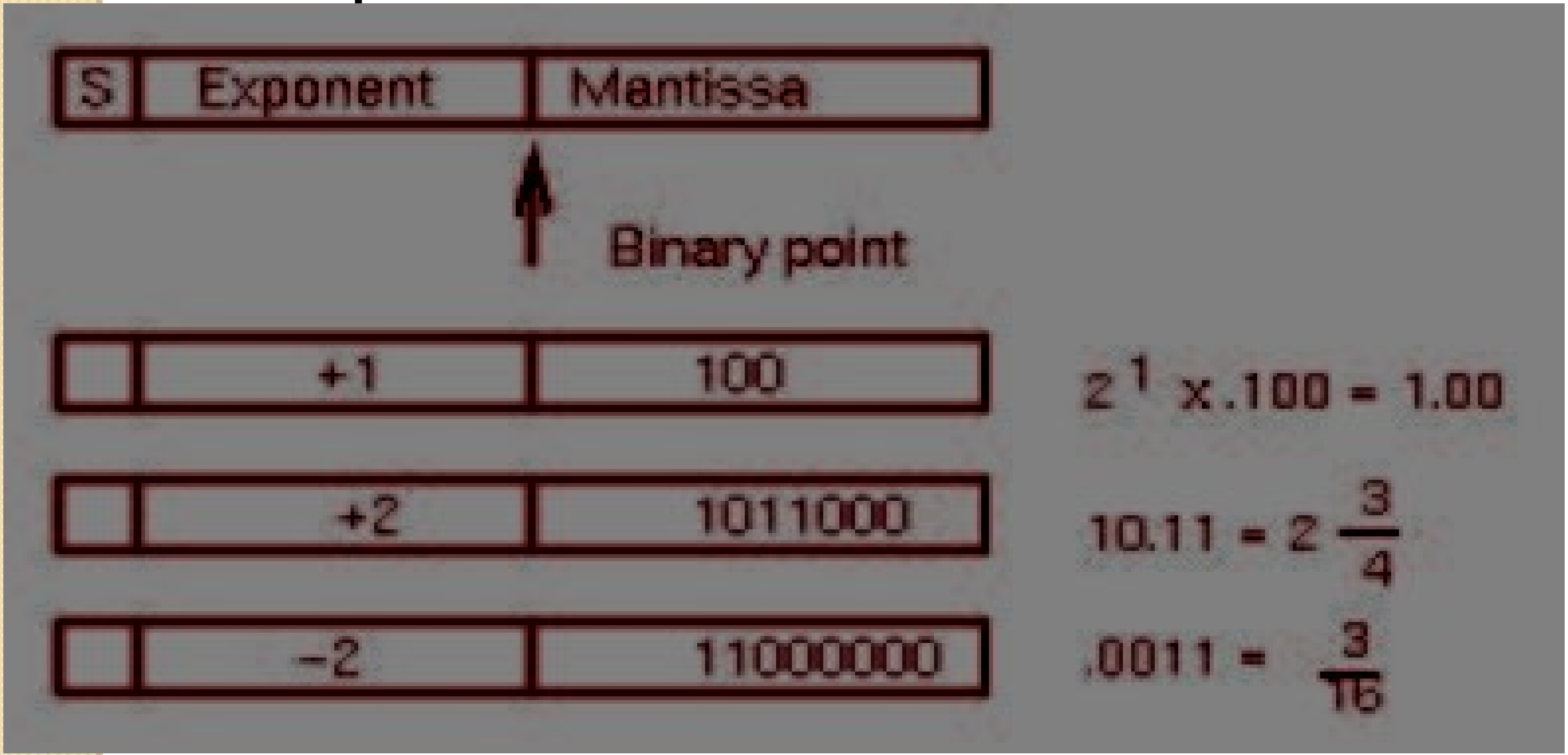
- hope you already know how to do this !

Integer operations

- Arithmetic
  - Binary: addition, subtraction, multiplication, division, modulo + - * / %
  - Unary: negation, identity -
- Relational: less, greater, equal, not equal, Assignment < > = != =
- Bit Operations: AND, OR, shift & | << >>

Real numeric data

- Floating Point (real): hardware representations



Real numeric data

- 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

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.

Example floating point numbers

- $+1 = 2^0 * 1 = 2^{127-127} * (1).0b$
  0 01111111 000000...
- $+1.5 = 2^0 * 1.5 = 2^{127-127} * (1).1b$
  0 01111111 100000...
- $-5 = -2^2 * 1.25 = 2^{129-127} * (1).01b$
  1 10000001010000...
- 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 -10^{22} to $+10^{23}$, yielding numbers in the range 10^{-308} to 10^{308}.

Decoding IEEE format

- Given E, and M, the value of the representation is:
  $(-1)S \times (1 + Mf\ ) \times 2(E-128)$
- where Mf is a fractional value.

Parameters	Value
E = 255 and M != 0	An invalid number
E = 255 and M = 0	∞
0 < E < 255	$2E-127$ 1.M
E = 0 and M != 0	$2E-126$ .M
E = 0 and M = 0	0

Floating Point Operations

- Converting real numbers to binary
- Usual arithmetic and comparison operations – implemented in hardware if possible
- Built-in mathematical functions – sin, sqrt, etc

Fixed point 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;

Using decimal data

- What is Z=X+Y?
  X = 12.345;

  Y = 9876.54;

- 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.

Implementing fixed decimal data

- Algorithm for code generation by compiler:

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

   Compiler knows scale factor (S=3 for X, S=2 for Y) and remembers it

   True value printed by dividing stored integer by 10^s

2. To add X+ Y, align decimal point. Adjust S by 1 (=3-2) by multiplying by 10.
3. 10*Y+X = 9876540 + 12345 = 9888885, Compiler knows S=3

Other numeric data

- Short integers (C) – 16 bit, 8 bit
- Long integers (C) – 64 bit
- Byte – 8 bits
- complex – requires two data values
- rational numbers – quotient of two integers

Enumerations

- Enumerated types allow the programmer to specify types which can take on a limited number of symbolic values.

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

typedef enum  color { orange, red, white } color ;
```

- Implemented as small integers with values:

  A = 0, B = 1, C = 2, D = 3;   orange = 0, red =1, white= 2; etc

```
NewType X, Y, Z;    color a, b, c;

X = A;   a = b;
```

- Why not simply write: X=0 instead of X=A?
  - Readability   a =0;   Vs a = orange ?
  - Error detection   a= 4; // improper, but legal

Enumeration Example

- In Pascal:

  enum { fresh, soph, junior, senior} ClassLevel;

  enum { old, new } BreadStatus;

  BreadStatus = fresh;   An error which can be detected

- Defining an enumerated type effectively gives you a set of named constants.
- In C and C++, you could use an enumerated type for your lexeme types.

Character

- Representation
  - Single 8-bit byte can represent up to 256 characters
  - ASCII is a 7 bit 128 character code
  - Unicode (Java) is a 16-bit code
- In C, a char variable is simply 8-bit integer numeric data
  - usually supported in hardware
  - Operations – relational, class membership (isdigit e.g.)

Boolean

- Value true or false
- Could use 1 bit for storage efficiency
- Frequently use byte or word for efficiency
  - use a particular bit
  - use 0 for true, everything else for false
- Operations: and, not, or
- Not supported in C

Boolean Implementation

- Could implement as single bit but usually use smallest addressable memory
  - unit (byte or word) for efficiency
  - values of true and false
    - defined values of true and false as in java
    - use state of a specific bit
    - use 0 for false; anything else is true – have to be careful since complement of a true value may not be false

Subtypes

- A is a subtype of B if every value of A is a value of B.
- A subtype shares the operations of the type it was derived from.
  - Pascal allows the programmer to define subranges of integers
- type SmallInteger = 1..20;
  - In Java, byte and short are subtypes of integer - they don’t have a separate set of operations defined
  - In C almost everything (except floating point) is a subtype of integer.
- Subranges have smaller storage requirements and better type checking but generally slower operations.

Bitstrings

- Combine multiple data values into a single storage unit
  - one byte could hold 8 independent bits of information
- Allow multiple boolean values to be packed into smaller space
- Harder to use because of having to extract bits individually

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

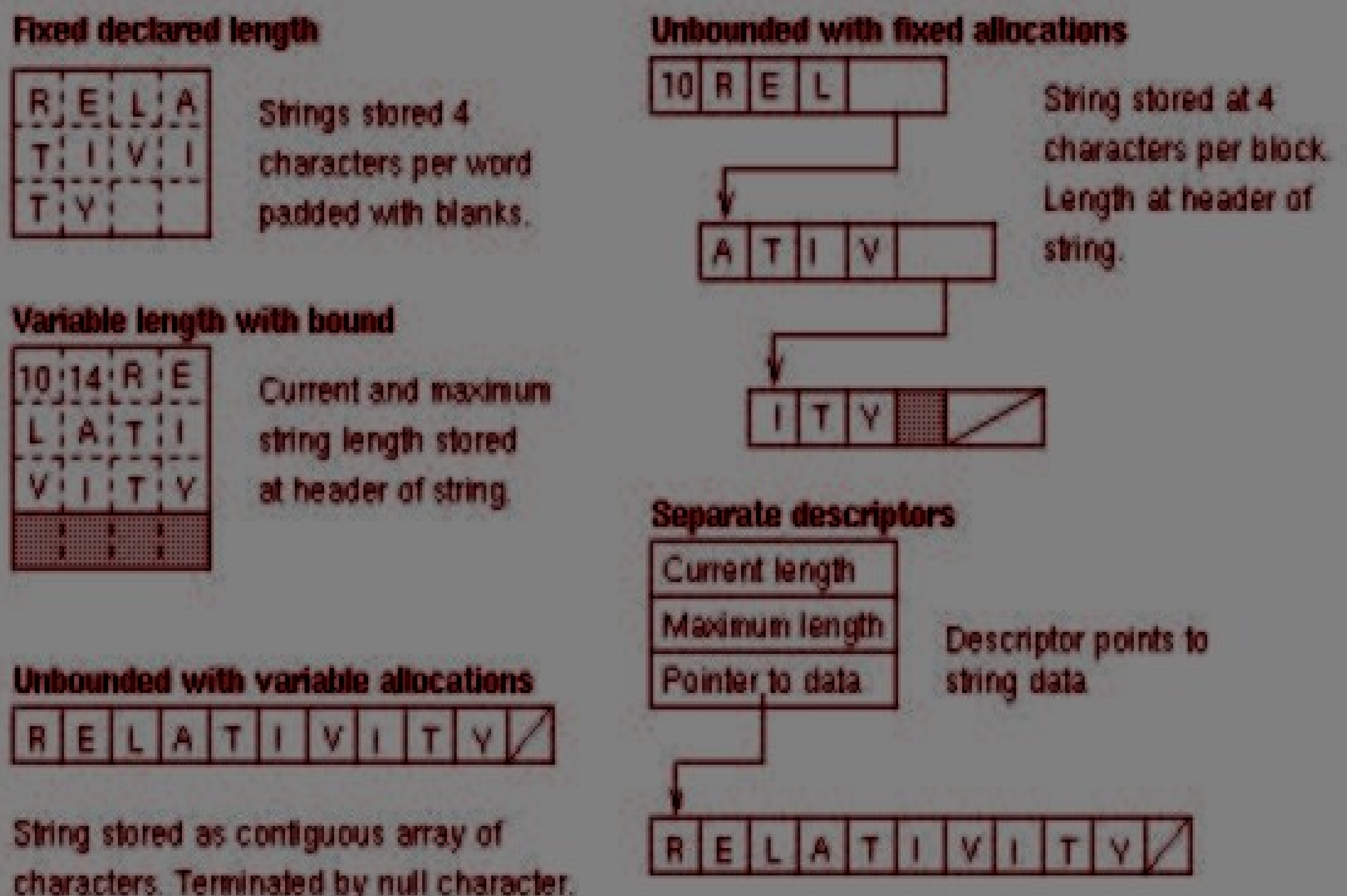
  var C packed array [1..10] of char - Pascal

- Variable length: upto alimit

  DCL D CHAR(20) VARYING - PL/I - 0 to 20 characters

- Variable length: no limit

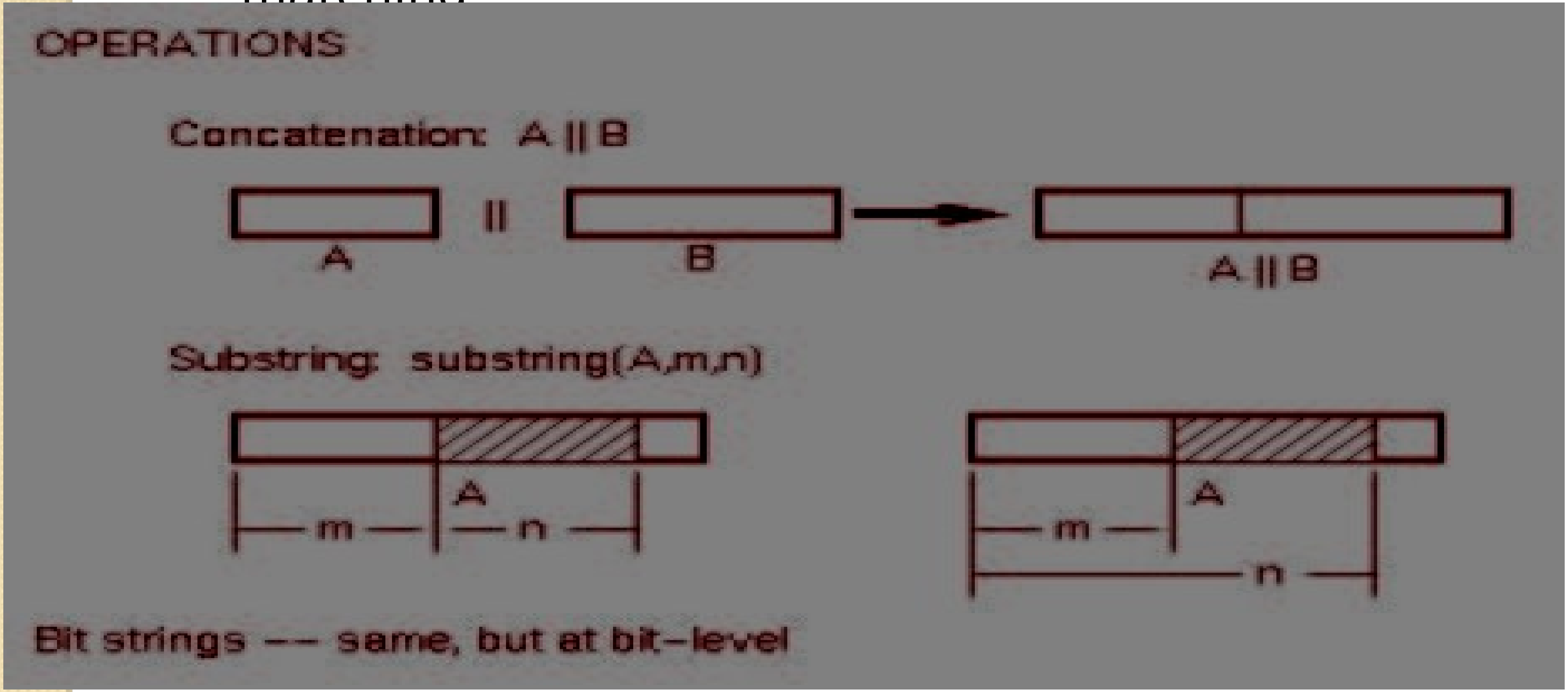
  E = "ABC" - SNOBOL4 - any size, dynamic



Fixed declared length
R E L A
T I V I
T Y
Strings stored 4 characters per word padded with blanks.
Variable length with bound
10 14 R E
L A T I
V I T Y
Current and maximum string length stored at header of string.
Unbounded with variable allocations
R E L A T I V I T Y
String stored as contiguous array of characters. Terminated by null character.
Unbounded with fixed allocations
10 R E L
A T I V
I T Y
String stored at 4 characters per block. Length at header of string.
Separate descriptors
Current length
Maximum length
Pointer to data
Descriptor points to string data
R E L A T I V I T Y

String operations

- Concatenation
- Comparisons – based on encoding
- Substring – based on either position or pattern matching



String operations

- Special case of C:
  - F = "ABCDEFG" - C - size programmer defined, terminated by null character ’\0’, can't change later
  - char a [10];
  - In C, arrays and character strings are the same. Except, the last element of a character string must be ’\0’.
- Implementation of element access:
  L-value(A[I]) = L-value(A[0]) + I

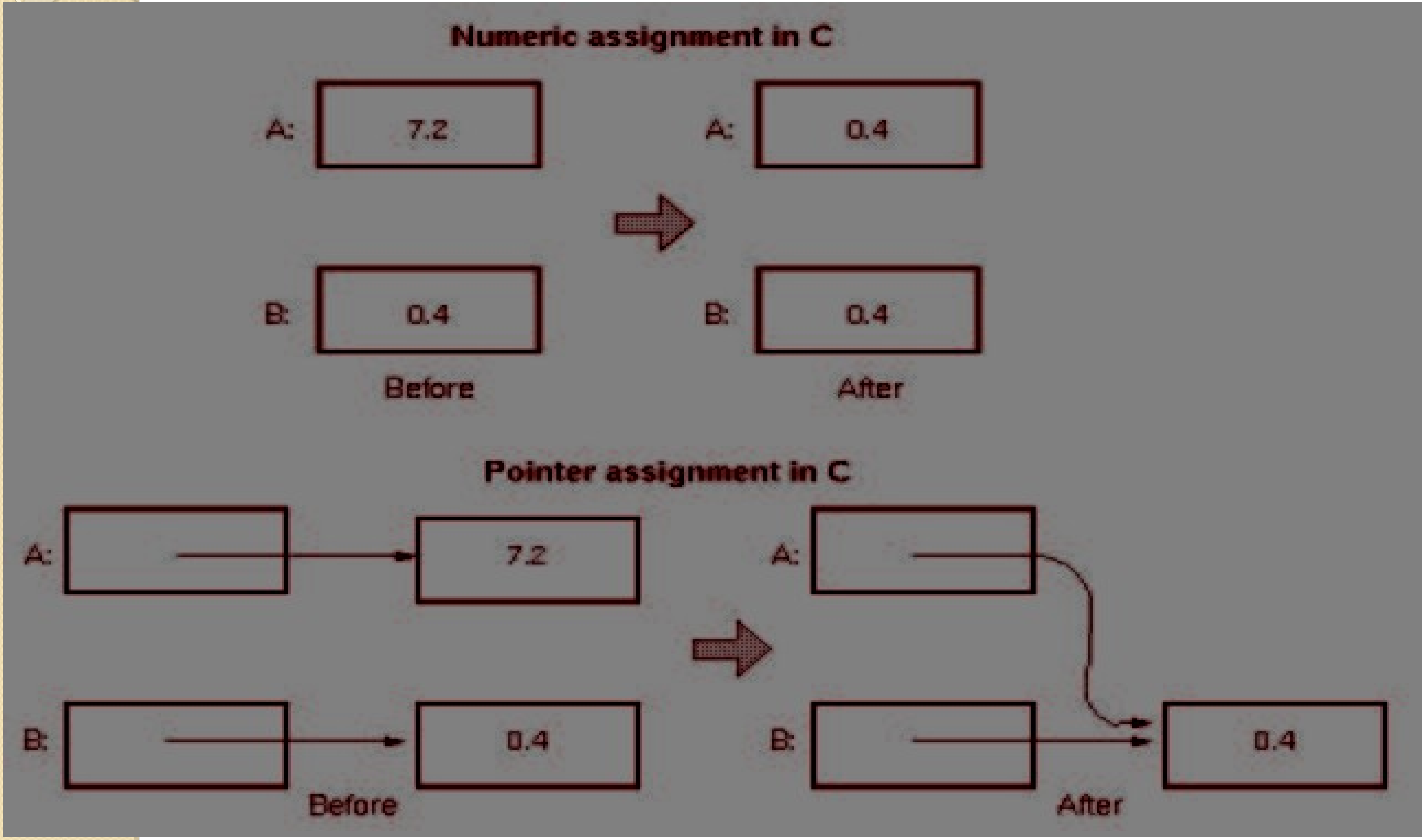
Pointer data

- Data object whose value is a location
  - pointers may reference a single type as in C, Pascal, Ada
  - pointers may reference any type as in Smalltalk
  - object-oriented languages use a mixture
- Use of pointers to create arbitrary data structures such as linked lists
  malloc (C) or new (C++)

Pointer data

- In general a very error prone construct and should be avoided – memory leaks, dangling pointers
- Some languages allow the programmer to manipulate pointers directly – C, C++
- Other languages hide pointers in the implementation – Java
- Implementation – either absolute or relative address possible

Pointer aliasing



Pointer Operations

- Creation (new, malloc) – allocate memory for a data object for the pointer to refer to
- dereferencing – retrieve the data that a pointer is refering to
- deletion (delete, free) – free up the memory used by the object a pointer is refering to

- EXTRA SLIDES

Files

- A special kind of data object
  - represented in secondary memory (disk)
  - lifetime extends beyond that of the program
  - used for I/O and scratch storage
- Three types
  1. sequential – variable length, linear sequence of components of the same type, usually characters
  2. direct access – unordered sequence of components each of which can be accessed at random
  3. indexed sequential – like direct access except components are ordered by key value
- Operations: open, read, write, check for EOF, close
- Implementation: generally based on the file system of the OS

Declarations

- Declarations serve several purposes
  - determines storage requirements
  - ocation in program determines variable lifetime which is used for storage management
  - allows translator to choose between polymorphic versions of an operation
  - facilitates type checking
  - initialization (usually optional)
  - storage representation in a few languages (COBOL)
- Some languages require explicit declarations – C, C++, Java
- Other languages allow implicit declarations – variables beginning with [I..N] are
- integer by default in Fortran
- Some languages don’t have types – Perl
- Some languages can infer types from the context – ML

Type Inference

- Consider
  x = 3 * 5
- x must be an integer because the result of the operation is an integer
  float y = a + 3.0
- a must be a float because the operation needs two arguments of the same type

Declarations for Programmed Defined Operations

- Prototypes or declarations are often used to give the signature
  - return type
  - number of arguments
  - order of arguments
  - type for each argument

Type Checking

- A value stored in hardware is just a sequence of bits. It could represent anything – integer, floating point number, characters, instruction. In order to interpret the bits, the type needs to be known.
- Type checking is used to determine whether operations (or functions) have correct number and type of arguments can be static or dynamic
  - Static type-checking done at compile time. It is more efficient. A symbol table is used to store the information needed by the compiler for each data object - type for variables, type and value for constants, argument number, order and types as well as return type for functions.
  - Dynamic type-checking is done at execution-time. It is less efficient, both in speed and memory, can make things hard to debug but is more flexible. A type-tag needs to be stored along with the value(s) for each data object.
- Many languages have situations where static type-checking is not possible.(Superclass references in Java for instance.) Can use either dynamic type checking or leave these situations unchecked.
- A strongly-typed language is one in which all type-checking can be done at compile time

Batch Model of Processing

- Many program executions can be viewed as consisting of the following steps:
- program loaded
- access to external data (files) made available to program
- program executes
- program terminates
- Not all processes fit this model – think about an airline reservation system, Bron-coWeb, on-line banking
- Both data and program co-exist indefinitely.