

Fundamentals of Programming Languages

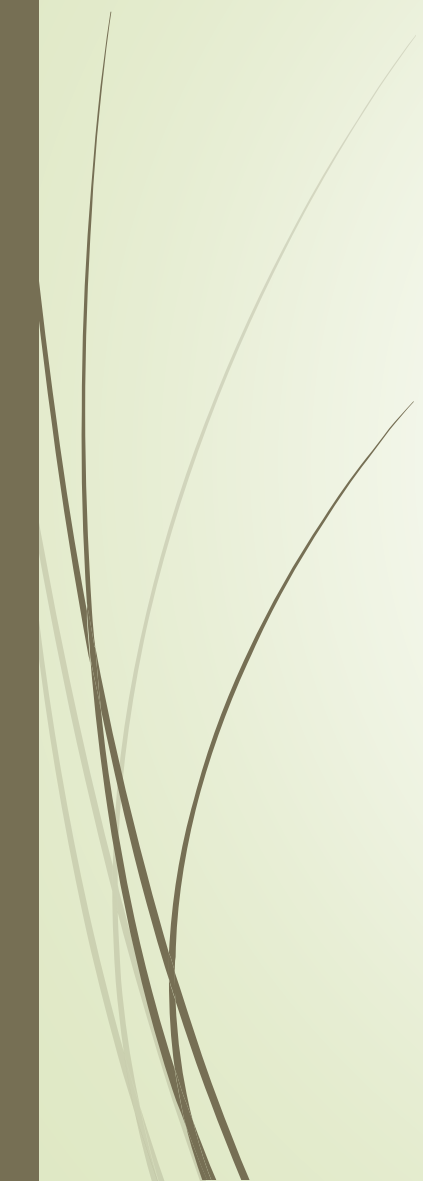
Program entity attributes. Attributes dynamic binding

Lecture 05

sl. dr. ing. Ciprian-Bogdan Chirila



Lecture outline

- Variable domain
 - Variable lifetime
 - Memory allocation
 - Variable value
 - Variable type
- 



Program attributes



- A PL operates with several entities:
 - Variables
 - Constants, literals
 - Subprograms
 - Types
 - Instructions
- Entities
 - may have a name – when an id is associated
 - may be anonymous
 - E.g. when objects are referred by pointers
- The name is just one possible attribute of an entity



Program attributes



In imperative PLs:

- a variables has
 - Name
 - Type
 - Memory address
- a subprogram has
 - Name
 - Formal parameters
 - Associated action sequence
- an instruction
 - The actions involved



Binding



- The association between an entity and its attributes is called binding.
- PL differ in the way the attributes are bound / linked to entities
- Attribute binding can be
 - Implicit
 - Explicit



Examples



- ▶ In Fortran for variables:
 - ▶ Name binding
 - ▶ at the first place the variable is used
 - ▶ Type binding
 - ▶ Depending on the variables name
 - ▶ I,J,K,L,M,N – integers
 - ▶ Otherwise – real
 - ▶ By explicit declaration
- ▶ In ML:
 - ▶ Variable name, type, value is bound in the moment of the assignment

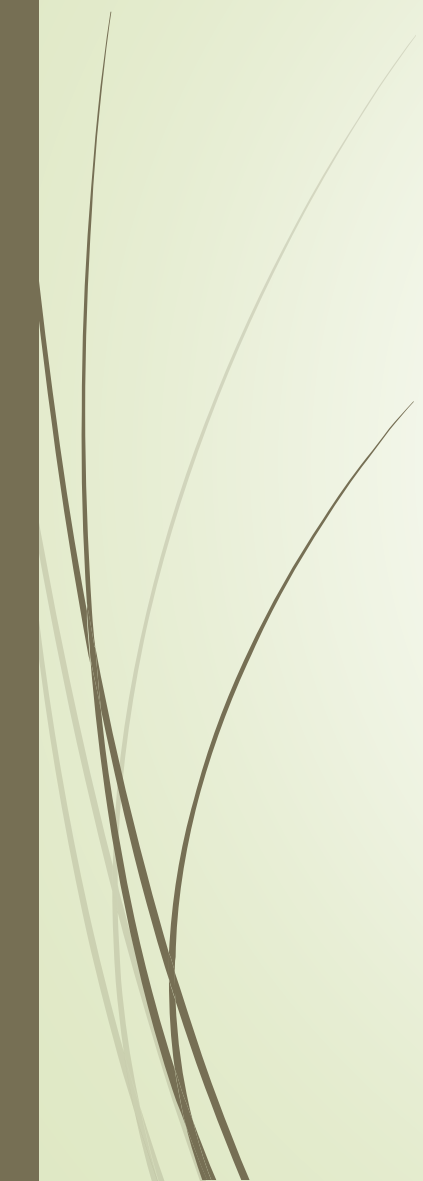


Examples

- Useful redundancy
 - Explicitly binding attributes
 - Declarations in Pascal, C, Java



Attribute binding moment

- Binding at language definition
 - Binding at compile time
 - Binding at execution time
- 



Binding at language definition

- In C:
 - special identifiers: char, int, float, ...
 - are associated with corresponding value sets
- In Pascal:
 - Constants: true, false, Maxint
 - Types: integer, real, char
 - Functions: abs, trunc, chr, ord
- In Java:
 - null is bound by definition
 - null is associated with the void pointer



Binding at compile time

- Variables types
 - `var i:integer; (Pascal)`
 - `int i; (C)`
- Constant values
 - `const a=3; (Pascal)`
- Type and value
 - `int a=3; (Java)`



Binding at execution time

- Assigning values to a variable
- Static binding
 - Before execution
 - In the language definition or
 - At compile time
 - Can not be changed afterwards
- Dynamic binding
 - At execution time
 - Can be changed afterwards



Variables

- Domain
- Life time
- Value
- Type
- Name
 - if not anonymous and referred by pointer



Variable domain

- The program zone where the variable is known and useful
- The variable
 - Is visible in the domain
 - Invisible outside the domain
- Domain concept related to
 - Context
 - Environment



Variable domain

- Context
 - All variables with values at some point
- Environment
 - Explicitly defined subdomain for one or more variables
 - E.g. the function body is an environment for local variables and parameters



Domain static binding

- Block oriented PL classic rules
 - Variable domain is the block where it was declared and its internal blocks
 - Variable is invisible outside the block where it was defined
- Variable domain
 - Is determined in terms of program lexical structure
 - Is determined statically by the program text
 - Does not depend on the execution dynamic
 - Any variable reference will be related by the compiler with its declaration (implicit or explicit)
- Thus, it results a domain static binding



Pascal example

```
program domain;
  var x:integer;
  procedure f;
  begin
    write(x) { refers to globally declared x }
  end;
  procedure f1(x:integer);
  begin
    f
  end;
begin { main program }
  x:=10;
  f; { 10 is printed }
  f1(5); { 10 is printed }
end.
```




Comments



- f procedure refers global x
- Does not matter from where is was called
- In static domain binding
 - The valid declaration is searched in the environment where it is referred
 - If it is missing then it is searched in the surrounding environments
- It is the case for:
 - Pascal, Ada, C, Java, Fortran, Modula 2



Domain dynamic binding

- Variable domain
 - is determined at program execution
 - Depends on its execution
- The variable binds to a declaration that
 - is not noticeable in the program text
 - is determined during execution
- A variable declaration gets available when
 - Is met on the execution path
 - Binds to it all further references to that variable name
 - Until a new declaration with the same name occurs



Lisp example

```
(setq x 10)
(defun f()
  (print x))
; may refer the global x or the
parameter x
```

```
(defun f1(x)
  (f))
```

- print x may refer to global x or parameter x
- binding is made at execution time



Comments



- (f)
 - 10 is printed
 - The value of global x
- (f1 5)
 - 5 is printed
 - The value of parameter x
- Domain dynamic binding affects program readability
- Facilitates the implementation of interpreted languages
- Present in functional languages
 - Lisp or APL

Lisp domain static binding

- present in Lisp new versions

 - Scheme

 - Common Lisp

- Example

```
>(defun f1(x) (f))
```

```
>(defun f( ) x)
```

```
>(f1 5)
```

```
*** - EVAL: variable X has no value
```



Comments



- The value of x from function f is searched
 - statically in the environment of f
 - then globally
- It is not defined there
- So, error occurs

Domain dynamic binding at programmer request

- In Common-Lisp
- Special local variables

```
>(defun f1(x)  
  (declare (special x))  
  (f))
```

```
>(defun f()  
  x)
```

```
>(f1 5)
```

```
5
```



Domain dynamic binding at programmer request

➤ Global defined variables

```
>(defvar x)
```

```
>(defun f1(x)  
  (f))
```

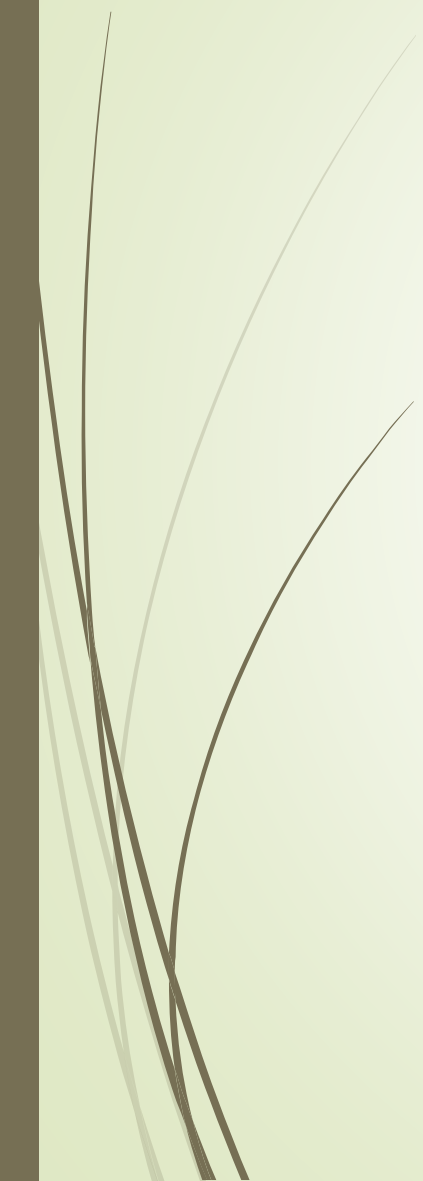
```
>(defun f(x)  
  x)
```

```
>(f1 5)
```

```
5
```




Variable lifetime

- The amount of time a certain memory zone is associated to the variable
 - The association of the memory zone to a variable is known as **allocation**
- 



Allocation



- Static
 - Before execution
 - A certain zone decided at compile time
 - Will remain associated the whole program execution
- Dynamic
 - Allocation is made during the program execution
 - The memory zone can be freed afterwards

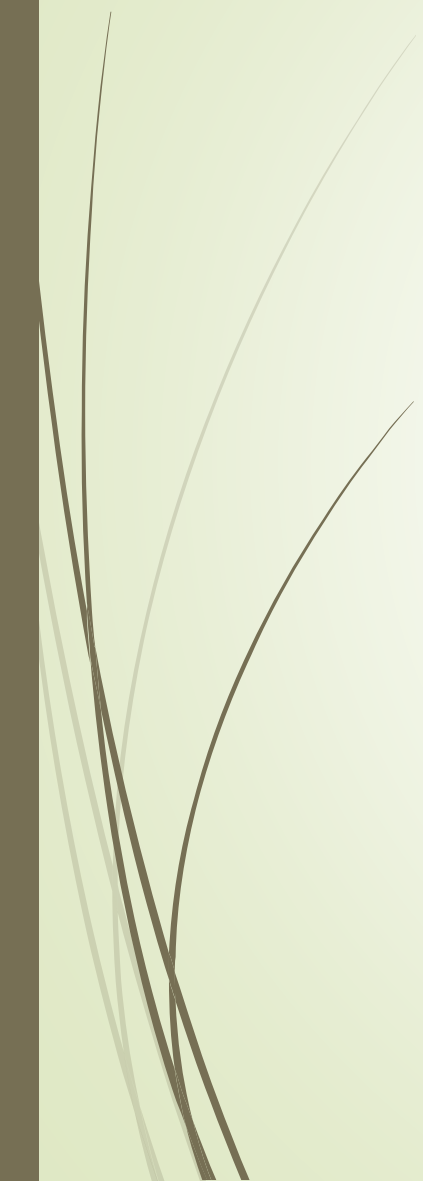


Dynamic allocation

- automatic
 - Without programmers request
- by request
 - by request from the programmer
 - by using instructions like: new, malloc



Memory allocation

- is not specific to the PL
 - depends also on the implementers choice
- 



Memory allocation examples

- Fortran and Cobol
 - Static allocation of variables in most implementations
 - Could be equipped with dynamic memory allocation as well
- Pascal, C, Java
 - For locally declared variables dynamic allocation is used
 - In a function call all locals are allocated on the stack
 - After the call the stack is cleaned
 - Memory allocation based on **stack** organization



Memory allocation examples

- Programmer defined allocation
 - In C programming language
 - inside a function
 - implicitly is dynamic
 - static by using the “static” keyword
 - outside functions
 - implicitly is static
- Lisp and Prolog
 - Allocating and releasing memory
 - Not based on the stack model
 - Objects may be created and destroyed arbitrary moments of the runtime
 - Dynamic languages



Variable value

- The value is bind dynamically
 - The assignment changes the variable value
- The value can be bound statically
 - Used in the case of constants
 - The value can not be modified during their lifetime



Binding moment

- At compile time
 - Constants in Pascal
 - Literals or literals expression in Ada
 - Define constants in C
 - All compile time bound constants are called **manifest constants**
- At execution time
 - The constant expression can contain variables and operands
 - C, Ada, Algol
 - `const int k=3*i+j;`
 - `k: constant integer:=3*i+j;`



Variable type

- Determines
 - the value set that a variable can have
 - the operation set that can create or modify these values
- Static binding
 - at compile time
 - Implicit
 - Fortran based on the first letters of the identifier
 - Pascal constants `const k=3;`
 - Explicit
 - Pascal `var x:integer;`
 - C, Java `int x;`



Variable type

- Lisp or ML
 - The type is dynamically bound
 - The same variable can have different typed values associated
- CAML (ML dialect)
 - # let a=2*2;
 - val a:int=4
 - #a;;
 - :int=4



Variable type



Lisp

```
defun f(x) (car x))
```

```
(setq y 'a)
```

```
(setq y '(a b))
```

```
(f y)
```