Estruturas de Linguagem

Francisco Sant'Anna

francisco@ime.uerj.br

http://github.com/fsantanna/EDL

- Nomes
- Binding (amarração)
- Variáveis
- Tempo de Vida

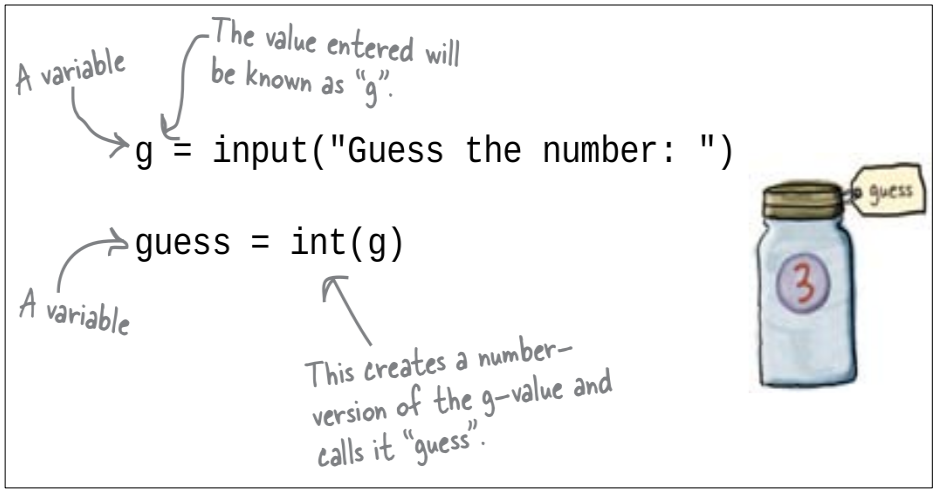
- Escopo

Binding (Amarração)

- Associação entre “entidade” e “atributo”
  - *binding time*
    - *language design time*
    - *language implementation time*
    - *preprocess time*
    - *compile time*
    - *link time*
    - *load time*
    - *run time*

Variáveis

- Uma “etiqueta” (ou nome) que representa uma região de memória
- Uma abstração da memória do computador
  - endereço
  - valor
  - tipo
  - **escopo**
  - **tempo de vida**



Créditos: "Head First Programming"

Binding de Memória

- Binding *nome* -> *célula/endereço de memória*
  - alocação
  - desalocação
- Tempo de vida
  - período entre alocação e desalocação
  - característica de execução
- Escopo
  - intervalo de visibilidade da variável
  - característica léxica

Tempo de Vida vs Escopo

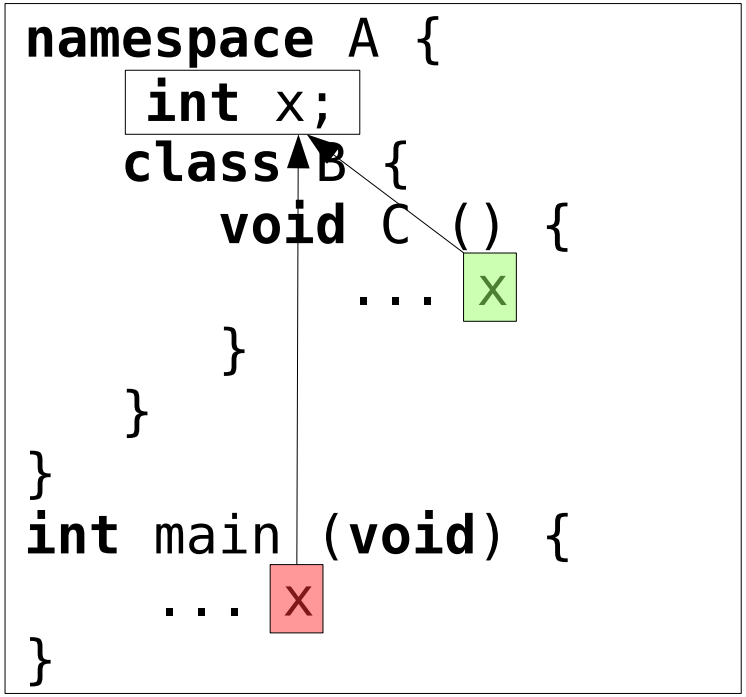
```c
#include <stdio.h>

int f (void) {
    [redacted]int v = 0;
    v = !v;
    return v;
}

int main (void) {
    printf("f = %d\n", f());
    printf("f = %d\n", f());
    printf("f = %d\n", f());
    printf("f = %d\n", f());
    printf("f = %d\n", f());
    return 0;
}
```

Tempo de Vida

- Estático (variáveis globais/estáticas)
- Dinâmico
  - pilha (variáveis locais)
  - heap
    - explícito (e.g., *malloc/free*)
    - implícito (e.g., *new/coletor*, construtores primitivos)

Escopo

- Estático / Léxico
  - determinado em tempo de compilação
- Dinâmico
  - depende da execução do programa



Locais e “não locais”

- Locais
- Não locais
  - Globais
  - Pacote, Namespace
  - OO
    - Variáveis de classe (Class.y)
    - Variáveis de instância (this.y)
  - *Upvalues*

```
{
  int x = ...;
  _printf(“%d\n, x + y);
}
```

Exemplo C++

Escopo léxico

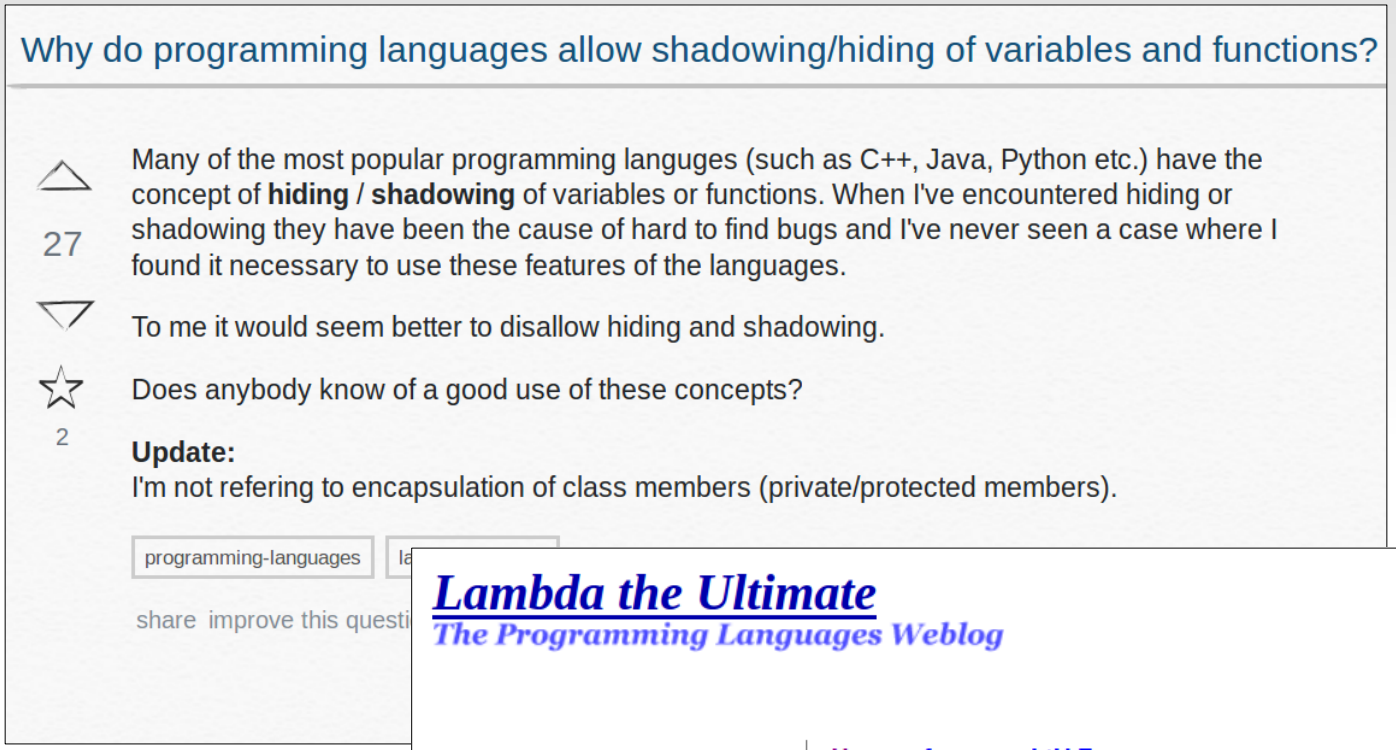
- Blocos
- Rotinas
- Classes
- Namespaces
- Arquivos
- Globais

```
int x;
void f (int x) {
    void g (int x) {
        int x;
        _printf(“%d\n, x);
    }
}
```

```
let a=10 in
  printf "> a=%d\n" a;
  let a=1 and b=a in
    printf ">> a=%d b=%d\n" a b;;
```

Hiding, Shadowing

Hiding, Shadowing



Why do programming languages allow shadowing/hiding of variables and functions?

27

Many of the most popular programming languges (such as C++, Java, Python etc.) have the concept of **hiding** / **shadowing** of variables or functions. When I've encountered hiding or shadowing they have been the cause of hard to find bugs and I've never seen a case where I found it necessary to use these features of the languages.

To me it would seem better to disallow hiding and shadowing.

2

Does anybody know of a good use of these concepts?

Update:
I'm not refering to encapsulation of class members (private/protected members).

programming-languages

share improve this questi

Lambda the Ultimate
The Programming Languages Weblog

XML

Home

Feedback

Home » forums » LtU Forum

Disallow shadowing?

I came across an early post on the topic of disallowing shadowing (message quoted below):

Hiding, Shadowing

```
x = 1

def f():
    x = 2
    print "dentro ", x

f()

print "fora ", x
```

- Declaração vs Atribuição
  - binding of scope
  - binding of value

Outro exemplo: global

So, imperative language designers of the future, heed my warning: SHARPLY DISTINGUISH BINDING FROM ASSIGNMENT OR BE FOREVER DAMNED.

By **Matt Hellige** at Tue, 2014-02-11 16:55 | **login** or **register** to post comments

Escopo dinâmico

- Se aplica a variáveis não locais
- Busca declaração ativa mais recente
  - na pilha, em tempo de execução
- Perl :)

```
local $x = 'global';

sub print_x {
    print "x = $x\n";
}

sub f {
    local $x = "f";
    print_x();
}

print_x();
f()
```

Outro exemplo

Escopo dinâmico

Quora

What are the advantages of dynamic scoping?

McCarthy about that *bug*

30

Like everything else, Dynamic Scoping is merely a tool. Used well it can make certain tasks easier. Used poorly it can introduce bugs and headaches.

I can certainly see some uses for it. One can eliminate the need to pass variables to some functions.

For instance, I might set the display up at the beginning of the program, and every graphic operation just assumes this display.

If I want to set up a window inside that display, then I can 'add' that window to the variable stack that otherwise specifies the display, and any graphic operations performed while in this state will go to the window rather than the display as a whole.

It's a contrived example that can be done equally well by passing parameters to functions, but when you look at some of the code this sort of task generates you realize that global variables are really a much easier way to go, and **dynamic scoping gives you a lot of the sanity of global variables with the flexibility of function parameters**.

share improve this answer

edited Nov 16 '09 at 17:18

answered Nov 26 '08 at 15:20

Adam Davis

58.6k • 41 • 208 • 302

Locais e “não locais”

- Locais
- Não locais
  - Globais
  - Pacote, Namespace
  - OO
    - Variáveis de classe (Class.y)
    - Variáveis de instância (this.y)
  - ***Upvalues***

exemplo (counter-03.lua)