Nomes, Escopos e Vínculos

Sérgio Queiroz de Medeiros
sergio@ufs.br

02 de abril de 2012

Nomes

- Um nome é uma cadeia de caracteres e pode se referir a variáveis, constantes, tipos, etc.
- Nomes são um mecanismo de abstração
- Podemos associar a um nome um fragmento de código complicado
  - Subrotinas são abstrações de controle
  - Classes são abstrações de dados

Vínculos

- Um vínculo é uma associação entre duas coisas, como um nome e a coisa que ele nomeia.
- Tempo de vínculo é o momento no qual um vínculo é criado (ou uma decisão de implementação é tomada)
- Decisões podem ser tomadas em diferentes momentos
  - Tempo de projeto da linguagem
  - Tempo de implementação da linguagem
  - Tempo de escrita do programa
  - Tempo de compilação
  - Tempo de ligação
  - Tempo de carregamento
  - Tempo de execução

Decisões sobre vínculos

- Vínculo dinâmico x Vínculo estático
- Implementações de linguagens baseadas em compiladores tomam decisões mais cedo
- Decisões tomadas em fases iniciais $\Rightarrow$ eficicência
- Decisões em fases posteriores $\Rightarrow$ flexibilidade

Tempo de vida de vínculos e objetos

- Eventos importantes:
  - Criação de objetos
  - Criação de vínculos
  - Desativação/reativação de vínculos
  - Destruição de vínculos
  - Destruição de objetos
- Referência pendente (*dangling reference*)

Objetos e mecanismos de alocação

- Objetos estáticos: endereços absolutos
- Objetos da pilha: geralmente são alocados/desalocados junto com a chamada/retorno de funções
- Objetos do heap: podem ser alocados/desalocados a qualquer momento
  - Maior custo de gerenciamento

Questões de alocação

- Variáveis globais e constantes são alocadas estaticamente

```
printf("alô mundo\n")
A = B / 3.14
```

- Variáveis de uma função
  - Alocação estática
  - Alocação dinâmica

Alocação no heap

- Heap é usado para guardar objetos alocados dinamicamente
  - Listas, conjuntos, strings
- Estratégias de alocação
  - Fragmentação interna x Fragmentação externa
  - First fit x Best fit

Coleta de lixo

- Alocação no heap é disparada por alguma operação em um programa
  - Instanciação de um objeto
  - Inserção de um elemento no fim de uma lista
- Desalocação pode ser implícita ou explícita
  - Explícita: C, C++, Pascal
  - Implícita: Lisp, Java, Lua, C#

Coleta de lixo

- Desalocação implícita $\Rightarrow$ complexidade da implementação e custo da execução
- Desalocação explícita $\Rightarrow$ erros de desalocação são comuns em programas reais
- Coleta automática de lixo (desalocação implícita) parece ser um consenso atualmente

Escopo estático

- Escopo $\Rightarrow$ região do texto de um programa na qual um vínculo está ativo
- Quase todas as linguagens possuem escopo léxico/estático
- Uma linguagem possui escopo léxico/estático quando o escopo é dado por regras puramente textuais
  - Vínculos entre nomes e objetos pode ser determinado em tempo de compilação

Regras de escopo estático

- Há um único escopo global
  - Basic
- Há o escopo global e escopos locais
- Geralmente cada subrotina/bloco introduz um novo escopo local

Regras de escopo estático

```
procedure P1(A1 : T1)
var X : real;
    ...
    procedure P2(A2 : T2)
        ...
        procedure P3(A3 : T3)
        ...
    procedure P4(A4 : T4)
        ...
        procedure F1(A5 : T5)
        var X : integer;
        ...
```

Ordem de declaração

- Qual o escopo de uma variável *x* declarada em um bloco?
- Em várias linguagesm (e.g., Algol 60, Lisp) todas as declarações devem aparecer no início do escopo
- Em Pascal nomes devem ser declarados antes de serem usados e o seu escopo é o bloco inteiro

Ordem de declaração

```
1  const N = 10;
2  ...
3  procedure foo;
4  const
5  M = N; (* erro semântico estático *)
6  ...
7  N = 20;
```

Ordem de declaração

```
1  const N = 10;
2  ...
3  procedure foo;
4  const
5  M = N; (* erro semântico estático *)
6  var
7  A : array [1..M] of integer;
8  N = real;
```

Ordem de declaração

- Em Java, C++, e Ada o escopo de uma variável começa quando ela é declarada
- Em Java e C++ os membros de uma classe são visíveis em todos os métodos
- C# = Java + Pascal

```
1  class A {
2  const int N = 10;
3  void foo() {
4    const int M = N;
5    const int N = 20;
```

Ordem de declaração em Lisp

- Há mais de uma opção
  - **letrec**: escopo é o bloco inteiro
  - **let*** e **let**: escopo é daqui até o fim do bloco

```
(let ((x 2) (y 3))
  (* x y))

(let ((x 2) (y 3))
  (let ((x 7)
        (z (+ x y)))
    (* z x)))
```

Declaração e Definição

```
struct manager;
struct employes {
  struct manager *boss;
  struct employes *next_employes;
  ...
};

struct manager {
  struct employes *first_employee;
  ...
};
```

Escopo dinâmico

- Numa linguagem com escopo dinâmico o vínculo entre nomes e objetos depende do fluxo de controle em tempo de execução
- APL, Snobol, dialetos de Lisp e Perl possuem escopo dinâmico
- Maior parte da verificação de tipos é feita em tempo de execução

Escopo dinâmico

```
01 a : integer
02
03 procedure first
04   a := 1
05
06 procedure second
07   a : integer
08   first()
09
10 a := 2
11 if read_integer() > 0
12   second()
13 else
14   first()
15 write_integer(a)
```

Vínculo com o ambiente referenciado

- No caso de uma referência para uma subrotina, quando as regras de escopo devem ser aplicadas?
  - Quando a referência é criada $\Rightarrow$ vínculo profundo (*deep binding*)
  - Quando a referência é usada $\Rightarrow$ vínculo superficial (*shallow binding*)
- Quando o vínculo é profundo, criamos um fecho (*fecho*) com as informações do ambiente referenciado

Fecho de subrotinas

```
01 procedure A(I : integer; procedure P);
02   procedure B;
03   begin
04     writeln(I);
05   end;
06  begin (* A *)
07    if I > 1 then
08      P
09    else
10      A(2, B);
11    end;
12  procedure C; begin end;
13
14  begin (* main *)
15    A(1, C);
16  end.
```

Subrotinas de primeira e de segunda classe

- Em geral, um valor em uma linguagem é de primeira classe se ele pode ser passada como um parâmetro, retornado de uma subrotina ou atribuído a uma variável
- Um valor de segunda classe pode ser passado como parâmetro, mas não pode ser retornado nem atribuído a uma variável

```
(define plus_x (lambda (x)
   (lambda (y) (+ x y))))
...
(let ((f (plus_x 2)))
    (f 3))              ;retorna 5
```

Referências

- Programming Language Pragmatics (Michael Scott)
  - Capítulo 3