Linguagens de Programação
Conceitos e Técnicas

Amarrações

Prof. Aléssio M. Jr

Conceituação

- Amarração (ou *binding*) é uma associação entre entidades de programação, tais como entre uma variável e seu valor ou entre um identificador e um tipo
- Enfoque na amarração de identificadores a entidades

Tempos de Amarração

Identificador ou Símbolo	Entidade	Tempo de Amarração
*	Operação de multiplicação	projeto da LP
int	Intervalo de inteiros	projeto da LP (JAVA) implementação do compilador (C)
variável	Tipo	compilação (C) execução (polimorfismo em C++)
função	Código correspondente da função	ligação
variável global	Variável em memória	carga do programa
variável local	Variável em memória	execução

- Amarração Estática X Dinâmica

Identificadores

- Identificadores são cadeias de caracteres definidas pelos programadores para servirem de referência a entidades de computação
- Objetivam aumentar a legibilidade, redigibilidade e modificabilidade
- LPs podem ser *case sensitive* e limitar o número máximo de caracteres
- Alguns identificadores podem ter significado especial para a LP
  - Palavras reservadas ≠ Palavra Chave ≠ Palavras Pré-definidas
  - FORTRAN

```
INTEGER REAL
REAL INTEGER
```

Ambientes de Amarração

- A interpretação de comandos e expressões, tais como a = 5 ou g(a + 1), dependem do que denotam os identificadores utilizados nesses comandos e expressões
- Um ambiente (ou *environment*) é um conjunto de amarrações
- Cada amarração possui um determinado escopo, isto é, a região do programa onde a entidade é visível

Ambientes de Amarração

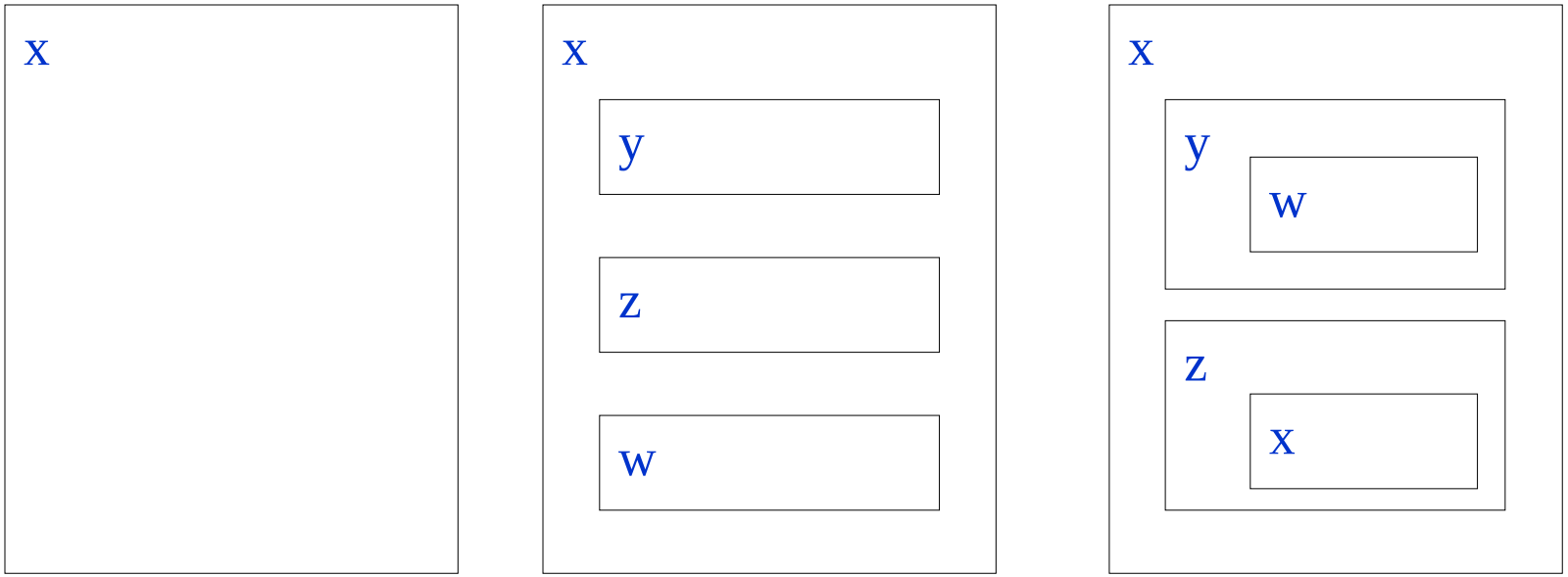
- Amarração de Identificador a Duas Entidades Distintas no Mesmo Ambiente

```
int a = 13;
void f() {
   int b = a;
   int a = 2;
   b = b + a;
}
```

Escopo

- Estático
  - definição do subprograma
  - tempo de compilação
  - texto do programa
- Dinâmico
  - chamada do subprograma
  - tempo de execução
  - fluxo de controle do programa

Escopo Estático



Bloco Monolítico | Blocos Não Aninhados | Blocos Aninhados

Escopo Estático

- Ocultamento de Entidade em Blocos Aninhados

```
void main() {
   int i = 0,  x = 10;
   while (i++ < 100) {
      float x = 3.231;
      printf("x = %f\n", x*i);
   }
}
```

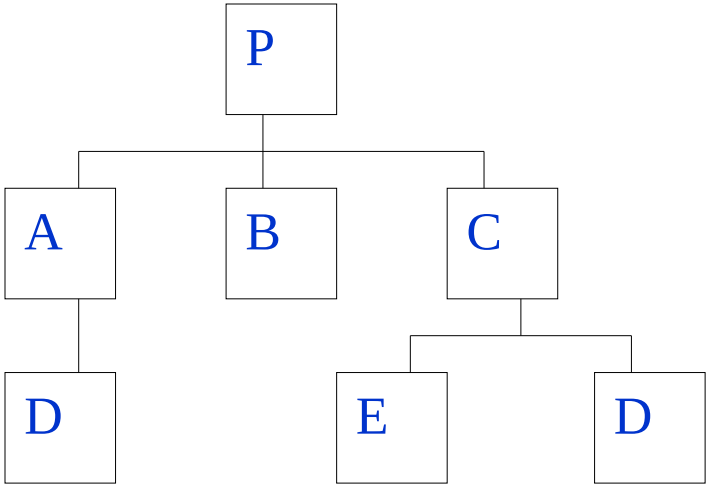
Escopo Estático

- Referência Seletiva em ADA

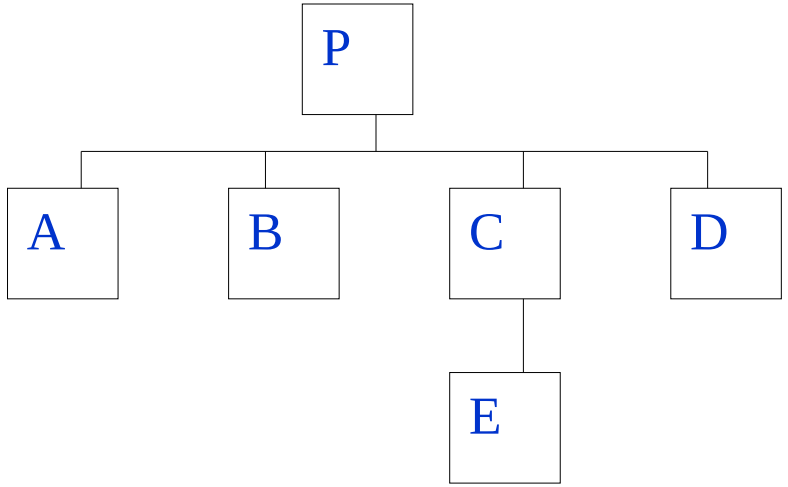
```
procedure A is
    x : INTEGER;
    procedure B is
        y : INTEGER;
        procedure C is
            x : INTEGER;
        begin
            x := A.x;
        end C;
    begin
        null;
    end B;
begin
    null;
end A;
```

Escopo Estático

- Problemas com Estrutura Aninhada



a



b

Escopo Estático

- Estrutura de Blocos de C

```
int x = 10;
int y = 15;
void f() {
        if (y - x) {
          int z = x + y;
        }
}
void g() {
    int w;
    w = x;
}
```

```
void main() {
    f();
    x = x + 3;
    g();
}
```

Escopo Dinâmico

```
procedimento sub() {
        inteiro x = 1;
        procedimento sub1() {
          escreva( x);
        }
        procedimento sub2() {
          inteiro x = 3;
           sub1();
        }
        sub2();
        sub1();
}
```

Escopo Dinâmico

- Problemas
  - Eficiência
  - Legibilidade
  - Acesso
  - Confiabilidade
- Não usado por LPs

Definições e Declarações

- Definições produzem amarrações entre identificadores e entidades criadas na própria definição
- Declarações produzem amarrações entre identificadores e entidades já criadas ou que ainda o serão

Definições e Declarações

- Localização de Definições de Variáveis em C++

```
void f() {
   int a = 1;
   a = a + 3;
   int b = 0;
   b = b + a;
}
```

Declaração de Constantes

- Em C

```
const float pi = 3.14;
#define pi 3.14
```

- Em JAVA

```
final int const1  = 9;
static final int const2  = 39;
final int const3 = (int)(Math.random()*20);
static final const4 = (int)(Math.random()*20);
final int j;
Construtor () {
    j = 1;
}
```

Definições e Declarações de Tipos

Definições Tipos em C

```
struct data {
    int d, m, a;
};
```

```
union angulo {
    int graus;
    float rad;
};
```

```
enum dia_util {
    seg,  ter,  qua,
    qui, sex
};
```

Declarações Tipos em C

```
struct data;
typedef union angulo curvatura;
typedef struct data aniversario;
```

Definições e Declarações de Variáveis

Definições de Variáveis em C

```
int k;
union angulo ang;
struct data d;
int *p, i, j, k, v[10];
```

Definições com Inicialização

```
int i = 0;
char virgula = ',';
float f, g = 3.59;
int j, k, l = 0, m=23;
```

Definições e Declarações de Variáveis

- Definições com Inicialização Dinâmica

```
void f(int x) {
   int i;
   int j = 3;
   i = x + 2;
   int k = i * j * x;
}
```

- Definições com Inicialização em Variáveis Compostas

```
int v[3] = { 1, 2, 3 };
```

Definições e Declarações de Variáveis

- Declaração de Variáveis em C

```
extern int a;
```

- Declaração de Variáveis em C++

```
int r = 10;
int &j = r;
j++;
```

Definições e Declarações de Subprogramas

- Definição de Subprogramas em C

```
int soma (int a, int b) {
   return a + b;
}
```

- Declaração de Subprogramas em C

```
int incr (int);
void f(void) {
   int k = incr(10);
}
int incr (int x) {
   x++;
   return x;
}
```

Definições Compostas Seqüenciais

- Definições Seqüenciais em C

```
struct funcionario {
   char nome [30];
   int matricula;
   float salario;
};
struct empresa {
   funcionario listafunc [1000];
   int numfunc;
   float faturamento;
};
int m = 3;
int n = m;
```

Definições Compostas Seqüenciais

- Definições Seqüenciais em ML

```
val par = fn (n: int) => (n mod 2 = 0)
val negacao = fn (t: bool) => if t then false else true
val impar = negacao o par
val jogo = if x < y then par else impar
```

Definições Compostas Recursivas

- Definição Recursiva de Função em C

```
float potencia (float x, int n) {
   if (n == 0) {
      return 1.0;
   } else if (n < 0) {
      return 1.0/ potencia (x, -n);
   } else {
      return x * potencia (x, n - 1);
   }
}
```

- Tipo Recursivo em C

```
struct lista {
   int elemento;
   struct lista * proxima;
};
```

Definições Compostas Recursivas

- Definições Mutuamente Recursivas em C

```
void segunda (int);
void primeira (int n) {
   if (n < 0) return;
   segunda (n – 1);
}
void segunda (int n) {
   if (n < 0) return;
   primeira (n – 1);
}
```

Definições Compostas Recursivas

- Erro em Definição de Função strcmp em C

```
int strcmp (char *p, char *q) {
    return !strcmp (p, q);
}
```

- Explicitação de Recursividade em Função ML

```
val rec mdc = fn ( m:int, n: int) = >
   if m > n then mdc (m – n, n)
   else if m < n then mdc (m, n – m)
   else m
```