Lista de Exercícios de Linguagens de Programação IV
Universidade Federal do Amazonas
Departamento de Ciência da Computação
Marco Cristo

Procedural: Nomes, Ligações, Escopos e Procedimentos

1) O que é um apelido (alias)? Quais as vantagens e desvantagens relacionadas com o uso de apelidos?

   Um nome alternativo para um objeto. Vantagens: legibilidade de código. Desvantagens: possibilidade de referências inválidas e inconsistências.

2) O que é escopo?

   Trecho de um programa para o qual um conjunto de ligações é válida.

3) O que é um bloco? Qual a relação comum entre bloco e escopo, introduzida pelo Algol e válida para a maioria das linguagens até hoje?

   Bloco é um conjunto de instruções de um programa determinado por indicadores de início e fim do conjunto. O Algol introduziu a regra de que todo o bloco determina um escopo.

4) Quais as vantagens e desvantagens de ligações estáticas?

   Vantagem: código eficiente, ciclo de vida perene, não necessidade de pilha; Desvantagem: perda de flexibilidade como, por exemplo, o uso de tanta memória quanto necessário ou limitações na capacidade de recursão.

5) Quais são as (mais importantes) vantagens e desvantagens de linguagens tipadas?

   Vantagem: melhora legibilidade na medida em que se representam estruturas de dados complexas, facilita a detecção de erros, possibilita certas otimizações em tempo de compilação; Desvantagem: pode dificultar a legibilidade e a conveniência na escrita de programas pequenos ou que usam apenas tipos de dados simples.

6) **PROJETO**: Escreva um programa em C com três funções. Uma declara um vetor de cem milhões de entradas na pilha, outro no *heap* e outro estaticamente. Chame cada uma das funções cem mil vezes e mostre o tempo necessário para cada caso. Como você explica este resultado?

7) O que é um procedimento (sub-programa)?

   Parte de um programa que executa uma tarefa específica e é relativamente independente do restante do código. É caracterizado por possuir uma única entrada (salvo raras exceções como em Fortran), reter o controle de execução do programa até ser finalizado, quando retorna o controle ao código que o invocou.

8) Qual a diferença entre parâmetros formais e reais?

Formais: aqueles definidos no cabeçalho e usados no corpo do procedimento (definidos no procedimento). Reais: aqueles passados pelo código que invoca o procedimento (definidos no chamador).

9) Qual a diferença entre funções sobrecarregadas, polimórficas e genéricas?

Sobrecarregadas: funções distintas que possuem o mesmo nome; polimórficas: funções que apresentam natureza distinta quando aplicadas a diferentes sub-tipos de um tipo; genéricas: a mesma função aplicada a diferentes tipos de dados.

10) Qual a vantagem de operadores sobrecarregados?

Facilitam a legibilidade e a conveniência de escrita de muitas operações.

11) Considere o seguinte programa em sintaxe C:

```
void swap(int a, int b) {
   int temp; temp = a; a = b; b = temp;
}
void main() {
        int value = 2, list[5] = {1, 3, 5, 7, 9};
        swap(value, list[0]);
        swap(list[0], list[1]);
        swap(value, list[value]);
}
```

Quais são os valores das variáveis *value* e *list* depois de cada instrução *swap*, considerando que a passagem de parâmetros é feita por (a) valor, (b) referência, (c) valor-resultado?

```
a)  value = 2, list = {1, 3, 5, 7, 9}
    value = 2, list = {1, 3, 5, 7, 9}
    value = 2, list = {1, 3, 5, 7, 9}

b)  value = 1, list = {2, 3, 5, 7, 9}
    value = 1, list = {3, 2, 5, 7, 9}
    value = 2, list = {3, 1, 5, 7, 9}

c)  value = 1, list = {2, 3, 5, 7, 9}
    value = 1, list = {3, 2, 5, 7, 9}
    value = 2, list = {3, 1, 5, 7, 9}
```

12) O código abaixo representa uma passagem de parâmetro por referência em C++. Que modificações você faria em foo para conseguir uma semântica de passagem de parâmetros de acordo com o modelo valor-resultado:

```
ftype_t foo(ptype_t &parameter) {
   ... foo statements using parameter ...
}
```

No modelo valor-resultado, o procedimento manipula uma cópia do dado original e ao fim da execução salva o valor da cópia no dado original. Logo, a seguinte versão de foo representa a semântica de uma passagem valor-resultado em C++, de forma mais fiel:

```
ftype_t foo(ptype_t &copy) {
   ptype_t parameter = copy;
   ... foo statements using parameter...
   copy = parameter;
}
```

13) Considere o seguinte pseudo-código.

```
1. procedure main
2.          a: integer := 1
3.          b: integer := 2
4.          procedure middle
5.                  b: integer := a
6.                  procedure inner
7.                          print a, b
8.                          a: integer := 3
9.                  -- body of middle
10.                 inner()
11.                 print a, b
12.         -- body of main
13.         middle()
14.         print a, b
```

Suponha que este código foi feito em uma linguagem que possui as mesmas regras de ordem de declaração de C (mas com procedimentos aninhados, como em Pascal) – isto é, nomes devem ser declarados antes de serem usados e o escopo de um nome se estende de sua declaração até o fim do bloco. Para cada *print*, indique que declarações de a e b estão no ambiente de referência. O que o programa imprime? Ou o compilador irá detectar erros semânticos estáticos? Repita o exercício considerando as regras de ordem de declaração do C# (nomes devem ser declarados antes de usados, mas o escopo do nome é o bloco inteiro em que é declarado) e Modula-3 (nomes podem ser declarados em qualquer ordem e escopo é o bloco inteiro em que foi declarado).

Com as regras do C, a linha 7 se refere aos a e b declarados nas linhas 2 e 5, respectivamente; linha 11 se refere aos a e b declarados nas linhas 8 e 5, respectivamente; e a linha 14 se refere ao a e b declarados nas linhas 2 e 3, respectivamente. O programa imprime 1 1 3 1 1 2. Com regras do C#, o compilador iria gerar erros por uso antes de definição para a nas linhas 5 e 7. Com as regras do Modula-3, a linha 7 se refere ao a e b declarado nas linhas 8 e 5, respectivamente; a linha 11 se refere ao a e b declarados nas linhas 8 e 5, respectivamente; e a linha 14 se refere ao a e b declarados nas linhas 2 e 3, respectivamente. O programa imprime 3 3 3 3 1 2.

14) Considere o seguinte fragmento de código C:

```
{       int a, b, c;
        ...
        {
           int d, e;
           ...
           {
                   int f;
                   ...
           }
           ...
        }
        ...
        {
           int g, h, i;
           ...
        }
        ...
}
```

Assuma que cada variável inteira ocupe 4 bytes. Qual o espaço total necessário para as variáveis neste código?

Variáveis a, b, e c existem ao longo da execução do bloco mais externo. Variáveis d, e, e f são necessárias apenas no primeiro bloco aninhado e terão seu espaço em memória posteriormente ocupado por g, h e i. Assim, um total de 4 × 6 = 24 bytes é necessário.

15) Considere o seguinte pseudo-código:

```
x : integer -- global

procedure set x(n : integer)
        x := n

procedure print x
        write integer(x)

procedure foo(S, P : function; n : integer)
        x : integer := 5
        if n in {1, 3}
           set x(n)
        else
           S(n)
        if n in {1, 2}
           print x
        else
           P

set x(0); foo(set x, print x, 1); print x
set x(0); foo(set x, print x, 2); print x
set x(0); foo(set x, print x, 3); print x
set x(0); foo(set x, print x, 4); print x
```

Assuma que a linguagem usa escopo dinâmico. O que o programa imprime se a linguagem usa ligação superficial? E se usa ligação profunda? Porquê?

Com ligação superficial, set_x e print_x sempre acessam a variável local x de foo. O programa imprime 1 0 2 0 3 0 4 0. Com ligação profunda, set_x acessa a global x quando n é par e a local x de foo quando n é ímpar. Similarmente, print_x acessa a global x quando n é 3 ou 4 e o local x de foo quando n é 1 ou 2. O programa imprime 1 0 5 2 0 0 4 4

16) Quais as diferenças entre funções matemáticas e funções em linguagem procedurais?

Uma função matemática mapeia valores do seu domínio para a sua imagem de forma determinística. Uma função pode interagir com o mundo de muitas outras formas. Ela pode produzir valores que não são determinísticos ou que dependem de outros elementos que não seus parâmetros reais (ex: variáveis estáticas e globais ou entradas do mundo real). Ela também pode mudar seu ambiente por meio de efeitos colaterais. Estes não incluem apenas mudanças em variáveis globais ou saída, mas mudanças nos valores dos parâmetros passados se eles não são passados por valor. Além disso, note que enquanto funções matemáticas podem ser intratáveis ou não computáveis, este não é o caso de funções em linguagens de programação.

17) Considere uma expressão como a + b passada para uma sub-rotina em Fortran. Há alguma diferença semântica entre passar esta expressão como uma referência para um valor temporário sem nome (o caso de Fortran) ou passá-la por valor (como em C ou PASCAL)? Isto é, o programador pode dizer a diferença entre um parâmetro que é um valor e uma referência para um temporário?

O programador pode pensar nisso como coisas distintas, mas o comportamento é rigorosamente o mesmo.