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

Procedural: Tipos de Dados

1) O que acontece quando um elemento não existente de um vetor é referenciado em C?

   O programa acessa a memória na posição solicitada, mesmo embora esta posição não faça parte da área alocada ao vetor (se isso não for uma operação ilegal como, por exemplo, uma violação de uma área de memória destinada a outro programa pelo sistema operacional). Nenhum aviso é fornecido.

2) Quais os tipos comuns de problemas com ponteiros?

   [1] Apontar para uma área de memória desalocada, o que pode causar (a) checagens de tipo inválidas se área foi realocada para uma variável de tipo diferente daquela que o ponteiro apontava; (b) inconsistência se área foi realocada para uma variável do mesmo tipo daquela que o ponteiro apontava e este é usado para modificar seu valor; (c) falha do sistema de gerência de armazenamento se área passou a ser usada por ele para controle interno; (d) possibilidade de violação de abstrações de dados; (e) dificulta legibilidade ao exigir a deferência explícita em certas linguagens; [2] Deixar de apontar para uma área alocada o que pode tornar esta área inatingível se nenhum outro ponteiro pode referenciá-la (isto é conhecido como *memory leak*). [3] Apontar para qualquer área da memória, o que pode implicar em falhas de segurança e possibilidade de causar estragos em outros programas em execução (caso específico de C/C++).

3) Por que em várias linguagens um ponteiro é restrito a apontar para dados do mesmo tipo?

   Para facilitar o cálculo de endereços. Se p aponta para uma posição i da memória, p+1 aponta para a posição i+(tamanho tipo apontado por p). Note que se p pudesse apontar para vários tipos, seria necessário calcular a memória com endereçamento variável, o que é problemático.

4) Qual o uso mais comum de referências em C++? Por que o seu uso é aconselhado em lugar de ponteiros?

   Passagem de parâmetros por referência. Referências em C++, após inicializadas, não podem mais se referir a um novo objeto nem podem ser usadas como base para o cálculo de outros endereços. Assim, o uso de referências minimiza o acesso ao espaço de endereçamento do chamador (já que este não pode mais apontar pra um endereço relativo ao do ponteiro), contribui para a escrita de código mais legível (já que não precisa fazer deferências explícitas do tipo *p) e minimiza a possibilidade de uma 'alocação órfã' (já que a referência não pode ser usada para apontar pra outro lugar).

5) O que é uma linguagem fortemente tipada?

   É uma linguagem em que erros de tipos são sempre detectados (note que este é um conceito um tanto frouxo, por exemplo, devido a regras de coerção). Para isso acontecer, é necessário que *todos* os operandos tenham seus tipos determinados, em tempo de compilação ou em tempo de execução. Este não é o caso de C++, por exemplo, que

permite o tipo *union* que não é checado e abusa de coerção. Note que embora linguagens como Ada, C# e Java sejam consideradas fortemente tipadas (porque não há mecanismos implícitos que possibilitem que um erro de tipo não seja detectado), nas três, o programador pode forçar tipos com *casting* explícito.

6) Quais as principais vantagens e desvantagens na representação de tipos booleanos com um único bit?

   Vantagem: economia de memória. Desvantagem: necessidade de alinhamento e acesso alinhado, pois arquiteturas de hardware são projetadas para a recuperação eficiente de bytes na memória e não bits.

7) Considerando segurança e custo de implementação, o que é melhor: *tombstone* ou *lock-and-key*?

   Segurança: tombstones, pois não se limita ao *heap* e garante a unicidade de mapeamento entre ponteiro e objeto; Implementação: lock-and-key, pois se limita ao heap e não mantém estruturas para sempre o que evita inclusive ter que descobrir quando liberá-las.

8) Quais os ganhos e perdas de Java ao decidir não incluir ponteiros, como em C++?

   Ganhos: evita problemas relacionados com uso de ponteiros já descrita em resposta anterior. Perdas: não permite o livre endereçamento da memória, o que impossibilita o acesso direto, sem intermediação, à memória usada por outros processos, por exemplo.

9) A instrução, em C, &(&i) é válida?

   Não. Não faz sentido obter o endereço de um endereço de memória.

10) Suponha que estamos compilando para uma máquina com caracteres de 1 byte, shorts de 2 bytes, inteiros de 4 bytes, reais de 8 bytes e com regras de alinhamento que requerem que cada dado primitive seja um múltiplo par do tamanho do elemento. Além disso, suponha que o compilador não reordena os campos. Quanto espaço será consumido pelo vetor abaixo. Explique.

```
a: array[0..9] of tuple
    s: short
    c: char
    t: short
    d: char
    r: real
    i: integer
```

    240 bytes. Dentro de cada elemento da matriz, s tem offset 0, c tem offset 2, t tem offset 4 (forçado por alinhamento), d tem offset 6, r tem offset 8 (forçado por alinhamento), e i tem offset 16. Adicionando o tamanho de i, temos 20 bytes, mas 20 não é múltiplo de 8. Nós então preenchemos cada elemento até 24 bytes de forma que r esteja alinhado em cada elemento da matriz. Como A tem 10 elementos, o tamanho total é 240 bytes.

Procedural: Fluxo de Controle

11) O que é incomum na forma como Python define blocos?

    Uso de tabulação sem palavra reservada para definir início e fim de bloco.

12) O que é incomum no comando de múltipla seleção do C (*switch*)?

Ele não é mutuamente exclusivo.

13) Qual o equivalente em Python da instrução C: for (i = 0; i < n; i ++) ...

for i in range(0, n): …

14) Por que linguagens contemporâneas não incluem goto?

Porque a maioria dos usos do *goto* contribui para a escrita de código de difícil compreensão.

15) Quais as vantagens e desvantagens de usar palavras reservadas únicas para terminar instruções compostas? (ex: if...end if; while... end while; em ADA)

Vantagem: legibilidade. Desvantagem: mais palavras reservadas.

16) Descreva uma situação em que a cláusula *else* do *for* em Python seria conveniente:

Qualquer situação em que há uma assimetria entre a saída do laço ao seu término e antes do seu término, via *break*. Nestas situações normalmente um teste adicional com um *flag* é usado para checar se o laço terminou normalmente ou via break. No caso de Pyhton, o *else* pode ser usado, evitando o *flag*. Uma desvantagem é que ele pode causar confusão de leitura (o programador pode imaginar que o *else* é executado caso o teste falhe no início).

17) Reescreva o código abaixo em C e Pyhton, usando uma instrução de múltipla seleção:

```
if ((k = 1) or (k = 2)) j := 2 * k - 1
if ((k = 3) or (k = 5)) j := 3 * k + 1
if (k = 4) j := 4 * k - 1
if ((k = 6) or (k = 7) or (k = 8)) j := k - 2
```

```
/* C */
switch (k) {
   case 1:
   case 2: j = 2 * k – 1; break;
   case 3:
   case 5: j = 3 * k + 1; break;
   case 4: j = 4 * k – 1; break;
   case 6:
   case 7:
   case 8: j = k - 2;
}
```

```
# Python
if k = 1 or 2:
        j = 2 * k – 1
elif k = 3 or 5:
        j = 3 * k + 1
elif k = 4:
        j = 4 * k – 1
elif k = 6 or 7 or 8:
        j = k - 2;
```

18) Reescreva o código abaixo em C e Pyhton, eliminando breaks sem usar gotos:

```
j = -3
for (i = 0; i < 10; i ++) {
   switch (j + 2) {
            case 3:
```

```
            case 2: j--; break;
            case 0: j += 2; break;
            default: j = 0;
        }
        if (j > 0)
            break;
        j = 3 - i;
}

/* C */
int i, j;
j = -3;
for (i = 0; i < 10; i ++) {
   j = j == 0? j - 1: 0;
   if (j > 0) i = 10; else j = 3 - i;
}

# Python
j = -3;
for i in range(0, 10):
   j = j - 1 if j == 0 else 0
   if j > 0: i = 10 else: j = 3 - i
```

19) Considere o código escrito em C, que evita o processamento de linhas em branco:

```
for (;;) {
   linha = leia_linha();
   if (em_branco(linha))
            break;
   processe_linha(linha);
}
```

Mostre como realizar a mesma tarefa usando laços *while* ou *do…while*, sem usar o *break* no meio do código. Como estas alternativas se comparam ao método apresentado, usando *for* com *break*?

Alternativa 1:

```
linha = leia_linha();
while (!em_branco(linha)) {
        processe_linha(linha);
        linha = leia_linha();
}
```

Alternativa 2:

```
do {
        linha = leia_linha();
        if (!em_branco(linha))
            processe_linha(linha);
        processe_linha(linha);
} while (!em_branco(linha));
```

Em ambos os casos, há código duplicado em relação ao *for*. Note que pra eliminar o código duplicado, seria necessário criar variáveis adicionais.