Universidade Federal de Minas Gerais
Ciência da Computação

Linguagens de Programação - Haniel Barbosa

Lista de Exercícios 3

Respostas

1. Quais componentes de um programa devem ser armazenados na memória?

   (a) As variáveis, funções e seus resultados
   (b) O programa em si e os estados que ele mantém
   (c) O estado e os resultados de funções
   (d) As variáveis e suas atribuições

   Letra (b)

2. Marque V ou F para as alternativas que definem as características de cada tipo de memória em C:

   (a) A memória estática tem gerenciamento automático V
   (b) A memória dinâmica alocada na pilha utiliza gerenciamento manual F (gerenciamento automático)
   (c) A memória dinâmica alocada na heap utiliza gerenciamento automático F (gerenciamento manual)
   (d) A memória dinâmica não é flexível F (é flexível)
   (e) A memória dinâmica tem gerenciamento mais complexo do que memória estática V
   (f) A memória estática armazena variáveis globais V
   (g) A memória dinâmica alocada na pilha utiliza gerenciamento automático V
   (h) A memória dinâmica alocada na heap utiliza gerenciamento manual V

3. Considerando os diferentes tipos de memória em C:

   (a) classifique qual tipo de memória está sendo utilizada para cada variável do programa em C apresentado a seguir
   valor_inicial: memória estática
   valor_intermediario: memória estática
   taxa: memória dinâmica
   valores: memória dinâmica

(b) apresente quais valores serão gerados para a lista construída ao fim da execução

valores[0] = 25
valores[1] = 75
valores[2] = 15

```c
#include <stdio.h>

int valor_inicial = 10;

int valor_intermediario = 5;

void calcula (int * valores ) {
   int taxa = 3;
   valores [0] = valor_inicial + valor_intermediario * taxa ;
   valores [1] = valores [0] * 3;
}

int main () {
   int * valores = (int *) malloc (3 * sizeof (int) ) ;
   calcula ( valores ) ;
   valores [2] = valor_inicial + valor_intermediario ;
}
```

4. Cite três exemplos de coletores de lixos que podem ser utilizados por uma linguagem de programação. Apresente um exemplo de aplicação para o qual um dos modelos, à sua escolha, é o mais adequado e justifique.

   Reference Counting Algorithm: Ideal para sistemas de monitoramento em tempo real já que a execução do coletor de lixo não vai parar a execução dele. O sistema também pode ter um trafego com muitas informações e grande quantidade de acesso de memória. Já que não precisa varrer o heap inteiro para poder desalocar uma determinada parte da memória.
   Copying Algorithm: Ideal para Servidores de email em que há um alto trafego de mensagens e precisa alocar e desalocar memória e nesse coletor isso ocorre de forma rápida, além de servidores de email não precisar funcionar em real time já que o coletor precisa da aplicação parada para poder ser executado.
   Mark and sweep: Ideal para sistemas de padaria onde não precisa alocar e desalocar memória com tanta frequencia já que o coletor de lixo precisa varrer o heap inteiro na fase de marcação, além de não precisar funcionar em real time já que o coletor precisa da aplicação parada para poder ser executado.

5. Considere uma heap e seu administrador abaixo, a classe *HeapManager*, em Python. Ela utiliza uma estratégia *first-fit* para encontrar o primeiro bloco de memória grande o suficiente para alocar uma requisição. Porém ela possui apenas a opção para alocamento de espaço na memória. Você deverá implementar a função *deallocate()*, para assim finalizar as funcionalidades de gerenciamento desta heap.

```
1  NULL = -1 # The null link
2
3  class HeapManager :
4    """ Implements a very simple heap manager ."""
5
6    def __init__(self,memorySize) :
7      """ Constructor . Parameter initialMemory is the array of
8          data that we will
9          use to represent the memory ."""
10     self.memory = [0] * memorySize
11     self.memory[0] = self.memory.__len__()
12     self.memory[1] = NULL
13     self.freeStart = 0
14
15   def allocate(self, requestSize) :
16     """Allocates a block of data , and return its address . The
           parameter requestSize is the amount of space that must
         be allocaed."""
17
18     size = requestSize + 1
19     # Do first-fit search: linear search of the free list for
          the first block
20     # of sufficient size .
21     p = self.freeStart
22     lag = NULL
23     while p != NULL and self.memory[p] < size :
24       lag = p
25       p = self.memory[p + 1]
26     if p == NULL :
27       raise MemoryError()
28     nextFree = self.memory[p + 1]
29     # Now p is the index of a block of sufficient size ,
30     # lag is the index of 'ps predecessor in the
31     # free list , or NULL , and nextFree is the index of
32     # 'ps successor in the free list , or NULL .
33     # If the block has more space than we need , carve
34     # out what we need from the front and return the
35     # unused end part to the free list .
36     unused = self.memory[p] - size
37     if unused > 1:
38       nextFree = p + size
39       self.memory[nextFree]= unused
40       self.memory[nextFree + 1] = self.memory[p + 1]
41       self.memory[p]= size
42     if lag == NULL :
43       self.freeStart = nextFree
44     else :
```

```
45        self.memory[lag + 1] = nextFree
46
47      return p + 1
48    def deallocate(self, address):
49      address = address - 1
50      p = self.freeStart
51      lag = NULL
52      while p != NULL and p < address:
53        lag = p
54        p = self.memory[p + 1]
55      if lag != NULL:
56        self.memory[lag + 1] = address
57      else:
58        self.freeStart = address
59      self.memory[address + 1] = p
```

6. Outra estratégia para gerenciamento de uma heap é o *best-fit*. Esta estratégia consiste em percorrer a lista de blocos livres em busca do pedaço de memória que seja o menor possível mas que seja grande o suficiente para comportar a área de memória requisitada. Se for encontrada uma área exatamente do tamanho da requisição, então pode-se interromper a busca, retornando a área encontrada. Do contrário, toda a lista deve ser percorrida, em busca do melhor pedaço de memória. A vantagem de *best-fit* é que esta estratégia não quebra áreas de memória muito grandes desnecessariamente. Se houver uma área de tamanho exato, *best-fit* a encontrará, não tendo, portanto, de quebrar nenhum bloco neste caso. Sendo assim, você deve implementar uma nova versão da classe *HeapManager* que utilize esta política de alocação de memória. Comece com uma cópia de *HeapManager* e então modifique o método allocate para implementar esta estratégia.

```
1  NULL = -1 # The null link
2  INF = float("inf")
3
4  class HeapManager :
5    """ Implements a very simple heap manager ."""
6
7    def __init__(self,memorySize) :
8      """ Constructor . Parameter initialMemory is the array of
9          data that we will
10         use to represent the memory ."""
11     self.memory = [0] * memorySize
12     self.memory[0] = self.memory.__len__()
13     self.memory[1] = NULL
14     self.freeStart = 0
15
16   def allocate(self, requestSize) :
17     """Allocates a block of data , and return its address . The
           parameter requestSize is the amount of space that must
```

```
      be allocaed."""
18
19    size = requestSize + 1
20    # Do first-fit search: linear search of the free list for
         the first block
21    # of sufficient size .
22    p = self.freeStart
23    lag = NULL
24    bestLag = NULL
25    bestBlockSize = INF
26    bestBlock = NULL
27    while p != NULL and bestBlockSize > size:
28      if self.memory[p] >= size and self.memory[p] <
           bestBlockSize:
29        bestBlock = p
30        bestBlockSize = self.memory[p]
31        bestLag = lag
32      lag = p
33      p = self.memory[p + 1]
34    if bestBlock == NULL:
35      raise MemoryError()
36    lag = bestLag
37    p = bestBlock
38    nextFree = self.memory[p + 1]
39    # Now p is the index of a block of sufficient size ,
40    # lag is the index of 'ps predecessor in the
41    # free list , or NULL , and nextFree is the index of
42    # 'ps successor in the free list , or NULL .
43    # If the block has more space than we need , carve
44    # out what we need from the front and return the
45    # unused end part to the free list .
46    unused = self.memory[p] - size
47    if unused > 1:
48      nextFree = p + size
49      self.memory[nextFree]= unused
50      self.memory[nextFree + 1] = self.memory[p + 1]
51      self.memory[p]= size
52    if lag == NULL :
53      self.freeStart = nextFree
54    else :
55      self.memory[lag + 1] = nextFree
56
57    return p + 1
58  def deallocate(self, address):
59    address = address - 1
60    p = self.freeStart
61    lag = NULL
```

```
62      while p != NULL and p < address:
63        lag = p
64        p = self.memory[p + 1]
65      if lag != NULL:
66        self.memory[lag + 1] = address
67      else:
68        self.freeStart = address
69      self.memory[address + 1] = p
```

7. Muitas linguagens de programação não possuem qualquer mecanismo de coleta automática de lixo. Um exemplo típico é C++. Ainda assim, é possível programar de forma mais segura via bibliotecas. Uma estratégia comumente adotada em C++ e baseada no uso de ponteiros deslocados automaticamente. Uma possível implementação deste tipo de ponteiro é dada logo abaixo:

```
1 template <class T > class auto_ptr {
2   private : T * ptr ;
3   public :
4   explicit auto_ptr ( T * p = 0) : ptr ( p ) { }
5   ~ auto_ptr () { delete ptr ; }
6   T & operator *() { return * ptr ; }
7   T * operator - >() { return ptr ; }
8 };
```

(a) A classe *auto ptr* utiliza pelo menos dois tipos diferentes de polimorfismo. Que tipos de polimorfismos são estes?

Polimorfismo paramêtrico presente na linha número 1 em que temos um tipo genérico T que pode representar diferentes tipos. Sobrecarga de operadores presentes na linha 6 e 7 representado pelos operadores * e -¿

(b) A função abaixo contém um problema de memória ou não? Em caso afirmativo, explique que falha é esta. Utilize a ferramenta valgrind para analisar este programa, por exemplo, tentando o comando **valgrind -v ./a.out**. Considere que uma falha de memória leva *valgrind* a fornecer algum aviso. Caso o erro não exista, justifique a sua resposta:

```
1 void foo0 () {
2  auto_ptr<std::string> p(new std::string("I did one P.O.F!\
3 n"));
4  std::cout << * p ;
5 }
```

Não ocorre problema de memória porque o destrutor da classe auto_pr ao final da execução libera o ponteiro alocado.
Podemos comprovar isso analisando o programa executando o comando do valgrind acima tendo como saída as mensagens:
All heap blocks were freed – no leaks are possible
ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

(c) Novamente: problema de memória ou não? Em caso afirmativo, explique que falha é esta. Em caso negativo, justifique. Note que exceções, neste caso, funcionam como em Java ou Python:

```
void foo1 (){
 try{
 auto_ptr<std::string> p(new std :: string ("Oi!\n") ) ;
 throw 20;
 } catch(int e){std::cout << "Oops:" << e << "\n";}
}
```

Mesmo caso/resposta da letra b.

(d) Última pergunta: problema de memória ou não? Em caso afirmativo, explique que falha é esta. Em caso negativo, justifique a sua resposta:

```
void foo2 (){
 try{
 std::string * p = new std :: string("Oi!\n");
 throw 20;
 delete p;
 } catch(int e) {std::cout << "Oops: " << e << "\n";}
}
```

Sim ocorre problema de memória, devido a perda de refência do ponteiro sendo assim a memória não será desalocada gerando memory leak.
Analisando o programa executando o comando do valgrind temos como saída as mensagens:
LEAK SUMMARY:
definitely lost: 32 bytes in 1 blocks
indirectly lost: 0 bytes in 0 blocks
...

8. Nesta questão você deve implementar um Tipo Abstrato de Dado em SML para representar uma biblioteca Math que lida com números inteiros e fornece algumas funções para o usuário. Você deve definir tanto a especificação quanto a implementação. O nome de sua structure deve ser MyMathLib, e você deve implementar quatro operações:

- fact: calcula o fatorial de um número
- halfPi: constante representando metade do valor de Pi
- pow: dado uma base e um expoente, calcule a potência
- double: dobra um número

NOTA: o valor halfPi deve ser real.

input: MyMathLib.pow(2,3)
output: val it = 8 : int

input: MyMathLib.double(6)
output: val it = 12 : int

```
signature MATH =
sig
  val fact : int -> int
  val halfPi : real
  val pow : int * int -> int
  val double : int -> int
end;

structure MyMathLib :> MATH =
struct
 fun fact 0 = 1
    | fact x=x*fact (x-1)
 val halfPi = Math.pi/2.0
 fun pow(n,0) = 1
    | pow (n,x) = n*pow (n, x-1)
 fun double x=x*2
end;
```

9. Utilizando a classe Node descrita abaixo, defina, em Python, um tipo abstrato de dado Stack que armazena objetos do tipo Node. O campo armazena uma string e o campo n aponta para o próximo elemento na pilha.

```
class Node :
    def __init__ ( self ) :
        self . n = 0
        self . e = ''
```

Você deve implementar os seguintes métodos:

- **add:** adiciona um Node na pilha
- **remove:** remove um Node da pilha e retorna o elemento desse Node
- **isNotEmpty:** retorna True se a pilha não é vazia, e False caso contrário.

O construtor deve iniciar o topo da pilha com um Node vazio.
Exemplo:

```
>>> s = Stack ()
>>> s . add (" Baltimore ")
>>> s . add (" Lord ")
>>> s . add (" Sir")
>>> s . isNotEmpty ()
True
>>> while ( s . isNotEmpty () ) :
    print ( s . remove () )
Sir
Lord
Baltimore
```

```
class Node:
  def __init__(self):
    self.n = None
    self.e = ''

class Stack:
  def __init__(self):
    self.head = Node()
  def add(self, val):
    node = self.head
    while node.n is not None:
      node = node.n
    newNode = Node()
    newNode.e = val
    node.n = newNode
  def remove(self) -> Node:
    node = self.head
    while node.n.n is not None:
      node = node.n
      pass
    ret = node.n
    node.n = None
    return ret
  def isNotEmpty(self) -> bool:
    return self.head.n is not None
```

10. Utilizando a mesma classe Node, defina agora, também um Python, um tipo abstrato de dado MinStack que implemente os mesmos métodos que Stack mais um método **getSmaller()**, o qual retorna o menor elemento da pilha. Note que este método apenas retorna esse elemento, não o remove da pilha. Você pode modificar a classe Node para esta implementação desde que ela ainda funcione para o TAD Stack, definido na questão anterior. **NOTA:** para comparar strings lexicograficamente, utilize os operadores relacionais: $<$, $>$, $\leq$, $\geq$, $=$, $\neq$.

    Exemplo:

```
>>> ms = MinStack ()
>>> ms . add ("C")
>>> ms . add ("A")
>>> ms . add ("B")
>>> ms . isNotEmpty ()
True
>>> ms . getSmaller ()''
A
```

```
class Node:
  def __init__(self):
    self.n = None
```

```
    self.e = ''
    self.mine = '{'

class MinStack:
  def __init__(self):
    self.head = Node()
  def add(self, val):
    node = self.head
    while node.n is not None:
      node = node.n
    newNode = Node()
    newNode.e = val
    newNode.mine = min(val, node.mine)
    node.n = newNode
  def remove(self) -> Node:
    node = self.head
    while node.n.n is not None:
      node = node.n
    ret = node.n
    node.n = None
    return ret
  def isNotEmpty(self) -> bool:
    return self.head.n is not None
  def getSmaller(self) -> str:
    node = self.head
    while node.n is not None:
      node = node.n
    return node.mine
```

11. Considere o método removeAll abaixo e responda:

```
def removeAll ( s ) :
    """ Removes all the elements from the data structure ."""
    while ( s . isNotEmpty () ) :
        print ( s . remove () )
```

(a) Qual o "contrato" que deve ser garantido pelos objetos passados para este método? Isto é pelos elementos de s passado para o método?

O contrato que deve ser garantido é que s independente do seu tipo, possa implementar as duas funções isNotEmpty e remove

(b) O que significa a expressão *duck typing*? E qual sua relação com este método?

A expressão refere-se a um tipo de codificação de linguagem dinamicamente tipada, em que o tipo implementado não é relevante desde que o seu comportamento seja o adquado."Se ele grasna como um pato e anda como um pato, ele é um pato" ou seja, se ele implementa o que o programa espera, seu tipo é adequado/desejado.
A relação que temos com o método acima é: s pode assumir qualquer tipo desde

que ele tenha seu comportamento esperado, ou seja conseguir que as funções isNotEmpty e remove possam ser implementadas.

12. Considere o programa abaixo e responda o que acontecerá em cada linha numerada. As opções possíveis são:

    (i) Algo será impresso. Neste caso, escreva o que será impresso.

    (ii) Um erro será produzido em tempo de execução

```
class Animal :
    def __init__ ( self , name ) :
        self . name = name
    def __str__ ( self ) :
        return self . name + " is an animal "

    def eat ( self ) :
        print ( self . name + ", which is an animal , is eating
            .")

class Mammal ( Animal ) :
    def __str__ ( self ) :
        return self . name + " is a mammal "

    def suckMilk ( self ) :
        print ( self . name + ", which is a mammal , is sucking
milk .")

class Dog ( Mammal ) :
    def __str__ ( self ) :
        return self . name + " is a dog "

    def bark ( self ) :
        print ( self . name + " is barking rather loudly .")

    def eat ( self ) :
        print ( self . name + " barks when it eats .")
        self . bark

def test () :
    a1 = Animal (" Pavao ")
    a2 = Mammal (" Tigre ")
    a3 = Dog (" Krypto ")
    print ( a1 ) # 1
    print ( a2 ) # 2
    print ( a3 ) # 3
    a1 . eat () # 4
    a2 . suckMilk () # 5
```

```
a2 . eat () # 6
a3 . bark () # 7
a3 . suckMilk () # 8
a3 . eat () # 9
a1 . bark () # 10
a1 = a3
a1 . bark () # 11
```

#1 - "Pavao is an animal"
#2 - "Tigre is a mammal"
#3 - "Krypto is a dog "
#4 - "Pavao, which is an animal, is eating"
#5 - "Tigre, which is a mammal, is sucking milk"
#6 - "Tigre, which is an animal, is eating"
#7 - "Krypto is barking rather loudly."
#8 - "Krypto, which is a mammal, is sucking milk."
#9 - "Krypto barks when it eats."
#10 - Erro
#11 - "Krypto is barking rather loudly."

13. Acerca de orientação a objetos, descreva o que é o "Problema do Diamante".

    É uma ambiguidade que surge quando duas classes B e C herdam de A, e a classe D herda de B e C. Se houver um método em A que B e C substituíram, e D não o substitui, então, qual versão do método D herda: a de B ou a de C? Para resolver esse problema deve reescrever o código de forma que isso não ocorra, a forma de implementar a solução varia dependendo da linguagem.
    O nome "problema do diamante" é devido ao formato do diagrama de herança de classes nessa situação. Nesse caso, a classe A está no topo, B e C separadamente abaixo dela, e D une os dois na parte inferior para formar uma forma de diamante.

14. Nesta questão, você deve adicionar exceções no TAD criado na questão 8. Você deverá restringir todos os valores manipulados pelas funções definidas a números positivos. Você também deverá criar uma função useMyMathLib : `int * string -> unit` que utiliza os métodos de MyMathLib e imprime o resultado das operações utilizando a função print de SML. O primeiro parâmetro é um valor a ser usado nas funções e o segundo é uma string com a função a ser usada. No caso de pow, suponha que sempre estaremos elevando um valor x a `x^x`.

    Essa função deve tratar as exceções disparadas por MyMathLib, exibindo a mensagem "Não posso lidar com valores negativos!". Você é livre para modificar a implementação da questão 1 como achar melhor, desde que não modifique o comportamento esperado.

    input: useMyMathLib(2, "pow")
    output: 4 val it = () : unit

input: useMyMathLib(∼ 3, ”fact”)
output: Não posso lidar com números negativos val it = () : unit

```
exception NegativeNum

structure MyMathLib :> MATH =
struct
  fun fact 0 = 1
    | fact x= if x > 0 then x*fact (x-1) else raise NegativeNum
  val half_pi = Math.pi/2.0
  fun pow(n, 0) = 1
    | pow (n,x)= if n > 0 then n*pow (n, x-1) else raise
       NegativeNum
  fun double x= if x > 0 then  x*2 else raise NegativeNum
end;

fun useMyMathLib(n, opr) =
  let fun f n "fact" = print(Int.toString(MyMathLib.fact(n)))
    | f n "pow" = print(Int.toString(MyMathLib.pow(n,n)))
    | f n "double" = print(Int.toString(MyMathLib.double(n)))
    | f n _ = raise Match
  in
    f n opr
  end
  handle NegativeNum => print("ãNo posso lidar com únmeros
     negativos!");
```

15. Nesta questão você deverá escrever uma calculadora interativa em Python, a qual recebe pela entrada padrão operações com +, -, *, / em formato de string. Você deverá tratar os seguintes erros:

    - Se a entrada não consistir de 3 elementos, dispare uma **FormulaError**, que é uma exceção customizada com a mensagem “A entrada não consiste de 3 elementos”.
    - Tente converter a primeira e a segunda entrada para float, trate cada **ValueError** que acontecer e dispare uma FormulaError com a mensagem “O primeiro e o terceiro valor de entrada devem ser numeros”.
    - Se o segundo elemento não for nenhum dos operadores aritméticos descritos acima, dispare uma exceção com a mensagem “x não é um operador válido”.

    Exemplo:

```
>>> 1 + 1
2.0
>>> 1 +
Traceback ( most recent call last ) :
  File "8. py", line 35 , in < module >
  n1 , op , n2 = parse_input ( user_input )
```

```
    File "8. py", line 8 , in parse_input
    raise FormulaError '(A entrada nao consiste de 3 elementos
       ')
__main__ . FormulaError : A entrada nao consiste de 3 elementos
>>>
```