Universidade Federal de Minas Gerais
Ciência da Computação

Linguagens de Programação - Haniel Barbosa

Lista de Exercícios 3

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

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

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

```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.

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. Você pode assumir que o endereço recebido sempre é o início de um bloco alocado.

```python
NULL = -1 # The null link

class HeapManager :
  """ Implements a very simple heap manager ."""

  def __init__(self,memorySize) :
    """ Constructor . Parameter initialMemory is the array of
        data that we will
        use to represent the memory ."""
    self.memory = [0] * memorySize
    self.memory[0] = self.memory.__len__()
    self.memory[1] = NULL
    self.freeStart = 0

  def allocate(self, requestSize) :
    """Allocates a block of data , and return its address . The
        parameter requestSize is the amount of space that must
       be allocaed."""

```

```
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
49    def deallocate(self, address):
50      pass
51
52  def test () :
53    h = HeapManager(10)
54    a = h.allocate(4)
55    print("a = ", a , ", Memory = ", h.memory)
56    b = h.allocate(2)
57    print("b = ", b , ", Memory = ", h.memory)
58
59  test ()
```

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 requisi-

tada. 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.

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:

```
template <class T > class auto_ptr {
  private : T * ptr ;
  public :
  explicit auto_ptr ( T * p = 0) : ptr ( p ) { }
  ~ auto_ptr () { delete ptr ; }
  T & operator *() { return * ptr ; }
  T * operator - >() { return ptr ; }
};
```

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

   (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:

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

   (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";}
}
```

(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";}
}
```

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

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
```

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
```

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?

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

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
```

13. Acerca de orientação a objetos, descreva o que é o "Problema do 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 -> string` que utiliza os métodos de MyMathLib e retorna a string correspondente ao 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 `xˆx`.

    Essa função deve tratar as exceções disparadas por MyMathLib, exibindo a mensagem "Nao posso lidar com numeros negativos!". Você é livre para modificar a implementação da questão 8 como achar melhor, desde que não modifique o comportamento esperado.

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

    input: useMyMathLib(∼3, "fact")
    output: Nao posso lidar com numeros negativos! val it = () : unit

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
>>>
```