Linguagens de Programação

Trabalhos

Carlos Bazilio
carlosbazilio@id.uff.br
http://www.ic.uff.br/~bazilio/cursos/lp

Observações Gerais

- Todo trabalho de implementação deve ser hospedado num repositório de código aberto, como Github, Gitlab ou Bitbucket.
- Trabalhos de implementação precisam ser apresentados; caso sejam permitidos grupos (máximo duplas), todos precisam estar presentes ativamente durante a apresentação, pois a avaliação é individual.

Exercício 1

- Para a linguagem escolhida, faça uma apresentação sucinta (vídeo-aula) contendo:
  - Histórico
  - Público alvo
  - Principais características
  - Exemplo básico
  - Exemplo avançado

Exercício 1.1

- Para a linguagem escolhida no exercício anterior, busque em alguma documentação da linguagem por um trecho da gramática BNF-alike desta que descreva a definição e chamada de função

Exercício 2

- Faça um programa para simular a manipulação de memória num heap
- Segue um programa exemplo e a gramática da linguagem a ser utilizada:

```
heap best
new a 5
new b 3
new c 8
exibe
del b
b = a
exibe
new d 2
del b
heap worst
...
```

```
<programa> ::= <instrucao>+
<instrucao> ::= <set_heap> | <new> |
<del> | <exibe> | <atribui>
<set_heap> ::= 'heap { 'first | 'best | 'worst | 'next }
<new> ::= 'new <id> <number>
<del> ::= 'del <id>
<exibe> ::= 'exibe
<atribui> ::= <id> '= <id>
```

Exercício 2

- Para implementação deste exercício deve-se:
  - Definir uma estrutura de dados para armazenar o heap (possivelmente um vetor) que, inicialmente, precisa conter apenas valores booleanos
  - Definir uma lista, a qual irá manter o registro das áreas livres do heap; cada nó da lista deverá conter 2 valores: i) o endereço inicial da área livre no heap; ii) a qtd. de blocos livres contíguos da área
  - Considerar o programa de entrada com as instruções (para simplificar os testes), o qual pode ser copiado para um vetor de strings
  - Executar cada instrução do programa informado, alterando estas estruturas de maneira a refletir esta execução

Exercício 2

- Este trabalho pode ser feito em duplas
- Quem quiser fazer sozinho, pode implementar apenas 2 das 4 estratégias de alocação
- Quem quiser fazer em trio deve implementar alocação dinâmica; por exemplo, contagem de referências;
  - Para o exemplo de código dado, perceba o que ocorre quando o comando “del b” (após o primeiro “exibe”) é removido.

Exercício 3

- Implementar um coletor automático de lixo em C utilizando a técnica de contagem de referências
- Para tal, deve-se criar uma biblioteca estática que ofereça as seguintes funções:
  - *endereço malloc2(tamanho)* → esta função realiza a alocação dinâmica e retorna o endereço da área criada
  - *void atrib2(endereco, endereço2)* → esta função realiza a atribuição de ponteiros em C

Exercício 3
Exemplo de Código

```c
#include <contref.h>

int main(int argc, char const *argv[]) {
    int *v = malloc2(sizeof(int));
    *v = 10;
    int *w = malloc2(sizeof(int));
    dump();
    *w = 20;
    atrib2(v, w);
    dump();
    char *c = malloc2(sizeof(char));
    *c = 'Z';
    dump();
    atrib2(w, NULL);
    dump();
    return 0;
}
```

Exercício 4

- Implementação do algoritmo mark-sweep

Exercício 4.1

- Instalar o coletor de lixo para C e rodar o pequeno exemplo disponível neste link: https://github.com/ivmai/bdwgc/blob/master/doc/simple_example.md
- Mais informações sobre a biblioteca podem ser obtidas nestes 2 links: http://www.hboehm.info/gc/ e https://github.com/ivmai/bdwgc/

Exercício 5

- Implementação de um simulador/depurador para ~*bazL* que ilustre as regras de escopo (estático e dinâmico)
- O programa fonte será composto de um conjunto de funções com o formato *func f() {}*
- A função com nome *mainBazL* (palavra reservada) é a que inicia a execução do programa
- As funções serão compostas da declaração de variáveis locais, atribuições, chamadas a outras funções e comandos de impressão, 1 por linha

Exercício 5

- O único tipo suportado por esta linguagem são valores inteiros
- O interpretador poderá supor que o código fornecido é livre de erros sintáticos
- A interpretação do código deverá ser feita passo a passo (linha a linha)
- Após a execução de cada linha (ou durante toda a execução), o interpretador deverá exibir o estado da pilha de ativação

Exercício 5

- Cada registro de ativação deve armazenar os parâmetros, variáveis locais e o endereço de retorno da função
- O interpretador deverá suportar 2 modos de execução: baseado em escopo estático e baseado em escopo dinâmico

Exercício 5
Exemplo de Código em ~*bazL*

```
locais z
func g(x, y) {
    z = x + y
}
func f(w, k, t) {
    z = w + k + t
    g(z, z)
}
func main() {
    locais z
    z = 0
    f(z, z+1, z+2)
    print z
}
```

Exercício 5

- Sintaxe BNF da linguagem ~*bazL* cujos programas devem ser interpretados

<programa> ::= <funcao>+
<funcao> ::= 'func <id> '(<list_ids> ') '{ <locais> <comando>+ '}
<list_ids> ::= <id> ', <list_ids> | <id>
<locais> ::= 'locais <list_ids>
<comando> ::= <atrib> | <chamada> | <impressao>
<atrib> ::= <id> '= <expr>
<chamada> ::= <id> '(<list_expr> ')
<list_expr> ::= <expr> ', <list_expr> | <expr>
<expr> ::= <arg> ['+ <expr>]
<arg> ::= <id> | <num>
<impressao> ::= 'print <list_ids>

Exercício 5
Simulação de Execução / Escopo Estático

```
1  locais z
2  func g(x, y) {
3          z = x + y
4  }
5  func f(w, k, t) {
6          z = w + k + t
7          g(z, z)
8  }
9  func main() {
10         locais z
11         z = 0
12         f(z, z+1, z+2)
13         print z
14 }
```

```
Comando: g(z,z) // linha 7
Pilha:
      global: (z=3)
      main: (z=0)
      f: (w=0, k=1, t=2, end_ret=13)

Digite <enter> para avançar a
Execução !!!
```

Exercício 5
Simulação de Execução / Escopo Estático

```
1  locais z
2  func g(x, y) {
3          z = x + y
4  }
5  func f(w, k, t) {
6          z = w + k + t
7          g(z, z)
8  }
9  func main() {
10         locais z
11         z = 0
12         f(z, z+1, z+2)
13         print z
14 }
```

```
Comando: z = x + y // linha 3
Pilha:
      global: (z=3)
      main: (z=0)
      f: (w=0, k=1, t=2, end_ret=13)
      g: (x=3, y=3, end_ret=8)

Digite <enter> para avançar a
Execução !!!
```

Exercício 6

- Implemente um pré-processador para C que:
  - Remova os comentários do código fonte
  - Expanda include's e define's (constantes e macros)
  - Remova as quebras de linhas, tabulações e espaços, sempre que possível (*)
- O programa deve ser apresentado com um executável que receba como parâmetro o arquivo .c a ser pré-processado
- Pode ser feito em qualquer linguagem
- (*) Um código que não apresente erros de compilação, deve continuar desta forma após o pré-processamento

Exercício 6
(Ainda sem expansão do Include)

```c
// Programa Alo Mundo em C
// Autor: Prof. Bazilio

#include "stdio.h"

int main(void) {
    printf("Alô mundo!\n");
}
```

```c
#include "stdio.h"
int main(void){printf("Alô mundo!\n");}
```

Exercício 7

- Utilizar a ferramenta de conversão entre linguagens implementada pela monitora Isabela: goto-transpiler.herokuapp.com
- Identificar melhorias ou construções ainda não traduzidas
- Entender o código disponível no Github (https://github.com/isaCarvalho/Transpiler), propor e/ou implementar a melhoria.

Exercício 8

- Estudar Haskell implementando os exercícios desta lista: http://www2.ic.uff.br/~bazilio/cursos/lp/material/ListaExerciciosProgFuncional.pdf
- Fazer uma das questões (sorteada aleatoriamente) ao vivo, para o professor, em 5 minutos. A avaliação levará em conta o grau de dificuldade de resolução da questão.

Exercício 9

- Implementar algum jogo utilizando alguma linguagem OO
- Para tal, recomenda-se o uso de alguma biblioteca específica para a implementação de jogos na linguagem escolhida, de forma a simplificar a implementação e tornar o jogo mais completo

Exercício 10

- Implementar algum programa usando conceitos de OO. O programa pode ser implementado em qualquer linguagem que implemente recursos OO. Os programas podem ser: jogos, sistemas CRUD (Create, Read, Update e Delete) para manipulação de qualquer tipo de dado – agenda, todo's, controle de estoque, etc.
- A implementação precisará conter Classes, ao menos 1 Interface ou Classe Abstrata, 1 Associação, 1 Herança e alguma operação Polimórfica

Inversão de Papéis

- Cada grupo de alunos deverá estudar um ou mais assuntos, elencados pelo professor, e preparar uma pequena monografia (texto) contendo:
  - Resumo, Principais Características (estruturação livre), Exemplos em, pelo menos, 2 linguagens distintas, e Referências
- Esta monografia, após pronta, deverá ser distribuída para os demais alunos para servir de material de estudo para a prova.
- Nesta parte do curso, as aulas serão para tirar dúvida sobre a preparação deste material