

# CURSO DE ENGENHARIA DE COMPUTAÇÃO

## Disciplina: Compiladores – Implementação

### Instruções:

“Atribui-se nota zero ao acadêmico que deixar de submeter-se as verificações de aprendizagens nas datas designadas, bem como ao que nela se utilizar de meio fraudulento” (Capítulo V, art. 39 do Regimento Geral do Centro Universitário de Anápolis, 2015).

### RESTRIÇÕES

- Obrigatórios:
  - o Desenvolvimento em Linguagem C, conforme **ISO/IEC 9899-1990**
  - o A tabela **ASCII** deverá ser utilizada.
  - o **Usar pilha para implementar o duplo balanceamento.**
  - o **Usar lista encadeada para a tabela de símbolos**
- O software deve ser executado (**sem a instalação de plug-ins**)
  - o Linux
    - gcc - versão máxima 6.1
  - o Windows
    - Dev-C++ 5.0 beta 9.2 (4.9.9.2) with Mingw/GCC 3.4.2
    - Code::Blocks 17.12
  - o Pode ser utilizado outro software, desde que garanta a execução em um dos explícitos acima.
  - o O software deverá funcionar apenas com a compilação e execução no software escolhido (**não utilizar nenhum outro comando ou software**);
- Somente as funções abaixo podem ser utilizadas:
  - biblioteca ctype.h - isdigit(), isalpha() e isspace();
  - biblioteca stdio.h - scanf, gets, fgetc, fgets, printf, puts, for, while, ou do..while, if, switch, fopen, fclose, eof, malloc, sizeof, realloc, free;
  - biblioteca de string.h - strcmp, strcpy, strlen;
- Quaisquer outras funções devem ser construídas manualmente.

**CASOS OMISSOS:** Se houver alguma regra ou situação omissa **deverá** ser informado, que **poderá** retificar este documento destacando a parte retificada.

### REGRAS 2021/1

#### Sintaxe da Linguagem:

- Funções
  - o acordar() {  
  
} dormir();
- Palavras Reservadas
  - o pensar()
  - o falar()
  - o tentar()  
outra\_vez
  - o tarefa()

- Tipos de Dados
  - o líquido
  - o sólido
  - o gasoso

**IMPORTANTE: Case Sensitive**

real <> Real <> REAL, então verifique exatamente como descrito (**letras minúsculas**);

1. Função **acordar()** { ..... } **dormir()**;

- 1.1. A função **acordar()** deve estar presente no arquivo, e deve conter apenas um, após inserir uma chave e ao final é necessário um **dormir** com parênteses e ponto e vírgula.
- 1.2. Variáveis globais podem ser inseridas antes da função **acordar()**.

2. **Declaração de variáveis**

- 2.1. A declaração de variável poderá ser feita em qualquer local do código especificando o tipo de dado da variável, exceto dentro das palavras reservadas.
- 2.2. Variáveis podem ser globais ou locais, e seu nome precisa ser único.
- 2.3. Sempre deve conter o tipo de dado:
  - 2.3.1. **sólido**
    - 2.3.1.1. Representam os números inteiros, e precisam somente do tipo de dado e nome da variável.
  - 2.3.2. **líquido**
    - 2.3.2.1. Representam os números reais.
    - 2.3.2.2. Como separador decimal será usado o símbolo “.”;
    - 2.3.2.3. Haverá a necessidade de especificar a quantidade de caracteres antes e depois do símbolo separador;
    - 2.3.2.4. Limitador de tamanho “[ ]”, a ser inserido após o nome da variável;
  - 2.3.3. **gasoso**
    - 2.3.3.1. Representam as strings;
    - 2.3.3.2. Seu tamanho, sendo maior ou igual a um;
    - 2.3.3.3. Limitador de tamanho “[ ]”, a ser inserido após o nome da variável;
    - 2.3.3.4. Todo valor inserido em uma variável string, deverá ser utilizado aspas duplas, com duplo balanceamento “ (abre aspas duplas) e ” (fecha aspas duplas);
- 2.3.4. Os limitadores são obrigatórios, se aplicáveis.
- 2.4. Todas as variáveis precisam do marcador \*\*. Após o “\*” deve-se ter um(01) símbolo de a...z (minúsculo) e após e se necessário pode ser inserido qualquer símbolo de a..z ou A...Z ou 0...9.
- 2.5. Nenhum outro caractere será aceito na formação das variáveis.
- 2.6. A linha deve ser finalizada com ponto e vírgula;
- 2.7. Poderá, em uma linha, haver mais de uma variável declarada para o mesmo tipo de dado, desde que separadas por vírgula;
  - 2.7.1. Não deve haver declaração de variáveis de tipos diferentes na mesma linha.
- 2.8. Atribui-se valores a uma variável utilizando o símbolo “:=” (dois pontos e igual). Na sua declaração ou após.
- 2.9. As atribuições de variáveis devem obedecer ao escopo da variável:
  - 2.9.1. Atribuições podem ser feitos tanto com valor, quanto com outra variável ou através de cálculos matemáticos.

3. Expressões

3.1. Matemáticos

- 3.1.1. Poderá haver operações matemáticas no decorrer do código
  - 3.1.1.1. + para soma;
  - 3.1.1.2. \* para multiplicação;
  - 3.1.1.3. - para subtração;
  - 3.1.1.4. / para divisão
  - 3.1.1.5. ^ para exponenciação.
  - 3.1.1.6. % para resto da divisão
- 3.1.2. Poderão ser “[ ]” utilizados para delimitar prioridades, caso não utilize considerar as regras de matemática;

3.2. Relacionais

- 3.2.1. Podem aparecer comparações de :
  - 3.2.1.1. Variável com variável;
  - 3.2.1.2. Variável com texto / número;

- 3.2.1.3. Texto / número com variável;
  - 3.2.1.3.1. A palavra texto utilizada também pode-se tratar de um número decimal ou inteiro, porém entre as aspas duplas.
- 3.2.1.4. Texto/número com texto/número;
- 3.2.2. Os seguintes operadores serão válidos:
  - 3.2.2.1. = igual;
  - 3.2.2.2. <> diferente;
  - 3.2.2.3. < menor;
  - 3.2.2.4. <= menor ou igual;
  - 3.2.2.5. > maior;
  - 3.2.2.6. >= maior ou igual;
  - 3.2.2.7. != não;
  - 3.2.2.8. != não igual;
- 3.2.3. Não serão válidos os operadores invertidos =<, => ou <<, >> ;
- 3.2.4. Não serão válidos, operadores duplicados: !=!=;
- 3.2.5. Operações matemáticas podem ser parte da comparação relacional

#### 4. Pensar

- 4.1. O comando de leitura – pensar – poderá ler mais de uma variável (de tipos diferentes no mesmo comando), porém as variáveis devem ser separadas por vírgula e declaradas anteriormente;
- 4.2. Não podem ser feitas declarações de variáveis dentro da estrutura de leitura.
- 4.3. Haverá sempre um duplo balanceamento utilizando os parênteses.
- 4.4. A linha deve ser finalizada com ponto e vírgula;

#### 5. Falar

- 5.1. O comando de escrita – falar – poderá escrever mais de uma variável;
- 5.2. Poderá mesclar texto e variável, desde que tenha o símbolo “ + ” que deve ser utilizado após (e antes) das aspas duplas do texto;
- 5.3. Podem ser escritas variáveis de tipos diferentes no mesmo comando, desde que declaradas anteriormente;
- 5.4. Os textos que precisarem ser escritos no comando devem estar dentro das aspas duplas.
- 5.5. Variáveis estarão fora das aspas duplas.
- 5.6. Se houver escrita de mais de uma variável deverá ser separada com “ , ” e já devem ter sido declaradas anteriormente.
- 5.7. Observar o agrupamento de conteúdo.
- 5.8. Não podem ser feitas declarações dentro da estrutura de escrita ou operações matemática;
- 5.9. Haverá sempre um duplo balanceamento utilizando os parênteses e aspas duplas para texto.
- 5.10. A linha deve ser finalizada com ponto e vírgula;

#### 6. Tentar

- 6.1. O comando de teste - tentar - deve conter obrigatório um teste e uma condição de verdadeiro, podendo ou não conter um comando de falso.
- 6.2. Nos comandos de verdadeiro **e/ou** falso podem conter várias linhas, e pode conter qualquer estrutura da linguagem, exceto declaração de variáveis.
- 6.3. Considere a necessidade de abrir e fechar o bloco de verdadeiro e/ou falso com “{(abre chave) e } (fecha chave) ” sempre.
- 6.4. A linha do teste não conterà finalização de linha (ponto e vírgula) as demais – condição verdadeira **e/ou** falsa - devem conter a finalização de linha com ponto e vírgula.
- 6.5. Os testes podem ser feitos conforme especificação para operadores relacionais item 3.2.1;
- 6.6. Os seguintes operadores serão válidos:
  - 6.6.1. Para texto:
    - 6.6.1.1. Operadores 3.2.2.1, 3.2.2.2, 3.2.2.7 e 3.2.2.8;
  - 6.6.2. Para números:
    - 6.6.2.1. Todos os operadores do item 3.2.2;
  - 6.6.3. Atenção às regras 3.2.3, 3.2.4 e 3.2.5;
- 6.7. Atenção às regras de variáveis explícitos no item 2;
- 6.8. Pode haver testes aninhados;

#### 7. tarefa

- 7.1. O laço de repetição – tarefa - possui a seguinte estrutura repetir (x1; x2; x3), onde:
  - 7.1.1. x1 refere a operação matemática na variável de controle;
    - 7.1.1.1. As especificações de operações matemáticas podem ser feitas conforme o explícito no item 3.1;
    - 7.1.1.2. Será aceito qualquer operação matemática, com variáveis e/ou números;
    - 7.1.1.3. Haverá a contração dos símbolos + ou - (\*\*a++ ou \*\*a--).

- 7.1.1.4. Os símbolos contraídos, podem aparecer somente depois do nome da variável **\*\*a++**, pós-fixada;
- 7.1.1.5. Pode não haver a especificação da variável de controle, mas manter o ; (ponto-e-virgula);
- 7.1.2. **x2** – refere-se à atribuição de valor inicial da variável;
  - 7.1.2.1. Pode-se iniciar uma variável com um valor fixo, ou com o conteúdo de outra variável (*comando de atribuição*), ou ainda não a iniciar.
  - 7.1.2.2. Utilizar comando de atribuição;
  - 7.1.2.3. Poderá ser utilizado qualquer tipo de dado;
  - 7.1.2.4. As variáveis já devem ter sido declaradas anteriormente;
  - 7.1.2.5. Podem haver mais de uma variável sendo iniciada, e devem ser separadas por vírgula;
  - 7.1.2.6. Pode não haver inicialização de variáveis, mas manter o ; (ponto-e-virgula);
- 7.1.3. **x3** refere-se ao teste que deve ser feito a cada interação;
  - 7.1.3.1. Utilize os mesmos critérios condicionais explícitos para o comando de teste, ver item 6;
  - 7.1.3.2. Pode não haver teste;
- 7.2. Delimitar os blocos com a utilização de "{" e "}";
- 7.3. Os comandos de leitura, escrita e teste pode ser executado dentro do laço, inclusive outro laço;
- 8. Espaços
  - 8.1. **Poderá** aparecer entre uma palavra reservada e o próximo comando;
  - 8.2. **Poderá** aparecer entre a vírgula e uma variável, ou a variável e uma vírgula, mas não irá interferir – seja na leitura, escrita ou declaração de variáveis;
  - 8.3. **Não pode** aparecer entre os comandos de teste com operadores duplicados (<=, >=, :=, !=, <>)
  - 8.4. **Não pode** "quebrar/interromper" a sequência de uma palavra reservada ou variável;
- 9. Finalização
  - 9.1. De linha:
    - 9.1.1. Considere o ; (ponto e vírgula);
- 10. Identação
  - 10.1. Não são obrigatórios, estão no documento somente para melhorar a visualização;
  - 10.2. Se aparecerem no comando de escrita, dentro de aspas duplas será considerado texto;
  - 10.3. Caso ocorram podem acontecer somente no início da linha;
  - 10.4. Não podem aparecer entre palavras reservadas, funções / módulos, declarações, em testes, atribuições, operações matemáticas ou leituras;
- 11. Duplo-Balanceamento
  - 11.1. Para os itens:
    - 11.1.1. Chave;
    - 11.1.2. Parênteses;
    - 11.1.3. Colchetes;
    - 11.1.4. Aspas duplas;
    - 11.1.5. Operações matemáticas pós-fixadas;
- 12. Memória utilizada
  - 12.1. O software deve ser capaz de fazer alocações dinâmica na memória, e ainda liberar a memória alocada, quando não está mais sendo utilizada e/ou *realocar a memória se for o caso (a critério)*. E se não houver memória emitir a mensagem de **ERRO** "Memória Insuficiente". E ainda ao final liberar toda a memória alocada;
  - 12.2. Apresentar o valor máximo de memória utilizada;
  - 12.3. A quantidade de memória deve ser parametrizável;
  - 12.4. **A Memória disponível não poderá ultrapassar 128 KB;**
  - 12.5. Alertar se a memória utilizada estiver entre 90 e 99% do valor disponível;
- 13. Tabela de Símbolos
  - 13.1. A estrutura mais simples aceita é uma matriz, qualquer outra estrutura superior será aceita. A complexidade da escolha da estrutura não afeta na nota;
  - 13.2. Deve conter (não necessariamente nesta ordem)
    - 13.2.1. Tipo de Dado

- 13.2.2. Nome da variável
- 13.2.3. Possível Valor
- 13.2.4. Função / módulo a que pertence
- 13.3. Se houver fórmulas, atribuições – se tiver todos as informações – **pode** resolver;

#### 14. Erros

- 14.1. Léxicos e Sintáticos:
  - 14.1.1. Devem finalizar a execução apresentar o número da linha e o problema;
- 14.2. Memória Insuficiente;

#### 15. Alertas

- 15.1. Semânticos:
  - 15.1.1. Mostrar a linha e o problema;
  - 15.1.2. Não finaliza a execução
- 15.2. Alertar caso a memória utilizada no momento seja entre 90 e 99% do total disponível;

#### Exemplos de código

<pre>acordar(){     gasoso **var[10];     liquido **dinheiro[2.2], **mesada[3.2], **gastar[3.2]:=0.00;     solido **cpu :=1;     solido **tEstE1;      **mesada := **gastar + [**gastar^3];     falar("quanto você recebe por dia");     pensar(**dinheiro, **tEstE1);      tentar(**mesada &gt;=[**dinheiro + 1000]){         falar("Deu certo!");     }outra_vez {         falar("Danou-se!");     }      tarefa(**cpu++; **cpu=0; **cpu&lt;=**mesada){         **dinheiro = **dinheiro +1;         falar("Tenho" + **dinheiro, **tEstE1 + ", será suficiente?");     }  }dormir();</pre>	<pre>acordar(){     liquido **dinheiro[2.2], **mesada[3.2];     solido **cpu :=1;      falar("quanto você recebe por dia");     pensar(**dinheiro);      tarefa(**cpu++; **cpu=0; **cpu&lt;=**mesada){         **dinheiro = **dinheiro +1;         falar("Tenho" + dinheiro + ", será suficiente?");         tentar(**mesada &gt;=**dinheiro){             falar("Deu certo!");         }outra_vez {             falar("Danou-se!"); }     }  }</pre>
---	---