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

- Desenvolvimento em Linguagem C, conforme **ISO/IEC 9899-1990**
- A tabela ***ASCII*** pode ser utilizada
- O software deve ser executado (**sem a instalação de plug-ins**)
  - Linux
    - gcc - versão máxima 6.1
  - Windows
    - Devc++ instalável - versão 4.9.9.2
    - Code::Blocks 16.01
  - Pode ser utilizado outro software, desde que garanta a execução em um dos explícitos acima.
- O software deverá funcionar apenas com a compilação e execução no software escolhido (**não utilizar nenhum outro comando ou software**)

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

REGRAS 2019/2

Sintaxe da Linguagem:

- Funções / Módulos
  - principal() {}
  - funcao() {}
    - retorno
- Palavras Reservadas
  - escrever()
  - ler()
  - testar()
    - verdadeiro
    - falso
  - repetir()
- Tipos de Dados
  - inteiro
  - caracter
  - real

IMPORTANTE: Case Sensitive

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

1. Poderá haver no arquivo vários "módulos/ funções" de programas, porém somente **um** deve chamar-se principal();
   1.1. Em caso de inexistência do módulo/função *principal()* deve-se apresentar o erro: *Módulo Principal Inexistente;*
   1.2. Módulos/ funções podem comunicar-se entre si;
   1.3. A chamada de uma *função* se dará pelo nome e os possíveis parâmetros;
   1.4. Módulos do tipo *funcao()* precisam necessariamente ter um nome após a palavra reservada funcao e antes dos parênteses

1.4.1. Nomes de funções precisam:
1.4.1.1. Marcador __ (02 símbolos underscore) Após o "__" deve-se conter um(01) símbolo de a...z ou A...Z ou 0...9 e após, **pode** ser inserido qualquer símbolo de a..z ou A...Z ou 0...9;
1.4.2. Após o nome deve-se conter necessariamente o "(" e ")" (abre e fecha parênteses);
1.4.2.1. Dentro dos parênteses **pode** conter parâmetros;
1.4.2.1.1. Se ocorrerem, devem ser informados tipo de dados e nome da variável;
1.4.2.1.1.1. Para os tipos e nome de variáveis ver item 2;
1.4.2.1.2. Os parâmetros não devem ser declarados novamente dentro da *função*;
1.4.3. Não existe limitação de quantidade de parâmetros na *funcao()*, porém se houver mais de 01 (um) deverão ser separados por vírgula (somente uma);
1.4.4. As funções sempre possuem um tipo de dado para o retorno;
1.4.5. O retorno é uma variável do mesmo tipo de dado da função;
1.5. A função *principal* não possui parâmetros;
1.6. Poderá haver função sem chamada;
1.7. Após cada função/ módulo deve-se inserir um delimitador de "{" início e "}" fim;
1.8. Independentemente da quantidade de linhas deve-se inserir o delimitador de início e fim da função / módulo;
1.9. Se a função não for inserida antes do módulo *principal*(), e for chamada no contexto do *principal()*, deve-se procurá-la em todo o arquivo, e validar a *funcao()* antes de continuar a validação do módulo *principal()*;

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 (ler, escrever, repetir, testar)
2.2. Variáveis podem ser globais, mas seu nome precisa ser único.
2.3. Sempre deve conter o tipo de dado:
2.3.1. **inteiro**
2.3.2. **caracter**
2.3.2.1. Seu tamanho, sendo maior ou igual a um;
2.3.2.2. Limitador de tamanho "[ ]", a ser inserido após o nome da variável;
2.3.2.3. Todo valor inserido em um caracter, deverá ser utilizado aspas duplas, com duplo balanceamento " (abre aspas duplas) e " (fecha aspas duplas);
2.3.3. **real**
2.3.3.1. Como separador decimal será usado o símbolo ".";
2.3.3.2. Haverá a necessidade de especificar a quantidade de caracteres antes e depois do símbolo separador;
2.3.3.3. Limitador de tamanho "[ ] " , a ser inserido após o nome da variável;
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 - somente um). Na sua declaração ou após.
2.9. As atribuições de variáveis devem obedecer ao escopo da variável:
2.9.1. Para *caracter* utilizar a atribuição com aspas duplas;
2.9.2. Para *inteiro* considerar somente o número inteiro;
2.9.3. Para *real* considerar casas antes e após o ponto, conforme descrito na declaração;
2.9.4. Atribuições podem ser feitos tanto com valor, quanto com outra variável ou através de cálculos matemáticos.

3. Expressões
3.1. Matemáticos
3.1.1. Poderão 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. Comparações com :
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. Note que sempre haverá uma variável;
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;
3.2.3. Não serão válidos os operadores invertidos =<, => ou ><, <>, <<, >> ;
3.2.4. Não serão validos, operadores duplicados, mesmo que válidos: !=!=;
3.2.5. Operações matemáticas podem ser parte da comparação relacional

4. Ler
4.1. O comando de leitura – ler – 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. Escrever
5.1. O comando de escrita – escrever – poderá escrever mais de uma variável;
5.2. Poderá mesclar texto e variável, desde que tenha o símbolo “,” que deve ser utilizado após (e/ou 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.
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. Testar

6.1. O comando de teste - testar - 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 (considere a necessidade de abrir e fechar o bloco com “{}”, **somente para mais de uma linha**), e pode conter qualquer estrutura da linguagem, exceto declaração de variáveis.
6.3. A linha do teste não conterá finalização de linha (ponto e vírgula) as demais – condição verdadeira **e/ou** falsa - devem conter a finalização de linha com ponto e vírgula.
6.4. Os testes podem ser feitos conforme especificação para operadores relacionais item 3.2.1;
6.5. Os seguintes operadores serão válidos:
6.5.1. Para texto:
6.5.1.1. Operadores 3.2.2.1 e 3.2.2.2
6.5.2. Para números:
6.5.2.1. Todos os operadores 3.2.2.1, 3.2.2.2, 3.2.2.3, 3.2.2.4, 3.2.2.5 e 3.2.2.6;
6.5.3. Atenção às regras 3.2.3, 3.2.4 e 3.2.5;
6.6. Atenção às regras de variáveis explícitos no item 2;
6.7. Haverá somente um parêntese em toda a estrutura do testar;
6.8. Podem haver testes aninhados;

7. Repetir
7.1. O laco de repetição – repetir possui a seguinte estrutura repetir *(x1; x2; x3)*, onde:
7.1.1. x1 – refere-se à atribuição de valor inicial da variável;
7.1.1.1. Pode-se iniciar uma variável com um valor fixo, ou com o conteúdo de outra variável (*comando de atribuição*), ou ainda não a iniciar.

7.1.1.2. Utilizar comando de atribuição;
7.1.1.3. Poderá ser utilizado qualquer tipo de dados;
7.1.1.4. As variáveis já devem ter sido declaradas anteriormente;
7.1.1.5. Podem haver mais de uma variável sendo iniciada, e devem ser separadas por vírgula;
7.1.2. x2 refere-se ao teste que deve ser feito a cada interação;
7.1.2.1. Utilize os mesmos critérios condicionais explícitos para o comando de teste, ver item 6;
7.1.2.2. Pode não haver teste;
7.1.3. x3 refere a operação matemática na variável de controle;
7.1.3.1. As especificações de operações matemáticas podem ser feitas conforme o explícito no item 3.1;
7.1.3.2. Será aceito qualquer operação matemática, com variáveis e/ou números;
7.1.3.3. Haverá a contração dos símbolos + ou -. ($a++ e/ou $a--).
7.1.3.4. Os símbolos contraídos, podem aparecer somente depois do nome da variável $a++;
7.2. Para blocos de **mais de uma linha** deve-se utilizar "{" e "}" para delimitar o início e fim;
7.3. Os comandos de leitura, escrita e teste pode ser executado dentro do laço de repetição, inclusive outro laço;
7.4. Ao final da linha do repetir não pode conter o ponto-e-vírgula;

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);
9.1.2. No caso da palavra reservada "testar" ou "repetir" **pode** ser adicionado **uma** quebra de linha;
9.2. Função / Módulo
9.2.1. Com a finalização "}", condicionado obrigando ao início "{"

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;

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 1024 KB;
12.5. Alertar se a memoria 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 e apresentar o número da linha e o problema;
    14.2. Problemas Semânticos não são erros;
    14.3. Memória Insuficiente;

15. Alertas
    15.1. Semânticos:
        15.1.1. Mostrar a linha e o problema;
    15.2. Alertar caso a memória utilizada no momento seja entre 90 e 99% do total disponível;

Exemplos de código

```
principal(){
  inteiro $a, $b2 := 7;
  real $cc[2.5];
  inteiro $d;
  $b2 := $a;
  escrever("Escreva um número ");
  ler ($a);
 testar ($a <= $b2)
    verdadeiro escrever(" A é maior", $a);
    falso escrever("B é maior", $b);
 repetir ($d = 1; $d<=100; $d:=$d+2){
    escrever("D", $d);
 }
}
```

```
principal(){
  inteiro $a, $b2 := 7;
  inteiro $d;
   caracter $nome[10];
  escrever("Escreva um número ");
  ler($a);
 $d := __soma ($a, $b2)
}

funcao __soma (inteiro $e, inteiro $aa){
   inteiro $num;
   $num := $e + $aa;
   retorno $num;
}
```

***** Informações para a 1ª VA.

- Será verificado a leitura de arquivo externo e identificação das palavras reservadas possíveis;
- Caso houver alguma formação que não se refira a uma palavra reservada, identificar somente que símbolo ou palavra "não identificados";
- Extra – autômato finito determinístico com todas as regras;

***** Informações para a 2ª VA.

- Fazer a inserção das variáveis e funções na tabela de símbolos, e alterações se necessário - ;
- Identificar a sequência das palavras reservadas – (ler, escrever, testar, repetir);
- Se qualquer regra acima descrita não obedecida – apresentar erro e finalizar a execução;
- Os erros precisam da identificação da linha e o problema;
- Se a execução acontecer sem erros, apresentar a tabela de símbolos completa (tipo de dado, nome, possível valor e escopo);
- Extra – Gramática Livre do Contexto

***** Informações para a 3ª VA.

- Todas as regras da 2ª VA;
- Verificação de comparados, operadores lógicos, relacionais e operações matemáticas;
    - Não haverá erro, somente alerta;

Geral:

- Variáveis e tabelas de símbolos:
    - Inserção
    - Declaração
    - Alteração
    - Tipo de dado
    - Escopo
    - Visualização da tabela ao final
    - Palavras
        - int
        - char
        - double
    - Delimitação de tamanhos
- Palavras reservadas
    - principal() - palavra
    - função() – palavra, parâmetros e utilização da função
    - ler() – palavra e sua formação dentro dos parênteses;
    - escrever() – palavra e sua formação dentro dos parênteses;
    - testar() – palavra e sua formação dentro dos parênteses;
    - verdadeiro – palavra;
    - falso – palavra;
    - repetir() – palavra e sua formação dentro dos parênteses;
- Operadores
    - São válidos somente os operadores identificados na sessão 3. Em nenhum caso de operadores duplos pode haver espaços;
- léxico
    - Escrita correta de todas as palavras – em caso de não encontrar a palavra, ERRO;
- Sintático
    - A formação de escrita + as opções necessárias para o funcionamento – em caso de não adaptação a regra ERRO;
    - Duplo balanceamento;
- Semântico
    - Tipos de dados em atribuições;
    - Tipos de dados em testes;
    - Se os tipos de dados forem diferentes – ALERTA;
- Tratamento de ERRO:
    - Descrever corretamente e com clareza o erro e finalizar a execução do programa;
- Tratamento de Alerta
    - Descrever corretamente e com clareza o alerta e imprimir na tela o alerta. NÂO finaliza o programa;
- Memória
    - Controle;
    - Alerta se entre 90 e 99% do total da memória disponível;
    - Finalização do programa em caso de uso de 100% da memória disponível e definida neste documento.

UniEVANGÉLICA
CENTRO UNIVERSITÁRIO

COMPILADOR