Compilador
Implementação – 2018/2

RESTRIÇÕES

- Desenvolvimento em Linguagem C, conforme **ISO/IEC 9899-1990**
- É **necessário** a utilização da tabela ASCII
- 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**)

HISTÓRICO

- 2015/2

Palavras Reservadas : inicio, leia, escreva, var, fim, se, entao, senao;

- 2016/1

Palavras Reservadas: begin, read, write, end, if, then, else
Tipos de Dados: int, char, dec;

- 2016/2

Palavras Reservadas: inicio, ler, escrever, fim, se, então, senão, fim se, para, fim para;
Tipos de Dados: inteiro, caractere, decimal

- 2017/1

Palavras Reservadas: main(){, gets, puts, if, then, else, for, }
Tipos de Dados: int, char, dec

- 2017/2

Palavras Reservadas: programa, leia, escreva, se, senão, para
Tipos de Dados: inteiro, caractere, real, fim

- 2018/1

Funções : principal() e funcao ()
Palavras Reservadas : leitura (), escrita (), se (), senão, para ()
Tipos de Dados: inteiro, caractere, decimal

CASOS OMISSOS

*Se houver alguma regra ou situação omissa **deverá** ser informada a professora, que **poderá** retificar este documento destacando a parte retificada.*

REGRAS 2018/2

Sintaxe da Linguagem:

- Funções / Módulos
  - main()
  - function ()
- Palavras Reservadas
  - in ()
  - out ()
  - switch
  - case
  - break
  - default
  - foreach()
- Tipos de Dados
  - integer
  - char
  - float

IMPORTANTE: Case Sensitive

float <> Float <> FLOAT, então verifique exatamente como descrito (letras minúsculas)

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

      1.4.1. Nomes de funções precisam:

1.4.1.1. Marcador ! 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 parenteses **pode** conter parâmetros

1.4.2.1.1. Se ocorrerem, devem ser informados tipo de dados e nome da variável

1.4.2.1.2. Para os nomes de variáveis não considerar as informações de tamanho no caso de char ou especificação de casas decimais em caso de float

1.4.2.1.3. Parâmetros devem ser necessariamente do mesmo tipo de dado, e mesmo tamanho.

1.4.2.1.4. Os parâmetros não devem ser declarados dentro da funçao;

1.4.3. Não existe limitação de quantidade de parâmetros na function*()*, porém se houver mais de 01 (um) deve ser separado por vírgula (somente uma).

1.5. A função main 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. Independente da quantidade de linhas deve-se inserir o delimitador de inicio e fim da função / módulo;

1.9. Se a function não for inserida antes do main(), e for chamada no contexto do main(), deve-se procura-la em todo o arquivo, e validar a function() antes de continuar a validacao do main();

```
main(){
..
..
!soma($n1,$n2);
..
}
function !soma(integer $n1, integer $n2){
        integer $n3;
        $n3=$n1+$n2;

        }
```

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, mas seu nome precisa ser único.

2.3. Os nomes das variáveis de maneira alguma deve se repetir, mesmo que em módulos diferentes.

2.4. Sempre deve conter o tipo de dado (**integer**, **char** (e seu tamanho – limitador de tamanho "[]") ou **float**(e as especificações de casas decimais - e seu tamanho – limitador de tamanho "[]"));

```
char $nome[50] ;
float $numero[2.2];
integer $outroNumero;
```

2.4.1. Os limitadores são obrigatórios, se aplicáveis.

2.5. Todas as variáveis precisam do marcador $. Após o "$" deve-se ter um(01) símbolo de a...z (minúsculo) e após se necessário pode ser inserido qualquer símbolo de a..z ou A...Z ou 0...9.

2.6. Nenhum caractere especial será aceito na formação das variáveis.

2.7. A linha deve ser finalizada com ponto e vírgula;

2.8. Poderá, em uma linha, haver mais de uma variável declarada para o mesmo tipo de dado, desde que separadas por vírgula;

2.8.1. Não deve haver declaração de variáveis de tipos diferentes na mesma linha.

2.9. O separador de casas decimais será o símbolo "." (ponto), considerando-se que deve explicitar a quantidade antes e depois da vírgula, separadamente – considerando valores separados.

2.10. Atribui-se valores a uma variável utilizando o símbolo "=" (igual e somente um). Na sua declaração ou após.

2.11. As atribuições de variáveis devem obedecer ao escopo da variável, para caractere utilizar a atribuição com aspas duplas, para inteiro considerar somente o número inteiro, e para decimal considerar casas antes e após o ponto, conforme descrito na declaração;

2.12. 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 – Considere + pra soma, * para multiplicação, - para subtração, / para divisão e ^ para exponenciação. Poderão ser "[]" utilizados para delimitar prioridades, caso não utilize considerar as regras de matemática;

3.2. Relacionais

3.2.1. Poderá ser variável com variável, variável com texto/número ou texto/número com variável – entenda que a palavra texto utilizada anteriormente também pode-se tratar de um número decimal ou inteiro, porém com as aspas duplas. (note que sempre haverá uma variável)

3.2.2. Os seguintes operadores serão válidos:

3.2.2.1. "==" (igual); "<>" (diferente); "<" (menor) ou "<=" (menor ou igual); ">" (maior) ou ">=" (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

3.3. Lógicos
3.3.1. São validos “&&” e “||”
3.3.2. Operadores lógicos são validos somente entre duas, ou mais condições relacionais;
3.3.3. **Não** pode nem necessita de parentes.
3.3.4. A precedência se dará sempre por &&, depois ||
3.3.5. Após um operador lógico deve-se sempre inserir um **espaço**;

4. Leitura - in
4.1. O comando de leitura – in – 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 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;

```
in($numero);
```

5. Escrita - out
5.1. O comando de escrita – out – 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. Observando que irá agrupar os conteúdos.
5.7. Não pode ser feita declarações dentro da estrutura de escrita.
5.8. Haverá sempre um duplo balanceamento utilizando os parênteses e aspas duplas para texto.

```
out (“texto”+$var, $var2+”texto2”);
```

6. Test - switch
6.1. O comando de teste - switch – fara uma correspondência entre o conteúdo da variável e as opções listadas;

6.2. Será necessário demonstrar o inicio das opções com um abre chave {, e o fim delas com fecha chave }

6.2.1. Para cada opção deve inserir o comando **case**

6.2.1.1. Se a variável verificada for do tipo char (somente um caractere) será disposto após o case entre aspas simples e após dois pontos (:).

6.2.1.2. Os comandos de leitura, escrita, teste e repetição podem ser inseridos a cada opção – e devem respeitar suas respectivas regras de formação;

6.2.1.3. A finalização da opção se dará com o comando break;

6.3. Podem haver vários case, mas no mínimo haverá 01 (um).

6.4. Após a inserção de todas as opções será inserido o comando default;

6.4.1. Comando que será executados caso todos as verificações anteriores tiverem como resultado 'negativo ou falso'.

6.4.2. Não e preciso colocar nenhuma opção após o default, para iniciar o bloco de código será necessário a inserção de dois pontos (:)

6.4.2.1. os comandos de leitura, escrita, teste e repetição podem ser inseridos a cada opção – e devem respeitar suas respectivas regras de formação;

6.4.2.2. A finalização da opção se dará com o comando break;

```
switch($op){
        case 1 :in($numero);
                out ("texto"+$var, $var2+"texto2");
        break;
        default  :in($numero);
                out ("texto"+$var, $var2+"texto2");

        break;
        }
```

7. Repetição - foreach

7.1. O laco de repetição – foreach - terá a seguinte estrutura foreach *(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.

7.1.2. x2 refere-se ao teste que deve ser feito a cada interação;
7.1.2.1. Pode-se comparar a variável inicializada anteriormente com número fixo, ou outra variável;
7.1.2.2. Os seguintes operadores serão válidos:
7.1.2.2.1. "==" (igual) , "<>" (diferente), "<" (menor) ou "<=" (menor ou igual); ">" (maior) ou ">=" (maior ou igual);
7.1.2.2.2. Não serão válidos os operadores invertidos =<, => ou ><, !=, <<, >>.
7.1.2.3. Se o teste feito for com texto com mais de um caractere deverá conter aspas duplas, ou com letras somente um caractere deverá conter aspas simples, respeitando o duplo balanceamento para ambos os casos;

7.1.2.4. Ou ainda não fazer teste;
7.1.2.5. Este teste não precisa necessariamente ser simples, pode ser duplo, triplo, etc..

7.1.3. x3 será o incremento ou decremento;
7.1.3.1. pode aparecer um incremento ou decremento com a variável, ou mesmo uma operação (adição, subtração, multiplicação ou divisão), ou ainda não incrementar;

7.2. Para blocos de mais de uma linha deve-se utilizar "{}" para delimitar o início e fim;

7.3. Os comandos de leitura, escrita, teste e repetição pode ser executado dentro do laço de repetição, inclusive outro laço;

8. Espaços
8.1. *Poderá* aparecer entre uma palavra reservada e o próximo comando (seja ele qual for); ex: in (" ; in(" ; in ("
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;
inteiro $a,$b;
8.3. ***Não pode*** aparecer entre os comandos de teste com operadores relacionais duplicados (<=, >=, == ou <>)
8.4. **Não pode** "quebrar/interromper" a sequência de uma palavra reservada ou variável.
8.5. **Deverá** aparecer antes e depois de um operador lógico;

9. Finalização
9.1. De linha:

9.1.1. Considere o **;** (ponto e vírgula)
9.1.2. No caso da palavra reservada “switch” **pode** ser adicionado **uma** quebra de linha;
9.2. Função / Módulo
9.2.1. Com a finalização “}”, condicionado obrigando ao inicio “{“

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 inicio 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 – chave, parênteses, colchetes, aspas (duplas ou simples)

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 10 MB.

13. Tabela de Símbolos
13.1. A estrutura mais simples aceita é uma matriz;
13.2. Deve conter (não necessariamente nesta ordem)
13.2.1. Tipo de Dado
13.2.2. Nome
13.2.3. Possível Valor
13.2.4. Função / modulo a que pertence
13.3. Se houver fórmulas, atribuições – se tiver todos as informações – **pode** resolver;

14. Erros

14.1. Léxicos e Sintáticos – devem finalizar a execução e apresentar o número da linha e o problema;
14.1.1. Todas as situações que não respeitarem as regras acima;
14.2. Problemas Semânticos não são erros;
14.3. Memória Insuficiente

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

16. Entregas
16.1. Dia 24/11 até as 12 hs
16.1.1. Análise Léxica
16.1.1.1. No final deve ser explicitada a tabela de símbolos, se a compilação for finalizada com sucesso;
16.1.2. Restrição de memória
16.1.3. O código **deve** ser enviado para o email (aline.lemos@docente.unievangelica.edu.br);

16.2. Em relação ao código serão avaliados:
16.2.1. Tabela de símbolos
16.2.2. Memória
16.2.3. Palavras reservadas / funções – separadamente;
16.2.4. Código funcionando corretamente;
16.2.5. Tratamento de Erros e Finalização;

16.3. Dia 15/12 até as 12 hs;
16.3.1. Analise Léxica
16.3.1.1. No final deve ser explicitada a tabela de símbolos, se a compilação for finalizada com sucesso
16.3.2. Análise Sintática
16.3.3. Analise Semântica
16.3.4. Restrição de memória
16.3.5. O código **deve** ser enviado para o email (aline.lemos@docente.unievangelica.edu.br)

16.4. Em relação ao código serão avaliados:
16.4.1. Tabela de símbolos
16.4.2. Memoria

16.4.3. Palavras reservadas / funções - separadamente
16.4.4. Código funcionando corretamente
16.4.5. Tratamento de Erros
16.4.6. Tratamento de Alertas
16.4.7. Duplos balanceamentos
16.4.8. Tipos de Dados – Atribuições e Comparações