Nomes: Amanda Maia, Augusto Gonçalves, João Pedro Carlos

1. INTRODUÇÃO

A GremioScript é uma linguagem de programação desenvolvida como parte da disciplina de Compiladores do curso de Ciência da Computação da UNESC. A seguir, apresentam-se suas principais construções e as regras que regem seu uso.

2. ESTRUTURA

A estrutura básica de uma função na linguagem segue o modelo abaixo:

```
void main {

      //Primeiro, declarações das variáveis
      //Segundo, declarações das funções
      inicio
      //Terceiro, o corpo com a lógica
      ;
      fim
}
```

3. DECLARAÇÕES

No início de cada função, é necessário declarar primeiramente as variáveis que serão utilizadas no código seguidas das funções.

3.1 VARIÁVEIS

Variáveis de um mesmo tipo podem ser declaradas em sequência, separadas por vírgulas, seguidas de dois pontos, indicando o tipo, e finalizadas com ponto e vírgula. O exemplo abaixo ilustra essa sintaxe:

```
i1 : integer;
f1, f2 : float;
s1, s2 : string;
```

3.1.1 Regras de identificação de variáveis

- **Caracteres permitidos**: letras (`a-z`, `A-Z`), números (`0-9`) e underline(`_`).
- **Primeiro caractere**: não pode ser número, apenas letra ou underline. Caso o primeiro caractere for uma letra ela deve ser minúscula.

- **Palavras reservadas**: não podem ser usadas como identificadores (`integer`, `for`, `while`, etc.)**.**
- **Tamanho:** Não há um limite para o tamanho dos nomes das variáveis.

3.2 FUNÇÕES

As funções devem ser declaradas após as variáveis, seguindo o padrão especificado abaixo:

1. Tipo do retorno (void, integer, float, string ou char).
2. Nome da função, com as mesmas regras de nomenclatura das variáveis.
3. Caso haja parâmetros, abrir parênteses, dentro dos parênteses são listados os parâmetros passados para função separados por ponto e vírgula.
4. Abre chaves.
5. Declarações de variáveis.
6. Declarações de funções.
7. Corpo com a lógica.
8. Return().

OBS: O tipo de retorno pode ser ‘void’, ‘integer’, ‘float’, ‘string’ ou ‘char’.

Segue um exemplo abaixo:

```
tipoDoRetorno nomeDaFuncao (integer parametro1; string parametro2){
        //declarações das variaveis
        //declarações de funções
    inicio
        //corpo com a logica
        ;
    fim
    // Todas funções que não são a main, deverão ter um return vazio caso //
    sejam void ou com UMA variavel/valor
        return()
        //ou
        return(algumaVariavel)
        //Valores fixos também são permitidos ex: return("Oi")
}
```

Funções podem ser chamadas a qualquer momento dentro do corpo da função desde que tenham sido declaradas previamente. Elas podem ser usadas tanto na atribuição de uma variável ou chamadas livremente.

```
Exemplo:
void main{
        retornoDaFuncao : integer;
        integer somaInteger(integer A; integer B){
                resultado : integer;
                inicio
                resultado = A + B;
                fim
                return (resultado)
        }

        void boasVindas{
                inicio
                cout << `Bem vindo!`;
                cout << `Menu: `;
                cout << `1 - Opção 1`;
                cout << `1 - Opção 2`;
                fim
                return()
        }
        inicio
                callfuncao boasVindas;
                retornoDaFuncao = callfuncao somaInteger(5,3);
        fim
```

4. COMANDOS

Os comandos são todas as operações que podem ser realizadas dentro do corpo da função, sendo necessário o uso do ; após cada comando.

4.1 COMANDO DE ENTRADA (CIN)

Através do comando de entrada *cin* é possível receber dados digitados pelo usuário no teclado e armazená-los em variáveis. Exemplo:

```
cin >> nomeDeVariavel;
```

4.2 COMANDO DE SAÍDA (COUT)

O comando de saída *cout* é utilizado para exibir mensagens ou valores na tela. Veja no exemplo abaixo:

```
//Exemplo 1:
      nomeAluno = “João Pedro”;
      cout << `Ola aluno ` << nomeAluno;
      cout << `Suas notas foram: ` << nota1, nota2, nota3;
      cout << `Suas notas foram: ` << nota1 << nota2 << nota3;
//Exemplo 2:
      nomeAluno = “João Pedro”;
      msg1 = “ bem vindo!”;
      cout << `Olá ` << nomeAluno << msg1;
//Exemplo 4:
      msg1 = “1 - Sim”
      msg2 = “2 - Não”
      cout << `Deseja retornar? ` << msg1, msg2;
```

4.3 ATRIBUIÇÃO DE VALORES

No caso de integer e float, é permitido todas as operações básicas (+, -, * e /) com o uso de () envolvendo números diretos ou variáveis. Também existe suporte para a atribuição de variáveis via chamada de função

As operações entre integers e floats é permitida, mas sempre que envolver um float ou uma divisão, a variável atribuída deverá ser do tipo float. Exemplo:

```
variavelA : integer;
X : float;
inicio
      X = 5 + variavelA * 3 / (5 + 3);
      X = callfuncao soma(variavelA, variavelB);
fim
```

Se tratando de string e char, é permitido a concatenação via variável ou string, note que caso dois chars sejam concatenados, o resultado será uma string:

```
Exemplo 1:
      char1, char2 : char;
      result : string;
      início
      char1 = ‘O’;
      char2 = ‘I’;
      result = char1 + char2 + char1 + char2; // "OIOI"
      fim
```

```
Exemplo 2:
        string1, string2, result : string;
        inicio
        string1 = "Ola";
        string2 = "mundo!";
        result = string1 + string2; // result agora é "Olamundo!"
        fim
```

4.4 IF E ELSE

Blocos condicionais podem ser criados utilizando 6 operadores e permite a interação entre variáveis do mesmo tipo e valores fixos. Variáveis do tipo integer e float também podem estar no mesmo bloco.

O bloco de condição else é totalmente opcional, não sendo necessário ser especificado. Para mais de uma condição, é utilizado IFs aninhados.

```
Exemplo 1:
inicio
        if (integerA > floatA){

        }else{

        };
fim
```

```
Exemplo 2:
        inicio
                if (integerA >= 18){
                        if(integerA<= 100){
                                cout << `Você é maior de idade e está vivo :)`;
                        }else{
                                cout << `Você provavelmente é bem velho ou mentiroso.`;
                        };
                }else{
                        cout << `Você é menor de idade`;
                };
        fim
```

```
Exemplo 3:
inicio
        if (nome == “Augusto”){
                cout << `Easter egg do augusto`;
        };
fim
```

Operadores suportados:

Operador	Descrição
==	Igualdade
!=	Desigualdade
<	Menor que
>	Maior que
<=	Menor ou igual
>=	Maior ou igual

4.5 BLOCO DE REPETIÇÃO FOR

O bloco de repetição **for** é utilizado para executar um conjunto de instruções um número determinado de vezes.
Sua estrutura é composta por três partes principais:

1. **Inicialização**: onde é declarada e/ou inicializada a variável de controle do laço;
2. **Condição**: responsável por verificar se a execução do laço deve continuar;
3. **Incremento/Decremento**: etapa onde a variável de controle é atualizada a cada iteração.

Dessa forma, o **for** permite controlar o número de repetições de forma clara e compacta.

```
Exemplo 1:
inicio
        contador : integer;
        contador = 0;
        for(i = 0; i < 10; ++1){
                contador = contador + 1;
        };
fim
// Ao final do laço, contador = 10
```

```
Exemplo 2:
inicio
        tamanhoFor, contador : integer;
        tamanhoFor = 10;
        contador = 0;
        for(i = 0; i <= tamanhoFor; ++1){
                contador = contador + 1;
        };
fim
// Ao final do laço, contador = 11
```

4.6 BLOCO DE REPETIÇÃO WHILE

O bloco de repetição while se repetirá enquanto que sua condição seja verdadeira. Diferente do **for,** o **while** não possui inicialização e nem incrementos de variáveis, isso será feito pelo programador fora ou dentro do laço

Sua estrutura é composta por:

1. **Condição:** verificada no inicio de cada iteração, caso verdadeira o laço continua a se repetir, caso falso o laço se encerra
2. **Bloco de instruções:** é onde serão realizados os comandos que devem se repetir

```
Exemplo 1:
condicao : integer;

inicio
      condicao = 0;
      while(condicao < 5) {
            condicao = condicao + 1;
      };
fim

// O laço vai se repetir 5 vezes
```

```
 Exemplo 2:
 condicao : integer;

 inicio
       condicao = 5;
       while(condicao < 5) {
             condicao = condicao + 1;
       };
 fim
 // Não haverá nenhuma iteração no while
```

4.7 BLOCO DE REPETIÇÃO DO WHILE

O bloco de repetição **do while** é muito similar ao **while**, sua única diferença é que sempre vai realizar pelo menos uma iteração e verificar se sua condição é verdadeira apenas após passar pelo bloco uma primeira vez.

Sua estrutura é composta por:

1. **Bloco de instruções**: executado imediatamente, sem avaliação prévia.

2. **Condição**: avaliada após a execução do bloco. Se for verdadeira, o laço continua; caso contrário, é encerrado.

```
Exemplo 1:
condicao : integer;

inicio
      condicao = 0;

      do {
            condicao = condicao + 1;
      }
      while(condicao < 5);
fim
// O laço vai se repetir 5 vezes
```

```
Exemplo 2:
condicao : integer;

inicio
      condicao = 5;

      do {
            condicao = condicao + 1;
      }
      while(condicao < 5);
fim
// Haverá uma iteração apenas
```

5. COMENTÁRIOS

Os comentários servem para caso o programador queira deixar algo anotado no código sem interferir na compilação, são linhas do código que serão ignoradas na hora de compilar.

Existem dois tipos de comentários, o feito na linha e o feito em blocos bloco:

Em linha:

```
// Comentário feito na linha
```

Em Bloco:

```
/- Comentário
feito
em bloco -/
```

6. REGRAS LÉXICAS

- Nomes de variáveis e funções devem iniciar com uma letra (maiúscula ou minúscula) ou underline (`_`). Após o primeiro caractere, podem conter dígitos.
- Não são permitidos caracteres especiais em nomes de variáveis e funções.
- Nomes de variáveis não podem coincidir com palavras reservadas da linguagem.
- Caracteres do tipo *char* devem estar delimitados por aspas simples, contendo um único caractere ou nenhum. Exemplos: `' '`, `'@'`, `'a'`.
- Literais utilizados em instruções `<< cout` devem ser delimitados por crases. Exemplo: `` `Mensagem aqui` ``.
- *Strings* devem ser delimitadas por aspas duplas e estar contidas na mesma linha.

7. ERROS LÉXICOS

- *Strings* e literais não podem se estender para outra linha; não é permitido abrir aspas ou crase e fechá-las na linha seguinte.
- *Floats* devem conter ao menos um dígito após o ponto decimal.
- Valores inteiros e floats devem estar no intervalo de −2.147.483.648 a 2.147.483.647.
- Abrir uma string com `"` e não fechá-la caracteriza erro léxico.
  Abrir um literal com `` ` `` e não fechá-lo também caracteriza erro léxico.