Universidade Federal de Alagoas
Instituto de computação

Compiladores
GD
Especificação da Linguagem

Alunos: Davi José de Melo Silva, Gustavo Augusto Calazans Lopes
Professor: Alcino Dall'Igna Jr.

2020.2

1 Introdução

A linguagem GD foi criada baseada nas linguagens C, Python e JavaScript. Não é uma linguagem orientada a objetos, é case-sensitive e possui palavras reservadas e não possui nenhum tipo de coerção de tipos. Pode ser usada para criação de algoritmos simples. Possui estrutura de dados simples que é o arranjo unidimensional. Possui tipagem estática, estruturas condicionais e de repetições.

2 Estrutura geral do programa

Um programa feito em GD deve possuir:

- Todas as funções possuem escopos delimitados por chaves de abertura e fechamento, possuindo um tipo de retorno.
- Deve possuir uma função main com tipo e retorno obrigatório sendo um inteiro, para o retorno deve ser 0.
- Todas as linhas devem terminar com ponto e vírgula (“;”).
- Qualquer bloco o escopo vai ser delimitado por abertura e fechamento de chaves.

2.1 Ponto inicial do execução

O ponto inicial do programa é identificado através da criação de uma função principal chamada **main** que possui como retorno 0 através da palavra reservada **return**. O escopo de uma função é determinado por chaves de abertura e de fechamento e o uso de parênteses para abertura e fechamento de parâmetros de função No final de cada linha deve possuir o caractere “;” . Segue um exemplo:

```
int function main() {
    .
    .
    .
```

```
        return 0;
}
```

2.2 Definições de funções

Para criação de novas funções, deve-se utilizar a palavra reservada **function** sendo que antes dela deve possuir o tipo de retorno da função podendo ser (**int, float, bool, string, char, void**) sendo void utilizado para quando não houver nenhum tipo de retorno. Em seguida o nome da função que é um identificador único seguido de uma lista de parâmetros delimitados por parênteses. A lista de parâmetros deve ser composta pelo tipo seguido do identificador. Cada parâmetro é separado por vírgulas (“,”), e por fim, para definir o escopo dessa função utiliza-se abertura e fechamento de chaves. A seguir temos um exemplo:
<tipo de retorno> function <identificador da função>(<tipo do parâmetro 1> <identificador 1>, ...) {}

```
void function funcao1() {
        ...
}
```

3 Conjunto de tipos de dados e nomes

3.1 Identificadores

Identificadores possuem algumas regras, sendo que são case-sensitive e não devem possuir certos caracteres especiais.

- Podem começar somente com letras maiúsculas ou minúsculas;
- Não podem começar com números ou qualquer caractere especial;
- O único caractere especial permitido é o underline (“_”) ;
- Não podem ser palavras reservadas;
- Não é permitido o uso de espaço;

3.2 Palavras reservadas

As palavras reservadas são:
main, if, elif, else, for, while, function, int, float, bool, string, char, print, input, False, True.

3.3 Coerção

Linguagem estaticamente tipada, não permitindo coerção de variáveis de tipos diferentes.

3.4 Inteiro

int identifica a variável como um número inteiro de 32 bits, sendo uma sequência de dígitos entre 0 a 9.
Ex:

```
int  inteiro;
```

3.5 Ponto Flutuante

float identifica variáveis como um número com ponto flutuante de 32 bits, sendo um sequência de dígitos entre 0 a 9 seguido de um ponto (“.”) e demais sequências de dígitos.
Ex:

```
float pontoFlutuante;
```

3.6 Caracteres e cadeias de caracteres

Para apenas um caractere, podendo ser desde números, letras maiúsculas e minúsculas ou caracteres especiais. Utiliza-se a palavra reservada **char** seguida do identificador. Seu valor é delimitado por aspas simples (‘.’).
Ex:

```
char caractere = ‘a’;
```

Já uma cadeia de caracteres é aceito os mesmo caracteres, porém delimitados por aspas duplas e identificados através da palavra reservada **string**.
Ex:

string string1 = “Esta é uma sequência de caracteres.”

3.7 Booleanos

Booleanos são identificados através da palavra reservada **bool** e pode ter somente dois resultados possíveis, sendo eles **True** e **False** que também são duas palavras reservadas da linguagem.
Ex:

bool teste = True;

3.8 Arranjos Unidimensionais

Um Arranjo Unidimensional ou Array é declarado como na linguagem C, onde o tamanho do arranjo é gerenciado pelo programador e não pode ser mudado durante a execução do programa. Definimos o seu formato como sendo:
<tipo> identificador[Tamanho]
Ex:

int array1[9];

3.9 Valores padrão

TIPO	VALOR PADRÃO
int	0
float	0.0
char	‘’ (vazio)
string	“” (vazio)

bool	false

4 Conjunto de operadores

4.1 Aritméticos

- “+” Soma de dois operandos
- “-” Subtração de dois operandos
- “/” Divisão de dois operandos
- “*” Multiplicação de dois operandos
- “%” Resto da divisão de dois operandos

4.2 Relacionais

- “==” Igualdade entre dois operandos
- “!=” Diferença entre dois operandos
- “<” Operador “menor que”
- “>” Operador “maior que”
- “<=” Operador “menor ou igual que”
- “>=” Operador “maior ou igual que”

4.3 Lógicos

- “!” Operador “negação”
- “&&” Operador "conjunção"
- “||” Operador “disjunção”

4.4 Concatenação de cadeias de caracteres

Para concatenar duas cadeias de caracteres utiliza-se **“.”** sendo sua associatividade da esquerda para a direita, só é possível concatenar com o tipo string. Exemplo:

string string1 = “Testando”;
string string2 = “ Concatenação”;
string string3 = string1 . string2;

4.5 Precedência e associatividade

“+” “-” (soma e subtração)	Da esquerda para direita
“*” “/” (multiplicação e divisão)	Da esquerda para direita
“!” (negação)	Da direita para esquerda
“<”, “<=”, “>” e “>=” (comparação)	Da esquerda para direita
“==” e “!=”	Da esquerda para direita
“&&” “||”	Da esquerda para direita
“.” (concatenação)	Da esquerda para direita

4.6 Operações multiplicativas e aditivas

Quando possuímos duas variáveis do mesmo tipo realizando operações, o tipo resultante será o mesmo. Mas ao fazer uma operação multiplicativa ou aditiva de um ponto flutuante com inteiro, o resultado será um ponto flutuante.

5 Instruções

Cada bloco de instruções na linguagem seu escopo são delimitados por chaves de abertura e fechamento **“{}”** e cada linha é finalizada com o caractere **“;”**.

5.1 Atribuição

Uma atribuição é feita utilizando o caractere de igualdade **“=”** sendo que do lado esquerdo deve possuir o tipo seguido do nome do identificador e após a igualdade sendo seu valor do mesmo tipo declarado. Exemplo:

<tipo> identificador = valor;

5.2 Estruturas condicionais

A estrutura condicional de **if** e **elif** sempre terá um retorno do tipo booleano. Dentro de parênteses deve ser colocado essa condição lógica seguida dos delimitadores de escopo. Para o uso do **else** não usa parênteses e nem coloca uma condição lógica, é utilizada quando as condições **if** e **elif** são falsas. Após um **if**, é possível utilizar um sequência de **elif** e somente um único **else** para cada cláusula if que foi criada.
Exemplo:

```
# Estrutura de uma via
if(<condição>) {
        ...
}

# Estrutura de uma ou mais vias
if(<condição>) {
        ...
} elif(<condição>) {
        ...
}

# Estrutura duas vias
if(<condição>) {
        ...
} else {
        ...
}
```

5.3 Estrutura de repetição por controle lógico

A repetição ocorre enquanto a condição for verdadeira, quando for falsa o loop para. Para utilizar utilizamos a palavra reservada **while** seguida de parênteses com a condição que deve continuar executando o loop. O escopo deve ser delimitado por chaves de abertura e fechamento. Exemplo:

```
while(<condição>) {
        ....
}
```

5.4 Estrutura de repetição por controle de contador

Como em C, para utilizar a estrutura de repetição por contador utiliza-se o comando **for** seguido de parênteses separados por três partes separadas por **";"**. Na primeira parte colocamos o contador (podendo atribuir um valor para iniciar ou se preferir deixar com seu valor inicial), no segundo a condição lógica para parada e no terceiro o contador para incrementar ou decrementar. O escopo também é delimitado por abertura e fechamento de chaves. Exemplo:

```
for (i = 0; i < 10; i = i + 1) {
        ...
}
```

5.5 Entrada e saída

As funções de entrada e saída são **input** para entrada e **print** para saída.

Para a função de leitura **input** devemos declarar a variável previamente a qual será recebida o valor, assim atribuindo o valor lido a variável que foi passada por parâmetro. Para uso de arrays (vetores) deve ser lido através de uma estrutura de repetição acessando cada posição que foi previamente definida.

Para a função de saída **print** é possível passar uma cadeia de caracteres mesmo que não foi previamente salva em uma variável. Pode-se também passar variáveis por parâmetro para exibição de seu valor na tela. Caso seja passado mais de uma variável para exibição deve ser separado cada uma através de vírgulas (**","**).

Exemplos:

Entrada:

```
int variavel1;
input(variavel1);
```

Saída:

```
int parametro1 = 1;
print(“Isto é um print com parâmetro número %d”,  parametro1);

print(“Este é um print simples”);
```

5.6 Funções

As funções funcionam baseada na linguagem C, onde primeiro definimos o tipo de retorno mas caso não tenha nenhum será o tipo void, seguido da palavra reservada **function** e do nome dessa função e dentro de parênteses os parâmetros que essa função espera receber. Cada parâmetro deve ser escrito com o tipo seguido do nome dela e caso tenha mais de um é separado por “,”. Para retornar algum valor utiliza-se a palavra reservada **return**. Exemplo:

```
<tipo> function <nome da função> (<tipo> parametro1, <tipo> parametro2, ...)
{
    ....
    return <valor>;
}
```

6 Programas de exemplos

6.1 Alô mundo!

```
int function main() {
   print("Hello World");
   return 0;
```

```
}
```

6.2 Fibonacci

```
void function fibonacci(int qnt) {

  if(qnt == 1) {
    print("0");
  } elif(qnt >= 2) {
    print("0, 1");
  }

  int i = 2;
  int numberToPrint;
  int prev2number = 0;
  int prev1number = 1;

  while(i < qnt) {
    numberToPrint = prev1number + prev2number;
    prev2number = prev1number;
    prev1number = numberToPrint;
    print(", %d", numberToPrint);
    i = i + 1;
  }
}

int function main() {
  int qnt;
  input(qnt);
  fibonacci(qnt);

  return 0;
}
```

6.3 Shell sort

```
void function shellSort(int size, int vetor[size]) {

    int i;
    int j;
    int value;

    int h = 1;
    while(h < size) {
        for(i = h; i < size; i = i + 1) {
            value = vetor[i];
            j = i;
            while(j > h - 1 && value <= vetor[j - h]) {
                vetor[j] = vetor [j - h];
                j = j - h;
            }
            vetor[j] = value;
        }
        h = h / 3;
    }

    for(i = 0; i < size/ i = i + 1) {
        print("%d ", vetor[i]);
    }

}

int function main() {
  int qnt;
  input(qnt);
  shellSort(qnt);

  return 0;
}
```