Universidade Federal de Alagoas

Compiladores
C- -

Paulo Bernardo
pbaf@ic.ufal.br

Ricardo Alves
ras@ic.ufal.br

Dezembro, 2019

Sumário

1 Introdução

A linguagem C– tem como base as linguagens C e Python, C– admite escopo global e não admite cast. Não possui coerção (conversão implicita de tipos) e por ser estaticamente tipada não possui tratamento de erros para detecção de tipos e sua compatibilidade de tipos é feita por nome. C– não é orientada a objetos e é case-sensitive.

2 Estrutura geral de um programa

Um programa escrito deve possuir:

- Todas as funcões devem ser declaradas antes de poderem ser utilizadas. A declaração de uma função deve ser iniciada pelo seu tipo de retorno, seguido pelo seu identificador (começando em letra maiúscula), após isso a lista de parâmetros dever ser dada entre parênteses e seus itens separados por vírgula, seu corpo dever estar contido dentro de chaves. Cada item dos parâmetros deve conter apenas o tipo da variável e o seu respectivo identificador. Funções sem o **return** retornam o valor padrão de cada tipo.

  Exemplo:

```
returnType Identifier(typeA paramA, typeB paramB,...) {
    ...
    code
    ...
}
```

- Um ponto de início de execução que se dá pela função inicial **Main** com tipo de retorno obrigatório do tipo int.

```
int Main() {
    ...
    code
    ...
    return 0;
}
```

3 Tipos de dados e nomes

3.1 Definições de variáveis globais/locais

Variáveis globais devem possuir seu tipo precedido da palavra "global". Exemplo:

```
global int var;
```

Funções/Procedimentos devem ser declarados previamente antes de serem implementados e o modo de passagem de parâmetros é por cópia.

3.2 Palavras reservadas

Lista de palavras reservadas: *and, bool, char, during, else, false, float, from, get, global, if, increment, int, not, or, print, return, string, to, true.*

3.3 Identificadores

Tem o tamanho máximo de 48 caracteres e as seguintes regras:

- Todas as variáveis devem iniciar com letra minúscula e funções/procedimentos devem ser iniciadas com letra maiúscula.
- Os caracteres, a partir do segundo, podem ser letras, números ou underline.
- Não é permitido o uso de espaços.
- Não é permitido o uso de palavras reservadas como identificador.

3.4 Comentários

Apenas comentários por linha, identificados por '#'

3.5 Inteiro

Deve ser identificado pela palavra reservada *int* e representa um número inteiro de 64 bits. Seus literais são uma sequência de números inteiros.
Exemplo:

```
int integer;
```

3.6 Ponto Flutuante

Deve ser identificado pela palavra reservada *float* e representa um número real de 64 bits. Seus literais são da forma:

```
digit+(\.digit+)?
```

Exemplo:

```
 float floating_point;
```

3.7 Caractere

Deve ser identificado pela palavra reservada *char* e representa 1 byte que guarda um valor de 0 a 127 referente a seu símbolo na tabela ASCII.
Exemplo:

```
char character;
```

3.8 Arranjos Unidimensionais

Um vetor deve ter seu tipo seguido pelo seu tamanho especificado entre colchetes e logo após o seu identificador. Seus literais dependem do tipo utilizado e devem ser inclusos dentro de colchetes seperados por vírgula.
Exemplo:

```
<Type>[size] identifier;

<Type>[size] identifier = [<values>];

<Type>[size] Identifier(<parameters>) {}
```

Os elementos de um arranjo podem ser acessados da seguinte maneira:

```
identifier[position];
```

Se uma quantidade de elementos num arranjo for diferente da que foi declarada, duas coisas podem acontecer:

- Quantidade menor do que o declarado:
  - O restante do arranjo é preenchido com os valores padrões do tipo do arranjo.
- Quantidade maior do que o declarado:
  - Erro.

3.9 Cadeia de Caracteres

Deve ser identificado pela palavra reservada *string* e representa uma sequência de caracteres:
Exemplo:

```
string str;
```

3.10 Booleano

Deve ser identificado pela palavra reservada "bool"e representa somente 2 possíveis valores: true e false.
Exemplo:

```
bool boolean;
```

3.11 Operaçoes de cada tipo

Tabela 1:

Tipo	Operações
int	Atribuição, Aritiméticos, Relacionais
float	Atribuição, Aritiméticos*, Relacionais
char	Atribuição, Relacionais, Concatenação
string	Atribuição, Relacionais, Concatenação
bool	Atribuição, Relacionais**, Lógicas

* operador de resto não incluso
** somente os operadores '==' e '!=' são aceitos.

3.12 Valores padrão

Tabela 2:

Tipo	Valor
int	0
float	0.0
char	' ' (0)
string	null
bool	false

4 Conjunto de Operadores

4.1 Aritimeticos

Tabela 3:

Operação	Símbolo
Soma	+
Subtração	-
Unário Negativo	-
Multiplicação	*
Divisão	/
Resto	%

4.2 Relacionais

Tabela 4:

Operação	Símbolo
Igual	==
Menor que	<
Maior que	>
Menor ou igual	<=
Maior ou igual	>=
Diferente	!=

4.3 Lógicos

Tabela 5:

Operação	Símbolo
Negação Unária	not
Conjunção	and
Disjunção	or

4.4 Concatenação

É formada pelo operador "&", que ao ser aplicado concatena os dois lados da operação se os dois lados forem cadeia de caracteres.
Exemplo:

```
string str = string1 & string2;
```

4.5 Tipo das operações

É definido de operador a operador.

4.6 Precedência e Associatividade

Tabela 6:

Operador	Associatividade	Precedência
-	Direita	1
not	Direita	2
%, /, *	Esquerda	3
+, -	Esquerda	4
<, >, <=, >=	Nenhuma	5
==, !=	Nenhuma	6
and	Esquerda	7
or	Esquerda	8
&	Nenhuma	9

Parênteses podem alterar a precedência.

5 Instruções

5.1 Atribuição

Definida pelo operador '=', sendo o lado esquerdo o identificador a receber o valor e o direito o valor ou expressão a ser atribuído.

5.2 Estrutura condicional de uma e duas vias

Definida da seguinte forma:

```
if expressao_logica {
    ...
} else {
    ...
}
```

5.3 Estrutura iterativa com controle lógico

Definida da seguinte forma:

```
during expressao_logica {
    ...
}
```

5.4 Estrutura iterativa controlada por contador

Definida da seguinte forma:

```
from id = expressao_aritmetica to expressao_aritimetica increment expressao_aritmetica {
    ...
}
```

5.5 Entrada e saída

- Entrada:

```
<Type> identifier = get();
```

  Se o tipo da entrada for diferente do tipo da variável ocorre um erro.

- Saída:

```
print();
```

6 Programas Exemplos

6.1 Hello World

```
int Main() {

    print("Hello World");

    return 0;
}
```

6.2 Fibonacci

```
void Fibonacci(int value) {

    int a = 1;
    int b = 1;
    int next;

    if value == 0 {
        return;
    }

    during next <= value {
```

```
        print(a);
        print(", ");
        next = a + b;
        a = b;
        b = next;

    }

    print(a);
    print("\n");

    return;
}

int Main() {

    int value = get();

    Fibonacci(value);

    return 0;
}
```

6.3 Shell Sort

```
int[] ShellSort(int values[], int size) {

    int i;
    int j;
    int num;
    int gap = 1;

    during gap < size {
        gap = gap * 3 + 1;
    }

    during gap > 0 {
        from i = gap to size - 1 increment 1 {
            num = values[i];
            j = i;
            during (j > gap - 1) and (num <= values[j - gap]) {
                values[j] = values[j - gap];
                j = j - gap;
            }
            values[j] = num;
        }
        gap = gap / 3;
```

```
    }

    return values;
}

int Main() {

    int i;
    print("Enter the length of array:");
    int size = get();
    int values[size];
    print("Enter the array");

    from i = 0 to size - 1 increment 1 {
        values[i] = get();
        print(values[i]);
        print(" ");
    }

    values = ShellSort(values, size);
    from i = 0 to size - 1 increment 1 {
        print(values[i]);
        print(" ");
    }
}
```

7 Especificação da Linguagem de Programação

Os analisadores léxico e sintático da linguagem serão implementados em Java, utilizado o analisador preditivo recursivo.

8 Especificação dos Tokens

8.1 Lista de Tokens

A lista de tokens é definida por:

```
public enum CategoryList {

    Tunknown, Tmain, Tglobal, Tint, Tfloat, Tstring, Tbool, Tchar,
    Tvoid, TvecInt, TvecFloat, TvecStr, TvecBool, TvecChar,
    Tcomma, Tif, Telse, Tduring, Tfrom, Tto, Tincrement, TsemiCol,
    TbegBrac, TendBrac, TbegSqBrac, TendSqBrac,
    TbegCrBrac, TendCrBrac, Tprint, Tget, Treturn, TopConc,
    TopAnd, TopOr, TopNot, TopAtr, TopEq, TopDif,
    TopAdd, TopSub, TopDiv, TopMult, TopMod, TopLowThen,
```

```
        TopLowThnE, TopGreThen, TComment,
        TopGreThnE, TfuncId, TnameId, TcteInt, TcteFloat,
        TcteString, TcteBool, TcteChar

    }
```

9 Especificação das Expressões Regulares

9.1 Expressões Regulares Auxiliares

```
letter = ‘[a-zA-Z]’
letterLow = ‘[a-z]’
letterHig = ‘[A-Z]’
digit = ‘[0-9]’
symbol = ‘[ /\\!@#$%&*()_\-=+\[\]{}><?.;:,"’]’
```

9.2 Lexemas

```
Main:
    Tmain = ‘Main’

Identificador:
    TnameId = ‘(letterLow)(letter|digit|_)*’
    TfuncId = ‘(letterHig)(letter|digit|_)*’

Comentário:
    TComment = ‘#.*’

Tipos Primitivos:
    Tint = ‘int’
    Tfloat = ‘float’
    Tchar = ‘char’
    Tstring = ‘string’
    Tbool = ‘bool’
    Tvoid = ‘void’
    Tglobal = ‘global’

Delimitadores:
    Escopo:
        TbegCrBrac = ‘\{’
        TendCrBrac = ‘\}’

    Parâmetros:
        TbegBrac = ‘\(’
        TendBrac = ‘\)’
```

```
    Array:
        TbegSqBrac = '\['
        TendSqBrac = '\]'

    Finalizador:
        TsemiCol = ';'

    Separador:
        Tcomma = ','

Definiçoes de tipos:
    TcteInt      = 'digit+'
    TcteFloat    = 'digit+(\.digit+)?'
    TcteBool     = '(true|false)'
    TcteChar     = ''(letter|digit|symbol)''
    TcteString   = '"(letter|digit|symbol)*"'

Palavras reservadas de fluxo:
    Tif = 'if'
    Telse = 'else'
    Tfrom = 'from'
    Tduring = 'during'
    Tto = 'to'
    Tincrement = 'increment'
    Treturn = 'return'

Operadores lógicos:
    TopAnd = 'and'
    TopOr = 'or'
    TopNot = 'not'

Operadores Aritiméticos:
    TopAdd = '(+)'
    TopSub = '(-)'
    TopMult = '(*)'
    TopDiv = '(/)'
    TopMod = '(%)'

Operadores relacionais:
    TopEq = '(==)'
    TopDif = '(!=)'
    TopGreThen = '(>)'
    TopGreThnE = '(>=)'
    TopLowThen = '(<)'
    TopLowThnE = '(<=)'

Operador de Concatenação:
    TopConc = '&'
```