Universidade Federal de Santa Catarina
Departamento de Computação

Disciplina: Construção de Compiladores.
Docente: Marlon de Matos de Oliveira
Discentes: Gabriel Estevam - 15104138
Vinicius Zanon - 15102833

Manual da Linguagem

1 Introdução

Este documento irá descrever todas as regras referentes às características léxicas e sintáticas da linguagem. O manual possui uma abordagem de alto nível, com definições objetivas e acompanhadas de exemplos, permitindo o entendimento da linguagem e a construção de programas.

2 Características da Linguagem

Uma linguagem pode ser definida como um conjunto de símbolos, palavras e regras usadas na construção de sentenças que expressam e processam instruções para os computadores. A seguir, será exposto características que denotam essa linguagem, seguindo uma estrutura lógica da gramática disponibilizada. Para a correta escrita de um código na linguagem é necessário a inserção de espaços (ou símbolos finalizadores, e quando necessário, '\n', ',' ou ';') entre os lexemas utilizados.

2.1 Comentários

Inscrições feitas durante trechos de comentários não irão participar da lógica do programa. A delimitação é feita através dos símbolos `/*`, no início e `*/` no fim. Não é definido comentário de linha.

Exemplo:

```
/*  Trecho em comentário. */
```

2.2 Tipos de dados

Os tipos de dados são responsáveis por definir as características do dado que poderão ser armazenados em variáveis, passados como parâmetro de funções ou participação de expressões lógicas e comparativas.

Tipo	Definição/Intervalo	Exemplo
integer	Número inteiro entre 0 e 9999.	1, 2, 0, 1012, 9999
float	Número com precisão decimal. 8 dígitos com ponto flutuante entre a 2ª e penúltima posição.	0.1549, 3215.321
string	Cadeia de caracteres de tamanho máximo 32. Delimitação entre “ e ”.	“Satoshi Nakamoto”
char	Caracter único. Delimitação entre ‘ e ’.	‘A’, ‘-’, ‘?’, ‘4’
literal	Cadeia de caracteres de tamanho máximo 32. Delimitação entre ~ e ~.	~Saída: ~
void	Tipo de dado nulo ou não especificado.	-

2.3 Declaração de variáveis

Uma variável armazena um dado de tipo específico da linguagem. Possui um identificador que será único durante a execução do programa. Não poderá iniciar com número ou carácter especial e deve ter tamanho mínimo 1 e máximo 64. Não poderá ser idêntico a um tipo de dado ou palavra reservada da linguagem.

A declaração de todas as variáveis do programa devem ser de uma única vez, em uma única linha.

2.3.1 Declaração única.

Definido pelo `identificador` seguido do caracter `:` e na sequência o `tipo` do dado e por fim o caracter `;`.

Exemplo:

```
x : integer;
```

2.3.2 Declaração múltipla.

Para declarar múltiplas variáveis de um mesmo tipo deve inserir o caracter `,` mais um espaço e o novo `identificador` antes do caracter `:`, quantas vezes forem necessário.

Exemplo:

```
x, y, z : integer;
```

2.3.3 Declaração de mais de um tipo.

Para declarar mais de um tipo de variável deve-se concatenar as declarações de cada tipo, única(2.3.2) ou múltipla(2.3.3) em uma única linha.

Exemplo:

```
x, y : integer; a, b : char; num : float; entry : string;
```

2.4 Operadores

Nesta seção serão definidos os operadores aritméticos, operadores de comparação e o operador de atribuição.

2.4.1 Operadores aritméticos

São utilizados para realizar operações matemáticas.

Operador	Definição	Exemplo
+	Soma	5 + 4
-	Subtração	5 - 4
/	Divisão	5 / 4
*	Multiplicação	5 * 4

2.4.2 Operadores relacionais

São utilizados para montar expressões lógicas

Operador	Definição	Exemplo
==	Igualdade	x == y
!=	Diferença	x != y
>	Maior	5 > y
>=	Maior ou igual	x >= 4
<	Menor	5 < y
<=	Menor ou igual	x <= 3

2.4.3 Operador de atribuição

O operador de atribuição é definido pelo caracter `=` , e é utilizado para armazenar um dado em uma variável oriundo de uma expressão aritmética ou retorno de uma função.

Exemplos:

```
x = 5 + 4
y = x * 3
```

2.5 Expressões aritméticas

A linguagem permite a criação de expressões aritméticas constituídas por variáveis, números e operadores aritméticos. Expressões podem ser concatenadas com outras expressões com os operadores aritméticos e ainda uma expressão pode ter precedência utilizando símbolos de parênteses `( )`.

Exemplos:

```
 x + y * z
 x + 5 / 2
(a + b) / c
```

```
(a / b + 3) * ((x + y) / 2)
```

2.6 Comandos de entrada e saída

Comandos de entrada e saída permitem realizar a leitura de dados através do teclado e imprimir dados na tela.

2.6.1 Entrada

Para realizar a entrada de dados através do teclado deve-se utilizar o comando `cin >>` seguido do identificador da variável que armazenará o dado e por fim o caracter `;` para encerrar o comando.

Exemplo:

```
cin >> entry;
```

2.6.2 Saída

Para imprimir dados na tela deve-se utilizar o comando `cout <<` seguido de um dado literal de algum tipo previsto na linguagem e por fim o caracter `;` para encerrar o comando. É possível acrescentar novas saídas a partir do mesmo comando adicionado `<<` antes do caracter `;` seguido de um novo dado literal, o identificador de uma variável ou ainda uma sequência de variáveis separadas pelo caracter `,` .

Exemplos:

```
cout << ~Saída~;
cout << ~Saída = ~ << x;
cout << ~Saída = [~ << x, y << ~]~;
```

2.7 Estruturas de Controle

2.7.1 Estrutura de Seleção ou Desvio Condicional

Os comandos de seleção permitem estabelecer pontos de desvios condicionais ao programa, isto é, decidir o fluxo de execução a partir de avaliação de alguma condição.

2.7.1.1 Seleção Simples

Utilizada quando necessita-se testar certa condição antes de executar uma ação/comando. Condição pode ser entendida como uma expressão lógica que ao ser avaliada pode retornar um valor falso ou verdadeiro. Sua composição é caracterizada pelo identificador de seleção simples `if` acompanhado dos caracteres parênteses `( )` englobando uma expressão condicional constituída por dois operandos entre um operador relacional. O primeiro operando sempre será uma variável enquanto o segundo pode ser uma variável ou um valor literal. O fim da declaração de uma seleção simples é identificada pelas chaves `{ };` , sendo que entre elas, será executado o `COMANDO`.

Exemplo:

```
if ( a == b ) {
     COMANDO
};
```

2.7.1.2 Seleção Composta

A seleção composta é utilizada quando tem-se duas alternativas a partir de uma mesma condição. Na seleção composta sempre haverá uma condição verdadeira e outra falsa. Sua composição é caracterizada pelos identificadores `if` e `else` . O primeiro identificador está disposto conforme visto na seleção simples. O segundo é iniciado com o identificador `else` acompanhado pelas chaves `{ }` , sendo que entre elas, será executado o `COMANDO` que caracteriza a condição de falsidade, já que o `COMANDO` da seleção simples `if` implica na ação dada a uma condição de verdade.

Exemplo:

```
if ( a == b ) {
     COMANDO
} else {
     COMANDO
};
```

2.7.1.3 Seleção Encadeada

A seleção encadeada permite aninhar comandos de seleção dentro de comandos de seleção. É utilizada quando for necessário avaliar uma condição dentro de um bloco de comandos pertencentes a um `if` ou a um `else`.

Exemplo:

```
if ( a == b ) {
     COMANDO
} else {
     if ( a > b ) {
          COMANDO
     };
};
```

2.7.2 Estruturas de Repetição

Os comandos de repetição permitem executar uma ou um conjunto de instruções por um número determinado de vezes.

2.7.2.1 Repetição com Teste no Início

A repetição com teste no início permite repetir diversas vezes o mesmo trecho do algoritmo, entretanto existe uma verificação antes de cada execução. O objetivo da verificação ou teste é validar se é permitido executar mais uma vez o trecho de código ou não. É formado pelo identificador `while` acompanhado dos caracteres parênteses `( )` englobando uma expressão condicional constituída por dois operandos entre um operador relacional. O primeiro operando sempre será uma variável enquanto o segundo pode ser uma variável ou um valor literal. O fim da declaração de uma repetição com teste no início é identificada pelas chaves `{ };` , sendo que entre elas, será executado o `COMANDO`.

Exemplo:

```
while ( a != b ){
     COMANDO
};
```

2.7.2.2 Repetição com Teste no Final

A repetição com teste no final é composta por uma condição de parada do laço de repetição no fim do bloco de instruções do laço. O teste no final permite executar ao menos uma vez o(s) comando(s) dentro do laço. É descrita com o identificador `do` acompanhada de chaves `{ }`, sendo que entre elas, será executado o `comando`, pelo menos uma vez, e se repetirá caso a expressão condicional do laço de repetição seja validada.

Exemplo:

```
do {
      COMANDO
} while( a >= b );
```

2.7.2.3 Repetição com Variável de Controle

As estruturas de repetição com teste no início ou no fim dependem que uma determinada condição aconteça para que o laço conclua sua execução, o que pode levar o algoritmo a uma execução infinita. A estrutura de repetição com variável de controle executa um determinado trecho do programa um número finito de vezes que já foi definido a priori. É composto pelo identificador `for` acompanhado pelos caracteres parênteses `( )`, englobando uma estrutura constituída pelo início da contagem da variável de controle `i`, pela determinação do seu valor final de incremento e pelo passo de incremento dessa variável. O fim da declaração de uma repetição com variável de controle é identificada pelas chaves `{ };` , sendo que entre elas, será executado o `COMANDO`. O passo é definido por um operador de incremento, que pode ser `++` ou `--` seguido de um valor inteiro.

Exemplo:

```
for ( i = a; i < b; ++ 2 ) {
      COMANDO
};
```

2.8 Funções

Uma função pode ser entendida como um sub-rotina que realiza uma tarefa específica quando chamada. Ela é utilizada para modular o programa e permitir a reutilização de código.

2.8.1 Declaração de Funções

A declaração de uma função é constituída por um tipo de retorno acompanhada ao nome da função e parâmetros de entrada: nenhum parâmetro ou um ou mais sequências de tipos acompanhados por identificadores de variáveis entre parênteses `( )` separados por `;` , caso seja mais de um parâmetro.

Seu corpo é delimitado por chaves `{ }` , sendo que entre elas, pode ser definido novas variáveis, funções e comandos. Por fim, é necessário utilizar o identificador `return` acompanhado do valor de retorno entre parênteses `( )`. O valor de retorno pode ser um literal, uma variável, ou ainda, não retornar nenhum parâmetro.

Funções podem ser declaradas diversas vezes e devem ser feitas em sequência.

Exemplo:

```
void soma ( integer ; integer  ) {
     c : integer;
     inicio
          c = a + b;
          cout << "Soma: " << c;
     fim
     return ( )
}

integer potencia2( integer ) {
     b : integer;

     integer multiplicacao( integer ) {
          m : integer;
          inicio
               m = c * c;
          fim
          return ( m )
     }

     inicio
```

```
            b = callfuncao multiplicacao (a); /* Verificar a
chamada de função */
        fim
        return ( b )
    }
```

2.8.2 Chamada de Funções

Para a utilização de uma chamada de função é necessário utilizar o identificador `callfuncao` seguido do seu nome e, caso exista, entre parênteses uma sequência de parâmetros.

Exemplos:

```
callfuncao soma;
result1 = callfuncao multiplicacao ( a );
result2 = callfuncao soma ( a , b );
```

2.8.3 Função Main

Todo programa deve ter uma função principal que irá ser chamada no início da execução do programa. Essa função é definida pelo identificador `void main` seguido de chaves `{ }`, entre elas, podem ser declaradas variáveis, funções e comandos.

Exemplo:

```
void main {

    inpt1, inpt2, outp: integer; /*Declaração Variáveis*/

    integer soma ( integer ; integer  ){ /*Função Soma*/
        resultado : integer;
        inicio
            resultado = a + b;
        fim
        return ( resultado )
    }
```

```
        inicio
                cout << ~Digite o 1 número: ~;
                cin >> inpt1;
                cout << ~Digite o 2 número: ~;
                cin >> inpt2;
                outp = callfuncao soma ( input1, input2 );
                cout << ~Soma: ~ << outp;
        fim
}
```

3 Erros Léxicos

Erros léxicos podem ser identificados, em sua maioria, na utilização de caracteres não presentes na gramática da linguagem. Ademais, extrapolação de valores atribuídos ao limite de cada tipo de variável, a escrita de comentários (ausência dos símbolos de fechamento), ou ainda, caracteres não esperados em determinados contextos, podem ser classificados como erros léxicos.

Exemplo:

```
void main {
        inpt1, inpt2, outp: integer; /*Declaração Variáveis
        integer soma ( integer , integer  ){
                resultado : integer;
                inicio
                        resultado = a+-b;
                fim
                return ( resultado )
        }
        inicio
                inpt1 = 123456;
                inpt2 = .15;
                cout << ~Digite o 1 número: ~;
                cin >> inpt1;
                cout << 'Digite o 2 número: ~;
                cin >> inpt2;
                outp = callfuncao soma ( integer, integer);
                cout <> ~Soma: ~ << outp;
        fim
}
```