Universidade Federal de Alagoas

Especificação da Linguagem AVI

Aluno: David Silva Alexandre

Professor: Alcino Dall'lgna Júnior

2021

Sumário

Introdução

A linguagem AVI foi criada para implementação dos analisadores léxico e sintático solicitados na disciplina de Compiladores. O principal objetivo é concluir a implementação destes, adquirindo assim mais conhecimento sobre a área em questão e consequentemente aprovação na disciplina.

Objetivo

O principal objetivo do presente trabalho é fornecer uma visão geral da linguagem, bem como definir especificações da mesma, para que seja possível um melhor entendimento da linguagem que será utilizada nos trabalhos da disciplina. Já no que diz respeito a criação da linguagem, o principal objetivo a ser alcançado foi a construção de um vocabulário com facilidade de leitura e escrita para facilitar a criação de programas.

Estrutura Geral de um Programa

Faremos uma descrição de como é a estrutura geral de um programa escrito em AVI, com informações sobre seu bloco de execução, declaração de variáveis e um exemplo da estrutura de um programa nesta linguagem. Sendo assim, temos

1. Bloco de Execução

   Como na linguagem C, nosso bloco de execução é composto por chaves "{}", que delimitarão o início e o fim de um bloco de execução. AVI não possui indentação, dessa forma, ignoraremos espaços em branco.

2. Declaração de variáveis

   Variáveis podem ser declaradas em qualquer parte do programa, a restrição aqui é que devem ser usadas sempre depois de serem declaradas, portanto, a declaração sempre deve vir antes do uso da variável. Variáveis criadas dentro de um bloco de execução só podem ser acessadas dentro do bloco que foram criadas. Cada variável poderá ter até 64 caracteres e não poderá conter acentuação ou começar com um dígito [0 ... 9]. Variáveis podem conter [0 ... 9], [a ... z] e [A ... Z].

3. Exemplo de Estrutura

   A seguir um exemplo da estrutura de um programa em AVI

```
main[]
{
    int numer;
    print[numer];
}
```

Especificação de Tipos

Temos vários tipos de dados em AVI, que incluem Inteiro, Ponto Flutuante, Caractere, Booleano, Cadeia de caracteres e os famosos Arranjos Unidimensionais(Arrays). A seguir, temos a especificação de cada tipo:

1. Inteiro
    - Tipo: Primitivo
    - Palavra Reservada: **int**
    - Operações: As operações disponíveis para o tipo inteiro são as Aritméticas(adição, subtração, multiplicação, divisão, unários, incremento e decremento), Relacionais(igual, diferente, maior, menor, maior ou igual, menor ou igual) e por fim as de Concatenação(com caracteres e cadeias de caracteres)
2. Ponto Flutuante
    - Tipo: Primitivo
    - Palavra Reservada: **float**
    - Operações: Para ponto flutuante temos operações Aritméticas(adição, subtração, multiplicação, divisão, unários, incremento e decremento), Relacionais(igual, diferente, maior, menor, maior ou igual, menor ou igual) e ainda Concatenação(com caracteres e cadeias de caracteres)
3. Caractere
    - Tipo: Primitivo(ascii)
    - Palavra Reservada: **char**
    - Operações: Para caracteres temos operações Relacionais(igual, diferente, maior, menor, maior ou igual, menor ou igual) e de Concatenação(com caracteres e cadeias de caracteres)
4. Booleano
    - Tipo: Primitivo
    - Palavra Reservada: **bool**
    - Operações: As operações disponíveis aqui são Lógicas(negação, conjunção, disjunção inclusiva, disjunção exclusiva), Relacionais(igual e diferente) e Concatenação(com caracteres e cadeias de caracteres)
5. Cadeia de caracteres
    - Tipo: Não Primitivo
    - Palavra Reservada: **string**
    - Operações: Incluem Relacionais(igual e diferente) e Concatenação(com caracteres e cadeias de caracteres)
6. Arranjos Unidimensionais
    - Tipo: Não Primitivo
    - Sintaxe “tipo” quantidade

Especificação de Constantes Literais

1. Inteiros

   Valores inteiros de até 32 bits. Coerção para float é permitida, caractere de acordo com a tabela ascii e string.
2. Float

   Para valores de ponto flutuante temos até 32 bits ou mais se estiver em notação científica. Permite coerção para string e inteiro.
3. Booleano

   Palavras reservadas true e false, permitindo coerção para inteiro e string.
4. Caracteres

   Qualquer caractere da tabela ascii. Coerção para inteiro de acordo com o valor correspondente na tabela ascii.
5. Cadeia de caracteres

   Sequência de caracteres em aspas duplas.

Conjunto de Operadores

Aqui serão mostrados todos os operadores que fazem parte da linguagem. Temos os seguintes tipos de operadores:

1. Aritméticos
   - Adição: +
   - Subtração e Unário Negativo: -
   - Multiplicação: *
   - Divisão: /
   - Incremento: ++
   - Decremento: --
2. Relacionais
   - Numéricos, Caracteres e Cadeia de caracteres: =, >=, <, <=, !=
   - Booleanos: =, !=
3. Lógicos
   - Booleanos: &, !, ^, |
   - Negação: !
   - Disjunção: ||
   - Conjunção: &&
4. Concatenação
   - Todos os tipos que suportam +=

Instruções

A seguir uma especificação das formas de controle

1. Condicional de uma via
    - Palavra reservada **if**
    - **if**[ExprLog]{bloco de execução}
2. Condicional de duas vias
    - Palavras reservadas **if** e **else** / símbolos "**?**" e "**:**"
    - **if**[ExprLog]{bloco de execução} **else**{bloco de execução}
    - .[ExprLog] **?** {bloco para verdadeiro} **:** {bloco para falso}
3. Estrutura iterativa com controle lógico
    - Palavra reservada **while**
    - **while**[ExprLog]{bloco de execução}
4. Estrutura iterativa controlada por contador
    - Palavra reservada **for**
    - **for[**varInt; expArit; expArit; expArit] bloco de execução
    - Note que para o for acima a primeira "expArit" é o ínicio e a segunda o fim. A terceira é o tamanho do passo, caso omitido, contém o padrão 1.
5. Entrada e Saída
    - **Entrada**: palavra reservada **scan**
    - **scan**[lista de variáveis separadas por vírgula];
    - **Saída**: palavra reservada **print**
    - print[lista de variáveis separadas por vírgula]
    - Vale ressaltar que caso venha um número após a vírgula, o valor da variável anterior será formatado com a quantidade do número
    - Exemplo: considere x como um inteiro, x = 1 e y um float y = 1, se fizermos print[a, 2, b, 3] teremos como saída 01 e 1000
    - No print constantes não são permitidas
    - Caso o número seja muito grande, usaremos notação científica para expressá-lo
6. Atribuição
    - Consideramos a atribuição como um **operador**
    - Operador = "tipo da variável" = "valor"

Funções

Exibiremos agora informações sobre funções, tais como modelos semânticos de passagem de parâmetro e afins.

- Ao iniciar o programa a função main será a primeira a ser chamada
- Iniciamos a declaração da função com a palavra reservada do tipo que será retornado
- Também pode iniciar com a palavra reservada void caso não tenha retorno
- Após a definição do retorno o nome deverá ser definido da mesma forma que o nome de uma variável
- Nas funções temos o nome da mesma, em seguida o bloco de parâmetros[tipo nome, tipo nome, ...] seguida do bloco de execução{...}
- No final da função, temos a palavra reservada return seguida do nome da variável que será retornada, acompanhada de um ponto e vírgula
- Confira um exemplo abaixo

```
int Exemplo[int parametro1, char parametro2]
{
    //comandos
}
```

Programas exemplo

1. **Alô mundo**

```
void main[]
{
    string alo = "Alo Mundo!"
    printf(alo);
}
```

2. **Programa Fibonacci**

```
void main[]
{
    int n;
    int lt = 0;
    int rt = 1;
    scan[n];

    if[n > 0]
    {
        print[lt];
    }
    if[n > 1]
    {
        print[rt];
    }

    while[n > 2]
    {
        n --;
    }
}
```

3. **Programa Shellsort**

```
int vetor(100);
int n;

void shellSort[]
{
    int h = 1 ;
    while[h < n]
    {
        h = h * 3 + 1 ;
    }

    h = h/3 ;

    int temp, j;
    while[h > 0]
    {
        for[i = h; i < n; i++]
        {
            temp = a(i) ;
            j = i ;
            while[j >= h && a(j-h) > temp
            ]{
                a(j) = a(j-h) ;
                j = j - h ;
            }
            a(j) = temp ;
        }
        h = h / 2 ;
    }

}
```

```
void main[]
{
    int i ;
    scan[n] ;
    for[i ; 0; n]
    {
        scan[vetor(i)] ;
    }
    shellSort[];
    for[i = 0; i < n; i++]
    {
        print[vetor(i)] ;
        print[" "] ;
    }
}
```