UNIVERSIDADE FEDERAL
DE ALAGOAS

UNIVERSIDADE FEDERAL DE ALAGOAS – UFAL

INSTITUTO DE COMPUTAÇÃO

CIÊNCIA DA COMPUTAÇÃO

COMPILADORES

Professor Alcino Dall'Igna Júnior

AUDREY EMMELY RODRIGUES VASCONCELOS

NATÁLIA DE ASSIS SOUSA

WILLIAM PHILIPPE LIRA BAHIA

MACEIÓ, AL 17 DE DEZEMBRO DE 2021

SUMÁRIO

1. Estrutura geral do programa

A linguagem ANW admite escopo de bloco, com recursão. Ou seja, um bloco interno tem acesso ao escopo de todos os seus pais, mas não tem acesso ao escopo de um bloco no mesmo nível que ele. O arquivo deve ter a extensão "*.anw*". O programa sempre deve começar com a palavra reservada "*start*". Após o início, deve-se abrir um bloco de código com "{" e esse bloco deve, obrigatoriamente, ser fechado com "};".

Toda instrução da linguagem ANW deve terminar com ";", incluindo a inicial. Toda instrução deve estar dentro de um bloco "{...}".

Um bloco de código marca o início e o fim de um escopo. Todo bloco deve ser iniciado com "{" e finalizado com "}".

Existem algumas palavras reservadas nesta linguagem, as quais você pode verificar no tópico 3.

Como uma linguagem fortemente tipada, ANW aceita inteiro, ponto flutuante, caractere, cadeia de caracteres, booleano e arranjo unidimensional com tipo definido.

As variáveis podem ser declaradas em qualquer lugar. Para declarar variáveis globais, basta declarar no bloco inicial "*start*". Variáveis respeitam o escopo de bloco.

```
start {
  function int[] shellSort(int[] array) {
    int n = array.size;
    int gap = (int) array.size / 2;

    for (; gap > 0; gap = (int) gap/2) {
      loop (int i; gap; n;) {
        int temp = array[i];

        int j;
        for (j = i; j >= gap & array[j - gap] > temp; j = j - gap) {
          array[j] = array[j - gap];
        }

        array[j] = temp;

      }
    }

    return array;
  }
};
```

Figura 1. Exemplo de implementação do algoritmo Shell Sort em ANW.

2. Nomes

Nomes são sensíveis à caixa. Não possuem limite de caracteres. Podem conter qualquer caractere alfabético e numérico. Só podem começar com caracteres alfabéticos minúsculos.
A expressão regular para aceitar um nome:
[a-z][a-zA-Z0-9]*

3. Palavras reservadas

int, float, char, string, bool, undefined, if, else, elseif, do, loop, while, for, start, get, put, function, return, size.

4. Tipos e estruturas de dados

4.1. Forma de declaração

Tipo	**Declaração**
Inteiro	int nome = 1;
Ponto flutuante	float nome = 2.1;
Caractere	char nome = ‘c’;
Cadeia de caracteres	string nome = “tipo”;
Booleanos	bool nome = false | true;
Arranjos	tipo[tamanho] nome;

4.2. Tipos de dados primitivos

4.2.1. Inteiro

Deve ser identificado pela palavra reservada *int* e representa um número inteiro de 64 bits. Seus literais são uma sequência de números inteiros.
Suporta operações de atribuição, aritméticas e relacionais.
Constante de inteiros: ((‘digit’)+);
Exemplo: *int* valor = 1;

4.2.2. Ponto flutuante

Deve ser identificado pela palavra reservada *float* e representa um número real de 64 bits.

Suporta operações de atribuição, aritméticas e relacionais.
Constante de ponto flutuante: ((‘digit’)+)(‘\.’)((‘digit’)+);
Exemplo: *float* valor2 = 2.5;

4.2.3. 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. Ao atribuir um caractere a uma variável, o caractere deve estar contido dentro de apóstrofos.
Suporta operações de atribuição, relacionais e concatenação.
Constante de caractere: (‘\‘’)(‘letter’ | ‘symbol’ | ‘digit’)(‘\’’);
Exemplo: *char* letra = ‘A’;

4.2.4. Cadeia de caracteres

Deve ser identificado pela palavra reservada *string* e representa uma sequência de caracteres. Ao atribuir uma string a uma variável, a string deve estar contida dentro de aspas duplas.
Suporta operações de atribuição, relacionais e concatenação.
Constante de cadeias de caracteres: (‘\“’)((‘letter’ | ‘symbol’ | ‘digit’)*)(‘\”’);
Exemplo: *string* palavra = “felicidade”;

4.2.5. Booleanos

Deve ser identificado pela palavra reservada *bool* e representa somente um dos dois valores: *true* ou *false*.
Suporta operações de atribuição, lógica, relacional de igualdade ou desigualdade e concatenação.
Constantes de booleanos: (‘true’ | ‘false’);
Exemplo: *bool* condicao = true;

4.2.6. Arranjos

Um arranjo deve ter seu tipo seguido pelo seu tamanho especificado entre colchetes e, em seguida, seu identificador. Os arranjos são armazenados em posições contíguas na memória. A propriedade *size* retorna o número de elementos em um arranjo.
Exemplo: *float*[5] medias;
int a = (*int*) array.*size* / 2;

4.2.7. Constantes com nomes

A linguagem contém três constantes nomeadas, sendo duas do tipo *float* e uma do tipo *char*:
PI = 3.14159

EULER = 2.71
NEW_LINE = '\n'
A expressão regular das constantes com nomes é a seguinte:
[A-Z][_A-Z]*

4.3. Equivalência de tipos

4.3.1. Forma de equivalência

Um caractere é essencialmente uma string com tamanho 1. Todo inteiro pode ser convertido para ponto flutuante e vice-versa. Qualquer valor pode ser convertido para booleano.

4.3.2. Coerções admitidas

Tipo *int* ou *float* para tipo *string*
Exemplo: string a = 1 + '';
Saída: a = '1'
Ao ser concatenado com um espaço em branco, o valor da variável "a" agora será uma *string*.

Tipo *int* ou *float* para tipo *booleano*
Exemplo: int val = 100;
boolean a = (val == 100);
Saída: a = true
Para converter inteiro em booleano, primeiro vamos inicializar um inteiro. Após isso, vamos declarar uma variável booleano e inicializá-la com o valor val comparando-a com um inteiro usando o operador "==". Se o valor corresponder, o valor "true" será retornado, caso contrário, "false" será retornado.

4.3.3. Conversão de tipo explícita

Tipo *float* para tipo *int*
Exemplo: int a = (int) 1.2;
Saída: a = 1
Uma vez que o *float* é maior do que o *int*, é possível converter um *float* em um *int* simplesmente fazendo um "down-casting", como nesse exemplo.

Tipo *int* para tipo *float*
Exemplo: int a = (float) 1;
Saída: a = 1.0
De forma análoga ao que acontece na conversão de *float* para *int*, nesse caso o valor *int* sofre um "up-casting".

5. Atribuição e expressões

5.1. Atribuição

Atribuições em ANW são feitas através do símbolo “=”, sendo o lado esquerdo referente ao identificador do tipo da variável, e o lado direito refere-se ao valor ou a expressão atribuída à mesma.

Exemplo: *int* numero;

numero = 1;

5.2. Expressões aritméticas, relacionais e lógicas

5.2.1. Aritméticas

Operadores	**Operações**
+	Soma entre operandos ou concatenação entre strings
-	Subtração entre operandos
*	Multiplicação entre operandos
/	Divisão entre operandos
+, - (unário)	Desempenha uma ação em um único operando, o resultado do operador unário (+) é o valor de seu operando, o operador unário (-) inverte o sinal de uma expressão de positivo para negativo ou vice-versa. Tem que estar ligado à constante/id, sem espaço entre.

5.2.2. Relacionais

Operadores	**Operação**
==	Igualdade entre operandos
!=	Desigualdade entre operandos
>=	Maior ou igual que
<=	Menor ou igual que
>	Maior que
<	Menor que

5.2.3. Lógicos

Operadores	Operação
!	Negação
&	Conjunção
|	Disjunção

5.2.4. Precedência e associatividade

Operadores (precedência)	Associatividade
!	direita para esquerda
- e + (unário)	direita para esquerda
* e /	esquerda para direita
+ e -	esquerda para direita
< <= > e >=	não associativo
== e !=	não associativo
&	esquerda para direita
|	esquerda para direita

6. Sintaxe e exemplo de estruturas de controle

Todos os blocos devem possuir chaves ({ }).

6.1. Comandos de seleção

O comando de seleção a ser utilizado é o *if* que seleciona um caminho baseado em cada condição. Para utilizar mais de uma condição existe *elseif* e o bloco opcional *else*, caso nenhuma das condições sejam atendidas.

if (condição){
    ...
} elseif (condição){

```
        ...
} else {
        ...
}
```

6.2. Comandos de iteração

6.2.1. Controle por contador

A estrutura faz uso da palavra reservada *loop*, o número de iterações pode ser definido pelo programador e o controle é feito por um contador.
Essa estrutura deve possuir um valor de início e um valor final. Sendo a inicialização e o incremento valores opcionais, ainda assim, é preciso colocar ';' entre todos os parâmetros. Todos os valores são inteiros. Caso o incremento não seja informado, o mesmo recebe o valor 1 por padrão.

```
…
int n = array.size;
int gap = (int) array.size / 2;
…
loop (int i; gap; n;) {
        ...
}
```

6.2.2. Controle lógico

Existem três tipos de controles lógicos, em um deles a estrutura utiliza a palavra reservada *while*, e enquanto a condição não for satisfeita (verdadeira), o bloco de código é executado. É realizada uma verificação pré-teste para verificar se a repetição ainda ocorrerá ou não.

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

A outra estrutura utiliza as palavras reservadas *do* e while, e enquanto a condição não for satisfeita (verdadeira), o bloco de código é executado. Mas, nesse caso, realizando uma verificação pós-teste para verificar se a repetição ainda ocorrerá ou não.

```
do {
        ...
}while(condição)
```

A terceira estrutura utiliza a palavra reservada *for*, o comando *for* executa um conjunto de instruções, um determinado número de vezes de acordo com uma condição enquanto uma variável de controle é incrementada ou decrementada a cada passagem pelo "laço".

```
for (valor_inicial; condição_final; valor_incremento){
        ...
 }
```

6.3. Desvios incondicionais

A linguagem ANW não tem desvios incondicionais.

7. Entrada e Saída

A entrada é feita por meio da instrução *get()*, que recebe como argumentos as variáveis que irão armazenar a entrada fornecida pelo usuário, com o tipo correspondente.

Exemplo:

```
int var1;
int var2;
get(var1, var2);
```

A saída é realizada pela instrução *put()*, que irá mostrar na tela o(s) valor(es) correspondente(s) ao algoritmo específico no programa, através dessa instrução. Caso deseje imprimir a saída com quebra de linha, basta utilizar a constante *NEW_LINE,* como demonstrado em seguida.

Exemplo:

```
int valor = 10;
put(valor);

float a = 10.5;
float b = 15.5;
```

put(a, NEW_LINE, b);

put (“hello world”);

8. Subprograma

Funções podem ser declaradas a qualquer momento em qualquer lugar do código, desde que dentro do bloco inicial “*start*”. Estas também podem ser chamadas em qualquer lugar. Funções são declaradas com a palavra reservada “*function*”. Toda função deve ter seu retorno especificado, incluindo os casos nos quais a função não retorna nada, usando a palavra reservada “*undefined*”.

O retorno da função é definido pela palavra reservada *return* e seu valor.

Todos os parâmetros de funções devem ser tipados.

Após declarar a assinatura da função, o corpo de execução desta deve estar contido dentro de um bloco “{...}”.

Para chamar uma função, deve ser utilizado o seu identificador, e dentro dos parênteses os valores que serão utilizados pela mesma. Com exceção do arranjo, toda vez que for passado algum parâmetro para a função, será feita uma passagem de parâmetro por valor. Isso significa que o computador faz uma cópia da variável passada como parâmetro, permitindo usar dentro de uma função essa cópia do valor da variável, não sendo possível alterar o valor da variável original (somente a cópia pode ser alterada). Para arranjos, existe um outro tipo de passagem, que é por referência. Quando fazemos a passagem desse tipo, uma referência, ou seja, um endereço de memória é que é passado para a função. Nesse endereço, está a variável original. Assim, se a variável for alterada dentro de uma função, ela é alterada originalmente.

Exemplo:

```
function int fibonacci(int limite) {
   int fib = 0;
   int aux = 0;
   int n;
         ...
         ...
         ...
   return n;
 }
```