UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO - IC
CIÊNCIA DA COMPUTAÇÃO

JOÃO VICTOR DE ALARCÃO AYALLA ALCÂNTARA
ASCANIO SAVIO DE ARAUJO NEVES
JACKSON BARBOSA DA SILVA

COMPILADORES
ESPECIFICAÇÃO DA LINGUAGEM AJA++

Sumário

1 - Estrutura geral do programa

1.1 - Escopo

A linguagem admite tanto escopo local, quanto global.

1.2 - Ponto de início de execução

O ponto inicial da execução é a função main que é obrigatória e possui algumas propriedades:

- A função main deve ser do tipo void;
- Só pode existir uma função main;

```
function void main{} [
  ...
]
```

Figura 1: Estrutura da função inicial do programa

1.3 - Definição de funções

As funções podem ser declaradas em qualquer lugar do código, entretanto não podem ser declaradas dentro de outras funções e são visíveis a partir do momento que são declaradas. Além disso, o corpo de cada função é delimitado pela abertura e fechamento de colchetes.

```
function itg func123{itg x, bool b, dbl c} [
  return 0;
]
```

Figura 2: Estrutura de uma função

1.4 - Definição de instruções

As instruções só podem ser declaradas dentro do escopo de funções, além de que cada instrução deve ser finalizada com ‘ ; ’.

1.5 - Variáveis

Em uma declaração de variável é possível definir mais de uma variável na mesma instrução utilizando o separador vírgula, desde que as variáveis sejam do mesmo tipo. A linguagem suporta, também, a inicialização das variáveis durante a declaração.

```
str s = "Hello World";
bool b;

function void main{} [
  itg a = 2;
]
```

Figura 3: Exemplo de declaração e inicialização de variáveis

Os nomes das funções e variáveis devem ser iniciados por letras [a-z, A-Z] e depois podem conter letras [a-z, A-Z], dígitos [0 - 9] (Pode ser definida pela expressão regular seguindo o padrão do flex: '[:upper:] | [:lower:]+[:upper:] | [:lower:] | [:digit:]*'). A linguagem é case sensitive.

2 - Tipos de dados e nomes

2.1 - Palavras Reservadas

As palavras reservadas da linguagem são:

main, function, return, if, elseif, else, for, while, append, void, itg, dbl, chr, bool, str, list, read, write, true, false, and, or, not.

2.2 - Inteiro

O tipo inteiro é definido pela palavra reservada itg, representando um número inteiro de 32 bits. Pode receber uma constante inteira ou outra variável do mesmo tipo. Seus literais são expressos por um sinal '+' ou '-' (opcional) e uma sequência de dígitos (1 ou mais).

```
function void main{} [
  itg a = 2;
]
```

Figura 4: Exemplo de declaração e inicialização de um inteiro

2.3 - Ponto flutuante

O tipo ponto flutuante é definido pela palavra reservada dbl, representando um número de ponto flutuante, utilizando 64 bits de tamanho. Pode receber uma constante de ponto flutuante ou outra variável do mesmo tipo. Seus literais são expressos por um sinal '+' ou '-' (opcional) e uma sequência de um ou mais dígitos, após isso um ponto e por fim uma sequência de um ou mais dígitos.

```
function void main{} [
  dbl pi = 3.14;
]
```

Figura 5: Exemplo de declaração e inicialização de um número de ponto flutuante

2.4 - Booleano

O tipo booleano é definido pela palavra reservada bool, e tem tamanho de 1 byte por ter apenas dois valores possíveis (true ou false). Pode receber uma constante booleana ou outra variável do mesmo tipo. Seus literais são expressos como ‘true’ ou ’false’.

```
function void main{} [
  bool flag = true;
]
```

Figura 6: Exemplo de declaração e inicialização de uma variável booleana

2.5 - Char

O tipo caractere é definido pela palavra reservada char, tem tamanho de 1 byte e guarda um código referente ao seu símbolo na tabela ASCII. Pode receber uma constante char ou outra variável do mesmo tipo. Seus literais são expressos por uma letra, símbolo ou dígito.

```
function void main{} [
  chr c = 'a';
]
```

Figura 7: Exemplo de declaração e inicialização de um caractere

2.6 - String

O tipo string é caracterizado por uma cadeia de caracteres cujos códigos estão na tabela ASCII e é definido pela palavra reservada str. Pode receber uma constante string ou outra variável do mesmo tipo e o seu tamanho varia de acordo com a última cadeia de caracteres atribuída. Seus literais são expressados por um conjunto de letras, símbolos ou dígitos.

```
function void main{} [
  str s = "Hello World";
]
```

Figura 8: Exemplo de declaração e inicialização de uma string

2.7 - Arranjos Unidimensionais

O tipo arranjo unidimensional é responsável por um array de um tipo e é definido pela palavra reservada list. O tamanho é dinâmico, no qual é inicializado como 0 após ser declarado e, ao serem inseridos elementos nessa lista (append), seu tamanho vai sendo alterado. A atribuição pode ser feita com a palavra reservada “append”, a qual insere um elemento no final da lista, ou pode ser feita diretamente no índice, utilizando a sintaxe: list(i) = element.

```
function void main{} [
  list (itg) l;
  l.append(1);
  l.append(2);
  l(0) = 3;
]
```

Figura 9: Exemplo de declaração e inicialização de arranjos unidimensionais

3 - Operações

3.1 - Tipos de operações

Os operandos de uma operação devem obrigatoriamente ser do mesmo tipo, logo o retorno de uma operação será do mesmo tipo dos operandos (válido para todas as operações com exceção da concatenação).

3.1.1 - Operadores Aritméticos

Operação	Símbolo	Exemplo
Adição	+	var_a + var_b

Subtração	-	var_a - var_b
Multiplicação	*	var_a * var_b
Divisão	/	var_a / var_b

3.1.2 - Operadores Relacionais

Operação	**Símbolo**	**Exemplo**
Igual	==	var_a == var_b
Diferente	!=	var_a != var_b
Menor que	<	var_a < var_b
Maior que	>	var_a > var_b
Menor ou igual	<=	var_a <= var_b
Maior ou igual	>=	var_a >= var_b

3.1.3 - Operadores Lógicos

Operação	**Símbolo**	**Exemplo**
Conjunção	and	var_a and var-b
Disjunção	or	var_a or var_b
Negação	not	not var_a

3.2 - Operações de cada tipo

Vale ressaltar que as operações não realizam coerção. Segue a tabela com todas as operações suportadas pela linguagem:

Operador	Tipos que realizam a operação
‘*’, ‘/’, ‘+’, ‘-’	itg, dbl
‘<’, ‘>’, ‘<=’, ‘>=’	itg, dbl
‘==’, ‘!=’	itg, bool, dbl, chr, str
‘and’, ‘or’, ‘not’	itg, bool

3.3 - Valores padrões

A linguagem possui valores padrões para cada tipo, mais especificamente para as variáveis que não tiveram inicialização. São eles:

Tipo	Valor
itg	0
dbl	0.0
bool	false
chr	‘’
str	“”

3.4 - Coerção

A linguagem não suporta coerção.

3.5 - Ordem de precedência e associatividade

A ordem de precedência vai de cima para baixo e tem a seguinte tabela:

Operador	Ordem de precedência	Associatividade
Negação	‘!’	Direita para esquerda
Operadores aritméticos	‘*’, ‘/’	Esquerda para direita
Operadores aritméticos	‘+’, ‘-’	Esquerda para direita
Operadores relacionais	‘<’, ‘<=’, ‘>’, ‘>=’	Não associativo
Operadores relacionais	‘==’, ‘!=’	Não associativo
Conjunção	‘and’	Esquerda para direita
Disjunção	‘or’	Esquerda para direita

4 - Instruções

4.1 - Atribuição

A instrução de atribuição é definida pelo símbolo ’ = ’, sendo o lado esquerdo o identificador a receber o valor e o lado direito o valor ou expressão a ser atribuído. Ambos devem possuir o mesmo tipo.

4.2 - Estruturas condicionais

Cada estrutura condicional deverá possuir obrigatoriamente no início um if que é a condição inicial, seguido de uma expressão lógica entre ’{’ e ’}’. Uma expressão lógica é aquela cujo resultado da avaliação é um valor do tipo bool. Caso exista mais de uma condição onde a expressão lógica atenda os requisitos, o fluxo do programa deverá entrar na primeira estrutura que atender o requisito. Ao finalizar o bloco de código daquele condicional, o fluxo do programa sai daquela estrutura condicional.

Há também a estrutura elseif que deverá estar entre if e else e também é precedida pela expressão lógica entre ’{’ e ’}’, podendo haver zero ou mais elsif entre eles.

Por fim, a última estrutura é o else que abrange todas as condições não abordadas pelas estruturas anteriores, podendo haver 0 ou 1 else.

A estrutura deve possuir colchetes para a abertura e fechamento do seu escopo.

```
if{expressão_logica} [
  ...
]
elseif{expressão_logica} [
  ...
]
else [
  ...
]
```

4.3 - Estruturas de repetição

4.3.1 - While

Definida pela palavra reservada **while**, seguida pela expressão lógica entre ‘{‘ e ‘}’. Enquanto essa expressão for ‘true’, a estrutura irá iterar. A estrutura deve possuir colchetes para a abertura e fechamento do seu escopo.

```
while{expressão_lógica} [
  ...
]
```

4.3.2 - For

Definida pela palavra reservada for. Essa é uma estrutura dividida em 3 partes:

1 - Parte de inicialização de variáveis a serem utilizadas no laço.

2 - Condição de parada, que quando verdadeira, faz com que a execução do laço termine.

3 - Incrementos para serem aplicados em variáveis após uma iteração do laço.

Vale ressaltar que as variáveis a serem utilizadas devem ser declarada antes do loop.

```
itg i;
for {i = 0; i < 10; i++} [
  ...
]
```

4.4 - Entrada e saída

4.4.1 - Entrada

A instrução de entrada, definida pela palavra reservada read, poderá receber um parâmetro e não é necessário especificar o seu tipo.

```
itg i;
read{i};
```

4.4.2 - Saída

A instrução de saída, definida pela palavra reservada write, deverá estar entre aspas duplas e poderá receber um parâmetro.

```
itg i = 10;
write{i};
```

4.5 - Funções

As funções podem ser declaradas em qualquer lugar do código (menos dentro de outras funções) e seu escopo é delimitado pelas chaves ’[’ ’]’ que são obrigatórias.
Cada função deve ser iniciada pela palavra reservada ’function’, seguido pelo seu tipo por parâmetros. Ela pode ter ou não parâmetros e caso tenha ela deve especificar entre ’{’ ’}’ cada parâmetro e seu tipo, além de separá-los por ’,’ .

```
function itg func{itg param1, bool param2} [
  ...
  return param1;
]
```

5 - Algoritmos exemplos

5.1 - Hello World

```
function void main {} [
  str msg = "hello world";
  write{msg};
]
```

5.2 - Série de Fibonacci

```
function void main {} [
    itg n;
    read{n};
    itg a = 0;
    itg b = 1;
    if {n >= 1} [
        write{a};
    ]
    if {n >= 2} [
        write{b};
    ]
    while {n >= 3} [
        itg x = a + b;
        a = b;
        b = x;
        write{b};
        n--;
    ]
]
```

5.3 - Shell Sort

```
function void shellsort {itg n, list (itg) l} [
    itg h = 1;
    while {h < n} [
      h = h * 3 + 1;
    ]
    h = h / 3;
    itg temp, j;
    while {h > 0} [
      for {i = h; i < n; i++} [
          temp = l(i);
            j = i;
            while {j >= h and l(j - h) > temp} [
              l(j) = l(j - h);
                j = j - h;
            ]
            l(j) = temp ;
        ]
        h = h / 2 ;
    ]
]
function void main {} [
    itg i;
    itg n;
    read{n};
    list (itg) l;
    for {i = 0; i < n; i++} [
        itg x;
        read{x};
        l.append{x};
    ]
    shellsort{n, l};
    for {i = 0; i < n; i++} [
        write{l(i)};
        write{" "};
    ]
]
```