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

MOPA LANGUAGE 2.0
ESPECIFICAÇÕES DA LINGUAGEM

Maceió, 2022

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

MOPA LANGUAGE 2.0
ESPECIFICAÇÕES DA LINGUAGEM

Trabalho solicitado pelo **Prof. Me**. **Alcino Dall'igna Júnior**, para composição da primeira parte da nota da disciplina de **Compiladores**, do semestre 2021.1. **Alunos:** José Eraldo dos Santos Neto, Jonas Santos de Almeida Alves e Rick Martim Lino dos Santos**,** do curso de Ciência da Computação.

Maceió, 2022

Sumário

Introdução

A linguagem MOPA LANGUAGE 2.0 foi desenvolvida baseando-se nas linguagens mais comuns, que foram apresentadas na graduação, tais como C e Python. O leitor perceberá no decorrer do documento que a linguagem faz-se de uma sintaxe bastante simples, com comandos que facilitam a memorização e, consequentemente, torna mais fácil o desenvolvimento de programas de computador.

De um modo geral, a MOPA LANGUAGE 2.0 utiliza o paradigma de programação estruturada, visando a melhor clareza, qualidade e a diminuição no tempo de desenvolvimento dos programas. Além disso, possui palavras reservadas, que foram selecionadas cuidadosamente para serem mais compreensíveis enquanto o programador utiliza a linguagem. Vale ressaltar que há a diferença entre letras maiúsculas e minúsculas (*case-sensitive*) e que não há a coerção, ou seja, a conversão automática de tipo gerada pelo compilador.

Espera-se que ao final da leitura dessa especificação, o leitor seja capaz de desenvolver alguns programas simples e consistentes, visto que a MOPA LANGUAGE 2.0 é de fácil entendimento. Aprecie a leitura e tenha um ótimo aprendizado!

Estrutura geral do Programa

Início de Execução:

O início de execução de um programa se dá por meio da declaração da função principal, utilizando a função main() e com o retorno int.

Ex.:

```
fun int main()
{
      .
      .
      .
      return 0;
}
```

Definições de funções e procedimentos:

As funções definidas pelo usuário poderão existir se forem declaradas fora do corpo de outra função e podem ser acessadas caso o programador já as tenha definido previamente, por meio da assinatura ou implementação. Além disso, a MOPA LANGUAGE 2.0 não aceita a passagem de subprogramas como parâmetro de funções.

A declaração de uma função inicia-se pela palavra reservada **fun** seguido do tipo de retorno, que pode ser de um único tipo: **int, float, bool, string** ou **char;** e o(s) parâmetro(s) acompanhados do seu identificador de tipo, separados por vírgula e entre parênteses. Além disso, os delimitadores do bloco da função são representados por chaves.

A declaração de uma função seguirá o mesmo padrão que foi definido acima, porém, para as funções que não há nenhum tipo de retorno, inicia-se utilizando a palavra reservada **proc.**

O exemplo a seguir evidenciam como dar-se a declaração de funções.

Ex.:

```
fun tipoDeRetorno id (listaDeParametros){
    .
    .
    .
    return x;
}

proc id (listaDeParametros){
     .
     .
     .
}
```

Definição de Instruções:

As instruções somente são declaradas dentro do bloco de funções ou de procedimentos, sendo encerrada com ';' sendo esse bloco delimitado por chaves "{}".

Conjunto de tipo de dados e nomes

Identificador:

Os identificadores possuem sensibilidade à caixa, quer dizer, entre letras maiúsculas e minúsculas, com o limite máximo de 31 caracteres e o seu formato será definido a partir da expressão regular: **("letter")("digit"|"letter)*.**

Exemplo de nome aceito pela linguagem: `storeData, data1;`
Exemplo de nome não aceito pela linguagem: `.departureCode, ab@cd, 45jfk, _737800, _.`

Keywords:

As palavras reservadas para a linguagem MOPA LANGUAGE 2.0 são as seguintes: **const, char, int, float, string, bool, proc, read, print, if, else, false, true, for, while, return, null.**

Comentários:

Os comentários são feitos utilizando "//". Assim, quando utilizamos o comentário em uma linha de código, tudo o que está após "//" será descartado. É possível comentar apenas por linha.

Ex.:

```
fun int main()
     {
       //string recifeTower = "Clear for landing on Runway
     18";
       //string pilotToRecifeTower = "Recife Tower Cleared for
     landing on Runway 18";

          return 0;
     }
```

Definição de variáveis:

Há duas formas de declarar uma variável:

1. Pode ser definida fora do escopo de uma função, que estará disponível globalmente para uso em todo o programa a partir do ponto de declaração;
2. Pode ser definida dentro do escopo de uma função como variável local, visível a partir do ponto da declaração.

As declarações são feitas utilizando o tipo da variável, que podem ser **int, float, bool, string e char;** seguidas de um identificador. A MOPA LANGUAGE 2.0 permite múltiplas declarações de variáveis do mesmo tipo, sendo separadas por vírgulas e encerradas por ponto e vírgula. Também, permite a declaração de uma variável seguida por atribuição de valor, por meio do operador de atribuição "=".

Ex.:

```
bool takeOffClearance = true;
int fuelEngine_left;
```

```
float towerFrequency_SBMO = 113.10;
char flight_ID = 'G31230';
string selectApproach = "ILS approach on Runway 18";
```

Definição de constantes:

A definição de uma constante segue o mesmo padrão da declaração de uma variável, exceto que inicia-se com a palavra reservada **const**, seguido da inicialização de valor na declaração.

Ex.:

```
const float ILS_frequency = 113.30;
```

Tipos de Dados

Todos os tipos e estruturas possuirão compatibilidade por nome.

Booleano:

É a identificação da variável como booleano, utilizado com a palavra reservada **bool**, que pode ser atribuído o valor de **true** ou **false**.

Ex.: `bool maceioAirportInOperation = true;`

Inteiro:

É a identificação da variável como inteiro de 32 bits, utilizado com a palavra reservada **int.** Seus literais são expressos como uma sequência de dígitos decimais.

Ex.: `int maceioTakeOffRunway = 2602;`

Ponto Flutuante:

É a identificação da variável como flutuante de 64 bits, utilizado com a palavra reservada **float.** Seus literais são expressos como uma sequência de dígitos decimais, seguido de um ponto e demais dígitos, que representam a parte fracionária.

Ex.: `float maceioAirportelevation = 118.01;`

Caractere:

É a identificação da variável como caractere de 8 bits, utilizado com a palavra reservada **`char.`** A constante literal do caractere é delimitada por apóstrofos.

Ex.: `char maceioAirportICAO = 'SBMO';`

Cadeia de caracteres:

É a identificação da variável como uma cadeia de caracteres, utilizado com a palavra reservada **`string.`** Seus literais são expressos como um conjunto mínimo de 0 caracteres e de tamanho máximo ilimitado. A constante literal é delimitada por aspas.

Ex.: `string maceioAirportId = "Aeroporto Internacional Zumbi dos Palmares";`

Arranjos Unidimensionais:

São compostos pelos tipos que foram determinados acima. A declaração se dá iniciando pelo tipo seguido de um identificador único, e delimitado por colchetes, que representará o seu tamanho. Caso aconteça um arranjo menor que o tamanho declarado, os espaços serão ocupados com o valor **`null`** e, quando passar dos limites definidos, é indicado erro.

Sua indexação ocorre mediante deslocamento, ou seja, a posição inicial é zero e a última equivale a tamanho -1, como ocorre na linguagem C. Caso tente ser atribuído um valor a uma posição inexistente do array, será acusado erro.

Ex.:

```
fun int main(){

      int departuresGatesInSBMO[9];
      float aircraftsTotalWeights[25];
      bool  aircraftsReadytoTakeOff[25];
      char departuresSBMO[35];
      string flightCompaniesInSBMO[100];
      .
      .
      .

      return 0;
```

```
}
```

Coerção:

A linguagem não aceita coerção entre variáveis de tipo diferentes. Todas as verificações de compatibilidade de tipo serão feitas de forma estática. Com exceção com a verificação de limites de array, que, caso seja definido por constante, também será feito estaticamente, mas, caso seja definido por variável, será feito dinamicamente.

Concatenação de String:

Caso o programador desejar realizar a concatenação entre duas strings, é necessário fazer o uso do operador "#".

Ex.:

```
fun int main(){

      string controlTower = "Flight G31230 ";
      string controlTowerOrder = "ready for take off.";
      string pilotFrequency;

      pilotFrequency = controlTower#controlTowerOrder;

      print(pilotFrequency);

      return 0;
}
```

O resultado será: `Flight G31230 ready for take off.`

Valores padrão dos tipos:

Tipo	Valor de Inicialização
int	0
float	0.0
char	''(caractere vazio)
string	""(string vazia)
bool	false

Constantes literais e suas expressões regulares:

As expressões regulares das constantes literais são definidas da seguinte forma:

1. Constante de inteiro: `((‘digit’)+)`
2. Constante de float: `((‘digit’)+) (‘\.’) ((‘digit’)+)`
3. Constante de char: `(‘\’’) (‘letter’|‘symbol’|‘digit’) (‘\’’)`
4. Constante de bool: `(‘true’|‘false’)`
5. Constantes de String: `(‘\”’) ((‘letter’|‘symbol’|‘digit’)*) (‘\” ’)`

Conjunto de Operadores

Aritiméticos:

Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Resto	%
Unitário Positivo	+
Unitário Negativo	-

Lógicos:

Negação Unitária	!
Conjunção	&
Disjunção	|

Atribuição:

Operador de atribuição	=

Operações suportadas:

Operador	Tipos que realizam a operação
!	bool
^, *, /, +, -	int, float
<, <=, >, >=	int, float, char, string
==, !=, =	int, float, bool, char, string
&, |	bool
#	string

Precedência e Associatividade

Operadores	Associatividade
'!'(not)	Direita para esquerda
'-'(unário negativo), '+' (unário positivo)	Direita para esquerda
'*'(multiplicação), '/'(divisão), '%'(resto)	Esquerda para direita
'+'(adição); '-'(subtração)	Esquerda para direita
'<', '>', '<=', '>='	Não associativo
'==', '!='	Não associativo
'&'	Esquerda para direita
'|'	Esquerda para direita
'#'(concatenação)	Esquerda para direita
'='(atribuição)	Direita para esquerda

Instruções

Entrada e Saída

A **entrada** de dados é feita por meio da função **`read()`**: A função read (identificador) atribui o valor lido da variável do identificador. É possível colocar mais de um parâmetro como entrada para a função read, separando os identificadores por vírgula.

Já a **saída** de dados é realizada pela função **`print`**(): A função print pode imprimir o valor na tela, como variáveis e string. É permitido a impressão de strings utilizando a cadeia a ser impressa entre a abertura e fechamento de aspas.
A saída poderá ser formatada, os identificadores dos tipos de formatação são: **`@d` inteiro, `@c` caractere, `@f` float, `@s` string, `@b` bool**. Caso a saída se encontre formatada, no código deverá ser especificado quais são as variáveis de cada um dos identificadores de formatação;
A impressão de cada tipo de dado ocupa um determinado espaço na
tela, tal que:

- Um caracter ocupa uma posição;
- Um booleano ocupa quatro ou cinco posições, a depender de seu valor (true ou false);
- Um inteiro pode representar a faixa de números entre -2.147.483.648 a 2.147.483.647, logo sua impressão pode ocupar até 11 posições;
- Um float pode representar a faixa de números entre 10^{-308} a 10^{308} .

Para imprimir com uma quebra de linha no final, utiliza-se a função **`println()`.**

Atribuição:

A atribuição é feita por meio do símbolo `'='`, no qual o lado esquerdo recebe o identificador e o lado direito é o valor ou a expressão a ser atribuída. Dessa forma, é necessário que os operadores e operandos sejam do mesmo tipo.

Ex.

```
fun int autoPilot(bool CMD_A, bool ALT_HDL, bool HDG_SEL, int
HDG, int HDG_SEL_SET…){
        .
        .
        .
        int heading_direction = 035;
        int altitude_in_feets = 9000;
        int vertical_speed = -2500;
        .
```

```
        .
        .
    }
```

Estrutura condicional:

A estrutura "**if**" relaciona-se com a expressão lógica ou variável booleana entre parênteses, seguido de um bloco de instruções definido entre chaves. Caso a instrução entre parênteses seja avaliada em **true**, as instruções que estão entre as chaves serão executadas; Caso a instrução seja avaliada em **false**, o fluxo de execução do programa não entrará no no bloco de instruções.

É possível usar o comando "**else**" , caso a instrução "**if**" não seja verdadeira, possibilitando a execução de um bloco de instruções definido entre chaves. Desse modo, verifica-se que o comando "**else**" só poderá ser inserido, caso seja definido um "**if**" antes dele; além disso, o uso deste comando é opcional. Também, é possível utilizar essa estrutura condicional no bloco de instruções de outra estrutura condicional, ou seja, dentro de um bloco "**if**" pode-se colocar outra sequência de "**if**".

Ex.

```
fun int autoPilot(bool CMD_A, bool ALT_HDL, bool HDG_SEL, int
HDG,int HDG_SEL_SET…){
        .
        .
        .
        if (CMD_A == true){
            if(ALT_HLD == false){
                ALT_HLD == true;
                return altitude_in_feets;
            }
            else{
                return altitude_in_feets;
            }
    }
```

Estrutura interativa com controle lógico:

A estrutura do "**while**" deve possuir uma expressão lógica entre parênteses e conter um bloco de instruções entre chaves. O loop continuará sua execução enquanto a

condição for verdadeira. No momento o qual a condição for avaliada como falsa, o loop encerrará e o fluxo do programa seguirá adiante.

Ex.

```
fun int autoPilot(bool CMD_A, bool ALT_HDL, bool HDG_SEL, int
HDG,int HDG_SEL_SET…){
          .
          .
          .
          while(HDG_SEL_SET < heading_direction){
               HEG_SEL_SET = HEG_SEL_SET + 1;
          }
          .
          .
          .
     }
```

Estrutura interativa controlada por contador:

A estrutura do comando **"for"** deve possuir uma variável de controle inteira, declarada entre parênteses, seguido por um bloco de instruções delimitado por chaves. Dentro do **"for"** o valor da variável de controle não pode ser alterado. Já no loop, os três valores **(x,y,z)** devem ser inteiros. O valor de **x** será o valor inicial do contador, **y** será o valor final e **z** será o valor de incremento, que realiza o passo a cada final de ciclo. Os três valores são pré-avaliados. Além disso, o valor final deve ser sempre maior do que o inicial para que o for seja executado.

Ex.

```
fun int aircraftWindowsSeats(bool windows_seat[]){

          int free_seats = 0;

          for(i : 0, 58, 1){
               if(windowsSeat[i]== True){
                    free_seats = free_seats + 1;
               }
          }

          return free_seats;
}
```

Desvios Incondicionais

Não foi implementado nenhum tipo de desvio incondicional na MOPA LANGUAGE 2.0.

Exemplo de Programas

Hello World:

```
fun int main () {
     print("Alo Mundo!");
     return 0;
}
```

Série de Fibonacci:

```
     proc fib(int n) {
          int n1 = 0, n2 = 1, n3;
          if (n == 0) {
               println("@d", n);
          }
          if (n == 1) {
               println("0, @d", n);
          } else {
          string separator = ",";
          print("0, 1, ");
          while (true){
               n3 = n1 + n2;
               print("@s@d", separator, n3);
               if (n3 >= n) {
                    return;
               }
               n1 = n2;
               n2 = n3;
          }
     }
}
fun int main() {
int n;
read(n);
fib(n);
return 0;

}
```

Shell Sort:

```
proc shellSort (int array [ ] , int n) {
      int h = 1, c , j;

      while( h < n ) {
            h = h * 3 + 1;
      }
      h = h / 3;
      while ( h > 0 ){
      for (int i : h , 1 , n){
            c = array(i);
            j = i;

            while ( j >= h & array [ j - h ] > c ) {
                  array [ j ] = array [ j - h ];
                  j = j - h ;
            }

                  array [ j ] = c;
            }
            h = h / 2;
      }
}

fun int main() {
      int n;
      println("Tamanho do array");
      read(n);
      int array [ n ];
      println("Digite os numeros que serao ordenados");
      for( int i : 0,1,n) {
            read( array [ i ] );
}

      println("lista dos valores digitados");
      for (int i : 0, 1, n){
            int a = array [ i ];
            println (a);
}

shellSort (array,n);

println("valores ordenados");
for(int i : 0 , 1 , n) {
      int b = array [ i ];
      println(b);
```

```
}
return 0;
}
```