UNIVERSIDADE FEDERAL
DE SANTA CATARINA
Campus de Araranguá

Construção de Compiladores

Manual da Linguagem Blocky

Matheus André Soares

Patrick Davila Kochan

Victor Camargo de Lima

Araranguá,

01 de maio de 2019

1. ESTRUTURA MÍNIMA DO PROGRAMA

Os programas escritos na linguagem Blocky possuem a declaração do bloco de maneira obrigatória para a construção de código. Esse bloco é envolto por chaves e precedido por *void main*, definindo a função principal, onde toda a lógica do programa estará centralizada.

Em seguida são realizadas as declarações de variáveis e de funções e então o corpo do programa é iniciado. Dessas três partes que compõe o bloco, apenas o corpo do programa é obrigatório. O corpo do programa é iniciado com a palavra reservada "inicio" e encerrado com a palavra reservada "fim".

2. DEFINIÇÕES

Comandos de entrada e saída

Cin: recebe uma informação do usuário e armazena em uma variável previamente definida.

Cout: utilizado para a impressão de dados ou informações para o usuário. Esse comando também permite exibir uma literal (cadeia de caracteres delimitados por @, ex.: @teste@), que pode ter até 32 caracteres.

Os comandos de entrada e saída não devem terminar com ponto e vírgula.

```
2   cin >> num
3   cout << @numero:@ << num
```

Figura 01: exemplo de comandos de entrada e saída.

Fluxos de controle

while: a estrutura de repetição *while* executa a repetição de um bloco de instruções enquanto uma condição imposta for verdadeira.

```
2  while(condição){
3      ...
4  }
```

Figura 02: exemplo do comando *while*.

for: executa a repetição de um bloco de instruções enquanto uma condição imposta for verdadeira. Para tal, deve-se utilizar uma variável de controle, a qual está atrelada um incremento que será executado a cada iteração do bloco, caso a condição seja verdadeira.

```
6  for(variavel; condição; incremento){
7      ...
8  }
```

Figura 03: exemplo do comando *for*.

do-while: a maneira como o *do* funciona é muito semelhante a estrutura *while*, porém a condição é testada sempre após executar o bloco de instruções.

```
10 do{
11     ...
12 }while(condição)
13
```

Figura 04: exemplo do comando *do-while*.

Estruturas de Decisão Simples

if: uma estrutura de decisão examina uma condição e decide quais instruções serão executadas de acordo com a condição atendida.

```
16 if(condição){
17     ..
18 }
19
```

Figura 05: exemplo do comando *if*.

else: utilizado em conjunto ao *if*, pois no momento em que a condição do *if* não for atendida pode existir outra instrução para ser executada.

É importante salientar que um *if* não necessariamente precisa ser seguido por um *else*.

```
21 if(condição){
22     ..
23 }
24 else{
25     ...
26 }
27
```

Figura 06: exemplo do comando *if-else*.

Operadores

Operadores aritméticos

\+ : Soma duas variáveis ou numerais;

\- : Subtrai duas variáveis ou numerais;

/ : Divide duas variáveis ou numerais;

* : Multiplica duas variáveis ou numerais.

Operadores unários

++ : Incrementa uma variável inteira;

-- : Decrementa uma variável inteira.

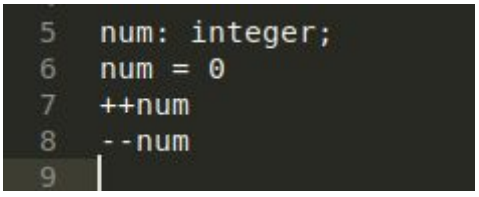


Figura 07: exemplo de declaração de uma variável *num,* com incremento e decremento.

Operadores de atribuição

= : Atribui algum conteúdo a uma variável;

>> : Sequência da estrutura de entrada cin;

<< : Sequência da estrutura de entrada cout.

Operadores de comparação

As comparações são realizadas no sentido de leitura esquerda para a direita.

>= : Comparação entre dois numerais que verifica se um é maior ou igual ao outro;

> : Comparação entre dois numerais que verifica qual é o maior número;

== : Comparação entre dois numerais que verifica se eles são exatamente iguais;

<= : Comparação entre dois numerais que verifica se um é menor ou igual ao outro;

< : Comparação entre dois numerais que verifica qual é menor número;

!= : Comparação entre duas variáveis que verifica se elas são diferentes.

Comentários

Comentários em programação são utilizados para realizar a devida documentação do software em questão, facilitando o entendimento do código. As partes do código que fazem parte do comentário são ignoradas no processo de compilação.

Para comentar pequenas informações no código utilizando a linguagem Blocky, deve-se utilizar o caractere %, sendo considerado um comentário em linha. Assim, toda linha que inicia com esse símbolo será destinada para o comentário em si.

Se o comentário possuir mais de uma linha, deve-se iniciar o mesmo com %* e a indicação do término é feita com *%. Assim, toda informação entre %* e *% será considerada comentário em bloco dentro do código e será ignorada no processo de compilação.

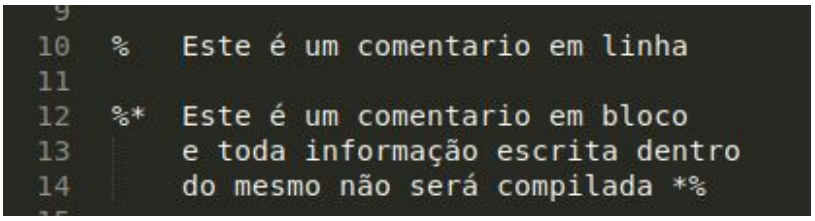
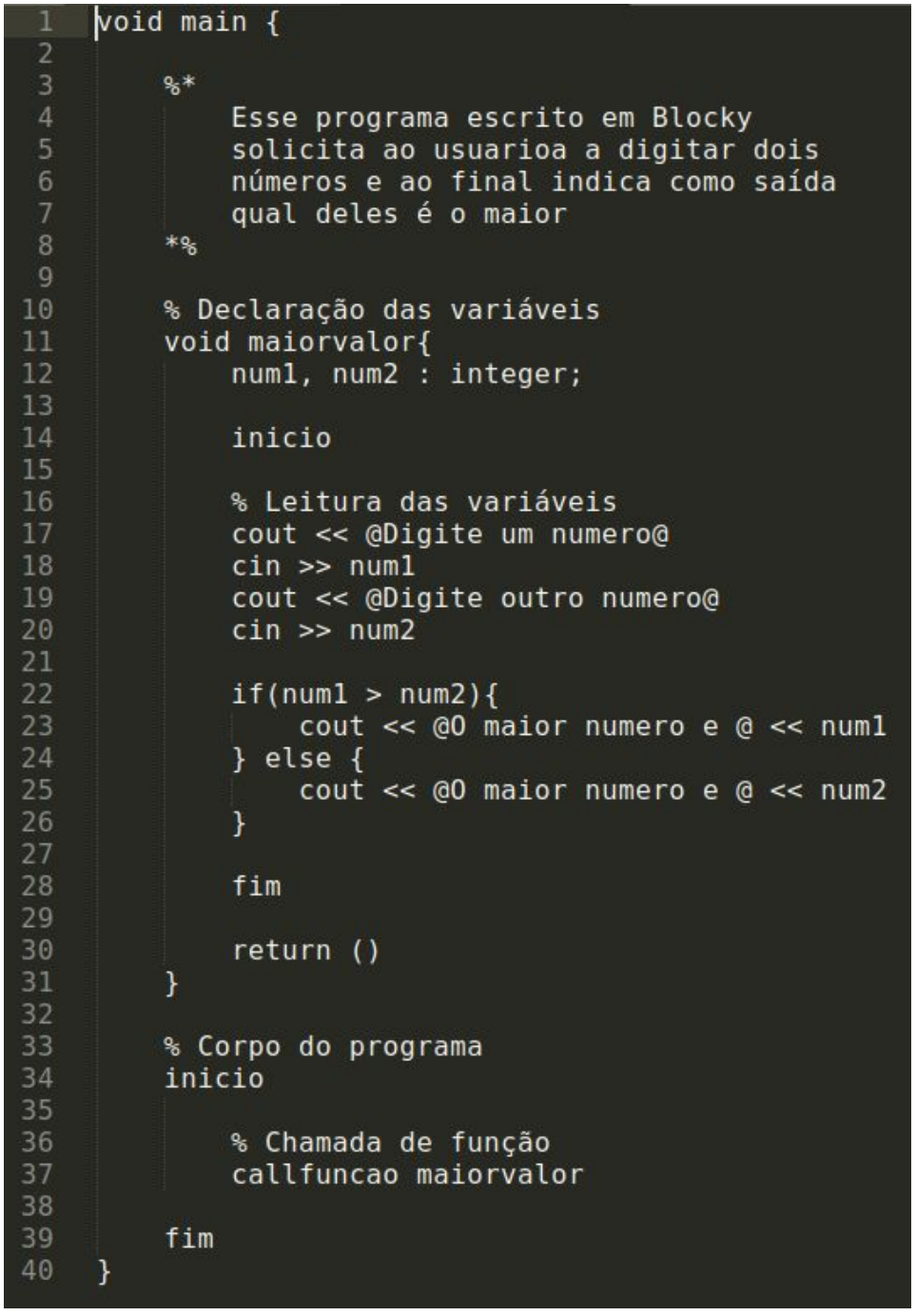


Figura 8: exemplo de comentários em linha e em bloco.

Funções

Funções são utilizadas em programação para a reutilização de código. Também conhecidas como sub-rotinas, as funções podem ou não apresentar parâmetros que são utilizados por elas buscando alcançar um resultado que depende do objetivo da própria função. Em Blocky, a utilização de parâmetros é indicada através do uso de parênteses, sendo que "(" indica o início e ")" indica o fim dos elementos que serão tratados como parâmetros. Parâmetros diferentes são separados por vírgula "," na chamada de função e por ";" na declaração.

Para realizar a definição de uma função e sua respectiva chamada/invocação, utiliza-se a seguinte sintaxe:



```
void main {

    %*
        Esse programa escrito em Blocky
        solicita ao usuarioa a digitar dois
        números e ao final indica como saída
        qual deles é o maior
    *%

    % Declaração das variáveis
    void maiorvalor{
        num1, num2 : integer;

        inicio

        % Leitura das variáveis
        cout << @Digite um numero@
        cin >> num1
        cout << @Digite outro numero@
        cin >> num2

        if(num1 > num2){
            cout << @O maior numero e @ << num1
        } else {
            cout << @O maior numero e @ << num2
        }

        fim

        return ()
    }

    % Corpo do programa
    inicio

        % Chamada de função
        callfuncao maiorvalor

    fim
}
```

Figura 09: exemplo de declaração e chamada de função.

3. REGRAS LÉXICAS

Identificadores

Um identificador pode ser considerado um nome para uma variável ou para uma função. Ela pode possuir no máximo 2^4 (32) caracteres, podendo ser letras e números. Obrigatoriamente deve iniciar com uma letra (excluindo ç). Os identificadores não devem possuir acentos.

Exemplo de identificadores válidos: *a123, salario, num1, a1bc,* etc.

Exemplo de identificadores inválidos: *123a, _salario, salário, maçã,* etc.

Variáveis

Variáveis são elementos que armazenam dados e podem ser acessadas através de identificadores previamente designados a elas. As variáveis podem ser de um dos quatros tipos disponíveis: *integer, float, char* e *string.*

Uma variável do tipo *integer* armazena um número pertencente ao grupo dos números inteiros, podendo ser positivo ou negativo. Pode possuir até 2^4 (32) caracteres. Exemplo: 1, -23, 1000.

Por outro lado, uma variável do tipo *float* representa um número decimal. Pode possuir até 2^4 (32) caracteres. A separação decimal é realizada através da utilização do caractere "." (ponto). Exemplo: 3.14, 12.345.

O tipo *char* indica que a variável contém apenas um caractere alfanumérico, escrito entre apóstrofos. Todos os caracteres compreendidos pela tabela ASCII são aceitos. Exemplo: 'k', 'l', 's'.

Uma variável *string* contém uma cadeia de caracteres, sendo delimitados por aspas. Essa variável pode possuir até 2^4 (32) caracteres. Todos os caracteres compreendidos pela tabela ASCII são aceitos. Exemplo: "melhor", "linguagem".

Durante a execução do programa, as variáveis podem alterar o dado que armazenam, mas estes devem sempre possuir o mesmo tipo para determinada variável. Assim, uma variável do tipo *integer* não poderá armazenar valores do tipo *char.*

Uma variável deverá ser declarada através de seu nome sucedida pelo tipo, podendo declarar mais de uma variável do mesmo tipo ao mesmo tempo. Assim, a sintaxe a ser

seguida é a seguinte, onde *nome, nome1* e *nome2* são nomes para as variáveis e tipo deverá ser um dos quatro já citados:

```
1  nome: tipo;
2  nome1, nome2: tipo;
3
```

Figura 10: exemplo da sintaxe de declaração de variáveis.

Por exemplo, caso desejamos declarar duas variáveis inteiras com identificadores *salario* e *dias,* pode-se escolher uma das seguintes formas:

```
1  salario: integer;
2  dias: integer;
```

```
5
6  salario, dias: integer;
7
```

Figura 11: duas maneiras de declaração de duas variáveis *integer*.

4. ERROS LÉXICOS

A linguagem Blocky considera como erros léxicos:

- A utilização de símbolos não existentes na linguagem;
- A atribuição de valores superiores ao limite estipulado: literais, identificadores, inteiros, floats e strings com mais de 32 caracteres.

5. EXEMPLO DE PROGRAMA ESCRITO EM BLOCKY

```
void main {
    %*
        Este programa requisita duas variaveis
        e imprime na tela todos os valores entre elas.
        A primeira variavel deve ser menor que a segunda.
        Caso contrario, uma mensagem de erro é impressa
    *%

    % Declaração das variáveis
    void iteracao{
        num1, num2 : integer;

        inicio

        % Leitura das variáveis
        cout << @Digite o menor numero@
        cin >> num1
        cout << @Digite o numero superior@
        cin >> num2

        if(num1 > num2){
            for(num1 = num1; num1 <= num2; num1++){
                cout << num1
            }
        }
        else{
            cout << @Numeros incorretos@
        }

        fim

        return ()
    }

    % Corpo do programa
    inicio

        % Chamada de função
        callfuncao iteracao

    fim
}
```

Figura 12: Exemplo de utilização da linguagem.