UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO

PEDRO JAVIER PANECA CORDOVA

Compiladores

Especificação da Linguagem

MACEIÓ/AL
2021.1

1. Introdução **3**

2. Estrutura Geral do Programa **3**
2.1 Ponto inicial do execução 3
2.2 Definições de funções 4

3. Conjunto de tipos de dados e nomes **4**
3.1 Identificadores 4
3.2 Palavras reservadas 5
3.3 Coerção 5
3.5 Inteiro 6
3.6 Ponto Flutuante 6
3.7 Caracteres e cadeias de caracteres 6
3.8 Arranjos Unidimensionais 7

4. Conjunto de operadores **8**
4.1 Lógicos 8
4.2 Relacionais 8
4.3 Aritméticos 8

5. Instruções **10**
5.1 Atribuição 10
5.3 Estrutura de repetição por controle lógico 11
5.4 Estrutura de repetição por controle de contador 11
5.5 Entrada e saída 12
5.6 Funções 12

6. Programas de exemplos **13**
6.1 Alô mundo 13
6.2 Série de Fibonacci 14
6.3 Shell Sort 15

1. Introdução

A linguagem PJ foi criada baseada nas linguagens C, Ruby e Python. É tipada estaticamente e possui estruturas condicionais e de repetição. Possui palavras reservadas, suporte a coerção, é case-sensitive e não é orientada a objetos. O seu propósito é para fins didáticos e criação de algoritmos simples.

2. Estrutura Geral do Programa

Um programa feito na linguagem PJ deve possuir:

- Uma função main com tipo e retorno obrigatório e igual a 0 para o compilador entender que o programa finalizou a sua execução com sucesso
- Os blocos e escopos são delimitados por abertura e fechamento de chaves
- As expressões devem terminar com ponto e vírgula ‘;’
- As funções possuem retorno e seu escopo é delimitado por chaves de abertura e fechamento

2.1 Ponto inicial do execução

O ponto inicial do programa é identificado através da criação de uma função principal chamada main que possui como tipo de retorno obrigatório sendo inteiro e retorno igual a 0 (zero) através da palavra reservada return.

O escopo de uma função é determinado por chaves de abertura e de fechamento e o uso de parênteses para abertura e fechamento de parâmetros de função. No final de cada linha deve possuir o caractere “;” . Exemplo:

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

2.2 Definições de funções

Para criação de novas funções, deve-se utilizar a palavra reservada **fn** sendo que antes dela deve possuir o tipo de retorno da função podendo ser int, float, bool (ou um array de qualquer tipo citado anteriormente), string, char, void, sendo void utilizado para quando não houver nenhum tipo de retorno, em seguida o nome da função que é um identificador único seguido de uma lista de parâmetros delimitados por parênteses.

A lista de parâmetros deve ser composta pelo tipo seguido do identificador. Cada parâmetro é separado por vírgulas (“,”), e por fim, para definir o escopo dessa função utiliza-se abertura e fechamento de chaves. Exemplos:

● Exemplo:

```
<tipo de retorno> function <identificador da função>(<tipo do
parametro 1><identificador 1>, ..., <tipo do parametro
I><identificador i>) {}

void fn get_user_data(string name, int amount, string books[amount]) {
      ...
}
```

3. Conjunto de tipos de dados e nomes

3.1 Identificadores

Os identificadores possuem algumas regras. De modo geral são case-sensitive e não devem possuir certos caracteres especiais:

- Podem começar somente com letras maiúsculas ou minúsculas;
- Não podem começar com números ou qualquer caractere especial;
- O único caractere especial permitido é o underline (“_”) ;
- Não podem ser palavras reservadas;
- Não é permitido o uso de espaço;

3.2 Palavras reservadas

As palavras reservadas são:

main, if, elsif, else, for, while, fn, int, float, bool, string, char, puts, gets, False, True.

3.3 Coerção

A linguagem não aceita coerção implícita, qualquer operação entre tipos diferentes deverá ser feita mediante o uso de **Cast** explícito.

Cast: Deve seguir a seguinte forma:

<tipo da variável que iremos fazer o cast>(identificador da variável)

Exemplo:

int number_int = 1;

float number = float(number_int)

Exemplo:

float num_1 = 2;

string numbers = ‘1’ . string(num_1)

Cast:

Ao fazer o Cast de um int ou ponto flutuante para booleano, se o número for maior que zero será considerado como True, caso contrário como False.

Ao fazer o Cast de um char ou string pegamos a soma dos seus caracteres de acordo com a tabela ASCII, se a soma dos mesmos for maior que zero será considerado como True, caso contrário como False.

3.4 Booleanos

bool identifica os booleanos e possuem somente dois resultados possíveis, sendo eles através das palavras reservadas True e False. Caso essa variável não seja inicializada com algum valor, seu valor padrão será False. A seguir é possível ver um exemplo de declaração de variável com inicialização e um exemplo só de declaração.

Exemplos:

bool teste = True;

bool valor_booleano;

Valor padrão: False

3.5 Inteiro

int identifica a variável como um número inteiro de 32 bits, sendo seu valor literal uma cadeia de números inteiros entre 0 e 9.

Exemplo:

int inteiro = 10;

Valor padrão: 0

3.6 Ponto Flutuante

float identifica variáveis como um número com ponto flutuante de 64 bits, sendo seu valor literal uma sequência de dígitos entre 0 e 9 seguido de um

ponto (“.”) e demais dígitos (com precisão de 8 dígitos). Ao estourar esse limite é desconsiderado a parte que passou do limite.

Exemplo:

float pontoFlutuante = 15.47;

Valor padrão: 0.0

3.7 Caracteres e cadeias de caracteres

char identifica os caracteres, podendo salvar números, letras maiúsculas e minúsculas ou caracteres especiais. Utiliza-se a palavra reservada char seguida do identificador. Seu valor é delimitado por apóstrofos. A seguir temos um exemplo de declaração com inicialização de valor seguido de um que não é inicializado e depois é atribuído um valor a essa variável.

Exemplo:

char caractere = ‘a’;

Nesse caso temos uma variável chamada “caractere” que ao ser declarada já foi atribuída o valor “a” a ela.

Se a variável não for inicializada com algum valor, seu valor padrão será vazio. É possível inicializá-la com o valor vazio também.

Exemplo:

char caractere;

caractere = ‘a’;

Exemplo: Inicializando com valor vazio char caractere = ‘’;

Já uma cadeia de caracteres aceita os mesmos valores que um char pode possuir, porém seu valor é delimitado por aspas e identificados através da palavra reservada string. Da mesma forma é possível inicializar a variável já com um valor ou com o valor vazio. Seu valor padrão ao não ser atribuído nenhum valor será vazio.

Exemplo:

string text = “Esta é uma sequência de caracteres.”

Valor padrão:

char: ‘’ (vazio)

string: “” (vazio)

3.8 Arranjos Unidimensionais

Um Arranjo Unidimensional ou Array é declarado como na C, porém seu funcionamento é como no Java, onde cada array sabe seu próprio tamanho, o mesmo não pode ser mudado durante a execução do programa. A propriedade do array para acessar seu tamanho seria array.length()

Definimos o seu formato como sendo:

<tipo> identificador [tamanho]

Exemplo:

int array1[];

4. Conjunto de operadores

4.1 Lógicos

- “!”: Operador “negação”, é um operador unário que retorna a negação de seu operando. Fica a esquerda do operando.
- “&&” Operador "conjunção" (E lógico), é um operador que irá retornar um valor booleano, caso ambos operandos sejam verdadeiros retorna True, caso contrário, retorna False.
- “||” Operador “disjunção” (OU lógico), é um operador que irá retornar um valor booleano, caso um dos operandos for verdadeiro será retornado True, e se ambos forem falsos, retorna False.

As operações citadas acima somente podem ocorrer entre tipos booleanos, em caso de números ou strings pode ser feito o uso do **Cast** explícito para fazer a sua transformação para booleano.

Exemplo:

```
float num_1 = 4.00

Int num_2 = 3.00

bool resultado =  bool(num_1) && bool(num_2)
```

4.2 Relacionais

- “**==**”: Operador que irá retornar um resultado verdadeiro ao verificar dois operandos, um a esquerda do operador e o outro a direita, caso sejam iguais.
- “!=”: Operador que irá retornar um resultado verdadeiro ao verificar dois operandos, um a esquerda do operador e o outro a direita, caso sejam diferentes.
- “<”: Operador “menor que” que irá retornar um booleano de valor True ao verificar dois operandos, um a esquerda do operador e o outro a direita, caso o da esquerda seja menor que o da direita, caso contrário retornará False.
- “>”: Operador “maior que” que irá retornar um booleano de valor True ao verificar dois operandos, um à esquerda do operador e o outro a direita, caso o da esquerda seja maior que o da direita, caso contrário retornará False.
- “<=”: Operador “menor ou igual que” que irá retornar um booleano de valor True ao verificar dois operandos, um a esquerda do operador e o outro a direita, caso o da esquerda seja menor ou igual que o da direita, caso contrário retornará False.

- “>=”: Operador “maior ou igual que” que irá retornar um booleano de valor True ao verificar dois operandos, um a esquerda do operador e o outro a direita, caso o da esquerda seja maior ou igual que o da direita, caso contrário retornará False.

As operações citadas acima somente podem ocorrer entre tipos inteiros, em caso de números ou strings pode ser feito o uso do **Cast** explícito para fazer a sua transformação para booleano.

Exemplo:

string num_1 = ‘ab’

char num_2 = ‘a’

bool resultado = int(num_1) > int(num_2)

4.3 Aritméticos

- “+” Soma de dois operandos
- “-” Subtração de dois operandos
- “-” Unário Negativo
- “/” Divisão de dois operandos
- “*” Multiplicação de dois operandos
- “%” Retorna um número inteiro restante da divisão dos dois operandos.

Operações de divisão por zero em ponto flutuante será retornado um NaN (Not a Number).

4.4 Concatenação de cadeias de caracteres

Para concatenar duas cadeias de caracteres utiliza-se “.”, a concatenação de dois valores string, retorna outra string que é a união dos dois operandos. Sendo sua associatividade da esquerda para a direita. Supondo que tenhamos duas variáveis, string1 e string2 com respectivamente os valores, “Testando ” e “concatenação.” que será atribuída a uma nova variável chamada string3, o resultado da concatenação dessas duas variáveis resultaria em:

string string1 = “Testando ”;

string string2 = “concatenação”;

string string3 = string1 . string2;

Valor de string3: “Testando concatenação”.

4.5 Precedência e associatividade

Quanto maior o valor, maior a precedência.

Operação	Precedência
“-” (unário negativo)	6
“*” “/” (multiplicação e divisão)	5
“+” “-” (soma e subtração)	4
“==” “!=” (igualdade e desigualdade)	2
“<” “<=” “>” “>=” (comparação)	2
“!” (negação)	2
“&&” “||” (operadores and e or)	2
“.” concatenação	3
= (atribuição)	1

Operação	Associatividade
“-” (unário negativo)	da direita para esquerda
= (atribuição)	da direita para esquerda
“*” “/” (multiplicação e divisão)	da esquerda para direita
“+” “-” (soma e subtração)	da esquerda para direita
“==” “!=” (igualdade e desigualdade)	da esquerda para direita
“<” “<=” “>” “>=” (comparação)	da esquerda para direita
“!” (negação)	da esquerda para direita
“&&” “||” (operadores and e or)	da esquerda para direita
“.” concatenação	da esquerda para direita

Parênteses terão precedência sobre todos os operadores. No caso de haver parênteses aninhados, o parênteses mais interno terá maior precedência, e assim consecutivamente.

O operador unário ! de negação tem associatividade à direita, ou seja, quer dizer que primeiro a expressão da direita será avaliada antes de fazer a negação. Da mesma forma, os operadores de atribuição vão resolver as expressões à direita antes de fazer a atribuição.

5. Instruções

Cada bloco de instruções na linguagem seu escopo são delimitados por chaves de abertura e fechamento “{}” e cada linha é finalizada com o caractere “;”.

5.1 Atribuição

Uma atribuição é feita utilizando o caractere “=” sendo que do lado esquerdo deve possuir o nome do identificador da variável e após o operador de atribuição a expressão a ser atribuída, que deve ser do mesmo tipo do identificador. Exemplo de uma atribuição:

tipo identificador = expressão;

5.2 Estruturas condicionais

Na linguagem PJ usamos a palavra reservada if para executar uma ação caso uma condição lógica seja verdadeira, podemos usar a cláusula elsif para obter várias condições que serão testadas em sequência. Ainda possui uma cláusula else que será executada caso todas as condições anteriores sejam falsas, onde condição pode ser qualquer expressão que seja avaliada como verdadeira ou falsa. Para cada declaração de if podemos ter várias cláusulas elsif em sequência e apenas uma cláusula else que deverá ser a última. A ordem de avaliação é em sequência. As condicionais só serão avaliadas até uma ser verdadeira, a partir dela nenhuma mais será avaliada. Caso nenhum os if e elsif em sequência forem verdadeiros, o else ao final será executado, caso exista. A seguir temos alguns exemplos:

Exemplo: Estrutura if com apenas uma via

```
if(<condição>) {
      ...
}
```

Exemplo: Estrutura if seguido de um elsif

```
if (<condição>) {
      ...
} elsif (<condição>) {
      ...
```

```
}
```

Exemplo: Estrutura if seguido de um else

```
if(<condição>) {
      ...
} else {
      ...
}
```

Exemplo: Estruturas combinadas

```
if(<condição>) {
      ...
} elsif (<condição>) {
      ...
} else {
      ...
}
```

5.3 Estrutura de repetição por controle lógico

A repetição ocorre enquanto a condição for verdadeira, quando for falsa o loop irá parar sua execução. O teste da condição ocorre antes que o laço seja executado. Desta forma se a condição for verdadeira o laço executará e testará a condição novamente. Declaramos usando a palavra reservada **while** seguida de parênteses com a condição, onde condição pode ser qualquer expressão que seja avaliada como verdadeira ou falsa. O escopo deve ser delimitado por chaves de abertura e fechamento. Exemplo:

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

5.4 Estrutura de repetição por controle de contador

Para utilizar a estrutura de repetição por contador utiliza-se o comando for seguido do identificador que irá fazer o controle. Passamos 3 informações por parâmetro, sendo eles o

valor inicial que o loop irá começar, valor final para o loop parar de executar e por fim o valor de seu incremento. Seu escopo também é delimitado por abertura e fechamento de chaves, a seguir temos uma demonstração de declaração:

```
for identifier in (initial_value, final_value, increment) {
      ...
}
```

Quando o identificador atingir um valor maior ou igual ao valor final especificado, o loop irá parar e as instruções dentro do loop não serão executadas.

5.5 Entrada e saída

As funções de entrada e saída são input para entrada e print para saída.

Para exibir uma informação na tela, utilizamos o comando print. Em PJ podemos passar uma saída formatada, sempre será retornada uma string. Para isso deve incluir especificadores de formato (começando com: %), e em seguida os argumentos separados por vírgulas, para assim então ser formatado e inserido na string resultante substituindo seus respectivos especificadores.

Para a função de leitura utilizamos o comando input, devemos declarar as variáveis que forem usadas para receber os valores previamente. Podemos receber mais de um valor, para isso separamos os parâmetros da função por “,”. Os valores são identificados através do espaço entre uma e outra. Se a quantidade de variáveis for diferente da quantidade passada para receber, o programa irá desconsiderar. Os parâmetros são passados entre parênteses.

A seguir podemos ver exemplos de leitura e escrita:

- Exemplo:

```
int name;
gets(name);
puts(name);
```

Obs: se houver o caso de executar a função gets() no fim do arquivo, ela receberá EOF

5.6 Funções

As funções funcionam baseada na linguagem C, onde primeiro definimos o tipo de retorno, seguido da palavra reservada function seguido de:

● Nome da Função.

● Lista de argumentos para a função, entre parênteses e separados por vírgulas. Deve possuir o tipo esperado do argumento a que será recebido seguido do nome dele.

● Escopo delimitado entre chaves { }.

Para retornar algum valor utiliza-se a palavra reservada return que deve retornar um valor do mesmo tipo que foi usado na assinatura da função. Caso essa função não tenha nenhum tipo de retorno, deve usar a palavra reservada void. A seguir temos um exemplo de como funciona a assinatura de uma função.

```
<tipo> fn <nome da função> (<tipo> parametro1, <tipo> parametro2, ...)
{

    ...
    return <valor>;
}
```

6. Programas de exemplos

6.1 Alô mundo

```
int fn main() {
    puts("Hello World");
    return 0;
}
```

6.2 Série de Fibonacci

```
void fn fibonacci(int num) {
    if (num == 0) {
        return;
    } elsif (num == 1) {
        puts("0");
        return;
    }

    int first = 0;
    int second = 1;
    int acc = first + second;

    while (acc < num) {
        puts(", %d", acc);
        first = second;
        second = acc;
        acc = first + second;
    }
}

int fn main() {
    int num;
```

```
    gets(num);
    fibonacci(num);

    return 0;
}
```

6.3 Shell Sort

```
int [] fn shell_sort(int unsorted_list[]) {
    int value;
    int i;
    int j;
    int h = 1;

    while (h < size) {

        for i in (h, size; i = i + 1) {
            value = unsorted_list[i];
            j = i;

            while (j > h - 1 && value <= unsorted_list[j - h]) {
                unsorted_list[j] = unsorted_list [j - h];
                j = j - h;
            }

            unsorted_list[j] = value;
        }

        h = h / 3;
    }
```

```
    return unsorted_list;
}

int fn main() {
    int unsorted_list[1000];
    int value;
    int size;
    int i = 0;

    while(gets(value) != EOF) {
        unsorted_list[i] = value;
        i = i + 1;
    }

    size = i;

    unsorted_list = shell_sort(unsorted_list);

    for i in (0, size - 1, 1) {
        puts("%d ", unsorted_list[i]);
    }

    puts("%d", unsorted_list[size - 1]);

    return 0;
}
```