UNIOESTE - Universidade Estadual do Oeste do Paraná
Centro de Ciências Exatas e Tecnológicas
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Compiladores

Especificação e definição da linguagem PortC

Gabriel Santos Ramos
Lucas Becker da Conceição

1. Introdução

Este documento refere-se à especificação da linguagem de programação PortC A especificação leva o nome da combinação de "Português" e "C". Ela define a finalidade da linguagem e tem como objetivo introduzir uma programação semelhante à C, mas ao Português.

2. Características da linguagem

PortC é uma linguagem de programação fortemente tipada baseada na gramática da linguagem C, porque esta é uma gramática bem conhecida na comunidade de programação, portanto, usar esta gramática ajudará os iniciantes de programação. Ao definir a sua gramática, serão agrupadas várias condições para que todos os códigos escritos em PortC tenham a mesma composição. O objetivo de adicionar esta funcionalidade à linguagem é evitar padrões de código diferentes, e o objetivo é utilizar a mesma estrutura e ordem para todos os códigos. O PortC executando a mesma tarefa é exatamente o mesmo (exceto para nomes de variáveis). Esta funcionalidade também tem o efeito secundário de reduzir a dificuldade de geração de código de leitura, pois sempre terá a mesma estrutura.

3. Tipo de dados

↳ Inteiro

Identificador: **INT**

Os inteiros representam números naturais positivos e negativos, incluindo zero. Seu intervalo de representação terá um limite inferior igual a -2^{31} e um limite superior igual a $2^{31}-1$, e é representado por 4 bytes.

Ponto flutuante

Identificador: **FLUT**

Os números de ponto flutuante são representações aproximadas de números racionais. Seu intervalo de representação é de 4.38×10^{-16} a 4.38×10^{16}. A separação entre o todo e a parte decimal é feita por meio de pontos.

↳ String

Identificador: **TEXTO**

Usando o tipo TEXTO, 128 caracteres da tabela ASCII podem ser representados. O valor deve ser atribuído informando os caracteres obrigatório entre aspas duplas. Por padrão, nenhum valor é atribuído a uma variável desse tipo.

↳Char

Identificador: **CHAR**

Usando o tipo CHAR, 1 caracter da tabela ASCII pode ser representado. O valor deve ser atribuído informando o caracter obrigatório entre aspas duplas. Por padrão, nenhum valor é atribuído a uma variável desse tipo.

4. Operadores lógicos

- **Disjunção**

  Operador: ||

  Representa à operação binária do OR lógico.Caso um dos operandos seja verdadeiro o resultado é verdadeiro, e falso se nenhum operando for verdadeiro.

- **Conjunção**

  Operador: &&

  Representa à operação binária do AND lógico. Caso todos os operandos sejam verdadeiros o resultado é verdadeiro, e falso caso um dos operandos não seja verdadeiro.

- **Negação**

  Operador: !

  Representa à operação unária do NOT lógico. Então a sua operação consiste em inverter o valor lógico do operador bit a bit.

-

5. Operadores aritméticos

- **Adição**

  Operador: +

  Se refere à operação binária que retorna a soma de dois operandos.

- **Subtração**

  Operando: -

  Se refere à operação binária que retorna a diferença entre dois operandos.

- **Multiplicação**

  Operador: *

  Diz respeito à operação binária que multiplica dois operandos.

- **Potência**

  Operando: ^

Se refere à operação binária que eleva o primeiro operando (base) ao segundo operando (expoente).

6. Operadores relacionais

↳ Igualdade

Operador: ==

Reflete a uma operação binária que retorna true se os dois operandos possuem valores iguais, e false caso contrário.

↳ Maior que e Maior igual que

Operador: >, >=

"Maior que", reflete a operação binária que retorna true caso o primeiro operando seja maior que o segundo. Já a operação "Maior igual que", condiz com o caso de que o primeiro valor é maior ou igual ao segundo.

↳ Menor que e Menor igual que

Operador: <, <=

Ambas operações são análogas às de "Maior que" e "Maior igual que". Entretanto, o primeiro valor deve ser menor, no primeiro caso, ou menor igual no segundo, do segundo valor.

7. Estruturas de saltos

↳ Condicional

Estrutura:

```
Se (<condição>){
<expressão>
}senao{
<expressão>
}
```

Consiste em uma estrutura de decisão que testa a "<condição>" e executa o bloco definido a seguir caso o resultado do teste seja verdadeiro. Caso contrário, a execução avança para a próxima estrutura de decisão. Apenas a cláusula "se" possui expressões que podem ser lógicas e aritméticas, a estrutura deve ser acompanhada por chaves, independente do tamanho da expressão.

8. Estruturas de repetição

↳ Laço de repetição por condição sem execução obrigatória

Estrutura:

```
Enquanto (<condição>){
        <expressão>
}
```

Consiste em uma estrutura que executa um bloco de código enquanto a condição de controle seja verdadeira, onde o teste é feito sempre antes da execução do bloco. O laço é definido pela palavra reservada "enquanto"

seguida da condição entre parênteses. O bloco a ser executado enquanto o teste for verdadeiro é definido entre chaves.

- **Laço de repetição por condição com uma execução obrigatória**

Estrutura:
Para (valor inicial, condição de parada, instrução da etapa) {
Inicio
<expressão>
Fim;}

Consiste em uma estrutura que executa um bloco de código enquanto a condição de controle é verdadeira, e o teste é sempre executado após a execução do bloco. Este loop é definido pela palavra reservada "para", o primeiro parâmetro pode ser vazio ou a inicialização de uma variável de controle, o segundo é seguido pela condição de parada que controla o laço e o terceiro por uma instrução que deve executar ao final de cada repetição. Use chaves para definir o bloco a ser executado quando o teste for verdadeiro.

9. Atribuição

Operador: =

As variáveis têm apenas um operador de atribuição. Consiste em uma operação binária do tipo "a = b", onde a variável mais à esquerda (a) recebe o valor da variável mais à direita (b), eb também pode assumir a forma de valores literais, operações matemáticas ou caracteres. As variáveis devem ter o mesmo tipo antes de serem atribuídas.

10. Palavras reservadas

Palavra Reservada	Descrição
programa	Define o início de um programa.
fimprograma	Indica o fim de um programa.
se	Inicia uma estrutura condicional (if).
senao	Caso contrário, em uma estrutura condicional.
fimse	Indica o fim de uma estrutura condicional (if).
para	Inicia um loop 'for'.
enquanto	Inicia um loop 'while'.
entao	Usado em algumas linguagens como parte da estrutura condicional.
leia	Utilizado para entrada de dados.
escreva	Utilizado para saída de dados.
inteiro	Define um tipo de variável inteira.
flut	Define um tipo de variável de ponto flutuante.
char	Define um tipo de variável de caractere.
texto	Define um tipo de variável de texto (string).
retorno	Retorna um valor de uma função.

11. Construção de identificadores

Você pode declarar variáveis usando letras, números e caracteres underline. Seus nomes só podem iniciar com uma letra e diferenciam maiúsculas de

minúsculas. O comprimento máximo do nome da variável deve ser 255 caracteres.

A declaração das variáveis é feita através da definição e nome do tipo. Você pode atribuir valores ao declarar. Variáveis do mesmo tipo podem ser declaradas na mesma linha, separadas por vírgulas.

12. Estrutura geral do programa/sintaxe

Para executar o código, é necessário apenas um bloco, separado pelas chaves para fechar o comando princ(int princ). As variáveis podem ser declaradas em qualquer parte do bloco, contanto que sejam declaradas antes do uso da variável. Cada instrução deve ser finalizada com “ ; ”.

13. Definição formal EBNF da linguagem PortC

↳ Abreviações utilizadas

<declaração_var> = <dv>
<estrutura_repeticao> = <er>
<declaração_estrutura> = <de>
<operação> = <op>
<operacao_relacional> = <op_rel>
<operando_relacional> = <sign_rel>
<operando_logico> = <sign_lo>
<operacao_logica> = <op_lo>
<operacao_aritmetica> = <op_ar>
<expressão> = <expr>
<especificação_var> = <ev>
<Alfabeto, Número, Underscore> = <ANU>
<operando_atribuição> = <oa>
<bloco_de_codigo> = <bk>

↳ Programa

<Portc> -> < " programa”<bk>>
<bk> -> “{“<expr>”}”

↳ Expressão

<expr> -> [<dv> | <dv> <expr>] [<de> | <de> <expr>] [<er> | <er> <expre>] [<op> | <op> <expr>] [<op_ar> | <op_ar> <expr>]

↳ Declaração de variável

<dv> -> <tipo> <nome_var> <ev>
<tipo> -> INT | FLUT | CHAR| TEXTO
<nome_var> -> <alfabeto> <ANU>
<ev> -> <,> <nome_var> <ev> | <atribuicao> <ev> | <;>
<atribuicao> -> <recebe> <valor> | <recebe> <nome_var>
<recebe> -> "="
<valor> -> int | flut | char | texto
<ANU> -> <alfabeto> | <alfabeto> <ANU> | <numero> | <numero> <ANU> | <underscore> | <underscore> <ANU>
<alfabeto> -> "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
<número> -> "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<underscore> -> "_"

*<valor> Refere-se a possíveis valores adequados para o tipo especificado.

↳↳Atribuição de valor

<atribuicao> -> <nome_var> <recebe> <valor> | <nome_var> <recebe> <nome_var>
<recebe> -> "="
<valor> -> int | flut | char | texto

*<valor> Refere-se a possíveis valores adequados para o tipo especificado.

↳↳Declaração de estrutura

<de> -> “SE(”<stmt>”)<bk> | “SE(”<stmt>”)<bk>“SENAO”<bk>

↳ Estrutura de repetição

<er> -> “ENQUANTO(“<stmt>”)”<bk> | “PARA<bk>(“<stmt>”)”

↳↳Operações, relacional e lógica

<op> -> <op_rel> | <op_lo>
<op_rel> -> <nome_var><sign_rel><nome_var>
<op_lo> -> <nome_var><sign_lo><nome_var>

↳↳Operandos

<sign_rel> -> “==” | “>” | “<” | “>=” | “<=”
<sign_lo> -> “||” | “&&” | “!” | “|&”

↳↳Operação aritmética

<op_ar> -> <var> <operando> <var> [<esp>] “ ;”
<var> -> <valor> | <nome_var>
<esp> -> <operando> <var> <esp> | <operando> <var>
<operando> -> + | - | * | ^

↳ Gramática de Atributos

1. Sintaxe: <assign> → <var> = <expr>
Semântica: .expected_type ← .actual_type

2. Sintaxe: <expr> → ><var>[2] + <var> [3]
Semântica: .actual_type ← se ([2].actual_type = int) & ([3].actual_type = int)
int
senao
flut
Predicado: <expr>.actual_type = .expected_type

3. Sintaxe: <expr>→ <var>
Semântica: <expr>.actual_type ← <var>.actual_type
Predicado: <expr>.actual_type = <expr>.expected_type

4. Sintaxe: <var> → A | B | C
Semântica: <var>.actual_type ← look-up(<var>.string)

1. <var>.actual_type ← look-up(A) (regra 4)
2. <expr>.expected_type ← <var>.actual_type (regra 1)
3. <var>[2].actual_type ← look-up(A) (regra 4)
4. <var>[3].actual_type ← look-up(A) (regra 4)
5. <expr>.actual_type ← int ou flut(regra 2)
6. <expr>.expected_type = <expr>.actual_type é TRUE ou FALSE (regra 2)

↳ Regras de Associatividade

A regra de associatividade é determinada da direita para esquerda.
As regras de precedência de operadores para avaliação de expressões definem a ordem pela qual os operadores de diferentes níveis de precedência são avaliados
Exemplo: **a + b * c**
Supondo a= 3; b=4 e c= 5.
Se a expressão for avaliada da direita para a esquerda: **23**
Se avaliada da esquerda para a direita: **35**

Níveis de precedência mais utilizado (hierarquia de prioridades)
1. parênteses
2. operadores unários
3. ** (exponenciação) quando apresentar
4. *, /
5. +, -

Regra de associatividade de operadores baseados em C.
Esquerda: * / % + -
Direita: ++ e --
]