



Compiladores – Prova #1

Nome:

Matrícula:

Data:

Observações:

- (a) A prova é individual e sem consulta, sendo vedado o uso de calculadoras e de telefones celulares.
- (b) A interpretação dos comandos das questões faz parte da avaliação.
- (c) A nota da prova será igual a 0 (zero) caso o estudante consulte algum material durante a prova, ou receba ou ofereça qualquer ajuda a outro estudante durante a prova.
- (d) As questões podem ser resolvidas a lápis ou à caneta. Entretanto, a resposta final deve ser **destacada** de forma clara (circulada, sublinhada, reforçada, indicada, etc...) para que a questão seja devidamente corrigida.
- (e) O grampo da prova **não deve ser removido**. Caso o grampo seja removido, a nota da prova será igual a 0 (zero).

Parte A

1. (7 pontos) Complete a sentença: “A compilação pode ser dividida em 6 fases, as quais podem ser classificadas em duas partes: análise e síntese. Outra organização possível separa as 4 primeiras fases na interface de vanguarda e as 2 últimas fases na interface de retaguarda”.

2. Considere a gramática livre de contexto G abaixo:

$$\begin{aligned} S &\rightarrow S \text{ concat } L \mid S \text{ replace } L L \mid \epsilon \\ L &\rightarrow a \mid b \mid c \end{aligned}$$

De acordo com as convenções de notação, responda as questões abaixo?

- (i) (2 pontos) Quais são os não-terminais de G ?
 S e L .
- (ii) (2 pontos) Quais são os terminais de G ?
concat, replace, ϵ , a , b e c .

- (iii) (1 ponto) Qual é o símbolo de partida de G ?
 S (não-terminal da primeira produção listada).

3. (8 pontos) Assinale a alternativa correta. Considere o seguinte código de máquina de pilha:

```
value-l a
push 5
push 2
φ
value-r b
⋮
```

Marque a expressão, em notação infixada, que é avaliada por este código. Assuma que as operações ϕ e $\ddot{=}$ sejam binárias e que o ordem de extração dos operando da pilha seja a seguinte: primeiro o operando à direita, em seguida o operando à esquerda.

- (X) $a := (5 \phi 2) \ddot{=} b$
- (B) $a := b \ddot{=} (2 \phi 5)$

- ## Parte B

```
do { cmd } while ( expr );
```

```
cmd ; while ( expr ) { cmd }
```

rótulo loop
código para cmd
código para expr
gotrue loop

$$S \rightarrow (S)S \mid \epsilon$$
[illegible]

7. Em relação aos analisadores gramaticais recursivos descendentes:

- (i.) (8 pontos) Defina uma gramática não-ambígua que gere expressões formadas pelos valores lógicos **t** e **f** (verdadeiro e falso, respectivamente) e a conjunção (**^**). Assuma que a conjunção é uma operação binária associativa à direita.

$$S \rightarrow t \wedge S \mid f \wedge S \mid t \mid f$$

- (ii.) (12 pontos) Construa um analisador gramatical recursivo descendente para a gramática definida no item anterior.

Primeiro a gramática deve ser reescrita, usando a fatoração à esquerda, para permitir a implementação de um analisador recursivo preditivo:

$$\begin{aligned} S &\rightarrow t R \mid f R \\ R &\rightarrow \wedge S \mid \epsilon \end{aligned}$$

```
1 void S()
2 {
3     if (lookahead == 't' or lookahead == 'f')
4     {
5         reconhecer(lookahead);
6         R();
7     } else
8         erro();
9 }
10
11 void R()
12 {
13     if (lookahead == '^')
14     {
15         reconhecer('^');
16         S();
17     }
18 }
```

Parte C

8. Uma constante inteira em base binária (token **NUM**) é uma cadeia não-vazia formada pelos caracteres 0 e 1 sendo que, exceto pelo número zero, nenhuma outra cadeia deve iniciar com o caractere zero. Um identificador (token **ID**) é uma cadeia não-vazia composta pelos caracteres **a** e **b**, nas quais dois caracteres consecutivos devem ser sempre distintos.

Considere a seguinte implementação de um analisador léxico que identifica estes tokens. Assuma

- (a) que o atributo de **NUM** é o seu valor em base decimal (por exemplo, para o lexema **101** geraria um token **NUM** cujo atributo seria igual a 5),
- (b) que o atributo de **ID** seja o número de caracteres que compõem o lexema (por exemplo, o lexema **abab** geraria um token **ID** cujo atributo seria igual a 4),
- (c) que a função `erro()` esteja devidamente implementada,
- (d) que os tokens foram devidamente declarados, e
- (e) que o código esteja sintaticamente correto segundo a linguagem C++.

```

1 using token = std::pair<int, int>;
2
3 token_t scanner()
4 {
5     while (not std::cin.eof())
6     {
7         auto c = std::cin.get();
8
9         if (c == '0' or c == '1')
10        {
11            int valor = 0;
12
13            while ((not std::cin.eof()) and (c = std::cin.get(), c == '0' or c == '1'))
14            {
15                valor *= 10;
16                valor += c - '0';
17            }
18
19            std::cin.unget();
20
21            return { NUM, valor };
22        } else if (isalpha(c))
23        {
24            int len = 1, prev = c;
25
26            while ((not std::cin.eof()) and (c = std::cin.get(), c == 'a' or c == 'b'))
27            {
28                if (c == prev)
29                    break;
30
31                ++len, prev = c;
32            }
33
34            std::cin.unget();
35
36            return { ID, len };
37        } else
38            erro();
39    }
40
41    return { EOF, -1 };
42 }

```

(i.) (24 pontos) Identifique três erros semânticos presentes nesta implementação. Para cada erro, indique o número da linha onde ele ocorre, descreva e justifique o erro e proponha uma correção para este erro. Esta proposição pode ser feita de forma descritiva, sem necessariamente mostrar o código C++ correspondente à correção.

1. Na linha 22, o código aceita qualquer caractere alfabético como primeiro elemento da cadeia de um identificador, porém a descrição limita os caracteres dos identificadores apenas a a e b. Isto pode ser corrigido pela reescrita da condição:

```

22     if (c == 'a' or c == 'b')

```

2. Na linha 15, a conversão de binário para decimal usa a base errada: da forma que está escrito, a cadeia de entrada seria entendida como se já estivesse em base 10, convertendo '101' para 101 e não 5. Para remover este erro basta corrigir a base:

```

15         valor *= 2;

```

3. O bloco que inicia na linha 9 não trata da regra que apenas o número zero pode iniciar com o caractere zero. Esta verificação pode ser adicionada antes da linha 11:

```

10         if (c == '0' and (not std::cin.eof()))
11         {
12             auto d = std::cin.get();
13
14             if (d == '0' or d == '1')
15                 erro();
16             else
17                 std::cin.unget();
18         }

```

(ii.) (6 pontos) Forneça uma cadeia de caracteres que a implementação acima reconheceria como um token válido mas que viola uma ou mais dentre as regras de formação de tokens apresentadas.

Por causa do erro 1, a cadeia 'cab' seria considerada válida; por causa do erro 3, a cadeia '0010' seria identificada erroneamente como válida.