Segunda Prova de Construção de Compiladores.
Primeiro Semestre de 2003.
Departamento de Computação – UFSCar.
José de Oliveira Guimarães.
Turma A (Terça).

Lembre-se: justifique tudo a menos de menção em contrário.

Entregue apenas a folha de respostas. Isto é, não entregue esta folha ou o rascunho.

1. (2.0) Escolha e faça UM e apenas UM dos itens abaixo.

(a) Explique o que é sistema de tempo de execução. Cite duas das suas funções.

(b) O que é um interpretador ? Cite uma vantagem de interpretadores sobre compiladores e uma de compiladores sobre interpretadores.

2. (1.5) Faça uma gramática com operadores binários `&&` e `||` e operadores unários prefixados `#` e `*` de tal forma que:

(i) `||` tenha maior precedência do que `&&`;

(ii) `&&` seja associativo à esquerda (então `a && b && c` quer dizer `(a && b) && c`) e `||` seja associativo à direita;

(iii) os operadores unários tenham igual precedência e maior do que os operadores binários;

3. (2.0) Retire a recursão à esquerda e fatore a seguinte gramática.

```
E ::= E T (+|-) | T
T ::= T * F | T / F | F
F ::= N
```

4. (4.0) Dados os métodos de análise sintática

```
void p() {
   if ( token ∈ first(A) )
      a();
   else if ( token ∈ first(B) )
      b();
   else
      error();
}

void q() {
   if ( token ∈ first(C) )
      c();
   else if ( token == 'a' ) {
      nextToken();
      b();
   }
}
```

faça as produções `P` e `Q` correspondentes da gramática e responda às seguintes questões:

(a) `a` ∈ `first(C)` ?

(b) `a` ∈ `follow(Q)` ?

Justifique. Não é necessário explicar a regra que justifica as respostas. Apenas cite a regra. Assuma que a gramática utilizada está fatorada à esquerda, não possui recursão à esquerda e não seja ambígua.

5. (2.5) Faça:

(a) a expressão regular do JLex que reconheça a seguinte seqüência de caracteres: dígito (um único) seguido de qualquer número de letras maiúsculas (zero ou mais) seguido de uma ou mais letras

minúsculas e terminando com 0 ou 1.

(b) o desenho do autômato finito que reconheçe a expressão regular descrita acima.

6. (2.0) Otimize os seguintes trechos de código. Faça apenas as otimizações que um compilador poderia fazer. Indique na sua resposta exatamente onde você fez as otimizações.

(a) (0.5)

```
L1: cmp i, n
    goto>= L2
    ; codigo para o corpo do while
    goto L1
L2:
```

(b) (1.5)

```
void f(int j, int n) {
   int m = j + n;
   if ( j > n )
      cout << j << endl;
   else {
      int i = 0;
      i++;
      while ( i < n/2 ) {
         j = j + 85*i;
         i++;
      }
      f( 3*j, n );
   }
}
```

7. (1.0) LMM é uma linguagem visual que permite a programação por meio de movimentos das mãos. Um programa em LMM é uma fita de vídeo composta por uma seqüência de quadros em formato jpeg. Um compilador LMM interpreta a fita produzindo código executável. Cada comando corresponde a um movimento diferente das mãos, que não precisa ser *exatamente* igual ao movimento definido pela linguagem. Mas deve ser bem próximo. Baseado nesta definição, faça os dois itens seguintes.

(a) Explique como seria uma gramática de LMM. Certamente a gramática não pode ter regra do tipo `P ::= a A | b B` onde `a` e `b` são tokens. Não há caracteres que podem ser tokens.

(b) Suponha que tenhamos algo *equivalente* à produção acima. O `a` é um comando, representado de alguma forma. Explique como o compilador decidiria entre as duas produções da regra acima. `a` e `b` são comandos e corresponderiam a movimentos das mãos.