UNIP – Universidade Paulista	
Curso.....:	Bach. em Ciência da Computação
Disciplina..:	Compiladores e Computabilidade
Professor...:	Leandro Carlos Fernandes

-:: Lista de Exercícios #4 ::-

Tópico: Geração de Código

1) Argumente sobre ao menos três pontos positivos na prática da geração de código intermediário antes da síntese do código alvo propriamente dito.

2) Qual a diferença entre as representações de código intermediário chamadas *HIR*, *MIR* e *LIR*? Por que existem todos esses tipos e não há apenas um, já que o código de máquina já baixo nível?

3) Reescreva cada uma das atribuições a seguir na forma de instruções de três endereços, tal qual se espera que seja o código intermediário correspondente a essas construções:
   a) x = a + b / (c + d);
   b) x = a++ * ++b + c * 2;
   c) MF = ((NP1 + NP2) / 2 + EX) / 2;
   d) area = pi() * raio * raio;

4) Linguagens de alto nível usualmente oferecem três comandos de decisão diferentes: *if*, *if* ... *else* e *switch*. Abaixo há três exemplos de uso destes comandos, para os quais pede-se que construa o código MIR correspondente.

```
x = 0;
if (a <= b) {
  x = 1;
}
y = x;
```

```
x = 0;
if (a != b) {
  x = 1;
}
else {
  x = 2;
}
y = x;
```

```
x = 0;
switch (a) {
  case 1: x = 10;
          break;
  case 2: x = 20;
          break;
  default:
          x = 0;
}
y = x;
```

5) O conjunto de comandos a seguir corresponde a um fragmento de um algoritmo que, sendo informado três valores inteiros distintos, os escreve em ordem crescente na tela.

```
if (a < b) {
   if (c < a) {
      prim = c; seg = a; ter = b;
   }
   else if (c < b) {
      prim = a; seg = c; ter = b;
   }
   else {
      prim = a; seg = b; ter = c;
   }
}
else {
   if (c < b) {
      prim = c; seg = b; ter = a;
   }
   else if (c < a) {
      prim = b; seg = c; ter = a;
   }
   else {
      prim = b; seg = a; ter = c;
   }
}
println("%d %d %d", prim, seg, ter);
```

Escreva o código intermediário resultante de sua tradução na forma de instruções de três endereços.

6) Dê o código intermediário para as seguintes construções:

a)
```
for(int i = 0; i < 10; i++)
    x = i * 2;
```

b)
```
int i = 0;
while(i < 10) {
    x = i * 2;
    i++;
}
```

c)
```
int i = 0;
do {
    x = i * 2;
} while(i++ < 10);
```

Você observou alguma peculiaridade entre os códigos gerados? O que foi e qual seria a explicação disto?

7) Considere os trechos de programa em C a seguir e dê o código intermediário equivalente à sua tradução.

a)
```
ehPalin = 1;
tam = strlen(str);
for(i = 0; i < tam/2; i++) {
  if (str[ i ] != str[ (tam-1) - i ])
    ehPalin = 0;
}
```

b)
```
for(i = 0; i < (n - 1); i++)
  for(j = 0; j < (n - (i + 1)); j++)
     if( list[j] > list[j+1] )
        swap( list[j], list[j + 1] );
```

Nota: suponha que list seja um vetor de inteiros de 32 bits e que swap é uma função que troca os conteúdos de duas variáveis informadas como parâmetro (passagem por referência).

8) Reescreva o código resultante do exercício anterior na forma de *quádruplas*.

9) Converta as instruções de *quádruplas* que você obteve no exercício anterior para o formato de *triplas*.

10) Dados os trechos de código intermediário abaixo, reescreva-os na forma original utilizando uma sintaxe C-*liked*.

```
_L1:   _t1 := b * b
       _t2 := 4 * a
       _t3 := _t2 * c
       D := _t1 – _t3
       if D < 0 goto _L2
       param D
       _t4 := call sqrt,1
       _t5 := –b + _t4
       _t6 := 2 * a
       x1 := _t5 / _t6
       param D
       _t7 := call sqrt,1
       _t8 := –b - _t7
       _t9 := 2 * a
       x2 := _t8 / _t9
       goto _L3
_L2:   x1 := 0
       x2 := 0
_L3:   ...
```

```
_L1:   if b == 0 goto _L2
       _t1 := -a
       _t2 := _t1 / b
       _t3 := a / b
       param _t3
       param 2
       _t4 := call log,2
       _t5 := _t2 * _t4
       _t6 := -c
       _t7 := _t6 / b
       _t8 := c / b
       param _t8
       param 2
       _t9 := call log,2
       _t10 := _t7 * _t9
       e := _t5 + _t10;
       goto _L3
_L2:   param "Erro”
       call print, 1
_L3:   param e
       call print, 1
```

11) Dê cinco exemplos de otimizações que podem ser realizadas no código intermediário gerado por um compilador.