Tópicos de resolução do miniteste de AED de 2005-04-20

1.

a) *Linha 17:* membro-função não pode ser "const" porque altera os dados; remover a palavra "const".
Linha 20: tipo errado de retorno; deve ser "return *this;"
Linha 25: função é do tipo "void", pelo que não deve retornar nenhum valor; mudar "return 0;" para "return;" ou remover a instrução.

b)
```
XX!!   ou        XX!!
1                9
2                7     (depende da alteração feita na linha 24)
```

2.

a)
```
static double c_para_k(double c) { return c+273.15; }
static double f_para_k(double f) { return (f+459.67)/1.8; }
static double k_para_c(double k) { return k-273.15; }
static double k_para_f(double k) { return 1.8*k-459.67;}
```

b)
```
static double Temperatura::para_k(double valarg, char unidade)
{
    switch(unidade) {
    case 'K': return valarg;
    case 'C': return c_para_k(valarg);
    case 'F': return f_para_k(valarg);
    default: throw Erro();
    }
}
static double de_k(double valarg, char unidade)
{
    switch(unidade) {
    case 'K': return valarg;
    case 'C': return k_para_c(valarg);
    case 'F': return k_para_f(valarg);
    default: throw Erro();
    }
}
```

c)
```
Temperatura::Temperatura(double val, char unidade)
{ kelvin = para_k(val,unidade); }
```

d)
```
bool Temperatura::operator==(const Temperatura &t) const
{ return kelvin==t.kelvin; }
```

e)
```
double Temperatura::operator[](const char unidade) const
{ return de_k(kelvin,unidade); }
```

f)
```
Temperatura & Temperatura::somar(double var, char unidade)
{ kelvin += para_k(var,unidade); return *this; }
```

g)
```
void Temperatura::escrever(ostream & os, const char unidade) const
{ os << de_k(kelvin,unidade) << " " << unidade << endl; }
```

h)
```
ostream & operator<<(ostream & os, const Temperatura &t)
{ t.escrever(os,'C'); return os; }
```

3.

```
10    -10   -10   10    10
20    -20   -10   20    10
11    -11   -10   11    10
```

Os objectos que são instâncias de subclasses de uma outra classe (neste caso Medida),podem ser referenciados através de variáveis do tipo da sua superclasse, o que acontece nas funções g, f e h. Contudo, o comportamento exibido é o da subclasse respectiva apenas quando esta referência é realizada através de apontador, e os métodos invocados são virtuais (polimorfismo): caso da função g. Caso contrário, o comportamento exibido é o da superclasse: caso das funções f e h.

4.
a)
Complexidade espacial: **O(1)**, porque todas as variáveis têm tamanho fixo.

Complexidade temporal: todas as instruções executam em tempo constante menos os ciclos. Os dois ciclos embrincados são executados, no pior dos casos, N(N-M) = N^2-NM vezes, Assumindo que M<<N, a complexidade temporal é **O(N^2)**, porque o corpo do ciclo interior executa em tempo constante.

b)
Complexidade espacial: permanece a mesma, porque o tamanho de todas as variáveis usadas é independente do tipo genérico T (incluindo os argumentos da função, porque são referências).

Complexidade temporal: A única operação que depende do tipo genérico T é a comparação existente no corpo do ciclo interior. Por exemplo, se T for "string" a comparação pode ser linear no número de caracteres. Assim, a complexidade temporal pode ser generalizada para **O(N^2)*O(K(T))** em que K(T) representa a complexidade da operação de comparação de elementos do tipo T. [No caso da alínea (a) K(T) é constante, logo O(K(T)) = O(1).]

FIM.