Universidade do Minho	**Escola de Engenharia da Universidade do Minho** Mestrado Integrado em Eng. Electrónica Industrial e Computadores Programação	**2011/2012** **MIEEIC** **(1º Ano)** **2º Sem**

Teste Nº2 - Época Normal - Data 08/06/2012, Duração 1h45m

Nome:_____ Nº Al:_____

Responda às seguintes questões, preenchendo a tabela com a **opção correcta (em maiúsculas) (Correcto:x Val / Errado: -x/3 Val).** Suponha que foram realizados as inclusões das bibliotecas necessárias.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

GRUPO I (6 Valores)

1. O tratamento de "exceções" utilizado em C++, permite:

A) Eliminar todos os erros sintáticos dos programas.
B) Tornar os programas mais fáceis de usar.
C) Alterar o fluxo normal dos programas, em casos especiais.
D) Eliminar todos os erros lógicos e sintáticos dos programas.
E) Nenhuma das respostas anteriores.

2. A classe vector da STL de C++ é um tipo abstrato de dados.

A) Sim, mas só se se declarar no código fonte que vector é do tipo abstrato.
B) Não, pois o tipo de dados abstrato é a lista (list da STL).
C) Sim, pois contém dados e operações sobre esses dados.
D) Não, pois é um dos tipos base de dados do C++ (o array).
E) Não pois encapsula os dados e operações.

3. Considere a seguinte definição do objeto nomes: `list<string> nomes;` Qual poderá ser o protótipo da função pesquisar(), que tenta averiguar se um dado nome já existe numa lista de nomes?

A) bool pesquisar (const list<string> li, const object nom);
B) void pesquisar (list<char *> lista, const str x);
C) bool pesquisar (const list<string>, const string);
D) bool pesquisar (vector<string>, void string);
E) Nenhuma das respostas anteriores.

4. Uma pilha (stack) e uma fila (queue) de objectos:

A) Permitem implementar listas duplamente ligadas.
B) São pequenas variantes de listas de objectos implementadas com apontadores.
C) Podem ser implementadas a partir de listas de objectos.
D) Têm todos os métodos de acesso tipicamente utilizados nas listas.
E) Nenhuma das respostas anteriores.

5. O método de ordenação por partição (Quick Sort):

A) É sempre mais rápido do que a ordenação por inserção e quase sempre mais rápido do que a ordenação MergeSort.
B) Escolhendo para Pivot sempre o maior valor do array, tem complexidade no tempo $O(n^2)$ e complexidade no espaço O(n).
C) No caso médio (valores aleatoriamente distribuídos) tem complexidade no tempo O(n*log(n)) e complexidade no espaço O(n*log(n)).
D) No caso médio tem uma complexidade no espaço de ordem inferior à dos métodos de ordenação por inserção e ao Bubble Sort pois faz a partição e ocupa menos espaço.
E) Nenhuma das anteriores.

6. A função `repetidos` apresentada a seguir, determina se existe algum valor repetido num vetor especificado como argumento:

```
template class T
public boolean repetidos(T v1[]) {
  ordena(v1);
  for(int i=0 ; i<v1.size()-2 ; i++ )
      if ( v1[i]==v1[i+1] ) return true;
  return false;
}
```

Por forma a que o tempo de execução da função repetidos seja O(n*log(n)) e o espaço adicional ocupado seja O(1) o método ordena deverá ser:

A) Ordenação por Partição;
B) Ordenação por Inserção;
C) Ordenação por MergeSort;
D) Ordenação por BubbleSort;
E) Nenhuma das Anteriores

7. Considere o seguinte programa:

```
// fichs inclusão e declarações gerais
int f(vector<int> arr) {
  queue<string> q;
  stack<string> s;
  if (n < 1) return -1;
  for (int i=0; i < arr.size(); i++) {
     q.push(arr[i]);
  }
  for ( ; !q.empty(); ) {
     s.push(q.front()); q.pop();
  }
  for ( ; !s.empty(); ) {
     cout << s.top() << " "; s.pop();
  }
  return 0;
}
```

A) No final da execução, tanto a pilha como a fila terão size 1 (tamanho 1).
B) Se a função for invocada com um arr={2, 3, 4}, aparecerá no ecrã: 2 3 4
C) Se função for invocada com um arr={}, isto é um vetor vazio, o programa "estoura".
D) No final do programa a fila estará vazia mas a pilha não (pois recebeu todos os elementos da fila).
E) Nenhuma das respostas anteriores.

8. Supondo o seguinte código aplicado a uma árvore binária, selecione a afirmação verdadeira:

```
1 void trav(TNode *X) {
2    if(X!=0) {
3        trav(X->right());
4        //usar nodo
5        trav(X->left());
6    }
7 }
```

A) Se a linha 4 for substituída por `delete X;` poderia ser utilizado para apagar completamente a árvore binária.
B) O código implementa uma travessia em pós-ordem de uma árvore binária.
C) O código implementa uma travessia em pré-ordem de uma árvore binária.
D) O código implementa uma travessia em ordem de uma árvore binária.
E) Nenhuma das anteriores.

9. Considere o seguinte fragmento de código C++:

```
vector<int> v1; int n;
cin >> n;
for(int i=0; i<n; i++)
   for(int j=2; j<n; j++) {
       cout << i << " - " << j << endl;
       v1.push_back(i*j);
   }
```

A complexidade temporal do fragmento de código é:

A) T(n)=O(n*log(n)).
B) T(n)=O(1).
C) T(n)=O(n^2).
D) T(n)=2*O(n).
E) Nenhuma das anteriores.

10. Considere o seguinte fragmento de código C++:

```
vector<int> v1; int n;
cin >> n;
for(int i=0; i<n; i++)
   for(int j=i; j<i+2; j++) {
       cout << i << " - " << j << endl;
   }
```

A complexidade temporal do fragmento de código é:

A) T(n)=O(n*log(n)).
B) T(n)=O(n).
C) T(n)=O(n^2).
D) T(n)=2*O(n).
E) Nenhuma das anteriores.

11. Considere o seguinte fragmento de código C++:

```
vector<int> v1; int n;
cin >> n;
for(int i=0; i<n; i++)
   for(int j=i; j<i+2 ; j++)
      for(int k=i+2; k>2 ; k = k/2)
         v1.push_back(i+j+k);
for(vector<int>::iterator it = v1.begin();
    it!=v1.end(); it++)
        cout << *it << " ";
```

A complexidade temporal do fragmento de código é:

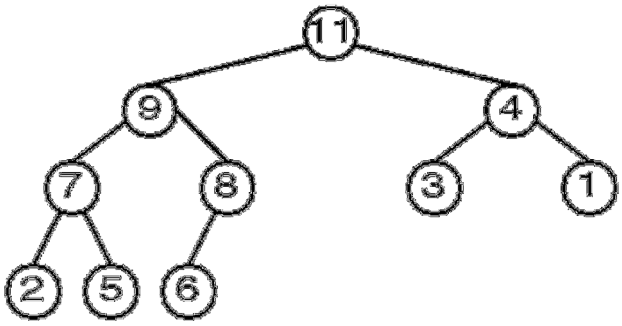
A) A complexidade temporal é T(n)=O(n*log(n)).
B) A complexidade temporal é T(n)=O(n^2*log(n)).
C) A complexidade temporal é T(n)=O(n^2).
D) A complexidade temporal é T(n)=O(n^3).

Nenhuma das anteriores.

12. A operação de pesquisa de um elemento numa lista:

A) Possui complexidade temporal linear se a implementação da lista é baseada em vetores ou em listas ligadas.
B) Possui complexidade temporal constante se a implementação da lista é baseada em vetores ou em listas ligadas.
C) Possui complexidade temporal linear se a implementação da lista é baseada em vetores, e constante se baseada em listas ligadas.
D) Não é possível utilizando listas ligadas pois os nós teria de estar em posições consecutivas.
E) Nenhuma das anteriores.

Suponha a seguinte árvore binária:



13. A travessia em ordem da árvore resulta na seguinte sequência:

A) 2 7 5 9 8 6 11 3 4 1
B) 11 9 7 2 5 8 6 4 3 1
C) 2 7 5 9 6 8 11 3 4 1
D) 2 7 9 5 6 8 11 4 3 1
E) Nenhuma das anteriores.

14. A travessia pós-ordem da árvore resulta na seguinte sequência:

A) 2 5 7 8 6 9 3 1 4 11
B) 11 9 7 2 5 8 6 4 3 1
C) 2 7 5 9 6 8 11 3 4 1
D) 11 9 7 2 5 6 8 4 3 1
E) Nenhuma das anteriores.

15. A operação de remoção de um elemento numa lista:

A) Possui complexidade temporal linear se a implementação da lista é baseada em vetores, e constante se baseada em listas ligadas.
B) Não é possível em vetores pois não podem ficar "buracos" no vetor.
C) Possui complexidade temporal linear se a implementação da lista é baseada em vetores ou em listas ligadas.
D) Possui complexidade temporal constante se a implementação da lista é baseada em vetores ou em listas ligadas.
E) Nenhuma das anteriores.

16. A operação de inserção de um elemento numa pilha:

A) Possui complexidade temporal linear se a implementação da pilha é baseada em vetores e constante se baseada em listas ligadas.
B) Não é possível em vetores pois não é possível comprimir elementos para arranjar espaço para o elemento a inserir.
C) Possui complexidade temporal linear se a implementação da pilha é baseada em vetores ou em listas ligadas.
D) Possui complexidade temporal constante se a implementação da pilha é baseada em vetores ou em listas ligadas.
E) Nenhuma das anteriores.

GRUPO II (7 Valores)

2) O proprietário de uma cadeia de parques de estacionamento deseja manter informação sobre os seus clientes e localização das respetivas viaturas. Para isso pretende implementar duas classes em C++ que permitam representar a informação sobre o estado de cada parque de estacionamento (CParqueEstac) e o estado de cada cartão cliente (CInfoCartao). Implemente as classes CParqueEstac e CInfoCartao de acordo com a especificação das alíneas seguintes Suponha que foram incluídos todos os #include necessários. A declaração das classes é a seguinte:

```
1  class CInfoCartao { // classe auxiliar
2    public:
3      string nome;
4      bool presente;
5  };
6
7  class CParqueEstac{
8     int vagas;
9     const int lotacao;
10    vector< CInfoCartao> clientes;
11    int num_clientes;
12    int pos_cliente(const string &nome) const;
13  public:
14   CParqueEstac(int lot, int nmax_clientes);
15   ~CParqueEstac();
16   bool novo_cliente(const string &nome);
17   bool retira_cliente(const string &nome);
18   bool entrar (const string &nome);
19   bool sair (const string &nome);
20   int num_lugares() const;
21   int num_lugares_ocupados() const;
22  };
```

2.1) Implemente o construtor:

```
CParqueEstac::CParqueEstac(int lot,
              int nmax_clientes)
```

que aceita como parâmetros a lotação do parque e o número máximo de clientes com acesso ao parque (p. ex. um parque de 200 lugares pode servir uma clientela de 300 pessoas). Ambos os valores são constantes. O construtor deve criar um array para guardar informação sobre os clientes (inicialmente o parque não tem clientes e está vazio). Nota: cuidado ao iniciar o membro dado constante `lotacao`.

2.2) Implemente o membro-função que retorna o número de viaturas presentes no parque.

```
int CParqueEstac::num_lugares_ocupados() const
```

2.3) Implemente o membro-função:

```
bool CParqueEstac::novo_cliente(const string
&nome)
```

A função retorna true caso consiga inserir o novo cliente, isto é, o número máximo de clientes ainda não foi atingido e o cliente ainda não existe. Este está inicialmente fora do parque.

2.4) Implemente o membro-função que retorna o índice do cliente no vetor de clientes. Retorna -1 caso o cliente não exista.

```
int CParqueEstac::pos_cliente(const string
&nome) const
```

2.5) Implemente o membro-função que regista a entrada de um cliente. A função retorna false se o cliente não puder entrar (porque não existem vagas, não está registado ou a sua viatura já está dentro do parque).

```
bool CParqueEstac::entrar(const string &nome)
```

2.6) Implemente o operador de comparação. Considere que dois parques são iguais se tiverem a mesma lotação e o mesmo número máximo de clientes.

```
bool CParqueEstac::operator==(const CParqueEstac
&pe) const
```

GRUPO III (7 Valores)

3) Pretende-se implementar uma classe para modelar o atendimento aos clientes num banco. Existem duas filas, uma prioritária e outra normal, e os clientes podem realizar dois tipos de operações: depósitos e levantamentos.

```
class CCliente
{
    string operacao;
    double montante;
  public:
    CCliente(string o="deposito", double m):
            operacao(o), montante(m) {}
    friend class CBanco;
};

class CBanco
{
    queue<CCliente> filaPrioritaria;
    queue<CCliente> filaNormal;
    double valorEmCofre;
  public:
    CBanco(double valor=10000):
           valorEmCofre(valor) {}
    void insereCli(CCliente cte, string fila);
    bool clientesEmEspera();
    CCliente proximoClienteServir();
    double montanteaMovimentar();
};
```

3.1) Implemente o membro-função que insere o cliente na fila indicada, que pode tomar dois valores ("`normal`" ou "`prioritaria`"), para realizar uma `operação`, que pode ser de "deposito" ou de "`levantamento`" de um dado `montante`. Contudo um cliente que realiza um levantamento, independente do valor passado à função, deverá sempre ser colocado na fila "`normal`". As operações de "`deposito`" incrementam o membro-dado `valorEmCofre` do banco, enquanto que nas operações de "levantamento" o `valorEmCofre` sofre um decremento do valor da operação.

```
void CBanco::insereCli(CCliente cte, string fila)
```

3.2) Implemente o membro-função que retorna verdadeiro se ainda existir algum cliente à espera de ser atendido, em quaisquer filas.

```
bool CBanco::clientesEmEspera()
```

3.3) Implemente o membro-função que retorna o próximo cliente a ser servido e remove-o da fila respetiva. Os primeiros clientes a serem servidos são os da `filaPrioritaria`, e só a seguir são atendidos os clientes da `filaNormal`. Caso não existam clientes à espera em qualquer das filas deve ser escrita no écran uma mensagem de erro.

```
CCliente CBanco::proximoClienteServir()
```

3.4) Implemente um membro-função que escreve no écran e retorna a soma dos montantes a movimentar (depósitos e levantamentos) por todos os clientes que se encontram à espera de ser servidos na `filaPrioritaria` e `filaNormal`:

```
double CBanco::montanteMovimentado()
```

3.5) No Banco, por vezes existe a necessidade de juntar ambas as filas numa fila única. Dependendo do número de elementos em cada fila, o modo como as filas são juntas é distinto. No caso de existirem menos de 20 elementos no total das duas filas e menos de 5 na fila prioritária, os elementos da fila prioritária são colocados à cabeça da nova fila resultado e só depois os elementos da fila normal. Em qualquer outro caso, os elementos são colocados intercalados, i.e. primeiro um da fila prioritária, depois um da fila normal, depois um da prioritária e assim consecutivamente. Usando as funcionalidades das classes modelo de STL, escreva, da forma o mais simples possível (sem preocupações de eficiência de código) uma função-membro que junte as duas filas colocando o resultado na `filaNormal`.

```
void CBanco::juntarFilas()
```