Universidade do Minho	**Escola de Engenharia da Universidade do Minho** Mestrado Integrado em Eng. Electrónica Industrial e Computadores Complementos de Programação de Computadores	**2013/2014** **MIEEIC** **(1º Ano)** **2º Sem**
Docentes: Luís Paulo Reis, Pedro Pimenta e C. Filipe Portela		
Exame - Época de Recurso - Data 04/07/2014, Duração 1h45m+15min (com consulta)		

Nome:_____ Nº Aluno:_____

Responda às seguintes questões, preenchendo a tabela com a **opção correta (em maiúsculas) (Correto: x Val / Errado: -x/3 Val).** Suponha que foram realizados as inclusões das bibliotecas necessárias e "using namespace std;".

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

GRUPO I (6 Valores)

1. Considere o seguinte fragmento de código:

```
ifstream fich ("f1.txt");
if (!fich) {cerr << msg1 << endl; return 1; }
do //(...)
while (!fich.eof())
cout << msg2 << endl; return 0;
```

A) As mensagens apropriadas à situação são: msg1: "Não se consegue abrir o ficheiro!" e msg2: "Não se consegue fechar o ficheiro!".
B) As mensagens apropriadas à situação são: msg1: "Não se consegue criar o ficheiro!" e msg2: "O ficheiro não existe!".
C) O ficheiro f1.txt vai poder ser usado tanto para saída como para entrada de dados
D) Na linha da msg2 devia estar cerr em vez de cout, porque fich.eof() é um teste de erro.
E) Nenhuma das anteriores

2. A passagem de parâmetros na função seguinte é por referência: `void f(int & a, float & b);`

A) Isso quer dizer que a e b vão ser certamente modificados no código.
B) Isso quer dizer que a função f(), ao executar, pode afetar os valores a e b.
C) Isso quer dizer que a e b não vão ser afetados na execução da função f().
D) A função f() só poderia afetar a e b se fosse declarada assim: `void f(int * a, float * b);`
E) Nenhuma das anteriores.

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

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

A) Como resultado da execução do código, poderá obter-se no ecrã: 1 2 3 4 5
B) Como resultado da execução do código, poderá obter-se no ecrã: 0 1 2 3 4
C) Como resultado da execução do código, poderá obter-se no ecrã: 5 4 3 2 1;
D) Como resultado da execução do código, poderá obter-se no ecrã: 4 3 2 1 0;
E) Nenhuma das anteriores.

4. O resultado da operação top() realizada numa stack após a sequência de operações: `push(2); push(10); pop(); top(); push(5); pop();` é:

A) 2
B) 10
C) 5
D) Dá erro pois a *stack* está vazia.
E) Nenhuma das opções anteriores está correta

5. A diferença entre uma classe e um objeto é que:

A) Todas as classes são pré-definidas na linguagem e os objetos são definidos pelo utilizador
B) Os objetos são sempre criados a pedido do utilizador com a instrução new
C) As classes permitem ao utilizador definir novos tipos de dados e criar objetos por instanciação de uma classe
D) Classe e objeto são duas designações para a mesma coisa
E) Classes têm dados e métodos enquanto os objetos só têm os métodos da classe.

6. Considere que str1 e str2 são duas variáveis do tipo *string*. Quando utiliza a expressão: *str1 += str2* , está a:

A) Comparar os endereços das duas *strings*;
B) Comparar lexicograficamente o conteúdo das duas *strings*;
C) Colocar em str1 a soma aritmética do conteúdo de str1 com str2;
D) Colocar em str1 o resultado de concatenar str1 com str2;
E) Nenhuma das opções anteriores está correta.

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

```
vector<int> v1; int n; cin >> n;
for(int i=1; i<n; i++)
   for(int j=1; j<n; j+=10) {
       cout << i << " - " << j << endl;
   }
```

A complexidade temporal do fragmento de código é:

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

8. Um programador inexperiente escreveu o seguinte código muito mal indentado:

```
if (n < 10) if (n > 0) cout << "Op1" << endl;
else cout << "Outro numero" << endl;
```

Indique qual das seguintes afirmações é verdadeira:
A) Se a variável n assumir o valor 8, no ecrã será escrito “Op1”
B) Se a variável n assumir o valor 8, no ecrã será escrito “Op2”
C) Se a variável n assumir o valor 15, no ecrã será escrito “Op1”
D) Se a variável n assumir o valor 15, no ecrã será escrito “Op2”
E) Nenhuma das opções anteriores está correta.

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

```
vector<int> v1; int n; cin >> n;
for(vector<int>::iterator it = v1.begin();
    it!=v1.end(); it++) v1.push_back(10);
for(int i=0; i<n; i++)
   for(int j=0; j<5 ; j++)
      for(int k=1; k<n ; k = k*2)
         cout << *it << " ";
```

A complexidade temporal do fragmento de código é:
A) A complexidade temporal é T(n)=O(n^3).
B) A complexidade temporal é T(n)=O(n*log(n)).
C) A complexidade temporal é T(n)=O(n^2*log(n)).
D) A complexidade temporal é T(n)=O(n^2).
E) Nenhuma das anteriores.

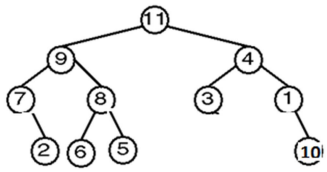
10. A pesquisa binária é tipicamente mais rápida do que a pesquisa sequencial.
A) A pesquisa binária só é mais rápida se o vetor não estiver ordenado caso contrário é melhor pesquisar sequencialmente.
B) A pesquisa binária só é mais rápida para vetores pequenos (menos de 1000 elementos).
C) Normalmente é duas vezes mais rápida pois divide em cada iteração o vetor em duas partes.
D) A pesquisa binária é mais rápida desde que o vetor esteja ordenado e tenha um número razoável de elementos.
E) A pesquisa binária não é aplicável para pesquisar vetores que estejam ordenados.

11. Afirmar que um algoritmo possui complexidade temporal constante significa que:
A) O tempo de execução do algoritmo é o mesmo em qualquer computador
B) O tempo de execução do algoritmo é independente do tamanho dos dados de entrada
C) O tempo de execução do algoritmo é igual ao tamanho dos dados de entrada a multiplicar por uma constante
D) O tempo de execução do algoritmo é linearmente dependente do tamanho dos dados de entrada
E) Nenhuma das anteriores

12. Qual é o método de ordenação preferível para vetores de média/grande dimensão?
A) Ordenação por Partição (QuickSort) pois tem complexidade temporal O(n*log n);
B) Ordenação por Inserção pois tem complexidade temporal o(n^2);
C) Ordenação por MergeSort pois tem uma complexidade temporal muito melhor que o Quicksort;
D) Ordenação por QuickSort pois o processo de ordenação deste método é muito rápido quick devido a gastar menos espaço de memória/disco que os restantes métodos;
E) Nenhuma das Anteriores

13. O método de ordenação por bubble sort:
A) Tem complexidade temporal O(n*log(n)) e espacial O(n).
B) Tem complexidade temporal O(n*log(n)) e espacial O(1).
C) Tem complexidade temporal O(n^2) e espacial O(1).
D) Tem complexidade temporal O(n^2) e espacial O(n).
E) Nenhuma das anteriores.

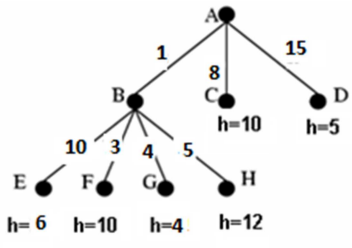
14. Supondo a seguinte árvore, indique o que é escrito no écran executando a função trav passando-lhe com parâmetro o nó raiz (topo) da árvore.




```
void trav(TNode *N) {
   if(N!=0) {
      cout << N->value() << " ";//imprime valor do nó
      trav(N->right());
      trav(N->left());
   }
}
```

A) 11 4 1 10 3 9 8 5 6 7 2
B) 11 4 1 10 3 9 8 5 6 7 2
C) 2 7 6 5 8 9 10 1 3 4 11
D) 5 6 8 2 7 9 10 1 3 4 11
E) Nenhuma das anteriores.

15. Supondo a seguinte árvore de pesquisa em que cada arco apresenta o custo do operador correspondente e h é uma função heurística que estima um custo para a solução, diga qual o nó expandido em seguida utilizando cada um dos seguintes métodos: i) Pesquisa em largura; ii) Pesquisa Gulosa;



A) i) Nó C ii) Nó G
B) i) Nó C ii) Nó E
C) i) Nó E ii) Nó G
D) i) Nó E ii) Nó E
E) Nenhuma das anteriores.

16. Diga quais os nós expandido em seguida utilizando respetivamente a pesquisa i) Custo Uniforme e a ii) A*.
A) i) Nó F ; ii) Nó F
B) i) Nó F ; ii) Nó G
C) i) Nó C ; ii) Nó F
D) i) Nó C ; ii) Nó G
E) Nenhuma das anteriores.

17. Supondo o seguinte código indique o que é escrito no écran.

```
class Base {
 public:
  int m;
  Base(int n=0): m(n) { cout << "Bas! ";}
};
class Derived: public Base {
 public:
  double d;
  Derived(double de=0.0):d(de){ cout << "Der! ";}
};
int main()
{
  Derived cDerived;
  cout << "Bas! ";
  Base cBase;
  cout << "Der! ";
}
```

A) Der! Bas! Bas! Der!
B) Bas! Bas! Der! Bas! Der!
C) Bas! Der! Bas! Der!
D) Bas! Der! Der! Bas!
E) Nenhuma das Anteriores

GRUPO II (8 Valores)

Considere a classe CMaquinaDeTrocos, usada para guardar moedas e notas de diferentes valores e dar troco aos utilizadores. Esta classe inclui informação sobre: i) o identificador único da máquina; ii) o número de moedas de 1 euro disponíveis; iii) o número de moedas de 2 euros disponíveis e; iv) o número de notas de 5 euros disponíveis. ma implementação incompleta da classe CMaquinaDeTrocos apresentada a seguir:

```
class CMaquinaDeTrocos
{
  static int nDeMaquinas; //numero total de máquinas criadas
  int identificador;   // identificador igual à ordem de criação dos objetos do tipo começando em 1
  int nMoedas1Euro;
  int nMoedas2Euros;
  int nNotas5Euros;
  int dinheiroTotal;   //soma pesada das moedas e notas
public:
  CMaquinaDeTrocos(){};
  CMaquinaDeTrocos(int, int, int);  // a implementar
  void actualizaMaquina(int, int, int);  // a implementar
  int daTroco(int);   // a implementar
  bool temMoedasSuficientes(int, int);   // a implementar
  void guardaMaquina(const char *);   // a implementar
  int leMaquina(const char *);   // a implementar
  void printMaquina(){ cout << identificador << " " << nMoedas1Euro << " " << nMoedas2Euros
                            << " " << nNotas5Euros << " = " << dinheiroTotal << endl; }
  int  getIdentificador() const { return identificador;}
  int  getMoedas1Euro() const { return nMoedas1Euro;}
  int  getMoedas2Euros() const { return nMoedas2Euros;}
  int  getNotas5Euros() const { return nNotas5Euros;}
  int  getTotal() const { return dinheiroTotal;}
  void setIdentificador(int id) {identificador = id;}
  void setMoedas1Euro(int m1eu) {nMoedas1Euro = m1eu;}
  void setMoedas2Euros(int m2eu) {nMoedas2Euros = m2eu;}
  void setNotas5Euros(int n5eu) {nNotas5Euros = n5eu;}
  void setTotal(int tot){ dinheiroTotal = tot;}
};
int CMaquinaDeTrocos::nDeMaquinas=1;
```

2.1) Implemente o construtor da classe CMaquinaDeTrocos que recebe 3 argumentos: o número de moedas de 1 euro; o número de moedas de 2 euros, o número de notas de 5 euros. O construtor deve atualizar o número de máquinas, o identificador e o dinheiro total também

2.2) Implemente um membro-função actualizaMaquina(int m1eu, int m2eu, int n5eu) que acrescenta moedas e notas à máquina e atualiza o dinheiro total.

2.3) Implemente o membro-funcão temMoedasSuficientes() que recebe 2 argumentos: 1) o tipo de moeda/nota a usar no troco (1 para troco só com moedas de 1 euro; 2 para troco só com moedas de 2 euros e; 5 para o troco só com notas de 5 euros); 2) o montante do troco a dar. A função devolve false se não houver moedas suficientes da qualidade especificada para dar o montante especificado. Devolve true no caso de haver quantidade suficiente do tipo de moeda/nota escolhido para satisfazer o montante necessário.

2.4) Implemente o membro-função daTroco() que escolhe a combinação de moedas que constituem o troco cujo valor é recebido como parámetro. O algoritmo usado para dar o troco segue os seguintes passos: 1) dá o maior número de notas de 5 euros e se faltar dinheiro preencha o valor em falta com 2). o maior número de moedas de 2 euros e, se faltar dinheiro preencha o valor em falta com // 3) moedas de 1 euro. A função devolve num inteiro o número de notas de 5 euros (dígito das centenas), o número de moedas de 2 euros (dígito das dezenas) e de 1 euro (dígito das unidades). Exemplo: chamando com daTroco(13) devolve 211. Nota: Assuma que existem sempre quantidades de notas e moedas suficientes para dar o troco e que o número de notas/moedas de cada tipo a dar de troco é sempre inferior a 10 (representável num único dígito).

2.5) Implemente o membro-função guardaMaquina() que guarda em quatro linhas consecutivas do ficheiro o identificador da máquina, o número de moedas de 1 euro, o número de moedas de 2 euros e o número de notas de 5 euros.

2.6) Implemente o membro-função `int leMaquina(const char *)` que lê do ficheiro cujo nome é passado como parâmetro um conjunto de máquinas, cada qual representada em 4 linhas consecutivas com: identificador da máquina; número de moedas de 1 euro; número de moedas de 2 euros e; número de notas de 5 euros. A função deve selecionar, de entre as máquinas lidas aquela com maior montante total e armazenar os valores lidos dessa máquina nos membros-dado correspondentes. A função devolve o montante da máquina.

2.7) Implemente uma função para criar máquinas de trocos `void CMaquinaDeTrocos::criaMaquinas(int nM1[], int nM2[], int nM5[], int n)` que recebe três vetores com n inteiros cada e cria as respetivas máquinas de trocos, gravando-as ainda em ficheiros com nomes maquinaID.txt em que ID é o identificador dado a cada maquina

2.8) implemente o programa principal de teste de todas as funções anteriores

GRUPO III (6 Valores)

3.1) Implemente os seguintes membros-função para a classe *Queue* (fila de inteiros) descrita nos slides/aulas da disciplina.

a) `void retira_nX12(int n, int x1, int x2)` que retira um máximo de n elementos da fila desde que tenham valores compreendidos entre x1 e x2 (inclusive). No caso de encontrar um valor distinto, não o deve retirar e deve terminar a sua execução.

b) `int da_a_volta_maior()` que elimina o maior dos três elementos que se encontram no topo da fila e introduz esse elemento na cauda da lista. Devem ser considerados os casos em que a lista tem menos de três elementos ou está mesmo vazia. A função deve retornar o elemento que "deu a volta" ou 0 caso a fila esteja vazia.

3.2) Na classe *List* descrita nos slides/aulas da disciplina implemente o método: `int escreve_sequencias(ostream &os);` Este método percorre a lista e sempre que determinar uma sequência de elementos com valores consecutivos, escreve essa sequência. Exemplo (supondo uma lista com [3,5,6,7,8,1,2,4,5,6,8,10,12,2,3]. A função escreve: Seq 5 6 7 8 Seq: 1 2 Seq; 4 5 6 Seq 2 3 . No final, a função deve retornar o número de sequências encontrado (neste caso 4)

3.3) Suponha a classe *BTree* descrita nos slides/aulas da disciplina.

a) Implemente o método `int contaEntre(int x, int &folhas, int &folhasX);` que conte o número de elementos da árvore que são maiores que x. A função deve escrever no écran todos os elementos nestas condições e retornar a sua contagem. Deve ainda retornar no parâmetro folhas o número de folhas da árvore e no parâmetro folhasX, o número de folhas maiores que x.

b) Construa uma função `int eliminar_menores();` que supondo uma árvore *BTree* elimine todos os filhos de elementos que têm valor inferior ao respetivo pai/antecessor. Quando encontrar um elemento nestas condições a função deve eliminar o filho e todos os respetivos filhos desse filho (i.e. toda a sub-árvore respetiva).