Nome da/do estudante: _____

***NOTA**: nas questões que envolverem alterações de código, procure minimizar o número de instruções alteradas e indique as alterações usando os números das linhas que surgem na margem esquerda do código fornecido.*

1. [2.8 valores = 0.8 + 0.8 + 0.6 + 0.6]

Um estudante que queria desenvolver um programa para calcular a sua classificação média escreveu o código seguinte:

```
 1  int main()
 2  {
 3    int n=1, grade, sum=0;
 4    float average;
 5    cout << "Grade " << n << "? "; cin >> grade;
 6    while (0 <= grade <= 20)   // gama de classificações possíveis é [0..20]
 7    {
 8      sum = sum + grade;
 9      n++;
10      cout << "Grade " << n << "? "; cin >> grade;
11    } ;
12    average = sum / n;
13    cout << "Average = " << fixed << setprecision(1) << average << endl;
14    return 0;
15  }
```

a) [0.8] Quando o estudante executou o programa pela primeira vez, verificou que não era era possível terminá-lo como ela esperava, isto é, inserindo uma classificação fora da gama possível: [0..20]. Corrija o código para resolver este problema.

Correção: `while (0 <= grade && grade <= 20)`

Explicação para o facto de o ciclo não terminar (não necessária na resposta): a instrução `while (0 <= grade <= 20)` está sintaticamente correta! É equivalente a `while ((0 <= grade) <= 20)`; a 1ª comparação, `0 <=grade`, tem como resultado 0 ou 1 (é assim que é representado false e true, na linguagem C/C++), pelo que a comparação seguinte será `0 < 20` ou `1 < 20`, que será sempre true; daí o ciclo nunca terminar. Alguns compiladores assinalarão a incorreção com um "warning".

b) [0.8] Depois de corrigir o erro acima indicado, o estudante correu outra vez o programa. Inseriu a sequência de números, **10** <enter> **15** <enter> **11** <enter> **-1** <enter>, tendo obtido a saída "**Average = 9.0**" (em vez do resultado esperado, 12)! Explique o que aconteceu e indique a correção necessária para evitar este erro.

O valor de 'n' usado no cálculo de 'average' é superior em 1 unidade ao que devia ser.

Correção: `average = sum / (n-1);`

c) [0.6] Após corrigir o erro anterior, o estudante executou o programa diversas vezes e verificou que a parte decimal do resultado era sempre zero, mesmo em situações em que isso não era esperado, como quando a sequência de valores introduzidos foi: **10**, **15**, **12**, **-1** (a média é 12.3 mas a saída foi 12.0). Explique o que aconteceu e corrija o código para evitar este erro.

Os operandos da operação `"sum/(n-1)"` são ambos inteiros, logo é feita uma 'divisão inteira'.

Correção: converter um dos operandos para 'float', já que 'average' é do tipo 'float'
→ `average = (float) sum / (n-1);`

d) [0.6] Numa das execuções do programa, o primeiro valor digitado foi **1o** (o dígito **1**, seguido da letra minúscula **o**) em vez do número **10**. Apresente, de forma aproximada, o que surgiu no terminal, explique sucintamente a causa e aponte uma solução (indique apenas o que é necessário fazer, sem escrever código).

O que surgiu no terminal … (aprox.)	Explicação breve:
Grade 1? **1o** Grade 2? Grade 3? Grade 4? Grade 5? … (nunca mais pára)	Na 1.a iteração, o dígito 1 foi atribuído a 'grade'; o 'o' ficou no *buffer* do teclado. A segunda leitura de 'grade' falhou porque o 'o' é um caráter inválido para um inteiro; a 'input stream' ficou num estado de erro e, daí em diante, todas as tentativas de leitura dão erro. Solução: 'limpar' o buffer de entrada, para retirar o 'o' que lá ficou – cin.ignore() - e limpar o estado de erro da 'input stream' – cin.clear()

2. [2.0 valores = 0.8 + 0.6 + 0.6]

Considere a função **swapElems** (à esquerda) e o código usado para testá-la (à direita):

```
 1  bool swapElems(vector<int> v, int i, int j)
 2  {
 3    if ( i>=0 && j<v.size() )      (*)
 4    {
 5      int t = v[i];
 6      v[i] = v[j];
 7      v[j] = t;
 8      return true;
 9    }
10    else
11      return false;
12  }
```

```
13  int main()
14  {
15    vector<int> v1(5);
16    for (int i=0; i<v1.size(); i++) v1[i]=i;
17    if ( swapElems(v1,1,3) )
18      for (int i=0; i<v1.size(); i++)
19        cout << v1[i] << endl;
20    else
21      cout << "Invalid swap !\n";
22    return 0;
23  }
```

a) [0.8] Um programador, ao testar o código, ficou surpreendido porque, apesar de não ter recebido a mensagem `"Invalid swap !"`, o conteúdo do vetor permaneceu inalterado. Explique o que aconteceu e indique a correção necessária para este problema. (*) *O teste correcto seria* `(i>=0 && i<v.size() && j>=0 && j<v.size())` *mas isso não influencia a resp.*

O vector foi passado 'por valor'. Apesar de ter sido feita uma troca de elementos, ela foi feita sobre uma cópia do vector original que, por isso, permaneceu inalterado.

Correção: passar o ' vector por referência' →
`bool swapElems(vector<int> &v, int i, int j)`

b) [0.6] Considere que pretende "sobrecarregar" (*overload*) a função **swapElems()** desenvolvendo uma função com uma funcionalidade semelhante que opera sobre um *array* (de **C**) em vez de um vetor. O código desta função pode ser obtido através de pequenas alterações ao código de **swapElems()** fornecido. Indique essas alterações.

```
1: bool swapElems(int v[], int numElems, int i, int j)

3: if ( i>=0 && j<numElems )
```

c) [0.6] Considere que pretende escrever uma função *template* que permita trocar elementos de um vetor contendo qualquer tipo de dados. Indique as modificações que seria necessário efetuar para transformar o código da função **swapElems()** fornecida numa função *template*.

```
template <typename T>
bool swapElems(vector<T> v, int i, int j)
{ ...
  T  t = v[i];
  ...
```

3. [3.2 valores = 0.8 + 0.8 + 0.8 + 0.8]

Considere o seguinte programa (incompleto) e a saída respetiva:

```
int main() {
  Time t1, t2(17,25,43);   // t2 representa 17h 25m 43s

  t1.show(); cout << endl;
  t2.show(); cout << endl;

  return 0;
}
```

saída:

0:0:0
17:25:43

a) [0.8] Defina a classe **Time** (apenas os atributos e métodos estritamente necessários para compilar o programa).
Nota: não implemente os métodos da classe, por enquanto.

```
class Time {
  public:
    Time();
    Time(int h, int m, int s);
    void show() const;
  private:
    int h, m, s;
};
```

b) [0.8] Implemente os métodos que declarou na classe **Time**.

```
Time::Time {
  h = m = s = 0;
}

Time::Time(int h, int m, int s) {
  this->h = h;
  this->m = m;
  this->s = s;
}

void Time::show() const {
  cout << h << ":" << m << ":" << ":" << s;
}
```

c) [0.8] Considere que pretende substituir as instruções de saída de dados da função **main()** pela instrução:
cout << t1 << endl << t2 << endl;
Indique as alterações necessárias ao código que apresentou anteriormente por forma a que o compilador não assinale erros de compilação.

```
É preciso fazer o "overload" do operador  <<.
Uma vez que não estão definidos métodos getXXX() na classe Time,
é necessário declarar este operador como friend da classe:

class Time {
   friend ostream & operator<<(ostream &out, const Time &t)
   public:
    ...
};

... e definir a respetiva função:

ostream & operator<<(ostream &out, const Time &t) {
   out << t.h << ":" << t.m << ":" << ":" << t.s;
   return out;
}

NOTA (não necessária na resposta): teria sido possível outra solução, sem declarar o operator<<
como friend de Time se estivessem definidos métodos getH(), getM() e getS() na classe Time; mas o
que se pretendia era a solução que implicasse menor nº de alterações do código, como dito no início
```

d) [0.8] Considere que pretende registar as atividades que executa durante o dia, o tempo em que iniciou cada atividade e a respetiva duração. Com base na classe **Time** defina a(s) estrutura(s) de dados que usaria para fazer esse registo, tendo em conta que: as atividades devem ser indexadas pela sua descrição (ex: "exame de PROG", "almoço", "jogging", ...); há algumas atividades que podem ser executadas mais do que uma vez por dia (ex: "jogging").

```
Tendo em conta que se pretende indexar as atividades pela sua descrição (por uma string)
e que esta pode estar repetida, usaria um multimap, cuja chave seja uma string.

O valor associado a cada chave deve representar o início e a duração de cada atividade (2 tempos).

Para isso usaria uma struct:
struct Activity {
  Time begin, duration;
};

A estrutura a usar seria pois:

  multimap<string,Activity> act;


NOTA: também seria aceite uma resposta em que 'duration' fosse, por exemplo, um inteiro (nº de
minutos que durou a atividade)
```

Nome da/do estudante: _____

4. [6 valores = 1 + 2 + 1 + 2]

Considere um jogo de lotaria onde cada aposta contém 5 números, de 1 a 45, mais 2 números, de 1 a 9. As apostas são registadas num ficheiro de texto, uma aposta por linha, ordenada por ordem crescente dos números da aposta que estão separados por espaços. Por exemplo, o ficheiro conterá algo como:

```
1 7 21 36 37 2 5
3 8 14 25 40 1 8
20 24 32 40 45 2 3
```

a) [1] Escreva o código da função **`fileOpen()`** que tenta abrir um ficheiro com apostas. A função recebe como parâmetro o nome do ficheiro (**`filename`**) e, se a abertura tiver sucesso, devolve através de outro parâmetro (**`f`**) a *file stream* correspondente. O valor de retorno será **`true`** caso a abertura tenha sucesso ou **`false`** no caso contrário.

```
bool fileopen(string filename, ifstream &f) {

  f.open(filename.c_str());
  return f.is_open();

}
```

b) [2] Escreva uma função que tem como parâmetros o nome de um ficheiro contendo apostas, no formato anteriormente indicado, e um **`vector<vector<int>>`**. A função deve preencher o vetor com as aposta lidas do ficheiro. Caso o ficheiro contenha apostas incompletas (com menos de 7 números) deve ser apresentada no ecrã uma mensagem, por cada aposta, indicando o número da aposta em que isso aconteceu (a 1ª aposta tem o número 1). Apenas devem ser guardadas no vetor as apostas completas. A função retorna **`true`** no caso de leitura sem erros e **`false`** no caso contrário. Use a função da alínea anterior para abrir o ficheiro.

```
bool getBets(string filename, vector<vector<int> > &vec)
{
  int contador=1;
  bool erro;
  ifstream f;
  stringstream ss;
  string line;
  vector<int> aposta;

  if(! fileopen(filename, f))
    return false;

  while(! f.eof()){
    getline(f, line);
    if(! line.size()) break;

    stringstream ss(line);

    aposta.clear();
    while(! ss.eof()){
      int n;
      ss >> n;
      if(n > 0) aposta.push_back(n);
    }
    if(aposta.size() != 7){
      cout << "Erro na aposta " << contador++ << endl;
      erro = false;
    }
    else
      vec.push_back(aposta);
  }

  f.close();

  return erro;









}
```

c) [1] Escreva uma função que recebe por referência um **vector** contendo uma aposta e a apresenta no formato exemplificado a seguir (do lado direito). Cada linha acomoda 9 números.

Matriz com todos os números:

```
 1   2   3   4   5   6   7   8   9
10  11  12  13  14  15  16  17  18
19  20  21  22  23  24  25  26  27
28  29  30  31  32  33  34  35  36
37  38  39  40  41  42  43  44  45
```

```
 1   2   3   4   5   6   7   8   9
```

Exemplo de como a aposta {1,7,21,36,37,2,5} é apresentada:

```
 1                   7

       21

                          36
37

     2         5
```

```
void printBet(const vector<int> &aposta)
{
  int index = 0;

  for(unsigned int i = 1; i < 46; i++){
    if(index < 5 && aposta[index] == i){
      cout << setw(2) << i << " ";
      index++;
    }
    else cout << "   ";
    if((i % 9) == 0)
      cout << endl;
  }

  cout << endl << endl;

  for(unsigned int i = 1; i < 10; i++)
    if(aposta[index] == i){
      cout << setw(2) << i << " ";
      index++;
    }
    else cout << "   ";

  cout << endl;




}
```

d) [2] Escreva uma função que recebe um **vector** com uma aposta e outro **vector** com a chave, no mesmo formato, e calcula a pontuação da aposta, escrevendo-a no formato X+Y, como exemplificado a seguir.

Exemplo: aposta - {1, 7, 21, 36, 37, 2, 5} ; chave - {1, 5, 27, 36, 40, 3, 5}; saída da função - **pontos: 2+1**

```
void score(const vector<int> &aposta, const vector<int> &chave){
  unsigned int certas = 0;

  for(unsigned int i=0; i < 5; i++)
    for(unsigned int j = 0; j < 5; j++){
      //      cout << aposta[i] << "  " << chave[j] << endl;
      if(aposta[i] == chave[j])
        certas++;
      else if (aposta[i] < chave[j])
        break;
    }
  cout << certas << " + ";
  certas=0;
  for(unsigned int i=5; i < 7; i++)
    if(aposta[i] == chave[5] || aposta[i] == chave[6])
        certas++;

  cout << certas << endl;









}
```

Nome da/do estudante: _____

5. [6 valores = 1 + 1.5 + 1.5 + 2]

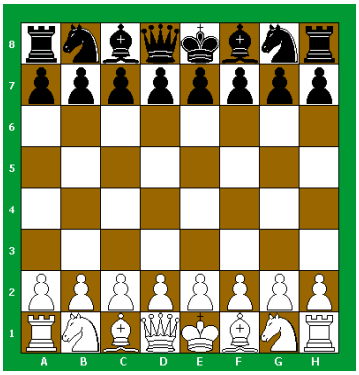
De modo a implementar um jogo de xadrez, um programador em C++ desenvolveu, entre outras, as classes **Peca** e **Tabuleiro**. Uma definição parcial destas classes é mostrada abaixo. Um tabuleiro de xadrez é codificado por uma matriz (8 x 8) de **Peca**. Uma peça é caraterizada pelo seu **nome** ("Rei", "Rainha", "Torre", "Bispo", "Cavalo" ou "Peao") e pela sua **cor**, um *booleano* cujo valor é **true** para as peças brancas e **false** para as pretas. As posições do tabuleiro que não têm peça devem conter uma "peça vazia", isto é, um objeto do tipo **Peca** em que o campo **nome** é preenchido com a *string* vazia e o campo **cor** pode ter qualquer valor.

A classe **Peca** representa uma peça de um tabuleiro de xadrez:

```
class Peca {
public:
  Peca();                                  // constrói uma "peça vazia"
  Peca(string nome, bool cor);
  string getNome() const;                  // retorna o nome de Peca
  bool getCor() const;                     // retorna a cor de Peca
  void setPeca(string nome, bool cor);     // atualiza os atributos de Peca
private:
  string nome; // o nome da peça pode ser: "Rei", "Rainha", "Torre", "Bispo", "Cavalo", ou "Peao"
               // a ausência de uma peça é assinalada usando a string vazia no nome
  bool cor;    // true para as peças brancas; false para as peças pretas
};
```

A classe **Tabuleiro** é usada para guardar informação sobre o estado do jogo:

```
class Tabuleiro {
public:
  Tabuleiro(); // reserva espaço para um tabuleiro (matriz 8 x 8) e preenche
               // todas as posições com uma "peça vazia" (ver texto acima)
  string getPecaNaPos(string pos) const; //retorna nome da peça na pos. dada
  vector<Peca> torrePodeTomar(unsigned int lin, unsigned int col,
                              bool cor) const; // retorna os nomes das
          // peças que podem ser tomadas pela torre cuja posição e cor é dada
  void pecasTomadas() const; //mostra nome das peças tomadas por cada jogador
private:
  vector< vector<Peca> > tabuleiro;  // tabuleiro de jogo
  set<string> pretasTomadas;  // peças pretas tomadas pelas brancas
  set<string> brancasTomadas; // peças brancas tomadas pelas pretas
};  //nota pós-exame: deviam ser multisets; são sets, apenas para simplificar
```



a) [1] Implemente o construtor da classe **Tabuleiro** que:
i) reserva espaço para o tabuleiro (matriz de 8 x 8 elementos) de **Peca** e
ii) preenche todas as posições do tabuleiro com uma "peça vazia".

```
#define DIMENSAO 8


Tabuleiro::Tabuleiro() {

  for(unsigned int i=0; i < DIMENSAO; i++){
    vector<Peca> linha;
    for(unsigned int j=0; j < DIMENSAO; j ++)
      linha.push_back(Peca());
    tabuleiro.push_back(linha);
  }




}
```

b) [1.5] Escreva a função **getPecaNaPos**() que recebe como parâmetro uma *string* com umas coordenadas em "xadrez standard", isto é, constituída por uma letra e um dígito (veja o sistema de coordenadas da figura) e retorna o nome da peça guardada nessa posição ou a *string* vazia se não houver lá nenhuma peça. Exemplo: considerando a posição inicial do jogo tal como é mostrado na figura, o valor retornado pela função será "Rei" se o parâmetro recebido for "E1", "Torre" se o parâmetro for "H8", ou "" (*string* vazia) se o parâmetro for "E5".

```
string Tabuleiro::getPecaNaPos(string pos) const {
  int linha, coluna;
  linha = DIMENSAO - (pos[1] -'0');
  coluna = pos[0] - 'A';

  return tabuleiro[linha][coluna].getNome();









}
```

c) [1.5] Escreva a função **torrePodeTomar()** da classe **Tabuleiro** que retorna a lista de peças que a torre cuja posição e cor são dados como parâmetros pode tomar. Uma torre pode movimentar-se para qualquer posição da mesma linha ou coluna em que está, com as seguintes restrições: uma torre não pode saltar nenhuma peça; uma torre de uma cor só pode tomar peças da outra cor. Os parâmetros de entrada são a linha e coluna da matriz tabuleiro em que está a torre e a sua cor.

```
vector<Peca> Tabuleiro::torrePodeTomar(unsigned int lin, unsigned int col, bool cor) const {
  vector<Peca> tomadas;

  // verificar na coluna para cima
  for(int l=lin-1; l >= 0; l--)
    if(tabuleiro[l][col].getNome() != ""){
      if(tabuleiro[l][col].getCor() != cor){
        tomadas.push_back(tabuleiro[l][col]);
        break;
      }
      else break;
    }
  // verificar na coluna para baixo
  for(int l=lin+1; l < DIMENSAO; l++)
    if(tabuleiro[l][col].getNome() != ""){
      if(tabuleiro[l][col].getCor() != cor){
        tomadas.push_back(tabuleiro[l][col]);
        break;
      }
      else break;
    }
  // verificar na linha para a esquerda
  for(int c=col-1; c >= 0; c--)
    if(tabuleiro[lin][c].getNome() != ""){
      if(tabuleiro[lin][c].getCor() != cor){
        tomadas.push_back(tabuleiro[lin][c]);
        break;
      }
      else break;
    }
  // verificar na linha para a direita
  for(int c=col+1; c < DIMENSAO; c++)
    if(tabuleiro[lin][c].getNome() != ""){
      if(tabuleiro[lin][c].getCor() != cor){
        tomadas.push_back(tabuleiro[lin][c]);
        break;
      }
      else break;
    }

  return tomadas;
}
```

d) [2] Escreva a função **pecasTomadas()** que mostra no ecrã o nome das peças tomadas tanto pelas brancas como pelas pretas. Os dois conjuntos de nomes devem ser separados por uma linha em branco.

```
void Tabuleiro::pecasTomadas() const {
  set<string>::iterator it;

  cout << "Tomadas pelas pretas: ";
  for(it = brancasTomadas.begin(); it != brancasTomadas.end(); it++)
    cout << *it << " ";
  cout << endl;
  cout << "Tomadas pelas brancas: ";
  for(it = pretasTomadas.begin(); it != pretasTomadas.end(); it++)
    cout << *it << " ";
  cout << endl;
}
```

FIM