**EEL670 - Linguagens de Programação**
**Departamento de Engenharia Eletrônica e de Computação - DEL-Poli/UFRJ**
**Prof.: Miguel Elias Mitre Campista**
**== Laboratório 7**

---

**1) Lista de exercícios:** Resolva os exercícios abaixo como se pede.

a) Escreva um programa para o jogo da velha. A ideia é criar um jogo entre um usuário humano, que escolhe a posição no tabuleiro para marcar um 'x' e o computador que marca sempre 'o' em uma posição aleatória desocupada. Para isso, crie uma classe `Velha` que programa o tabuleiro como uma matriz de caracteres inicializada através de alocação dinâmica de memória (use o operador `new`). Para isso, programe a matriz através de um ponteiro para um array de ponteiros de caracteres (`char **matriz`). Todos os elementos da matriz devem ser inicializados com o caractere '-'.

A classe `Velha` deve solicitar os índices na matriz como sendo a jogada do usuário. A partir dos índices, a classe deve verificar se a posição é válida e se essa posição já foi usada. Ainda, a classe `Velha` deve verificar se já houve vencedor ou se deu velha. A função principal abaixo ajuda na construção da classe `Velha`.

```
/*****************************************************************************/
int main () {
    enum jogador {USUARIO, COMPUTADOR};
    jogador vencedor;
    bool terminou = false;

    Velha velha;
    velha.imprime();

    while(!terminou) {
            int i, j;

            // Enquanto a posição não for válida, repete
            do {
                    cout << "Entre com a posição (i, j):";
                    cin >> i >> j;
            } while (!velha.usuarioJoga(i,j));

            velha.imprime();
            terminou = velha.verificaVencedor();

            if (terminou) {
                    vencedor = USUARIO; // Usuário venceu
            } else {
                    cout << "Computador joga...." << endl;
                    velha.computadorJoga();
                    velha.imprime();
                    terminou = velha.verificaVencedor();

                    if (terminou) vencedor = COMPUTADOR; // Computador venceu
            }
    }

    cout << "Vencedor: " << ((vencedor == 0) ? "Usuário" : "Computador")
         << ". Fim de jogo." << endl;

    return 0;
}
/*****************************************************************************/
```

b) Na função principal abaixo, a classe `Proxy` é utilizada como interface da classe `MinhaClasse` para o programador da função principal. Sendo assim, construa a classe `Proxy` levando em conta que o programador terá apenas os arquivos `main.cpp`, `proxy.h`, `proxy.o` e `minhaclasse.o` para gerar o executável.

O arquivo `proxy.h`, contém a definição da classe `Proxy`. Tenha em vista o uso do objeto da classe `Proxy` na função principal e a definição da classe `MinhaClasse`.

Lembre-se que a classe `Proxy` possui um ponteiro alocado dinamicamente para um objeto da classe `MinhaClasse` como atributo privado. Já o arquivo `proxy.cpp` contém a implementação de todos os métodos da classe `Proxy` definidos no arquivo `proxy.h`.

```
/***** Função Principal (main.cpp) *****/
#include <iostream>
#include "proxy.h"
using namespace std;

int main() {
  Proxy p;
  cout << p.getNome () << "\t"
       << p.getIdade () << endl;

  p.setNomeIdade ("Joao das Coves", 20);

  cout << p.getNome () << "\t"
       << p.getIdade () << endl;

  return 0;
}
```

```
/****** Saída na tela do programa *****/
None              0
João das Coves    20
```

```
/******* minhaclasse.h ******/
#include <iostream>
#include <string>

using namespace std;

class MinhaClasse {
    friend class Proxy;
    public:
           MinhaClasse (string, int);

    private:
            string nome;
            int idade;
};
```

**2) Programa para entrega dia 01/11/2024:** A entrega do programa será através do Google Classroom e consiste da devolução em único arquivo zip ou rar de todos os arquivos referentes ao código-fonte, um Makefile e um arquivo README que documente a utilização do programa. Todos os arquivos serão avaliados.

Escreva um programa que implemente uma classe `Disciplina` para gerenciamento de disciplinas da faculdade. A disciplina é definida a partir de uma `struct` (não é classe) contendo uma `string` com o nome da disciplina, uma `string` com o período de conclusão (p.ex., "2023.1"), o número de créditos da disciplina e um `double` para armazenar a nota final da disciplina. Já a classe `Historico` implementa uma estrutura do tipo `vector` para armazenar as disciplinas, além de um tamanho máximo para o número de disciplinas. As diferentes ações permitidas com o objeto da classe `Historico` devem usar operadores como se segue:

- **Impressão do catálogo inteiro de disciplinas cursadas e de uma única disciplina na tela:** devem ser realizadas respectivamente com `cout << historico` e `cout << disciplina`.

- **Inicialização dos dados referentes a uma disciplina:** deve ser feito através de `cin >> disciplina`.

- **Inserção ordenada de uma disciplina no histórico:** deve ser feita com o operador `+=`. Por exemplo, "`histórico += disciplina`" insere uma disciplina ordenada primeiramente pelo período de conclusão e em seguida pelo seu nome. Note que o nome, período de conclusão, número de créditos e nota devem ser inicializados previamente, antes da inserção.

  O operador `<` (ou `>`) deve ser implementado para que a comparação entre disciplinas seja possível. Por exemplo, "`disciplina1 < disciplina2`" deve retornar `true` caso o período de conclusão da `disciplina1` for menor que o da `disciplina2`. Se houver empate, os nomes das disciplinas devem ser avaliados. Ainda, a classe `Historico` não permite a inserção de disciplinas com o mesmo nome. Dessa forma, é importante implementar o operador `==` para verificar se a disciplina a ser inserida tem o mesmo nome de outra já existente. Por exemplo,

"`disciplina1 == disciplina2`" deve retornar `true` se os nomes das disciplinas forem os mesmos e `false`, caso contrário.

A inserção retorna o índice no `vector` do elemento inserido ou -1 caso a inserção não seja realizada.

- **Remoção de uma disciplina do histórico:** deve ser feito através do operador `-=`. Por exemplo, "`histórico -= disciplina`" remove a `disciplina` do histórico. A busca da disciplina deve ser feita a partir do nome da disciplina e a remoção no `vector` pode usar o método `erase`.

  A remoção retorna o índice no `vector` do elemento removido ou -1 caso a remoção não seja realizada. Este último caso pode acontecer se a disciplina não existir no histórico.

- **Busca de todas as disciplinas de um dado período do histórico:** deve ser feita através do operador `()` sobrecarregado. A busca é realizada a partir do período passado por valor, da seguinte maneira: `historico("2024.1")`. A busca retorna o `vector` de disciplinas do período ou um `vector` vazio, caso não existam disciplinas. Este `vector` é usado para exibição na tela do nome das disciplinas e de seus atributos. Dica: use `cout << disciplina`.

- **Edição da nota de uma disciplina no histórico:** deve ser feita através do operador `[]`, sendo que o nome da disciplina deve ser passado para o operador. O nome é usado para a busca da seguinte forma: `historico["nome_disciplina"] = novanota`.

- **Cálculo e exibição do CRA:** deve ser feito com o uso do operador `>>` implementado como um método da classe `historico`. O CRA é um objeto de uma classe que possui como parâmetro privado uma estrutura pra armazenamento do CRA de cada período e um atributo para armazenamento do CRA global. O CRA é calculado como a média das notas de todas as disciplinas no período ou todas do histórico de um aluno ponderada pelos respectivos créditos. O cálculo tanto do CRA por período quanto o global deve ser realizado da seguinte maneira: `historico >> cra`. A classe do objeto CRA possuirá um método de exibição de tanto do CRA por período quanto o global.

***Observação 1:*** *Crie um menu que permita a execução de todas as ações por intermédio da interação com o usuário. É permitido igualmente que as opções sejam passadas para o executável através de argc e argv.*

***Observação 2:*** *Implemente persistência de dados das disciplinas. Toda vez que um histórico é criado, este deve carregar todas as disciplinas já registradas e armazenadas em um arquivo de texto. Antes do encerramento do programa, o arquivo de texto deve ser totalmente atualizado.*

## == Respostas da Lista de Exercícios

### 1)

a)

```
/*************************************************************************/
/************************** Programa Principal ***************************/
#include <iostream>

#include "velha.h"

/* Programa do Laboratório 7:
   Programa de um jogo da velha com alocação dinâmica de memória
   Autor: Miguel Campista */

using namespace std;


int main () {
    enum jogador {USUARIO, COMPUTADOR};
    jogador vencedor;
    bool terminou = false;

    Velha velha;
    velha.imprime();

    while(!terminou) {
            int i, j;

            // Enquanto a posição não for válida, repete
            do {
                    cout << "Entre com a posição (i, j):";
                    cin >> i >> j;
            } while (!velha.usuarioJoga(i,j));

            velha.imprime();
            terminou = velha.verificaVencedor();

            if (terminou) {
                    vencedor = USUARIO; // Usuário venceu
            } else {
                    cout << "Computador joga...." << endl;
                    velha.computadorJoga();
                    velha.imprime();
                    terminou = velha.verificaVencedor();

                    if (terminou) vencedor = COMPUTADOR; // Computador venceu
            }
    }

    cout << "Vencedor: " << ((vencedor == 0) ? "Usuário" : "Computador")
         << ". Fim de jogo." << endl;

    return 0;
}

/*************************************************************************/
/***************************** Arquivo velha.h ***************************/
#include <iostream>
#include <iomanip>
#include <ctime>
#include <cstdlib>

using namespace std;

#ifndef VELHA_H
#define VELHA_H

class Velha {
    public:
            Velha ();
```

```
            // Alocação dinâmica requer programação explícita do destrutor
            ~Velha ();

            void imprime ();

            bool usuarioJoga (unsigned, unsigned);

            // computadorJoga não retorna bool pois não é necessário repetir jogada
            void computadorJoga ();

            bool verificaVencedor ();

    private:
            char ** matriz;
            unsigned posicoesOcupadas;

            bool verificaLimite (unsigned, unsigned);
            bool verificaPosicao (unsigned, unsigned);
            bool verificaVelha ();
};

#endif

/*****************************************************************************/
/***************************** Arquivo velha.cpp *****************************/
#include "velha.h"

Velha::Velha () {
    posicoesOcupadas = 0;
    matriz = new char * [3];
    for (unsigned i = 0; i < 3; i++) {
            matriz [i] = new char [3];
            // Vou inicializar cada uma das posições com '-'
            for (unsigned j = 0; j < 3; j++)
                    matriz [i][j] = '-';
    }
}

Velha::~Velha () {
    for (unsigned i = 0; i < 3; i++) {
            delete [] matriz [i];
    }
    delete [] matriz;
}

void Velha::imprime () {
    for (unsigned i = 0; i < 3; i++) {
            for (unsigned j = 0; j < 3; j++) {
                    cout << setw(3) << matriz [i][j];
            }
            cout << endl;
    }
}

bool Velha::usuarioJoga (unsigned i, unsigned j) {
    if (!verificaLimite(i, j)) {
            cout << "Posição inválida. Jogue novamente!" << endl;
            return false;
    }
    if (!verificaPosicao(i, j)) {
            cout << "Posição já ocupada. Jogue novamente!" << endl;
            return false;
    }

    matriz[i][j] = 'x';
    posicoesOcupadas++;

    if(!verificaVelha()) {
            exit (0);
    }
    return true;
}

void Velha::computadorJoga () {
    unsigned i, j;
    srand(time(0));
```

```
    do {
            i = rand() % 3;
            j = rand() % 3;
    } while (!verificaPosicao(i, j));

    matriz[i][j] = 'o';
    posicoesOcupadas++;

    if(!verificaVelha()) {
            exit (0);
    }
}

bool Velha::verificaVencedor () {
    for (unsigned i = 0; i < 3; i++) {
            if ((matriz[i][0] == matriz[i][1]) &&
                (matriz[i][1] == matriz[i][2])) {
                   if ((matriz[i][0] != '-') &&
                       (matriz[i][1] != '-') && (matriz[i][2] != '-'))
                           return true;
            }
            if ((matriz[0][i] == matriz[1][i]) &&
                (matriz[i][1] == matriz[2][i])) {
                   if ((matriz[0][i] != '-') &&
                        (matriz[1][i] != '-') && (matriz[2][i] != '-'))
                           return true;
            }
    }
    if ((matriz[0][0] == matriz[1][1]) &&
         (matriz[1][1] == matriz[2][2])) {
            if ((matriz[0][0] != '-') &&
                (matriz[1][1] != '-') && (matriz[2][2] != '-'))
                           return true;
    }
    if ((matriz[0][2] == matriz[1][1]) &&
         (matriz[1][1] == matriz[2][0])) {
            if ((matriz[0][2] != '-') &&
                (matriz[1][1] != '-') && (matriz[2][0] != '-'))
                           return true;
    }
    return false;
}

bool Velha::verificaLimite (unsigned i, unsigned j) {
    if ((i < 0)||(i > 2)||(j < 0)||(j > 2)) {
            return false;
    }
    return true;
}

bool Velha::verificaPosicao (unsigned i, unsigned j) {
    if (matriz[i][j] != '-') {
            return false;
    }
    return true;
}

bool Velha::verificaVelha () {
    if (posicoesOcupadas == 9) {
            imprime();
            cout << "Deu velha. Fim de jogo." << endl;
            return false;
    }
    return true;
}
/*****************************************************************************/
```

b)

```
/*****************************************************************************/
/***************************** Arquivo proxy.h *******************************/
#include <iostream>

using namespace std;

#ifndef PROXY_H
```

```
#define PROXY_H

class MinhaClasse;

class Proxy {
      public:
             Proxy ();
             ~Proxy ();
             string getNome ();
             int getIdade ();
             void setNomeIdade (string, int);

      private:
              MinhaClasse *ptr;
};

#endif

/*****************************************************************************/
/****************************** Arquivo proxy.cpp ****************************/
#include "proxy.h"
#include "minhaclasse.h"

Proxy::Proxy (): ptr (new MinhaClasse ("None", 0)) {}
Proxy::~Proxy () { delete ptr; }

string Proxy::getNome () { return ptr->nome; }

int Proxy::getIdade () { return ptr->idade; }

void Proxy::setNomeIdade (string n, int i) {
     ptr->nome = n;
     ptr->idade = i;
}

/*****************************************************************************/
```