

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) Escreva uma classe `Carrinho` para armazenar itens a serem comprados. Cada item é um objeto da classe `Produto` que possui os atributos privados tipo (p. ex., "brinquedo", "eletrodoméstico" etc.), marca e preço. A classe `Produto` ainda oferece um método `get` e `set` para cada atributo, um construtor que inicializa todos os seus atributos e o operador `<<` sobrecarregado.

Os produtos são armazenados na classe `Carrinho` em um `vector` de ponteiros para objetos da classe `Produto`. A classe `Carrinho` deve oferecer um método para inserir produtos e outro para remover produtos usando os operadores `+` e `-` sobrecarregados, respectivamente. Dessa forma, a inserção deve ser feita da seguinte maneira:

```
carrinho = carrinho + objetoBrinquedo;
```

e a remoção:

```
carrinho = carrinho - objetoBrinquedo;.
```

Note que o método de inserção deve inserir o produto no `vector` de ponteiros privados através da referência ao objeto da classe `Produto`:

```
v.at(pos) = &objetoBrinquedo;
```

A operação de remoção, por outro lado, deve tirar um produto específico do mesmo `vector`. Use o método `erase` da classe `vector` para a remoção. A classe `Carrinho` ainda deve implementar um construtor que define o número máximo de itens no carrinho (argumento passado para o construtor do `vector` privado) e o operador `<<` sobrecarregado para impressão na tela dos produtos atuais no carrinho. Lembre que a classe `Produto` também sobrecarrega o operador `<<`. A função principal deve ser escrita como abaixo. Verifique o que ocorre com o preço do brinquedo inserido no carrinho após a execução do método `set` correspondente.

```

/*****
int main () {
    Carrinho car (5);

    Produto brinquedo ("brinquedo", "estrela", 90.00);
    Produto arroz ("arroz", "Tio Joao", 20.00);
    Produto pneu ("pneu", "Goodyear", 150.00);

    car = car + brinquedo;
    car = car + arroz;
    car = car + pneu;

    cout << "*** Completo\n" << car;

    car = car - arroz;

    cout << "*** Reduzido\n" << car;

    brinquedo.setPreco(200.00);

    // Verifique se o preço do brinquedo foi atualizado no carrinho...
    cout << "*** Com preco dp brinquedo atualizado\n" << car;

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

- c) Continuando a Questão 1.b, implemente a inserção do produto no `vector` de ponteiros privados através do uso do operador `new` como se segue:

```
v.at(pos) = new Produto (objetoBrinquedo);
```

Lembre-se nesse caso de implementar o destrutor para a classe `Carrinho`. Ao executar a mesma função principal da Questão 1.a, o que ocorre com o valor do produto após ter o seu preço ajustado? Por que o comportamento do programa mudou?

- d) Ainda continuando a Questão 1.b, execute a função principal abaixo que cria uma cópia do objeto da classe `Carrinho`. O que acontece com o código?

```

/*****/
int main () {
    Carrinho car (5);

    Produto brinquedo ("brinquedo", "estrela", 90.00);
    Produto arroz ("arroz", "Tio Joao", 20.00);
    Produto pneu ("pneu", "Goodyear", 150.00);

    car = car + brinquedo;
    car = car + arroz;
    car = car + pneu;

    cout << "*** Completo\n" << car;

    car = car - arroz;

    cout << "*** Reduzido\n" << car;

    {
        // Cópia do carrinho é criada em um bloco interno à função principal
        Carrinho carCopia (car);

        cout << "\n*** Copia\n" << carCopia;
    }

    brinquedo.setPreco(200.00);

    cout << "*** Com preco dp brinquedo atualizado\n" << car;

    return 0;
}
/*****/

```

Corrija o programa através da criação de um construtor de cópia para a classe Carrinho.

- e) Dada a função principal abaixo, implemente as classes `Jogo` e `Personagem` e as funções globais necessárias em arquivos `*.cpp` e `*.h` para que o programa possa ser compilado e executado. A classe `Jogo` tem um `vector` de ponteiro de objetos da classe `Personagem` (`vector <Personagem *> v`).

```

/*****/
// includes...
using namespace std;

int main () {
    /* Construtor da classe Jogo possui argumento que define
    o número de personagens */
    Jogo jogo (4);

    /* Construtor da classe Personagem possui argumentos nome,
    nível de força e nível de inteligência */
    Personagem hulk ("Hulk", 90, 20);
    Personagem homemDeFerro ("Homem de Ferro", 60, 90);
    Personagem capitao ("Capitao America", 50, 70);
    Personagem thor ("Thor", 80, 60);

    /* Operador () sobrecarregado adiciona os personagens ao
    jogo de forma cascadeada */
    jogo(hulk)(homemDeFerro)(capitao)(thor);

    /* Operador [] sobrecarregado retorna ponteiro para objeto
    da classe Personagem a partir de índice e operador <<
    sobrecarregado imprime todas as características do
    personagem na tela */
    if (jogo ["Hulk"]) {
        cout << jogo ["Hulk"];
    } else {
        cout << "\nPersonagem nao encontrado!" << endl;
    }

    if (jogo ["Homem Formiga"]) {
        cout << jogo ["Homem Formiga"];
    }
}

```

```

    } else {
        cout << "\nPersonagem nao encontrado!" << endl;
    }

    /* Operador [] sobrecarregado retorna ponteiro para objeto
    da classe Personagem a partir de índice e operador <<
    sobrecarregado imprime todas as características do
    personagem na tela */
    cout << "\n== Personagens: " << endl;
    for (unsigned i = 0; i < jogo.getNumeroPersonagens (); i++) {
        cout << jogo [i];
    }

    /* Função global calculaEstatistica retorna ponteiro para
    Personagem que possui maior força ou maior inteligência,
    conforme o ponteiro para função passada como segundo
    argumento */
    cout << "\n*** Mais Forte:\n"
        << calculaEstatistica (jogo, maisforte);
    cout << "\n*** Mais Inteligente:\n"
        << calculaEstatistica (jogo, maisinteligente);

    return 0;
}
/*****

```

- 2) **Programa para entrega dia 17/09/2021:** 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 `Catalogo` para gerenciamento de filmes. O filme é definido a partir de uma `struct` (não é classe) contendo uma `string` com o nome do filme, uma `string` com o nome da produtora e um `double` para armazenar a nota do filme conforme a avaliação do dono do catálogo. Já a classe `Catalogo` implementa uma estrutura do tipo `vector` para armazenar os filmes, além de um tamanho máximo para o número de filmes. As diferentes ações permitidas com o objeto da classe `Catalogo` devem usar operadores como se segue:

- **Inserção ordenada de um filme no catálogo:** deve ser feita com o operador `+=`. Por exemplo, `catalogo += filme` insere um filme ordenado pelo nome no catálogo. Note que o nome, produtora e nota do filme devem ser inicializados previamente, antes da inserção. A inserção deve prever também a possibilidade de inserção de um `vector` de filmes, como uma operação de inserção em lote. Nesse caso, a inserção pode ser feita ao final do `vector` de filmes da classe e reordenada por completo a posteriori. Experimente o método `insert` da classe `vector`.

O operador `<` (ou `>`) deve ser implementado para que a comparação entre filmes seja possível. Por exemplo, `filme1 < filme2` deve retornar `true` caso o nome do `filme1` seja menor que o nome do `filme2`. Ainda, a classe `Catalogo` não permite a inserção de filmes com o mesmo nome. Dessa forma, é importante implementar o operador `==` para verificar se o filme a ser inserido tem o mesmo nome de outro já existente. Por exemplo, `filme1 == filme2` deve retornar `true` se os nomes dos filmes forem os mesmos e `false`, caso contrário.

- **Remoção de um filme do catálogo:** deve ser feito através do operador `-=`. Por exemplo, `catalogo -= filme` remove o filme do catálogo. A busca do filme deve ser feita a partir do nome do filme e a remoção no `vector` pode usar o método `erase`.
- **Busca de um filme no catálogo:** deve ser feita através do operador `()` sobrecarregado. A busca é realizada a partir do nome do filme passado por valor,

da seguinte maneira: `catalogo("nome")`. A busca retorna um ponteiro para o filme encontrado ou `NULL`, caso contrário.

- **Edição de um filme no catálogo:** deve ser feito através do operador `()`, sendo que o nome do filme e o valor a ser utilizado na atualização são passados para o operador. O nome é usado para a busca e, como tal, não pode ser alterado. A operação de edição do nome de um filme deve ser realizada através da remoção e inserção de um novo filme. Porém, caso a edição seja da produtora ou da nota, estas podem ser feitas respectivamente com `catalogo("nome", "novaprodutora")` ou `catalogo("nome", novanota)`. Um ponteiro para o objeto que acabou de ser editado deve ser retornado ou `NULL`, caso este não tenha sido encontrado.
- **Impressão do catálogo inteiro de filmes e do filme na tela:** devem ser realizadas respectivamente com `cout << catalago` e `cout << filme`.
- **Inicialização dos dados referentes a um filme:** deve ser feito através de `cin >> filme`.
- **Busca pelo filme mais bem avaliado:** Semelhante à operação de inserção, a implementação deve ser feita usando o operador `>` (ou `<`). Neste caso, porém, o operador deve ser usado na comparação entre um objeto filme e uma variável que armazene a nota máxima já encontrada, como por exemplo em `"filme > 0"`. O resultado dessa comparação que deve retornar `true` ou `false` dependendo da nota do filme.

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 dos filmes. Toda vez que um catálogo é criado, este deve carregar todos os filmes já registrados e armazenados em um arquivo de texto. Toda vez que o programa encerra, o arquivo de texto deve ser reescrito por completo.

== 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)

```

/*****
/***** Programa Principal *****/
#include <iostream>
#include <string>
#include <iomanip>
#include <vector>

```



```

#include "produto.h"
#include "carrinho.h"

/* Programa do Laboratório 7:
   Programa de um Carrinho de compras
   Autor: Miguel Campista */

using namespace std;

int main () {
    Carrinho car (5);

    Produto brinquedo ("brinquedo", "estrela", 90.00);
    Produto arroz ("arroz", "Tio Joao", 20.00);
    Produto pneu ("pneu", "Goodyear", 150.00);

    car = car + brinquedo;
    car = car + arroz;
    car = car + pneu;

    cout << "\n*** Completo\n" << car;

    car = car - arroz;

    cout << "\n*** Reduzido\n" << car;

    brinquedo.setPreco(200.00);

    cout << "\n*** Brinquedo atualizado\n" << car;

    return 0;
}

/*****
***** Arquivo produto.h *****/
#include <iostream>
#include <string>
#include <iomanip>

using namespace std;

#ifndef PRODUTO_H
#define PRODUTO_H

class Produto {
public:
    Produto (string, string, double);

    string getTipo ();
    string getMarca ();
    double getPreco ();

    void setTipo (string);
    void setMarca (string);
    void setPreco (double);

private:
    string tipo, marca;
    double preco;
};

ostream &operator<< (ostream &, Produto *);

#endif

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

ostream &operator<< (ostream &out, Produto *p) {
    out << setw(20) << "Tipo: " << setw(20) << p->getTipo () << endl;
    out << setw(20) << "Marca: " << setw(20) << p->getMarca () << endl;
    out << setw(20) << "Preco (R$): " << setw(20) << fixed << setprecision (2)
        << p->getPreco () << endl;

    return out;
}

```

```

Produto::Produto (string t, string m, double p): tipo (t), marca (m), preco (p) {}

string Produto::getTipo () { return tipo; }
string Produto::getMarca () { return marca; }
double Produto::getPreco () { return preco; }

void Produto::setTipo (string t) { tipo = t; }
void Produto::setMarca (string m) { marca = m; }
void Produto::setPreco (double p) { preco = p; }

/*****
/***** Arquivo carrinho.h *****/
#include <iostream>
#include <vector>
#include <iomanip>

#include "produto.h"

using namespace std;

#ifndef CARRINHO_H
#define CARRINHO_H

class Carrinho {
    friend ostream &operator<< (ostream &, Carrinho &);

public:
    Carrinho (int);

    Carrinho &operator+ (Produto &);
    Carrinho &operator- (Produto &);

private:
    unsigned conta, maxNum;
    vector <Produto *> v;
};

#endif

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

ostream &operator<< (ostream &out, Carrinho &c) {
    for (unsigned i = 0; i < c.conta; i++)
        out << c.v.at (i);
    return out;
}

Carrinho::Carrinho (int n): conta (0), maxNum (n), v (n) {}

Carrinho &Carrinho::operator+ (Produto &p) {
    if (conta < maxNum - 1)
        v.at (conta++) = &p;
    else cout << "Carrinho cheio..." << endl;

    return *this;
}

Carrinho &Carrinho::operator- (Produto &p) {
    for (unsigned i = 0; i < conta; i++) {
        if (v.at (i)->getTipo () == p.getTipo ()) {
            v.erase (v.begin () + i);
            conta--;
        }
    }

    return *this;
}
/*****/

```

- c) Os únicos arquivos que foram alterados em relação ao exercício anterior foram os relativos à implementação da classe Carrinho.

```

/*****/

```

```

/***** Arquivo carrinho.h *****/
#include <iostream>
#include <vector>
#include <iomanip>

#include "produto.h"

using namespace std;

#ifndef CARRINHO_H
#define CARRINHO_H

class Carrinho {
    friend ostream &operator<< (ostream &, Carrinho &);

public:
    Carrinho (int);

    ~Carrinho ();

    Carrinho &operator+ (Produto &);
    Carrinho &operator- (Produto &);

private:
    unsigned conta, maxNum;
    vector <Produto *> v;
};

#endif

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

ostream &operator<< (ostream &out, Carrinho &c) {
    for (unsigned i = 0; i < c.conta; i++)
        out << c.v.at (i);
    return out;
}

Carrinho::Carrinho (int n): conta (0), maxNum (n), v (n) {}

Carrinho::~Carrinho () {
    cout << "Destruindo..." << endl;
    for (unsigned i = 0; i < conta; i++) delete v.at (i);
}

Carrinho &Carrinho::operator+ (Produto &p) {
    if (conta < maxNum - 1)
        v.at (conta++) = new Produto (p);
    else cout << "Carrinho cheio..." << endl;

    return *this;
}

Carrinho &Carrinho::operator- (Produto &p) {
    for (unsigned i = 0; i < conta; i++) {
        if (v.at (i)->getTipo () == p.getTipo ()) {
            delete v.at (i);
            v.erase (v.begin () + i);
            conta--;
        }
    }

    return *this;
}
/*****

```

d) Os únicos arquivos que foram alterados foram os relativos à implementação da classe Carrinho.

```

/*****
/***** Arquivo carrinho.h *****/
#include <iostream>
#include <vector>
#include <iomanip>

```

```

#include "produto.h"

using namespace std;

#ifndef CARRINHO_H
#define CARRINHO_H

class Carrinho {
    friend ostream &operator<< (ostream &, Carrinho &);

public:
    Carrinho (int);

    Carrinho (const Carrinho &);

    ~Carrinho ();

    Carrinho &operator+ (Produto &);
    Carrinho &operator- (Produto &);

private:
    unsigned conta, maxNum;
    vector <Produto *> v;
};

#endif

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

ostream &operator<< (ostream &out, Carrinho &c) {
    for (unsigned i = 0; i < c.conta; i++)
        out << c.v.at (i);
    return out;
}

Carrinho::Carrinho (int n): conta (0), maxNum (n), v (n) {}

Carrinho::Carrinho (const Carrinho &c):
    conta (c.conta), maxNum (c.maxNum), v (c.maxNum) {
    for (unsigned i = 0; i < conta; i++) {
        v.at (i) = new Produto (*c.v.at (i));
    }
}

Carrinho::~Carrinho () {
    cout << "Destruindo..." << endl;
    for (unsigned i = 0; i < conta; i++) delete v.at (i);
}

Carrinho &Carrinho::operator+ (Produto &p) {
    if (conta < maxNum - 1)
        v.at (conta++) = new Produto (p);
    else cout << "Carrinho cheio..." << endl;

    return *this;
}

Carrinho &Carrinho::operator- (Produto &p) {
    for (unsigned i = 0; i < conta; i++) {
        if (v.at (i)->getTipo () == p.getTipo ()) {
            delete v.at (i);
            v.erase (v.begin () + i);
            conta--;
        }
    }

    return *this;
}
/*****/

```

e)

```

/*****/

```

```

/***** Programa Principal *****/
#include <iostream>
#include <string>
#include <vector>
#include <iomanip>

#include "personagem.h"
#include "jogo.h"
#include "globais.h"

/* Programa do Laboratório 7:
   Programa do Jogo de heróis
   Autor: Miguel Campista */

using namespace std;

int main () {
    /* Construtor da classe Jogo possui argumento que define
       o número de personagens */
    Jogo jogo (4);

    /* Construtor da classe Personagem possui argumentos nome,
       nível de força e nível de inteligência */
    Personagem hulk ("Hulk", 90, 20);
    Personagem homemDeFerro ("Homem de Ferro", 60, 90);
    Personagem capitao ("Capitao America", 50, 70);
    Personagem thor ("Thor", 80, 60);

    /* Operador () sobrecarregado adiciona os personagens ao
       jogo de forma cascadeada */
    jogo(hulk)(homemDeFerro)(capitao)(thor);

    /* Operador [] sobrecarregado retorna ponteiro para objeto
       da classe Personagem a partir de índice e operador <<
       sobrecarregado imprime todas as características do
       personagem na tela */
    if (jogo ["Hulk"]) {
        cout << jogo ["Hulk"];
    } else {
        cout << "\nPersonagem nao encontrado!" << endl;
    }

    if (jogo ["Homem Formiga"]) {
        cout << jogo ["Homem Formiga"];
    } else {
        cout << "\nPersonagem nao encontrado!" << endl;
    }

    /* Operador [] sobrecarregado retorna ponteiro para objeto
       da classe Personagem a partir de índice e operador <<
       sobrecarregado imprime todas as características do
       personagem na tela */
    cout << "\n== Personagens: " << endl;
    for (unsigned i = 0; i < jogo.getNumeroPersonagens (); i++) {
        cout << jogo [i];
    }

    /* Função global calculaEstatistica retorna ponteiro para
       Personagem que possui maior força ou maior inteligência,
       conforme o ponteiro para função passada como segundo
       argumento */
    cout << "\n*** Mais Forte:\n"
        << calculaEstatistica (jogo, maisforte);
    cout << "\n*** Mais Inteligente:\n"
        << calculaEstatistica (jogo, maisinteligente);

    return 0;
}

/***** Arquivo jogo.h *****/
#include <iostream>
#include <string>
#include <vector>

#include "personagem.h"

```

```

#ifndef JOGO_H
#define JOGO_H

class Jogo {
public:
    Jogo (int);

    Jogo &operator() (Personagem &);

    unsigned getNumeroPersonagens ();

    Personagem *operator[] (string s);
    Personagem *operator[] (unsigned);

private:
    vector <Personagem *> v;
};

#endif

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

Jogo::Jogo (int n): v (n) {}

Jogo &Jogo::operator() (Personagem &p) {
    static int number = 0;

    v.at (number++) = &p;

    return *this;
}

unsigned Jogo::getNumeroPersonagens () { return v.size (); }

Personagem *Jogo::operator[] (string s) {
    for (unsigned i = 0; i < v.size (); i++) {
        if (!(v.at (i)->getNome ().compare (s))
            return v.at (i);
    }

    return NULL;
}

Personagem *Jogo::operator[] (unsigned i) {
    if ((i < 0) || (i >= v.size ()))
        return NULL;
    else return v.at (i);
}

/*****
/***** Arquivo personagem.h *****/
#include <iostream>
#include <string>
#include <iomanip>

using namespace std;

#ifndef PERSONAGEM_H
#define PERSONAGEM_H

class Personagem {
    friend ostream &operator<<(ostream &, Personagem *);

public:
    //Personagem () {}
    Personagem (string, int, int);

    string getNome ();
    int getForca ();
    int getInteligencia ();

private:
    string nome;
    int forca, inteligencia;
};

```

```

#endif

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

ostream &operator<<(ostream &out, Personagem *p) {
    out << setw (20) << "Nome: " << setw (10) << p->nome << endl;
    out << setw (20) << "Forca: " << setw (10) << p->forca << endl;
    out << setw (20) << "Inteligencia: " << setw (10) << p->inteligencia << endl;

    return out;
}

Personagem::Personagem (string s, int f, int i):
    nome (s), forca (f), inteligencia (i) {}

string Personagem::getNome () { return nome; }
int Personagem::getForca () { return forca; }
int Personagem::getInteligencia () { return inteligencia; }

/*****
/***** Arquivo globais.h *****/
#include <iostream>

#include "personagem.h"
#include "jogo.h"

using namespace std;

#ifndef GLOBAIS_H
#define GLOBAIS_H

Personagem *maisforte (Jogo &jogo) {
    Personagem *maisForte;
    int maiorForca = 0;

    for (unsigned i = 0; i < jogo.getNumeroPersonagens (); i++) {
        if (jogo [i]->getForca () > maiorForca) {
            maisForte = jogo [i];
            maiorForca = jogo [i]->getForca ();
        }
    }

    return maisForte;
}

Personagem *maisinteligente (Jogo &jogo) {
    Personagem *maisInteligente;
    int maiorInteligencia = 0;

    for (unsigned i = 0; i < jogo.getNumeroPersonagens (); i++) {
        if (jogo [i]->getInteligencia () > maiorInteligencia) {
            maisInteligente = jogo [i];
            maiorInteligencia = jogo [i]->getInteligencia ();
        }
    }

    return maisInteligente;
}

Personagem *calculaEstatistica (Jogo &jogo, Personagem * (*p) (Jogo &)) {
    return (*p) (jogo);
}

#endif
/*****/

```