

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

- a) 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;
}
/* **** */
```

- b) Continuando a Questão 1.a, 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?

- c) Ainda continuando a Questão 1.a, 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*** Cópia\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`.

- d) 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 25/11/2022:** 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:

- **Impressão do catálogo inteiro de filmes e de um único filme na tela:** devem ser realizadas respectivamente com `cout << catalogo` e `cout << filme`.
- **Inicialização dos dados referentes a um filme:** deve ser feito através de `cin >> filme`.
- **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 (p.ex., `"catalogo += vector"`). 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.

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

- **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`.

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

- **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 o índice do filme encontrado no `vector` ou `-1`, caso contrário. Este índice é usado para exibição na tela do nome do filme e de seus atributos. Dica: use `cout << filme`.
- **Edição de um filme no catálogo:** deve ser feito através do operador `()`, sendo que o nome do filme, o nome do atributo a ser editado e o valor a ser utilizado na atualização são passados para o operador. O nome é usado para a busca e o nome do atributo indica o que deve ser editado. A operação de edição do nome, da produtora ou da nota pode ser feita respectivamente com `catalogo("nomefilme", "nomeatributo", "novonome")`, `catalogo("nomefilme", "nomeatributo", "novaprodutora")` ou `catalogo("nomefilme", "nomeatributo", novanota)`.

O índice no `vector` do objeto que acabou de ser editado deve ser retornado ou `-1`, caso este não tenha sido encontrado.

- **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 deve retornar `true` ou `false` dependendo da nota do filme, como por exemplo no trecho de código abaixo:

```
...
double max = filme.getNota ();
if (filme > max)
    max = filme.getNota ();
...
```

A busca retorna o índice do filme de maior nota. Este índice é usado para exibição na tela do nome do filme e de seus atributos. Dica: use `cout << 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. 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 <string>
#include <iomanip>
#include <vector>

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

/* Programa do Laboratório 8:
   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;
}
/*****

```

b) 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;
}
/*****

```

c) 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;
}
/*****

```

d)

```

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

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

/* Programa do Laboratório 8:
   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  
/*****
```