SÉTIMA LISTA DE EXERCÍCIOS DE PROGRAMAÇÃO III

1) Em que situações deve-se utilizar o operador "->" e em que situações deve-se utilizar o operador "." Em C++? Exemplifique.

Os operadores devem ser usados quando precisa-se acessar algum atributo ou método de um objeto, sendo diferenciados pelo tipo de objeto criado. O operador "." deve ser usado quando o objeto foi criado de maneira estática e o operador "->" deve ser usado quando o objeto foi criado de maneira dinâmica.

2) Em relação a sequência de heranças abaixo, com diferentes modificadores de acesso, informe quais membros (atributos e métodos) a classe CarroSport terá acesso e com qual modificador de acesso.

```
class MeioTransporte{
    public:
        int anoFabricacao;
    private:
        void imprimeAno(){}
};

class MeioTransporteTerrestre: private MeioTransporte{
    protected:
        int numEixos;
    public:
        void imprimeNumEixos(){}
        ~MeioTransporteTerrestre(){}
};

class Veiculo: public MeioTransporteTerrestre{
    protected:
        string placa;
        void imprimePlaca(){}
    public:
        Veiculo(){}
};
```

```
class Carro: protected Veiculo{
    private:
        string marca;
        string modelo;
    protected:
        void imprimeMarcaModelo(){}
        Carro(string _marca, string _modelo){}
        ~Carro(){}
    public:
        friend imprimeInfoCarro();
};

class CarroSport : public Carro {
    private:
        int potencia;
        double valor;
    public:
        void imprimePotencia(){}
};
```

Private:

int potencia;

double valor;

Public:

friend imprimeInfoCarros(){}

void imprimePotencia(){}

Protected:

void imprimeModelo(){}

Carro(){}

~Carro(){}

3) O que são Exceções? Quais são as palavras chaves utilizadas para manipular uma exceção e qual a função de cada uma? Como elas podem ser tratadas? Como elas podem ser lançadas pelo próprio programador? Implemente um código exemplificando o tratamento de exceções.

Exceções são uma forma de transferir o controle de uma parte do programa para outra sem que o programa seja finalizado com erro.

As palavras chaves utilizadas para manipular exceções são: try, throw e catch. A função da palavra try é criar um bloco de código que será executado e que pode gerar uma exceção. A palavra catch tem a função de indicar a captura da exceção gerada no bloco try. A função da palavra throw é lançar uma exceção quando o programa encontra um problema de execução.
Exceções podem ser tratadas utilizando um bloco try envolta do código que pode gerar a exceção seguido de um bloco catch que, além de capturar a exceção, terá dentro dele o código que irá tratar a exceção e retornar o fluxo do programa para o fluxo normal.
Uma exceção pode ser lançada pelo programador através do uso da palavra throw. Assim, o programador pode criar exceções personalizadas e captura-las em algum lugar do código.
Exemplo:

```
#include <stdexcept>

using namespace std;

double divisao(int a, int b)
{
    if(b==0)
    {
        throw "Nao eh permitido divisao por 0";
    }
    return (a/b);
}
```

```
int main(){
    int x;
    int y;
    double z;

    cout << "Digite o numerador: ";
    cin >> x;
    cout << "Digite o denominador: ";
    cin >> y;

    try {
        z = divisao(x, y);
        cout << "Resposta: " << z << endl;
    }catch (const char* msg) {
        cerr << msg << endl;
    }
    return 0;
}
```

4) O que são templates em C++? Implemente um código que exemplifique um Template para Função e um código que exemplifique um Template para Classe.

Templates são tipos genéricos baseados na ideia de polimorfismo paramétrico, ou seja, são uma forma que permite escrever funções ou tipo de dados genericamente, suportando operações idênticas sem depender do tipo de dado sendo manipulado.
Exemplo de template para função:

```cpp
template <typename tipo>
tipo compara_valores (tipo a, tipo b){
    if(a > b) return a;
    else return b;
}

int main()
{
    int x = 15, y = 20;
    float z = 10.55, k = 10.85;
    string s1 = "Aula", s2 = "Prova";
    char c1 = 'a', c2 = 'c';

    cout << ">>---Comparacao---<<\n" << endl;
    cout << "Maior Int: " << compara_valores<int>(x,y) << endl;
    cout << "Maior Float: " << compara_valores<float>(z,k) << endl;
    cout << "Maior String: " << compara_valores<string>(s1,s2) << endl;
    cout << "Maior Char: " << compara_valores<char>(c1,c2) << endl;

    return 0;
}
```

Exemplo de template para classe:

```cpp
template <typename tipo>
class Coletivo
{
public:
    vector<tipo> membros;

    void push (tipo membro){
        membros.push_back(membro);
    }
};
```

```cpp
class Pessoa{
    public:
        int idade;
        string nome;
        double altura;
        double peso;
};

class Cachorro{
    public:
        int idade;
        string nome;
        double peso;
};
```

```cpp
int main ()
{
    Pessoa *p1;
    Cachorro *c1;
    p1 = new Pessoa();
    c1 = new Cachorro();

    p1->idade = 25;
    p1->nome = "Joãozinho";
    p1->altura = 1.8;
    p1->peso = 85.5;

    c1->idade = 15;
    c1->nome = "Rex";
    c1->peso = 5.65;

    Coletivo<Cachorro*> cachorros;
    Coletivo<Pessoa*> pessoas;
    cachorros.push(c1);
    pessoas.push(p1);

    return 0;
}
```

5) Em uma herança é possível e correto chamar os métodos da superclasse dentro dos métodos da subclasse em C++ e Java? Para cada linguagem, explique e dê exemplos.

É possível, mas nem sempre é correto. Em Java, podemos chamar métodos da superclasse dentro de métodos da subclasse usando a palavra reservada super, que realiza a chamada, e isso pode ser feito inclusive em construtores. Em C++, se chamarmos um construtor da superclasse dentro do construtor da subclasse, ele vai construir dois objetos diferentes. Assim, o ideal é referenciar o construtor da superclasse na assinatura do construtor da subclasse, e não sendo chamado dentro do método.

Exemplo em Java:

```java
class Produto {
  /* ... */
  public void imprimir() {
    System.out.println(nome + "," + preco);
  }
}

class Livro extends Produto {
  /* ... */
  public void imprimir() {
    super.imprimir();
    System.out.println(autor + "," + paginas);
  }
}
```

Exemplo em C++:

```cpp
class Pessoa{
    protected:
    std::string nome;
    std::string telefone;
    std::string email;
    std::string endereco;
    std::string rg;
    std::string cpf;

    public:

    Pessoa(){
        std::cout << "Inicializando pessoa vazia" << std::endl;
    }

    Pessoa(std::string nome, std::string telefone, std::string email, std::string endereco, std::string rg, std::string cpf){
        std::cout << "Inicializando pessoa cheia" << std::endl;
        this->nome = nome;
        this->telefone = telefone;
        this->email = email;
        this->endereco = endereco;
        this->rg = rg;
        this->cpf = cpf;
    }
```

```cpp
class Aluno : private Pessoa{

  private:

    int codigo;

    std::string matriculaInstituicao;


  public:

    static int qtdAlunos;


    Aluno(std::string nome, std::string telefone, std::string email, std::string endereco, std::string rg, std::string cpf, int codigo, std::string matriculainstituicao) : Pessoa(nome, telefone, email, endereco, rg, cpf){

        std::cout << "Aluno" << std::endl;

        this->codigo = codigo;

        this->matriculaInstituicao = matriculaInstituicao;

        qtdAlunos++;

    }

};
```