***Importante***: Para item abaixo deve ser copiado trechos do código que cumprem o requisito e explicado, se não for aparente, o porquê o requisito é cumprido. Sejam bem explícitos. Deve ser indicado também o .h ou .cpp no qual o trecho do código está. Eu avaliarei o código do Github a partir desse documento para confirmá-lo e também para detectar possíveis erros. **Quem não seguir o que está indicado aqui, não terá o projeto avaliado e perderá a atividade.**

**Requisitos funcionais**

***Polimorfismo***

* Duas classes abstratas, sendo que uma classe abstrata herda da outra classe abstrata na hierarquia de classes;
* Cada classe abstrata deve ter pelo menos um parâmetro, o construtor vazio e o construtor default. Deve ter também um método não virtual, que não pode ser set ou get;
* Pelo menos três classes concretas na hierarquia de classes;
* Usar coerção de tipo C++ e não C, ou seja usar o **static\_cast** para fazer o máximo de reutilização de código.
* Usar um vector de classes concretas, o **dynamic\_cast** e o **typeid** como indicado aqui: <https://basecamp.com/2595605/projects/7018448/messages/33080741>
* Criar uma função no arquivo do **main**, que aceita um ponteiro da classe genérica e mostrar o seu uso para as classes concretas;

***Importante***: **Todos os Requisitos abaixo ainda devem ser feitos e serão avaliados**

***Todos os atributos e funções membros devem estar relacionados a classe***

* Pelo menos 4 atributos - Na classe Robocop.h, os atributos: idOriginal, velocidade, numeroDiretivas, VELOCIDADE\_MAXIMA, ...
* Pelo menos 4 funções membros sem incluir get e set - Na classe Robocop.h, os métodos: correr, falar, olharDireita, olharEsquerda, mostrarDetalhes,...

**Requisitos de implementação**

* Todos os atributos devem ser inicializados. Fez validação de dados - Sim, todos são inicializados nos construtores (Robocop.cpp)
* Três construtores, incluindo um construtor de cópia e construtor com parâmetros defaults. Verifica alocação dentro do construtor de cópia. - Sim, no arquivo Robocop.cpp
* Deve ter um atributo string - O array diretivas[1000] (Robocop.h)
* Um atributo static. Correta modelagem dos statics? Sim, o atributo numeroInstancias (Robocop.h)
* Um atributo const static - Os atributos VELOCIDADE\_MAXIMA, DIRETIVA\_PADRAO e FORCA\_MAXIMA (Robocop.h)
* Dois métodos constantes (não pode ser get) - Os métodos: mostrarDetalhes, olharEsquerda e olharDireita (Robocop.h)
* Um array - O array diretivas[1000] (Robocop.h)
* Uma função inline (não pode ser get ou set) - A função falar. (Robocop.h)
* Método com passagem por referência usando ponteiro - O método setDataTransformacao. (Robocop.h)
* Método static – deve ser chamado no main - O método getNumeroInstancias. (main.cpp)
* Composição com a classe Data. Fez uso do objeto criado? - Sim, o atributo dataTransformacao, usado para armazenar a data da transformação da Pessoa em Ciborgue. (Robocop.h, Robocop.cpp)
* O que é const deve ser const. Sim. (Robocop.h, Robocop.cpp)
* Alocação dinâmica de memória. A memória é desalocada? - Sim. (Destrutor no arquivo Robocop.cpp)
* friend Operator<< - Robocop.h
* Operator= - Robocop.cpp
* vector push\_back - vetorRobocopsTransformados (Robocop.h, Robocop.cpp)

Requisitos para as classes adicionais (pelo menos duas)

* Operator = - (Pessoa.cpp)
* Alocação dinâmica - se houver vazamento de memória a classe toda é desconsiderada - (Pessoa.cpp)
* Usar o destrutor - (Pessoa.cpp)
* Construtor de cópia - (Pessoa.cpp)
* Operator << friend - (Pessoa.h)
* Um const static

**Requisitos herança**

* Diagrama de classes (obrigatório salvar também o png do diagrama no gitHub) - Feito.
* Herança pública - Feito. (Robocop.h)
* Construtor de cópia, e sobrecargas dos operadores de atribuição (=) e << (cout << base) para a classe base e derivada - (Robocop.cpp, Pessoa.cpp)
* Usar Protected acessando diretamente os atributos na classe derivada - (Robocop.h)
* Alocação dinâmica de memória na classe base e derivada - (Robocop.cpp, Pessoa.cpp)
* Sobrescrita de método: chamar dentro do método da classe derivada o método correspondente da classe base usando :: - (Robocop.h, Pessoa.h)
* No main: criar um ponteiro da classe base para alocar memória para a classe derivada e chamar os vários métodos implementados - O ponteiro pessoa1. (main.cpp)