



Prof. Esther Colombini
esther@ic.unicamp.br

Atividade Prática 1: Herança, Sobrecarga, Sobrescrita e Alocação de Objetos

Entrega: 07/09/2023

1 Objetivo

O objetivo deste trabalho é permitir que o aluno aplique seus conhecimentos em herança, sobrecarga, sobrescrita, alocação dinâmica e estática de objetos usando C++.

2 Passos da Atividade

Passo 1: Configuração do Ambiente

Antes de iniciar a atividade, certifique-se de que o ambiente de desenvolvimento C++ está configurado. Você pode usar qualquer IDE de sua escolha (Visual Studio Code, Code::Blocks, Dev-C++, etc.) ou até mesmo um editor de texto simples com o compilador g++ instalado. Lembre-se que a instalação do compilador g++ é dependente de plataforma.

Passo 2: Escrevendo o Código

Desenvolva o mini-game chamado **Jewel Collector v1.0**. O objetivo deste jogo é que um robô, controlado pelo teclado, se desloque por um mapa 2D procurando comida e coletando as joias presentes no ambiente. Entretanto, neste momento, iremos nos concentrar apenas no desenvolvimento do ambiente de jogo. Para isso, as seguintes classes devem ser criadas:

1. **Item:** A classe Item é responsável por representar os elementos presentes no mapa do jogo. Cada item é caracterizado por dois atributos: o tipo, que é a identificação do item (por exemplo, joia, comida); e o valor, que é um número para indicar a relevância desse item no contexto do jogo. Lembre-se de criar os métodos *getters* e *setters* para os atributos da classe, se julgar necessário. Em seguida, crie as classes Jewel e Food que herdam as propriedades da classe Item.
2. **Treasure:** Implemente a classe Treasure que poderá conter várias joias e comidas dinamicamente. Use **vector** para criar coleções destes elementos dentro da classe. Implemente métodos para adicionar e remover comida e joias do baú. Como o baú possui muitos itens, você precisará sobreescrivê-lo para retornar o valor total do baú como sendo a soma dos valores individuais de cada item. Você também precisará usar sobreescrita de métodos para implementar o método de adição de elementos no baú. Para isso, crie o método *add()* usando sobreescrita para permitir a adição de itens que sejam comida e joias no baú.
3. **Map:** A classe Map deverá armazenar as informações do mapa 2D e implementar métodos para adição e remoção de itens no ambiente. Além de um método para imprimir o mapa na tela. A impressão do mapa deverá seguir a seguinte regra: Jewel como J, Treasure como T; Espaços vazios como ' '.

Crie um mapa com dimensões 10x10, insira algumas joias e um baú no ambiente. A quantidade de joias individuais e a posição de todos os elementos do mapa são definidas a seu critério. Ao exibir o mapa na tela certifique-se de imprimir a quantidade de joias e de comida que existe no baú, assim como os valores totais desses elementos. Teste se as funções de adicionar/remover itens do mapa e do baú estão funcionando corretamente. O programa é encerrado

apenas quando o usuário digita "exit" no prompt, caso contrário, o usuário pode optar por modificar o mapa ou exibi-lo na tela.

Passo 3: Criando o Arquivo Makefile

Um arquivo Makefile desempenha um papel fundamental na compilação de projetos maiores e mais complexos em linguagens de programação como C++ ou C. Ele automatiza o processo de compilação e torna a construção do projeto mais eficiente e organizada, permitindo o gerenciamento de dependências, a compilação incremental, a compatibilidade de plataforma e a documentação do processo de compilação.

Para o nosso projeto, crie um arquivo Makefile para compilar o programa jogo.cpp em um executável chamado **jogo**. Crie uma opção de apagar todos os .obj e todos os .exe criados.

Passo 4: Compilação e Execução

Abra um terminal no diretório onde os arquivos matriz.cpp e Makefile estão localizados. Use os comandos make para compilar o programa e, em seguida, executar o executável gerado.