INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: ALGORITMOS E ESTRUTURAS DE DADOS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: C++: NOÇÕES BÁSICAS 1

Prática 01

As práticas são descritas considerando o IDE Eclipse (versão 2022.3).
O estudante tem a liberdade de usar qualquer IDE que tenha familiaridade para o desenvolvimento C++ desde que consiga realizar a depuração (debug) no IDE.

Parte 1: Preparação inicial

Passo 0: Verifique o ambiente:

Verifique se o Eclipse se encontra instalado, se o CDT (programação C/C++) está instalado no Eclipse, e se existe um ambiente de compilação C/C++ instalado (por exemplo, **MinGW (preferencialmente)** ou Cygwin; se certificar que os pacotes **gcc**, **g++**, **gdb**, e **make** estão instalados.). Na dúvida, siga os próximos passos e veja se é possível criar e compilar um projeto C++ no eclipse.

Passo 1: Criando um novo projeto

1. Abra o Eclipse e selecione *File > New > C++ Project*;
2. Dê um nome adequado a seu projeto:

   Project name: Pratica01 [Nome default do executável que será gerado.]

   Project type: Empty Project

   Toolchains: MinGW ou Cygwin [Depende do ambiente C++ instalado.]

   Deixe os demais campos com os valores padrão e pressione o botão *Finish*.

Passo 2: Crie um novo arquivo fonte chamado **pratica01.cpp**.

Com o projeto selecionado: *File > New > Source File*. Dê o nome da janela aberta.

Passo 3: Em **pratica01.cpp**, crie a função `main()` exibindo apenas a mensagem "Primeira aplicação C++" usando a saída padrão de C++.

Use `cout << "string" << endl` para exibir a mensagem. Lembre-se de usar `#include <iostream>` e `using namespace std` para poder usar **`cout`**.

Passo 4: Compile a aplicação: *Project->Build all* ou *Ctrl+B*.

Verifique se houve erros durante o processo e corrija se necessário.

Passo 5: Rode a aplicação para testar:

Em *Run->Run configurations...*, selecione "*C/C++ Application*" e clique em "*New launch configuration*". O ambiente deve preparar uma configuração de execução com os dados do projeto aberto. Se não fizer, certifique-se de fazer a compilação antes. Nesse ponto já é possível rodar a aplicação clicando no botão *Run*. **(Alternativamente, clique no projeto com o botão direito e selecione "Run As ..." -> "Local C/C++ Application".)**

Parte 2: Criando e instanciando uma classe

Passo 1: Em **pratica01.cpp**, declare uma classe chamada `Veiculo` antes da função `main()`.

Não se esqueça do ponto-e-vírgula (;) depois da definição da classe.

Passo 2: Adicione um atributo chamado `nome` com tipo `string` a Classe com nível de visibilidade `private`.

A classe `string` faz parte da biblioteca padrão do C++.

Passo 3: Crie um construtor público (`public:`) no corpo da classe `Veiculo`, que receba um parâmetro do tipo `const char *` e o use para inicializar o atributo **`nome`**.

Para atribuir o atributo **`nome`** use `this->nome = string(param)`. (Essa sintaxe cria um objeto temporário da classe `string` a partir de `param` e esse objeto é copiado sobre o atributo `nome`). Faça também com que uma mensagem seja lançada na tela, informando que um novo objeto foi construído junto com o nome dele.

Passo 4: Adicione um destrutor à classe `Veiculo`.

Como no construtor, exiba na tela que o objeto foi destruído (incluindo o nome).

Passo 5: adicione o código abaixo no método `main()` de **pratica01.cpp**.

```
{
        Veiculo veiculo1("v1");

        {
                Veiculo veiculo2("v2");

                {
                        Veiculo veiculo3("v3");

                }
        }
}
```

ATENÇÃO: O blocos no código acima "{ ... }" são apenas para delimitar escopos e testar a destruição automática de objetos; não tem relação com os objetos em si, que podem vir declarado em qualquer lugar.

Passo 6: Compile e rode a aplicação, verificando a saída no console.

Veja que a ordem de criação e destruição é determinada pelos blocos onde os objetos são declarados. Ao final do bloco, os objetos criados são destruídos implicitamente.

Parte 3: Instanciando objetos no *Heap*

Passo 1: Modifique a função `main()`:

Faça com que os objetos sejam declarados como ponteiros e instanciados com o operador `new`, por exemplo:

```
Veiculo * obj1 = new Veiculo("v1");
```

ATENÇÃO: Comente o código da parte anterior para fins de apresentação.

Passo 2: Compile e rode a aplicação, verificando a saída no terminal.

Veja que agora os objetos são apenas criados; os destrutores não são chamados. É preciso usar o operador `delete` explicitamente para que isso aconteça.

Passo 3: Faça a destruição explícita de cada um dos objetos. (`delete`).

Como os objetos estão declarados dentro de blocos, a destruição deve ocorrer dentro deles. Do contrário, os ponteiros serão perdidos e haverá vazamento de memória.

Passo 4: Compile e rode a aplicação novamente.

A saída deverá ser igual à obtida inicialmente por causa dos blocos; se os objetos forem acessíveis em outros escopos, podem ser destruídos em uma ordem qualquer.

Parte 4: Trabalhando com visibilidade de membros

Passo 1: Na classe `Veiculo`, coloque um atributo inteiro chamado `num_rodas` que registra o número de rodas do veículo.

Faça com que o nível de visibilidade desse atributo seja privado (`private`).

Passo 2: Crie métodos públicos `setNumRodas()` e `getNumRodas()`, para setar e obter o número de rodas (`num_rodas`) respectivamente.

Declare as assinaturas dos métodos dentro da classe, porém implemente o corpo fora da classe, usando o operador `::` com a sintaxe abaixo:

```
<tipo> Classe::metodo(<parametros>) { <corpo> }
```

Passo 3: Na função `main()`, faça uso dos métodos criados, configurando o número de rodas dos veículos criados e obtendo eles em seguida.

Passo 4: Rode e teste a aplicação, verificando se os métodos estão funcionando adequadamente.

Passo 5: Crie um arquivo chamado **veiculo.h**: *New > Header File*.

Mova para esse arquivo a definição da classe `Veiculo` (mas não as implementações dos métodos). Dê o `#include` desse novo arquivo no **pratica01.cpp** usando “ ”.

Deve ser preciso incluir `<iostream>` e usar o *namespace* `std` para usar as funções de entrada e saída nos construtores.

Passo 6: Crie um novo arquivo fonte chamado **veiculo.cpp**.

Mova para esse arquivo as implementações dos métodos `setNumRodas()` e `getNumRodas()` da classe `Veiculo`. Verifique se será necessário incluir bibliotecas para que o arquivo compile adequadamente.

Passo 7: Rode e teste novamente, verificando se a compilação e execução ocorrem como esperado.

Parte 5: Trabalhando com sub-objetos

Passo 1: Defina uma nova classe chamada `Roda` no arquivo **veiculo.h** antes da classe `Veiculo`.

> Adicione construtor e destrutor padrão informando que o objeto foi construído/destruído, como em `Veiculo`, mas sem dizer o nome do objeto.

Passo 2: Na classe `Veiculo`, adicione um atributo privado chamada `rodas`, do tipo ponteiro para `Roda`.

> Inicialize esse atributo no construtor com valor `NULL`. Esse atributo será usado como um array dinâmico de rodas, cujo tamanho deve ser igual a `num_rodas`.

Passo 3: No método `setNumRodas()`, além de setar `num_rodas`, passe a instanciar o atributo `rodas` como um array do tipo `Roda` do tamanho especificado no parâmetro.

Passo 4: Rode e teste a aplicação.

> Verifique que ao instanciar o array de `Rodas`, os objetos do tipo `Roda` também foram automaticamente construídos. Isto é, diferente de Java, não é um array de ponteiros (ou referências) para `Roda`, mas de objetos do tipo `Roda`.
>
> Verifique também que ao destruir um `Veiculo`, os objetos `Roda` não foram destruídos automaticamente. É preciso modificar o destrutor de `Veiculo` adequadamente.

Passo 5: Modifique o destrutor de `Veiculo`, desalocando o array de objetos Roda.

> Use a sintaxe a `delete [] array`, ou será desalocando apenas o 1º elemento.

Passo 6: Rode e teste a aplicação.

> Verifique que dessa vez todos os objetos são destruídos adequadamente.

Parte 6: Trabalhando com o depurador

Passo 1: Coloque breakpoints nos construtores das classes `Veiculo` e `Roda`.

> No Eclipse, use duplo clique no número da linha ou o botão direito do mouse.

Passo 2: Rode a aplicação em modo de depuração.

> Use o menu *Run > Debug* ou *F11*.

Passo 3: Depois que a aplicação para no breakpoint, execute passo a passo para ver a ordem de criação dos objetos.

> Use *Run > Step Over* (*F6*) para passar por uma linha e *Run > Step Into* (*F5*) para entrar nela.

Parte 7 (Desafio/Opcional): Verificando o desempenho de funções *inline (Toolchain Cygwin)*

Passo 1: Configure o projeto para utilizar C++11.

Vá em *Project Properties > C++ Build > Settings*: no caso do Cygwin procure por "Dialect" e selecione "ISO C++11" em "Language standard" (Essa configuração permite usar o código a seguir)

Passo 2: Inclua a biblioteca **chrono** no seu programa, usada para medições de tempo:

```
#include <chrono>
```

Passo 3: Testando o desempenho das funções *inline*:

O código abaixo mede o tempo transcorrido para executar o trecho `//codigo`**:**

```
auto start = std::chrono::high_resolution_clock::now();

// codigo

auto finish = std::chrono::high_resolution_clock::now();
long elapsed = std::chrono::duration_cast<std::chrono::nanoseconds>
              (finish-start).count() / 1000;

cout << "tempo[us] = " << elapsed << endl;
```

Usando esse trecho de código, verifique a diferença no tempo de execução das funções de `Veiculo` quando declaradas *inline* e "*outline*" (fora da classe). Como os tempo são muito pequenos, é preciso fazer isso dentro de um laço (`for`) que executa milhões de chamadas às funções.