Laboratório - Fundamentos de C++

***Nota 1**: O código de exemplo "5.2-ponto-class-cpp-mais-com-cara-de-cpp" pode ser usado como referência para ajudar neste lab.*

***Nota 2**: Ao final do arquivo .*

Neste laboratório vamos criar uma classe chamada Horario contendo horas, minutos e segundos. Escreveremos a classe de forma que seja possível mostrar horários na tela na tela, medir a diferença de tempo entre dois horários e somar dos horários.

1) Estrutura do Projeto:

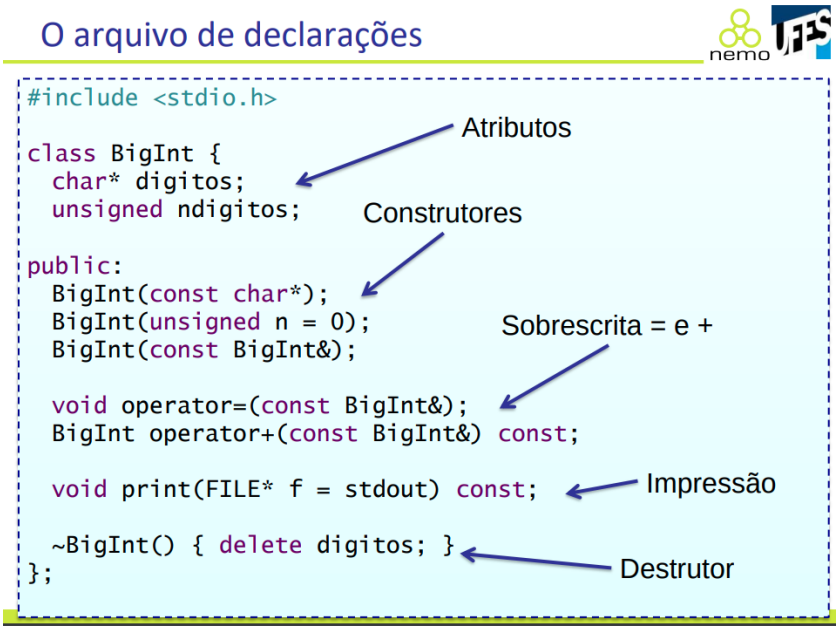
Os projetos em C++ tipicamente possuem bibliotecas, desenvolvidas de forma a serem reutilizáveis em outros projetos, e programas principais que usam estas bibliotecas para resolver problemas. No nosso projeto, criaremos uma biblioteca simples que vai conter apenas a classe Horario e um programa principal para testar as funcionalidades da biblioteca.

Crie uma pasta para o projeto e dentro dela construa a seguinte arquitetura de arquivos:

- horario/horario.h: arquivo de cabeçalho com a declaração da classe.
- horario/horario.cpp: arquivo com a implementação dos métodos da classe.
- programa.cpp: código-fonte do programa principal.
- Makefile: arquivo contendo as instruções de compilação do projeto.

2) Criação de Classes e de Objetos

No arquivo horario.h, crie a classe Horario contendo como atributos públicos os valores de horas, minutos e segundos. Para relembrar, a estrutura básica de classes é dada abaixo:



Em seguida, no arquivo main.cpp, crie a função main e, nela, crie dois objetos da classe Horario como abaixo:

```
Horario h1;
Horario h2;
```

Preencha os valores de horas, minutos e segundos e mostre os valores na tela usando o cout. Lembre-se de:

- Incluir as bibliotecas iostream e horario/horario.h
- Definir que o namespace std será utilizado adicionando “using namespace std;” após os includes.
- Retornar 0 ao final da função main.

3) Compilação

Para testar se o programa está funcionando, precisamos compilá-lo e executá-lo. Para isto, adicione as intruções abaixo no arquivo Makefile:

```
all:
	g++ -o programa programa.cpp horario/horario.cpp
clean:
	rm -rf programa
```

O alvo “all” é utilizado para criar executável “programa” a partir dos arquivos programa.cpp e horario.cpp. O alvo “clean” é utilizado para remover os arquivos gerados do projeto (por enquanto, apenas o arquivo executável “programa”).

Abra um terminal no VSCode através do menu Terminal > New Terminal (ou Novo Terminal). No terminal, digite “make”+Enter para compilar o programa. Se tudo estiver certo, nenhum erro aparecerá. Para executar o programa, digite “./programa”+Enter. Verifique que os valores que você mostrou na tela de fato apareceram.

Agora é com você: Modifique o arquivo programa.cpp de forma a criar os objetos usando o comando new, como fazíamos em Java. Você consegue se lembrar do que será diferente no programa? Após realizar as mudanças, compile e teste o código novamente.

*Se encontrar dificuldade, veja as respostas ao final e **preste atenção** no detalhe ao final da main. Você lembrou daquele detalhe? :)*

4) Construtores e Destrutores

Vamos modificar a classe Horario para adicionar um construtor que recebe os valores de horas, minutos e segundos, e um construtor de cópia. O arquivo horario.h deve conter a assinatura dos construtores, enquanto as implementações devem ser escritas no arquivo horario.cpp. Para ilustrar, façamos juntos o construtor de cópia. No arquivo horario.h:

```
class Horario
{
public:
        // (...)
        Horario(const Horario &h);
};
```

No arquivo horario.cpp:

```
#include "horario.h"

Horario::Horario(const Horario &h)
{
        horas = h.horas;
        minutos = h.minutos;
        segundos = h.segundos;
}
```

Agora é com você:

- Escreva o construtor que recebe os valores de horas, minutos e segundos.
- Escreva também um destrutor que por enquanto não faz nada. Lembre-se que o destrutor tem o mesmo nome da classe, mas começando com til (~).

- **[Possível Questão de Prova]** Use o comando cout para mostrar mensagens **diferentes** na tela quando os construtores e o destrutor forem chamados. Em seguida, modifique o arquivo programa.cpp como indicado abaixo. Quantas vezes os construtores e o destrutor serão invocados e onde? Execute o programa e verifique se você acertou. Se não, tente identificar os pontos em que as funções foram chamadas e você não tinha antecipado.

```
#include "horario/horario.h"
#include <iostream>

using namespace std;

void funcao_f(Horario h)
{
	cout << "funcao_f chamada" << endl;
}

void funcao_g(Horario &h)
{
	cout << "funcao_g chamada" << endl;
}

void funcao_h(Horario *h)
{
	cout << "funcao_h chamada" << endl;
}

int main()
{
	Horario h1(10, 4, 55);
	Horario h2(2, 12, 23);
	Horario h3(h1);
	Horario *h4 = new Horario(h2);
	Horario h5 = h1;
	h3 = h1;

	funcao_f(h1);
	funcao_g(h1);
	funcao_h(&h1);
	funcao_h(h4);

	// Lembrou deste detalhe importantíssimo?
	// A memoria alocada ao usar o "new" não é desalocada a menos que que o comando delete
	// seja utilizado.
	delete h4;
```

```
        return 0;
}
```

5) Entrada de dados

Modifique a função main para ler dados de horas, minutos e segundos usando o comando cin e use estes dados para criar um objeto da classe Horario.

6) Métodos de acesso aos atributos

Atualmente, a classe Horario permite que sejam atribuídos valores que não fazem sentido para os atributos, e.g., segundos = 537. Torne os atributos privados e crie métodos para atribuir e recuperar os valores dos atributos. Nos métodos, verifique se os valores são válidos e, se não forem, exiba uma mensagem de erro e não atualize os valores.

Depois de verificar que o trecho acima está funcionando, crie uma classe de exceção e lance-a quando um valor inválido para os atributos for encontrado.

7) Sobrecarga do operadores

Podemos mudar a forma como os operadores se comportam com objetos da nossa classe em C++. Nesta tarefa, vamos sobrescrever os seguintes operadores:

igual (operator=) : atribuir valores para os atributos.

stream (operator<<): define como os atributos são exibidos ao utilizar um objeto como entrada para o cout ou para um arquivo.

menos (operator-) : calcular a diferença entre dois horários.

mais (operator+) : calcular o tempo total ao somar dois horários.

A declaração dos operadores no arquivo horario.h será:

```
#include <iostream>

class Horario
{
        //(...)
        Horario operator+(const Horario &h) const;
        Horario operator-(const Horario &h) const;
        Horario& operator=(const Horario &h);
        friend ostream& operator<<(ostream &stream, const Horario &h);
```

```
};

ostream& operator<<(ostream &stream, const Horario &h);
```

[Possível questão de prova] Complete as implementações no arquivo horario.cpp:

```
Horario Horario::operator+(const Horario &h) const
{
        // (...)
}

Horario Horario::operator-(const Horario &h) const
{
        // (...)
}

Horario& Horario::operator=(const Horario &h)
{
        // (...)
}

ostream& operator<<(ostream &stream, const Horario &h)
{
        // (...)
}
```

[Possível questão de prova] Tente recordar:

- O significado do símbolo de & nos argumentos dos métodos
- const antes do nome de argumentos de métodos
- const após os parênteses em métodos
- a palavra reservada friend e o porquê dela não aparecer no arquivo horario.cpp
- A diferença entre o operator<< e os demais, e por que ele aparece dentro e fora da classe o arquivo horario.h .
- Quando no código da função main do item (4) o operador= seria invocado.

8) Once-only headers

Suponha que um arquivo cabeçalho de nosso projeto use a classe Horario e que o programa principal iniclua os dois arquivos cabeçalho, o da classe Horario o que usa a classe Horario. Ao tentar compilar o

programa principal, acontecerá um erro informando que a classe Horário está sendo redefinida. O mesmo aconteceria se ao usar biblioteca, utilizássemos dois arquivos cabeçalho que compartilham um terceiro. Para simular este fato, faça o include do arquivo “horario.h” duas vezes no arquivo programa.cpp e tente compilar o programa.

Para resolver este problema, precisamos adicionar uma estrutura de diretivas de compilação no arquivo horario.h para garantir que ele só seja incluído no programa uma vez:

```
#ifndef HORARIO_H
#define HORARIO_H

#include <iostream>

class Horario
{
        // (...)
};

#endif
```

As diretivas #ifndef ... #endif garante que o trecho de código entre as diretivas só seja incluído se o valor HORARIO_H não estiver definido. Este valor tipicamente é escolhido de forma a ser igual ao nome do arquivo, possivelmente com um prefixo denotando o projeto e pacote de código (e.g., PROJETO_PACOTE_CODIGO_H). Na primeira vez que o arquivo é incluído o valor não foi definido ainda, então o trecho é incorporado. Contudo, a primeira ação é definir o valor HORARIO_H. A partir deste ponto, se o arquivo for incluído novamente, a diretiva #ifndef HORARIO_H retornará falso e o código do não será mais incorporado.

Estas estruturas são chamadas de guardas de include (*include guards)*. Se quiser saber mais sobre, a explicação da Wikipédia sobre o assunto é bem boa: https://en.wikipedia.org/wiki/Include_guard .

Muitos compiladores suportam uma forma simplificada de especificar que o arquivo cabeçalho deve ser incluído uma vez, mas este comando não é parte do padrão de C/C++:

```
#pragma once

#include <iostream>

class Horario
{
        // (...)
};
```

9) Criação de Bibliotecas

Nesta seção, vamos transformar o código-fonte da classe Horario em uma biblioteca que possa ser usada em vários projetos. Para isto, além do ajuste no arquivo de cabeçalho feito na seção anterior, vamos modificar o arquivo Makefile:

```
# flags usadas no g++
# O -Wall ativa todos os warnings, mesmo aqueles que por padrao estao
# desligados por serem questoes menores.
FLAGS = -Wall

# Lista todos os arquivos com extensao .cpp que existem dentro da
# pasta horario
SRC = $(wildcard horario/*cpp)

# Substitui a extensao dos arquivos .cpp por .o nos arquivos existentes
# em SRC (ver linha acima)
OBJ = $(SRC:.cpp=.o)

# Target principal do makefile. Invoca os targets programa-v1 programa-v2 programa-v3
# listados abaixo.
all: programa

# Para todo arquivo .o, define que ele depende de um .cpp com mesmo nome.
# Sempre que o .cpp for modificado ou quando o .o não existir, sera' executado
# o comando abaixo para compilar o arquivo .cpp e produzir o arquivo .o.
%.o: %.cpp
	g++ $(FLAGS) -c $*.cpp -o $*.o

# A libhorario.a depende de todos os arquivos objeto existentes em OBJ.
# Sempre que os arquivos .o mudarem, e.g., como resultado da regra acima,
# o comando abaixo eh usado para produzir a biblioteca.
libhorario.a: $(OBJ)
	ar rsc libhorario.a $(OBJ)

# O programa depende da biblioteca e do arquivo .cpp . Sempre que um dos dois
# for modificado, sera' executado o comando  abaixo do target para compilar o
# arquivo .cpp e linkar o executável.
programa: libhorario.a programa.cpp
	g++ $(FLAGS) -o programa programa.cpp -L . -lhorario

# Remove os arquivos produzidos pelo processo de compilação e linkagem.
clean:
	rm -rf programa libhorario.a *.o horario/*.o
```

Após modificar o Makefile, use o comando make para produzir o executável. Em seguida, use o comando make novamente e veja que nada é feito dado que o código não foi modificado. Modifique o código, execute o comando make e verifique que dessa vez o programa foi compilado. Use o comando make clean para limpar os arquivos gerados no processo de geração da biblioteca e do executável.

Soluções

2) Criação de Classes e de Objetos

```
class Horario
{
public:
	int horas, minutos, segundos;
};
```

3) Criação de Classes e de Objetos

Programa.cpp: versão 1

```
#include "horario/horario.h"
#include <iostream>


using namespace std;


int main()
{
		Horario h1;
		Horario h2;


		h1.horas = 7;
		h1.minutos = 12;
		h1.segundos = 5;


		h2.horas = 2;
		h2.minutos = 55;
		h2.segundos = 30;
```

```
        cout << h1.horas << " " << h1.minutos << " " << h1.segundos << endl;
        cout << h2.horas << " " << h2.minutos << " " << h2.segundos << endl;


        return 0;
}
```

Programa.cpp: versão 2 (seção **agora é com você**)

```
#include "horario/horario.h"
#include <iostream>

using namespace std;

int main()
{
        Horario *h1 = new Horario();
        Horario *h2 = new Horario();

        h1->horas = 7;
        h1->minutos = 12;
        h1->segundos = 5;

        h2->horas = 2;
        h2->minutos = 55;
        h2->segundos = 30;

        cout << h1->horas << " " << h1->minutos << " " << h1->segundos << endl;
        cout << h2->horas << " " << h2->minutos << " " << h2->segundos << endl;

        // Lembrou deste detalhe importantíssimo?
        // A memoria alocada ao usar o "new" não é desalocada a menos que que o comando delete
        // seja utilizado.
        delete h1, h2;

        return 0;
}
```

4) Construtores e Destrutores

horario.h

```
class Horario
{
public:
	int horas, minutos, segundos;

	Horario(int horas, int minutos, int segundos);
	Horario(Horario &h);
	~Horario();
};
```

horario.cpp

```
#include <iostream>
#include "horario.h"

using namespace std;

Horario::Horario(int horas, int minutos, int segundos)
{
	this->horas = horas;
	this->minutos = minutos;
	this->segundos = segundos;
	cout << "Construtor" << endl;
}

Horario::Horario(Horario &h)
{
	horas = h.horas;
	minutos = h.minutos;
	segundos = h.segundos;
	cout << "Construtor de Copia" << endl;
}

Horario::~Horario()
{
	cout << "Destrutor" << endl;
}
```