Linguagens de Programação

Prof. Miguel Elias Mitre Campista

http://www.gta.ufrj.br/~miguel

Parte IV

Introdução à Programação em C++
(Continuação)

Relembrando da Última Aula...

- Ponteiros e strings
- Mais exemplos de programação orientada a objetos...

Empacotador de Pré-processadores

- Evita que o código seja incluído mais de uma vez
  - `#ifndef` – "se não definido"
    - Pula esse código se já tiver sido incluído
  - `#define`
    - Define um nome para que esse código não seja incluído novamente
  - `#endif`
  - Se o cabeçalho tiver sido incluído previamente
    - O nome estará definido e o arquivo .h não será incluído novamente
  - **Evita erros de múltiplas definições**

```
#ifndef TIME_H
#define TIME_H
... // code
#endif
```


Empacotador de Pré-processadores

- Utilize diretivas de pré-processador `#ifndef`, `#define` e `#endif` para formar um empacotador de pré-processador
  - O empacotador impede que os arquivos de cabeçalho sejam incluídos mais de uma vez em um programa
- Utilize o nome do arquivo do cabeçalho em caixa alta
  - Substitua o ponto por um sublinhado nas diretivas de pré-processador `#ifndef` e `#define` de um arquivo de cabeçalho

arquivo.h → ARQUIVO_H

Primeiro Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 1
 * Arquivo header
 * Autor: Miguel Campista
 */
#ifndef TIME_H
#define TIME_H

class Time {
      public:
            Time ();
            void setTime (int, int, int);
            void printUniversal ();
            void printStandard ();
      private:
             int hour; // 0 - 23
             int minute; // 0 - 59
             int second; // 0 - 59
};

#endif
```

Primeiro Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 1
 * Arquivo timeCap9Ex1.cpp
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
#include "timeCap9Ex1.h"

Time::Time () {
      hour = minute = second = 0;
}

void Time::setTime (int h, int m, int s) {
      hour = (h >= 0 && h < 24) ? h : 0; // valida horas
      minute = (m >= 0 && m < 60) ? m : 0; // valida minutos
      second = (s >= 0 && s < 24) ? s : 0; // valida segundos
}

void Time::printUniversal () {
      cout << setfill('0') << setw(2) << hour << ":"
           << setw(2) << minute << ":" << setw(2) << second;
}

void Time::printStandard () {
      cout << ((hour == 0 || hour == 12) ? 12 : hour % 12) << ":"
           << setfill('0') << setw(2) << minute << ":" << setw(2)
           << second << (hour < 12 ? " AM" : " PM");
}
```


Primeiro Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "timeCap9Ex1.h"

int main () {
    Time t;

    cout << "A hora Universal inicial eh: ";
    t.printUniversal();
    cout << "\nA hora Padrao inicial eh: ";
    t.printStandard();

    t.setTime(13, 27, 6); // Altera a hora

    cout << "\n\nA hora Universal alterada eh: ";
    t.printUniversal();
    cout << "\nA hora Padrao alterada eh: ";
    t.printStandard();

    t.setTime(99, 99, 99);

    cout << "\n\nHora depois de tentar valores invalidos";
    cout << "\nHora Universal: ";
    t.printUniversal();
    cout << "\nHora Padrao: ";
    t.printStandard();
    cout << endl;

    return 0;
}
```

Primeiro Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "timeCap9Ex1.h"
```

```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula9-ex1.exe
A hora Universal inicial eh: 00:00:00
A hora Padrao inicial eh: 12:00:00 AM

A hora Universal alterada eh: 13:27:06
A hora Padrao alterada eh: 1:27:06 PM

Hora depois de tentar valores invalidos
Hora Universal: 00:00:00
Hora Padrao: 12:00:00 AM
Pressione qualquer tecla para continuar. . .
```

```
    t.setTime(99, 99, 99);

    cout << "\n\nHora depois de tentar valores invalidos";
    cout << "\nHora Universal: ";
    t.printUniversal();
    cout << "\nHora Padrao: ";
    t.printStandard();
    cout << endl;

    return 0;
}
```

Manipulador de Fluxo Parametrizado `setfill`

- Especifica o caractere de preenchimento
  - O qual é exibido quando um campo de saída é maior que o número de dígitos no valor de saída
  - Por padrão, os caracteres de preenchimento aparecem à esquerda dos dígitos no número
- `setfill` é uma configuração "aderente"
  - Aplica-se a todos os valores subsequentes que são exibidos nos campos maiores que o valor que está sendo exibido

Manipulador de Fluxo Parametrizado `setfill`

- Toda configuração aderente (como um caractere de preenchimento ou a precisão de ponto flutuante) deve ser restaurada à sua configuração anterior quando não for mais necessária
  - Se não o fizer, isso pode fazer com que posteriormente uma saída seja formatada incorretamente em um programa

Declaração de Funções fora da Definição da Classe

- Função declarada em uma definição de classe, pode ser definida fora da definição da classe
  - Ainda se mantém no escopo da classe
  - É conhecida apenas por outros membros da classe, **a menos que**
    - Seja referida por meio do **objeto da classe**
    - Referencie-se por meio de uma **referência para um objeto da classe**
    - Referencie-se por meio de um **ponteiro para um objeto da classe**
    - Referencie-se por meio de um **operador binário de resolução de escopo**

Declaração de Funções fora da Definição da Classe

- Função definida no corpo da definição de uma classe
  - O compilador C++ tenta colocar `inline` as chamadas para a função
- Função definida fora do corpo da definição de uma classe
  - Deve ser colocada como `inline` explicitamente
    - Se quiser ter chance do compilador torná-la `inline`!

Declaração de Funções fora da Definição da Classe

- Definir uma pequena função dentro da definição de classe não melhora a engenharia do software
  - Os clientes da classe são capazes de ver a implementação da função
  - Código-cliente deve ser recompilado se a definição da função mudar
- Apenas as funções mais simples e mais estáveis (isto é, cujas implementações provavelmente não mudarão) devem ser definidas no cabeçalho de classe

Estudo de Caso da Classe `Time`

- Assim que definir uma classe `Time`, ela pode ser usada em declarações
  - `Time sunset;`
    - O que faz a sentença acima?
  - `Time arrayOfTimes[ 5 ];`
    - E essa?
  - `Time &dinnerTime = sunset;`
    - E essa?
  - `Time *timePtr = &dinnerTime;`
    - E essa?

Estudo de Caso da Classe `Time`

- Assim que definir uma classe `Time`, ela pode ser usada em declarações
  - `Time sunset;`
    - Cria um objeto
  - `Time arrayOfTimes[ 5 ];`
    - Cria um array de objetos
  - `Time &dinnerTime = sunset;`
    - Cria uma referência para um objeto
  - `Time *timePtr = &dinnerTime;`
    - Cria um ponteiro para um objeto

Alocação de Memória para Funções

- Os objetos contêm apenas dados
  - Logo, são muito menores do que se também contivessem funções

Alocação de Memória para Funções

- Aplicação do operador `sizeof` a um nome de classe ou a um objeto dessa classe informará somente o tamanho dos dados da classe
  - O compilador cria uma (única) cópia das funções, separada de todos os objetos da classe
    - Todos os objetos da classe compartilham essa cópia única já que o código de função não é modificável
  - Cada objeto precisa de sua própria cópia dos dados da classe, porque os dados podem variar entre os objetos

Escopo de Classe e Acesso a Membros de Classe

- O escopo de classe contém:
  - Membros de dados
    - Variáveis declaradas na definição de classe
  - Funções-membro
    - Funções declaradas na definição de classe
- As funções não-membro são definidas no escopo de arquivo

Escopo de Classe e Acesso a Membros de Classe

- Dentro do escopo de classe
  - Os membros de classe podem ser acessados por todas as funções-membro
- Fora do escopo de classe
  - Os membros de classe `public` são referenciados por meio de um ***handle***
    - Um nome de objeto
    - Uma referência a um objeto
    - Um ponteiro para um objeto

Escopo de Classe e Acesso a Membros de Classe

- Variáveis declaradas em uma função-membro
  - Têm escopo de bloco
    - Variáveis locais
  - São conhecidas apenas por essa função
- Ocultando uma variável de escopo de classe
  - Em uma função-membro, defina uma variável com o mesmo nome de uma variável com escopo de classe
  - Essa variável oculta pode ser acessada colocando o nome da classe seguido pelo operador de resolução de escopo (`::`) antes do nome da variável

Escopo de Classe e Acesso a Membros de Classe

- Operador de seleção de membro ponto (`.`)
  - Acessa os membros do objeto
  - Usado com o nome de um objeto ou com uma referência a um objeto
- Operador de seleção de membro seta (`->`)
  - Acessa os membros do objeto
  - Usado com um ponteiro para um objeto

Segundo Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 2
 * Arquivo countCap9Ex2.h
 * Autor: Miguel Campista
 */
#ifndef COUNT_H
#define COUNT_H

#include <iostream>

using namespace std;

class Count {
      public:
            // Configura o valor do membro de dados private x
            void setX (int value) {
                  x = value;
            }
            // Imprime o valor do membro de dados private x
            void getX () {
                  cout << x << endl;
            }
      private:
             int x;
};

#endif
```


Segundo Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 2
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "countCap9Ex2.h"

int main () {
    Count counter; // Objeto counter
    Count *counterPtr = &counter; // Ponteiro para counter
    Count &counterRef = counter; // Referência para counter

    cout << "Atribui 1 a x e imprime usando o nome do objeto: ";
    counter.setX(1);
    counter.getX();

    cout << "Atribui 2 a x e imprime usando a referência para o objeto: ";
    counterRef.setX(2);
    counterRef.getX();

    cout << "Atribui 3 a x e imprime usando o ponteiro para o objeto: ";
    counterPtr->setX(3);
    counterPtr->getX();

    return 0;
}
```

Segundo Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 2
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "countCap9Ex2.h"
```

```
C:\Users\Miguel\Documents\UFR\disciplinas\linguagens\projetos\aula9-ex2.exe
Atribui 1 a x e imprime usando o nome do objeto: 1
Atribui 2 a x e imprime usando a referência para o objeto: 2
Atribui 3 a x e imprime usando o ponteiro para o objeto: 3
Pressione qualquer tecla para continuar. . .
```

```
        counter.getX();

        cout << "Atribui 2 a x e imprime usando a referência para o objeto: ";
        counterRef.setX(2);
        counterRef.getX();

        cout << "Atribui 3 a x e imprime usando o ponteiro para o objeto: ";
        counterPtr->setX(3);
        counterPtr->getX();

        return 0;
    }
```


Funções de Acesso e Funções Utilitárias

- Funções de acesso
  - Podem ler ou exibir dados
  - Podem testar a veracidade ou falsidade das condições
    - Essas funções, em geral, são chamadas de funções predicado
- Funções utilitárias (tb chamadas de fçs auxiliares)
  - Funções-membro `private` que suportam a operação das funções-membro `public` da classe
  - Não fazem parte da interface `public` da classe
    - Não são usadas pelos clientes de uma classe

Terceiro Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 3
 * Arquivo countCap9Ex3.h
 * Autor: Miguel Campista
 */
#ifndef SALESP_H
#define SALESP_H

#include <iostream>

using namespace std;

class SalesPerson {
    public:
        SalesPerson ();
        void getSalesFromUser ();
        void setSales (int, double);
        void printAnnualSales ();
    private:
        double totalAnnualSales (); // Protótipo da função utilitária
        double sales [12]; // estimativa de vendas mensais
};

#endif
```


Terceiro Exemplo Usando Classes em C++

Função utilitária

```
/*
 * Aula 9 - Exemplo 3
 * Arquivo countCap9Ex3.h
 * Autor: Miguel Campista
 */
#ifndef SALESP_H
#define SALESP_H

#include <iostream>

using namespace std;

class SalesPerson {
    public:
        SalesPerson ();
        void getSalesFromUser ();
        void setSales (int, double);
        void printAnnualSales ();
    private:
        double totalAnnualSales (); // Protótipo da função utilitária
        double sales [12]; // estimativa de vendas mensais
};

#endif
```


Terceiro Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 3
 * Arquivo countCap9Ex3.cpp
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
#include "salesCap9Ex3.h"

SalesPerson::SalesPerson () {
    for (int i = 0; i < 12; i++)
        sales [i] = 0.0;
}

void SalesPerson::getSalesFromUser () {
    double salesFigure;

    for (int i = 1; i <= 12; i++) {
        cout << "Entre com a quantidade de ofertas para o mes "
             << i << ": ";
        cin >> salesFigure;
        setSales (i, salesFigure);
    }
}
```


Terceiro Exemplo Usando Classes em C++

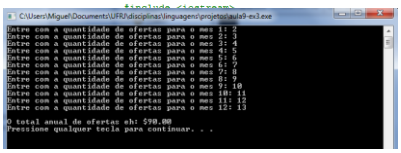
```
void SalesPerson::setSales (int month, double amount) {
    if (month >= 1 && month <= 12 && amount > 0)
        sales [month - 1] = amount;
    else
        cout << "Mes ou quantidade inválidos!" << endl;
}

void SalesPerson::printAnnualSales () {
    cout << setprecision(2) << fixed << "\nO total anual de ofertas eh: $"
         << totalAnnualSales () << endl;
}

double SalesPerson::totalAnnualSales () {
    double total = 0.0;
    for (int i = 0; i < 12; i++)
        total += sales [i];
    return total;
}
```

Terceiro Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 3
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "salesCap9Ex3.h"

int main () {
    SalesPerson s;

    s.getSalesFromUser ();
    s.printAnnualSales ();

    return 0;
}
```


Terceiro Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 3
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
```



Estudo de Caso da Classe `Time`

- Os construtores podem especificar argumentos-padrão
  - Podem inicializar membros de dados
    - Mesmo se não for fornecido nenhum valor em uma chamada de construtor
  - O construtor que assume um padrão para todos os seus argumentos é também chamado de construtor-padrão
    - Pode ser invocado sem nenhum argumento
    - É possível existir no máximo um construtor-padrão por classe

Quarto Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 4
 * Arquivo timeCap9Ex4.h
 * Autor: Miguel Campista
 */
#ifndef TIME_H
#define TIME_H

using namespace std;

class Time {
      public:
            Time (int = 0, int = 0, int = 0); // Construtor padrão

            void setTime (int, int, int);
            void setHour (int);
            void setMinute (int);
            void setSecond (int);
            int getHour ();
            int getMinute ();
            int getSecond ();

            void printUniversal ();
            void printStandard ();
      private:
            int hour; // 0 - 23
            int minute; // 0 - 59
            int second; // 0 - 59
};

#endif
```


Quarto Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 4
 * Arquivo timeCap9Ex4.cpp
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
#include "timeCap9Ex4.h"

// Construtor padrão inicializa cada membro de dados com zero;
// Logo, ele assegura que os objetos Time iniciem em um estado consistente
Time::Time (int hr, int min, int sec) {
      setTime (hr, min, sec);
}

void Time::setTime (int h, int m, int s) {
      setHour (h);
      setMinute (m);
      setSecond (s);
}

void Time::setHour (int h) {
      hour = (h >= 0 && h < 24) ? h : 0; // valida horas
}

void Time::setMinute (int m) {
      minute = (m >= 0 && m < 60) ? m : 0; // valida minutos
}

void Time::setSecond (int s) {
      second = (s >= 0 && s < 60) ? s : 0; // valida segundos
}
```


Quarto Exemplo Usando Classes em C++

```
int Time::getHour () { return hour; }

int Time::getMinute () { return minute; }

int Time::getSecond () { return second; }

void Time::printUniversal () {
      cout << setfill('0') << setw(2) << hour << ":"
           << setw(2) << minute << ":" << setw(2) << second;
}

void Time::printStandard () {
      cout << ((hour == 0 || hour == 12) ? 12 : hour % 12) << ":"
           << setfill('0') << setw(2) << minute << ":" << setw(2)
           << second << (hour < 12 ? " AM" : " PM");
}
```

Quarto Exemplo Usando Classes em C++

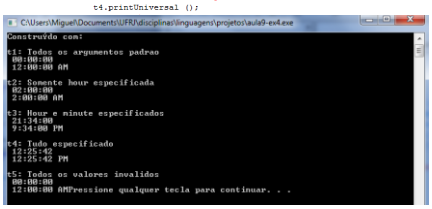
```
/*
 * Aula 9 - Exemplo 4
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "timeCap8Ex4.h"

int main () {
    Time t1; // Argumentos convertidos na config padrão
    Time t2 (2); // Hour especificada, o restante na config padrão
    Time t3 (21, 34); // Hour e minute especificados, second na config padrão
    Time t4 (12, 25, 42); // Hour, minute e second especificados
    Time t5 (27, 74, 99); // Valores inválidos

    cout << "Construido com:\n\nt1: Todos os argumentos padrao\n ";
    t1.printUniversal ();
    cout << "\n ";
    t1.printStandard ();

    cout << "\n\nt2: Somente hour especificada\n ";
    t2.printUniversal ();
    cout << "\n ";
    t2.printStandard ();

    cout << "\n\nt3: Hour e minute especificados\n ";
    t3.printUniversal ();
    cout << "\n ";
    t3.printStandard ();
```


Quarto Exemplo Usando Classes em C++

```
    cout << "\n\nt4: Tudo especificado\n ";
    t4.printUniversal ();
    cout << "\n ";
    t4.printStandard ();

    cout << "\n\nt5: Todos os valores invalidos\n ";
    t5.printUniversal ();
    cout << "\n ";
    t5.printStandard ();

    return 0;
}
```


Quarto Exemplo Usando Classes em C++

```
cout << "\n\nt4: Tudo especificado\n ";
t4.printUniversal ();
```



Construtores com Argumento Padrão

- Qualquer alteração nos valores de argumento-padrão de uma função exige que o código-cliente seja recompilado
  - Para assegurar que o programa permaneça funcionando corretamente
    - Mesma limitação encontrada em funções que declaram argumentos padrão

Destrutores

- Uma função-membro especial
- O nome é o caractere til (~) seguido pelo nome da classe
  - Por exemplo, `~Time`
- É chamado implicitamente quando um objeto é destruído
  - Por exemplo, isso ocorre quando um objeto automático é destruído porque a execução do programa deixou o escopo no qual esse objeto estava instanciado

Destrutores

- Não liberam a memória do objeto
  - Realizam uma "faxina de terminação"
  - Em seguida, o sistema reivindica a memória do objeto
    - Memória pode ser reutilizada para abrigar novos objetos

Destrutores

- Não recebem nenhum parâmetro e não retornam nenhum valor
  - Não especificam tipo de retorno (nem mesmo `void`)
    - É um erro de sintaxe:
      - Passar argumentos para um destrutor
      - Especificar um tipo de retorno (mesmo `void` não pode ser especificado)
      - Retornar valores de um destrutor
      - Sobrecarregar um destrutor

Destrutores

- Uma classe pode ter um único destrutor
  - **A sobrecarga de destrutores não é permitida**
- Se o programador não fornecer um destrutor explicitamente…
  - O compilador criará um destrutor "vazio"

Quando Construtores e Destrutores são Chamados?

- São chamados implicitamente pelo compilador
  - A ordem dessas chamadas de função depende da ordem segundo a qual a execução entra e sai dos escopos em que os objetos estão instanciados
- Geralmente,
  - As chamadas de destrutor são feitas na ordem inversa às chamadas de construtor correspondentes
    - **Último** objeto construído é o **primeiro** a ser destruído
- Entretanto,
  - As classes de armazenamento de objetos podem alterar a ordem segundo a qual os destrutores são chamados

Quando Construtores e Destrutores são Chamados?

- Para os objetos definidos no escopo global
  - Os construtores são chamados antes que qualquer outra função (incluindo `main`) nesse arquivo inicie a execução
  - Os destrutores correspondentes são chamados quando `main` termina

Quando Construtores e Destrutores são Chamados?

- Função `exit`
  - Força um programa a terminar imediatamente
    - Não executa os destrutores de **objetos automáticos**, mas executa os destrutores de objetos globais e estáticos
  - Em geral, é usada para terminar um programa quando é detectado um erro

```
A a;
void test() {
    static A b;
    A c;
    exit(0); // Não executa o destrutor de c
}
```


Quando Construtores e Destrutores são Chamados?

- Função `abort`
  - É semelhante à função `exit`
    - Mas força o programa a terminar imediatamente sem permitir que os destrutores de **qualquer objeto** sejam chamados
  - Normalmente, é usada para indicar uma terminação anormal do programa

```
A a;
void test() {
    static A b;
    A c;
    abort(); // Não executa o destrutor de a, b, c
}
```

Quando Construtores e Destrutores são Chamados?

- Para um objeto local automático
  - O construtor é chamado quando esse objeto é definido
  - O destrutor correspondente é chamado quando a execução sai do escopo do objeto
- Resumindo...
  - Os construtores e destrutores são chamados toda vez que a execução entra e sai do escopo do objeto
  - Os destrutores de objeto automático não serão chamados se o programa terminar com uma função `exit` ou `abort`

Quando Construtores e Destrutores são Chamados?

- Para um objeto local `static`
  - O construtor é chamado uma única vez
    - Quando a execução atinge pela primeira vez o local em que o objeto é definido
  - O destrutor é chamado quando `main` termina ou o programa chama a função `exit`
    - O destrutor não será chamado se o programa terminar com uma chamada para a função `abort`

Quando Construtores e Destrutores são Chamados?

- Os objetos global e `static` são destruídos na ordem inversa à que foram criados
  - Primeiro cria-se os objetos globais e depois os `static`
  - Inversamente, primeiro destrói-se os objetos `static` e depois os globais

Quinto Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 5
 * Arquivo createdestroy.h
 * Autor: Miguel Campista
 */
#ifndef CREATE_H
#define CREATE_H

#include <string>
#include <iostream>

using namespace std;

class CreateAndDestroy {
      public:
            CreateAndDestroy (int, string);
            ~CreateAndDestroy ();
      private:
            int objID;
            string message;
};

#endif
```


Quinto Exemplo Usando Classes em C++

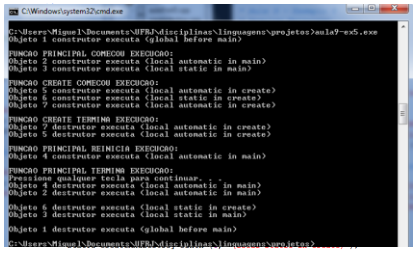
```
/*
 * Aula 9 - Exemplo 5
 * Arquivo createdestroy.cpp
 * Autor: Miguel Campista
 */
#include "createdestroy.h"

CreateAndDestroy::CreateAndDestroy (int ID, string messageString) {
      objID = ID;
      message = messageString;

      cout << "Objeto " << objID << " construtor executa "
           << message << endl;
}

CreateAndDestroy::~CreateAndDestroy () {
      cout << (objID == 1 || objID == 6 ? "\n" : "");
      cout << "Objeto " << objID << " destrutor executa "
           << message << endl;
}
```


Quinto Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 5
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include "createdestroy.h"

void create ();

CreateAndDestroy first (1, "(global before main)"); // objeto global

int main () {
      cout << "\nFUNCAO PRINCIPAL COMECOU EXECUCAO:" << endl;
      CreateAndDestroy second (2, "(local automatic in main)");
      static CreateAndDestroy third (3, "(local static in main)");

      create (); // Chama  função para criar objetos

      cout << "\nFUNCAO PRINCIPAL REINICIA EXECUCAO:" << endl;
      CreateAndDestroy fourth (4, "(local automatic in main)");
      cout << "\nFUNCAO PRINCIPAL TERMINA EXECUCAO:" << endl;

      return 0;
}

void create () {
      cout << "\nFUNCAO CREATE COMECOU EXECUCAO:" << endl;
      CreateAndDestroy fifth (5, "(local automatic in create)");
      static CreateAndDestroy sixth (6, "(local static in create)");
      CreateAndDestroy seventh (7, "(local automatic in create)");
      cout << "\nFUNCAO CREATE TERMINA EXECUCAO:" << endl;
}
```

Quinto Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 5
 * Arquivo Principal
 * Autor: Miguel Campista
```




```
            CreateAndDestroy seventh (7, "(local automatic in create)");
            cout << "\nFUNCAO CREATE TERMINA EXECUCAO:" << endl;
}
```


Estudo de Caso da Classe `Time`

- Retornando uma referência a um objeto
  - Alias para o nome de um objeto
    - Um *lvalue* aceitável que pode receber um valor
      - Pode ser usado no lado esquerdo de uma instrução de atribuição
    - Se uma função retornar uma referência `const`
      - Essa referência não poderá ser usada como um *lvalue* modificável

Estudo de Caso da Classe `Time`

- Retornando uma referência a um objeto
  - Uma forma "arriscada" de usar essa capacidade
    - Uma função-membro `public` de uma classe retorna uma referência a um membro de dados `private` dessa classe
      - O código-cliente poderia alterar os dados `private`
      - O mesmo problema ocorreria se retornasse um ponteiro para dados `private`
  - Retornar uma referência ou um ponteiro para um membro de dados `private` quebra o encapsulamento da classe

Sexto Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 6
 * Arquivo timeCap9Ex6.h
 * Autor: Miguel Campista
 */
#ifndef TIME_H
#define TIME_H

using namespace std;

class Time {
      public:
            Time (int = 0, int = 0, int = 0); // Construtor padrão

            void setTime (int, int, int);
            void setHour (int);
            void setMinute (int);
            void setSecond (int);
            int getHour ();
            int getMinute ();
            int getSecond ();

            void printUniversal ();
            void printStandard ();

            int &badSetHour (int); // retorno de referência "PERIGOSO"
      private:
             int hour; // 0 - 23
             int minute; // 0 - 59
             int second; // 0 - 59
};

#endif
```

Sexto Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 6
 * Arquivo timeCap9Ex6.h
 * Autor: Miguel Campista
 */
#ifndef TIME_H
#define TIME_H

using namespace std;

class Time {
      public:
            Time (int = 0, int = 0, int = 0); // Construtor padrão

            void setTime (int, int, int);
            void setHour (int);
            void setMinute (int);
            void setSecond (int);
            int getHour ();
            int getMinute ();
            int getSecond ();

            void printUniversal ();
            void printStandard ();

            int &badSetHour (int); // retorno de referência "PERIGOSO"
      private:
             int hour; // 0 - 23
             int minute; // 0 - 59
             int second; // 0 - 59
};

#endif
```

Protótipo de uma função que retorna uma referência

Sexto Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 6
 * Arquivo timeCap9Ex6.cpp
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
#include "timeCap9Ex6.h"

// Construtor padrão inicializa cada membro de dados com zero;
// Logo, ele assegura que os objetos Time iniciem em um estado consistente
Time::Time (int hr, int min, int sec) {
      setTime (hr, min, sec);
}

void Time::setTime (int h, int m, int s) {
      setHour (h);
      setMinute (m);
      setSecond (s);
}

void Time::setHour (int h) {
      hour = (h >= 0 && h < 24) ? h : 0; // valida horas
}

void Time::setMinute (int m) {
      minute = (m >= 0 && m < 60) ? m : 0; // valida minutos
}

void Time::setSecond (int s) {
      second = (s >= 0 && s < 60) ? s : 0; // valida segundos
}
```

Sexto Exemplo Usando Classes em C++

```
int Time::getHour () { return hour; }

int Time::getMinute () { return minute; }

int Time::getSecond () { return second; }

void Time::printUniversal () {
      cout << setfill('0') << setw(2) << hour << ":"
           << setw(2) << minute << ":" << setw(2) << second;
}

void Time::printStandard () {
      cout << ((hour == 0 || hour == 12) ? 12 : hour % 12) << ":"
           << setfill('0') << setw(2) << minute << ":" << setw(2)
           << second << (hour < 12 ? " AM" : " PM");
}

// Prática de programação não recomendada
int &Time::badSetHour (int hh) {
      hour = (hh >= 0 && hh < 24) ? hh : 0;
      return hour; // retorno de referência
}
```


Sexto Exemplo Usando Classes em C++

```
int Time::getHour () { return hour; }

int Time::getMinute () { return minute; }

int Time::getSecond () { return second; }

void Time::printUniversal () {
      cout << setfill('0') << setw(2
           << setw(2) << minute << ":
}

void Time::printStandard () {
      cout << ((hour == 0 || hour == 12) ? 12 : hour % 12) << ":"
           << setfill('0') << setw(2) << minute << ":" << setw(2)
           << second << (hour < 12 ? " AM" : " PM");
}

// Prática de programação não recomendada
int &Time::badSetHour (int hh) {
      hour = (hh >= 0 && hh < 24) ? hh : 0;
      return hour; // retorno de referência
}
```

Função que retorna uma referência para um atributo privado

Sexto Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "timeCap9Ex6.h"

int main () {
    Time t;

    // Inicializa hourRef com a referência retornada por badSetHour
    int &hourRef = t.badSetHour (20); // Hora válida

    cout << "Hora valida antes da modificacao: " << hourRef;
    hourRef = 30; // Usa hourRef para atribuir um valor inválido
    cout << "\nHora invalida depois da modificacao: " << t.getHour();

    // Perigoso: Chamada de função que retorna uma referência
    // pode ser usado como um lvalue!
    t.badSetHour (12) = 74;

    cout << "\n\n************************************************\n"
         << "Pratica perigosa de programacao!\n"
         << "t.badSetHour (12) como um lvalue, hora invalida: "
         << t.getHour ()
         << "\n\n************************************************\n"
         << endl;

    return 0;
}
```


Sexto Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "timeCap9Ex6.h"

int main () {
    Time t;
```

```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula9-ex6.exe
Hora valida antes da modificacao: 20
Hora invalida depois da modificacao: 30

************************************************
Pratica perigosa de programacao!
t.badSetHour (12) como um lvalue, hora invalida: 74

************************************************
Pressione qualquer tecla para continuar. . .
```

```
    cout << "\n\n************************************************\n"
         << "Pratica perigosa de programacao!\n"
         << "t.badSetHour (12) como um lvalue, hora invalida: "
         << t.getHour ()
         << "\n\n************************************************\n"
         << endl;

    return 0;
}
```


Atribuição-padrão Membro a Membro

- Operador de atribuição (=)
  - Pode ser usado para atribuir um objeto a outro objeto do mesmo tipo
    - Cada membro de dados do objeto à direita é atribuído ao mesmo membro de dados do objeto à esquerda
  - Isso pode provocar sérios problemas quando os membros de dados contêm ponteiros para memória alocada dinamicamente
    - Essa memória poderia ser desalocada...

Sétimo Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 7
 * Arquivo dateCap9Ex7.h
 * Autor: Miguel Campista
 */
#ifndef DATE_H
#define DATE_H

#include <iostream>

using namespace std;

class Date {
      public:
            Date (int = 1, int = 1, int = 2000);
            void print ();
      private:
            int month, day, year;
};

#endif
```

Sétimo Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 7
 * Arquivo dateCap9Ex7.cpp
 * Autor: Miguel Campista
 */
#include "dateCap9Ex7.h"

Date::Date (int m, int d, int y) {
      month = m;
      day = d;
      year = y;
}

void Date::print () {
      cout << month << '/' << day << '/' << year;
}
```


Sétimo Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 7
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include "dateCap9Ex7.h"

int main () {
    Date date1 (7, 4, 2010);
    Date date2; // Assume padrão 1/1/2000

    cout << "date1 = ";
    date1.print ();
    cout << "\ndate2 = ";
    date2.print ();

    date2 = date1; // Atribuição padrão de membro a membro

    cout << "\n\nDepois atribuicao padrao membro a membro, date2 = ";
    date2.print ();
    cout << endl;

    return 0;
}
```


Sétimo Exemplo Usando Classes em C++

```
/*
 * Aula 9 - Exemplo 7
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include "dateCap9Ex7.h"

int main () {
```

```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula9-ex7.exe
date1 = 7/4/2010
date2 = 1/1/2000

Depois atribuicao padrao membro a membro, date2 = 7/4/2010
Pressione qualquer tecla para continuar. . .
```

```
    date2 = date1; // Atribuição padrão de membro a membro

    cout << "\n\nDepois atribuicao padrao membro a membro, date2 = ";
    date2.print ();
    cout << endl;

    return 0;
}
```


Atribuição-padrão Membro a Membro

- Construtor de cópia
  - Permite que os objetos sejam passados por valor
    - É usado para copiar valores originais do objeto em um novo objeto passado a uma função ou que retornou de uma função
  - O compilador fornece um construtor-padrão de cópia
    - Copia cada membro do objeto original no membro correspondente do novo objeto (ou seja, é uma atribuição de membro a membro)
  - Também pode provocar sérios problemas quando os membros de dados contêm ponteiros para memória alocada dinamicamente

Atribuição-padrão Membro a Membro

- A passagem de um objeto por valor é adequada do ponto de vista de segurança
  - A função chamada não tem acesso ao objeto original no chamador, mas pode diminuir o desempenho ao fazer uma cópia de um objeto grande
- É possível passar um objeto por referência passando um ponteiro ou uma referência ao objeto
  - A passagem por referência oferece bom desempenho, mas menor segurança porque a função chamada recebe acesso ao objeto original

Atribuição-padrão Membro a Membro

- A passagem por referência `const` é uma alternativa segura de bom desempenho
  - Pode ser implementada com um parâmetro de referência `const` ou com um parâmetro de ponteiro para dados `const`

Oitavo Exemplo Usando Classes em C++

```
/* Aula 9 - Exemplo de Construtor de Cópia
 * Arquivo principal
 * Autor: Miguel Campista
*/
#include <iostream>
using namespace std;

class C {
        public:
        C (int s = 10) : size (s), data (new int [s]) {
                cout << "Construtor Padrão!" << endl;
        }
        ~C () { delete [] data; }
        void getSize () { cout << size << endl; }
        void getData () { cout << data [0] << endl; }

        private:
        int size;
        int *data;
};

int main () {
        C obj (10);
        {
                C copiaObj (obj);
                copiaObj.getSize ();
        }
        obj.getData ();
        return 0;
}
```


Oitavo Exemplo Usando Classes em C++

```
/* Aula 9 - Exemplo de Construtor de Cópia
 * Arquivo principal
 * Autor: Miguel Campista
miguel@pegasus-linux:~/UFRJ/disciplinas/linguagens/projetos$ ./a
Construtor Padrão!
10
0
*** glibc detected *** ./a: double free or corruption (fasttop): 0x0804a008 ***
======= Backtrace: =========
/lib/i686/cmov/libc.so.6[0xb7da1624]
/lib/i686/cmov/libc.so.6(cfree+0x96)[0xb7da3826]
/usr/lib/libstdc++.so.6(_ZdlPv+0x21)[0xb7f7a2e1]
/usr/lib/libstdc++.so.6(_ZdaPv+0x1d)[0xb7f7a33d]
./a(__gxx_personality_v0+0x2b0)[0x8048970]
./a(__gxx_personality_v0+0x1b4)[0x8048874]
/lib/i686/cmov/libc.so.6(__libc_start_main+0xe5)[0xb7d49455]
./a(__gxx_personality_v0+0x41)[0x8048701]
======= Memory map: ========
08048000-08049000 r-xp 00000000 00:11 1924       /mnt/ufrj/disciplinas/linguagens/projetos/
a
08049000-0804a000 rw-p 00000000 00:11 1924       /mnt/ufrj/disciplinas/linguagens/projetos/
a
0804a000-0806b000 rw-p 0804a000 00:00 0          [heap]
b7c00000-b7c21000 rw-p b7c00000 00:00 0
b7c21000-b7d00000 ---p b7c21000 00:00 0
b7d32000-b7d33000 rw-p b7d32000 00:00 0
b7d33000-b7e88000 r-xp 00000000 03:01 2523158    /lib/i686/cmov/libc-2.7.so
b7e88000-b7e89000 r--p 00155000 03:01 2523158    /lib/i686/cmov/libc-2.7.so
b7e89000-b7e8b000 rw-p 00156000 03:01 2523158    /lib/i686/cmov/libc-2.7.so
```


Oitavo Exemplo Usando Classes em C++

```
/* Aula 9 - Exemplo de Construtor de Cópia
 * Arquivo principal
 * Autor: Miguel Campista
*/
#include <iostream>
using namespace std;

class C {
        public:
        C (int s = 10) : size (s), data (new int [s]) {
                cout << "Construtor Padrão!" << endl;
        }
        C (const C &copia) : size (copia.getSize ()), data (new int [copia.getSize ()]) {
                cout << "Construtor Cópia!" << endl;
        }
        ~C () { delete [] data; }
        void getSize () { cout << size << endl; }
        void getData () { cout << data [0] << endl; }

        private:
        int size;
        int *data;
};

int main () {
        C obj (10);
        {
                C copiaObj (obj);
                copiaObj.getSize ();
        }
        obj.getData ();
        return 0;
}
```

Oitavo Exemplo Usando Classes em C++

```
/* Aula 9 - Exemplo de Construtor de Cópia
 * Arquivo principal
 * Autor: Miguel Campista
*/
#include <iostream>
using namespace std;

class C {
        public:
        C (int s = 10) : size (s), data (new int [s]) {
                cout << "Construtor Padrão!" << endl;
        }
        C (const C &copia) : size (copia.getSize ()), data (new int [copia.getSize ()]) {
                cout << "Construtor Cópia!" << endl;
        }
        ~C () { delete [] data; }
        void getSize () { cout << size << endl; }
        void getData () { cout << data [0] << endl; }

        private:
        int size;
        int *data;
};

int main () {
        C obj (10);
        {
                C copiaObj (obj);
                copiaObj.getSize ();
        }
        obj.getData ();
        return 0;
}
```

Construtor de Cópia

Oitavo Exemplo Usando Classes em C++

```
/* Aula 9 - Exemplo de Construtor de Cópia
 * Arquivo principal
 * Autor: Miguel Campista
*/
#include <iostream>
using namespace std;

class C {
        public:
        C (int s = 10) : size (s), data (new int [s]) {
                cout << "Construtor Padrão!" << endl;
        }
miguel@pegasus-linux:~/UFRJ/disciplinas/linguagens/projetos$ ./a
Construtor Padrão!
Construtor de Cópia!
10
0
        int size;
        int *data;
};

int main () {
        C obj (10);
        {
                C copiaObj (obj);
                copiaObj.getSize ();
        }
        obj.getData ();
        return 0;
}
```

Exemplo 1

- Escreva um programa que receba a largura e o comprimento de um retângulo e imprima as coordenadas cartesianas desse quadrado. Assuma que uma das coordenadas é o ponto (0, 0). O retângulo deve possuir uma largura e um comprimento default que devem ser alterados caso o usuário deseje.

?

```
/*
 * Aula 9 - Exemplo 8
 * Programa rectangleCap9Ex8.h
 * Autor: Miguel Campista
 */
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include <iostream>
#include "pointCap9Ex8.h"

using namespace std;

class Rectangle {
      public:
            Rectangle (float = 1, float = 1);
            ~Rectangle();

            float getLength ();
            float getWidth ();
            void getCoord ();
            void setLength ();
            void setWidth ();
            void setFourPoints ();

      private:
            float length, width;
            static const float max_length = 20, max_width = 20;

            Point points [4];

            bool checkLength (float);
            bool checkWidth (float);
};

#endif
```

```
/*
 * Aula 9 - Exemplo 8
 * Programa rectangleCap9Ex8.cpp
 * Autor: Miguel Campista
 */
#include "rectangleCap9Ex8.h"

Rectangle::Rectangle(float l, float w) {
      cout << "No construtor...\n";
      length = l;
      width = w;
      setFourPoints ();
}

Rectangle::~Rectangle() {
      cout << "No destrutor...\n";
}

float Rectangle::getLength () { return length; }

float Rectangle::getWidth () { return width; }

void Rectangle::getCoord () {
     for (int i = 0; i < 4; i++)
           points [i].getCoord ();
}

void Rectangle::setLength () {
     float l;
     cout << "Entre com o comprimento do quadrado: ";
     do {
          cin >> l;
     } while (checkLength (l));

     length = l;
}
```

Exemplo 1

```
void Rectangle::setWidth () {
     float w;
     cout << "Entre com a largura do quadrado: ";
     do {
          cin >> w;
     } while (checkWidth (w));

     width = w;
}

void Rectangle::setFourPoints () {
     for (int l = 0; l < 2; l++) {
          for (int w = 0; w < 2; w++)
               points[2*l + w].setCoord (l*length , w*width);
     }
}

bool Rectangle::checkLength (float x) {
     if ((x < 0) || (x > max_length)) {
          cout << "O comprimento deve estar entre 0 e 20.\n"
               << "Entre novamente com um novo comprimento.\n";
               return true;
     }
     return false;
}

bool Rectangle::checkWidth (float y) {
     if ((y < 0) || (y > max_width)) {
          cout << "O comprimento deve estar entre 0 e 20.\n"
               << "Entre novamente com um novo comprimento.\n";
               return true;
     }
     return false;
}
```

Exemplo 1

```
/*
 * Aula 9 - Exemplo 8
 * Programa pointCap9Ex8.cpp
 * Autor: Miguel Campista
 */
#include "pointCap9Ex8.h"

void Point::setCoord (float x, float y) {
     coord_x = x;
     coord_y = y;
}

void Point::getCoord () {
     cout << "(" << coord_x << ", " << coord_y << ")\n";
}
```


Exemplo 1

```
/*
 * Aula 9 - Exemplo 8
 * Programa pointCap9Ex8.h
 * Autor: Miguel Campista
 */
#ifndef POINT_H
#define POINT_H

#include <iostream>

using namespace std;

class Point {
      public:
            void setCoord (float, float);
            void getCoord ();
      private:
            float coord_x, coord_y;
};

#endif
```


Exemplo 1

```
/*
 * Aula 9 - Exemplo 8
 * Programa Principal
 * Autor: Miguel Campista
 */
#include "rectangleCap9Ex8.h"

using namespace std;

int main () {
    Rectangle r;
    string op;

    r.getCoord ();

    cout << "Mudar o valor padrao de comprimento e largura? (S/N) ";
    getline (cin, op);

    if (!op.compare("S") || !op.compare("s")) {
         r.setLength ();
         r.setWidth ();
         r.setFourPoints ();
         r.getCoord ();
    } else
         cout << "Tchau" << endl;

    return 0;
}
```

Leitura Recomendada

- Capítulos 9 do livro
  - Deitel, "*C++ How to Program*", 5th edition, Editora Prentice Hall, 2005