Curso de C/C++ Avançado

Aula 6 – Introdução ao C++

Allan Lima – http://allanlima.wordpress.com

creative commons

COMMONS DEED

SOME RIGHTS RESERVED

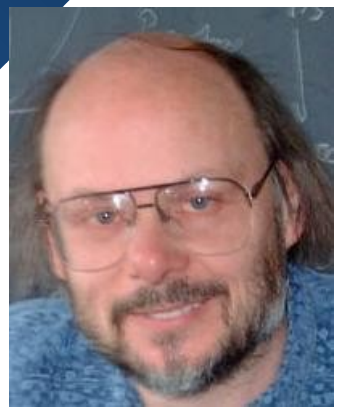
- Você pode:
  - copiar, distribuir, exibir e executar a obra
  - criar obras derivadas
  - fazer uso comercial da obra
- Sob as seguintes condições:
  - **Atribuição**. Você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante.
  - **Compartilhamento pela mesma Licença**. Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.
  - Para cada novo uso ou distribuição, você deve deixar claro para outros os termos da licença desta obra.
  - Qualquer uma destas condições podem ser renunciadas, desde que Você obtenha permissão do autor.
- **Veja aqui a licença completa**

Roteiro

- História da linguagem C++
- Diferenças entre C e C++
- Declaração de Classes
- Ponteiros para classes
- Herança

O Surgimento do C++

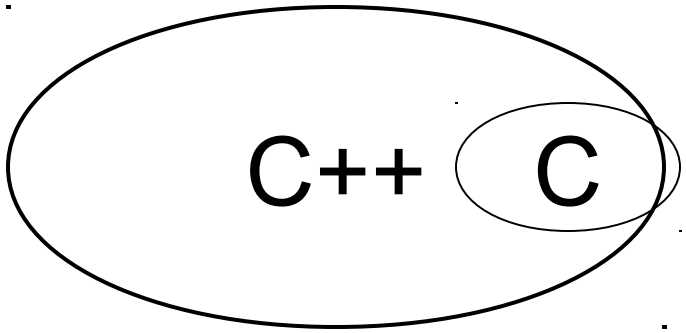
- Criada no Bell Labs em 1983
- Por Bjarne Stroustrup
- Possui a performance de C
- E as funcionalidades de outras linguagens como Simula e Algol
- Padronizada apenas em 1997



Bjarne Stroustrup

Comparativo entre C e C++

- Foi criada para ser C++ é uma extensão de C
- Mas não é 100% compatível com C



Comparativo entre C e C++

C	C++
Estruturada	Orientada a Objetos
malloc e calloc	new
free	delete
Passagem por valor	Passagem por referência
stdio	iostream
Variáveis declarada no início de um bloco	Variáveis declaradas em qualquer parte do bloco

Comparativo entre C e C++

C	C++
Inteiro como valor booleano	Tipo bool
Duas funções não podem ter o mesmo nome	Duas funções não podem ter o mesmo protótipo
Argumentos são sempre necessários	Valor defaut para os argumentos
Casts simples	Novos tipos de cast
String como array de caracteres	Tipo string

Exemplos

- *hello.cpp*
- *comparativo.cpp*

Classes

- São definições a partir das quais os objetos podem ser criados
- As classes determinam quais são os atributos e métodos de um objeto
- Sintaxe:

```
class nomeDaClasse {
    corpoDaClasse;
} listaDeObjetos;
```

Exemplo

```
class Retangulo {
    int largura;
    int altura;

    int area() {
        return largura * altura;
    }

    int perimetro() {
        return 2 *(largura + altura);
    }
} ret1; // declara uma variável do tipo Retangulo
```

Categorias de permissão

- Membros de uma classe podem ser:
  - public
    - Podem ser acessados em qualquer lugar
  - private
    - Só podem ser acessados pelos membros da própria classe
  - protected
    - Podem ser acessados apenas por membros da própria classe ou das suas sub-classes

Exemplo

```
class Retangulo {
        int largura;
    private:
        int altura;
    public:
        int area() {
            return largura * altura;
        }
    protected:
        int perimetro() {
            return 2 *(largura + altura);
        }
};
```

Obs.: Por default todo membro de uma classe é considerado **private**

```
// Exemplo de acesso:
int main() {
    Retangulo r;
    r.altura = 10; // Errado
    r.largura = 40; // Errado
    int a = r.area();
    a = r.perimetro(); // Errado
}
```

Classes

- Quando implementamos um método dentro de uma classe o compilador copia e cola o código toda vez que o método é chamado!
  - O método é dito inline
  - Isto torna o executável mais rápido
  - Mas deixa o executável bem maior
  - Só é bom para métodos muito curtos
- Qual a solução?
  - Utilizar o operador **::**

O Operador ::

- Permite a implementação de métodos fora da classe
- A classe passa a possuir apenas o protótipo do método
- O corpo pode ficar no mesmo arquivo ou em outro
- Sintaxe:
  - *nomeDaClasse::nomeDoMembro*
- Também podemos usar o modificador inline para que mesmo assim o método seja inline

Exemplo

```
class Retangulo {
    private:
        int largura;
        int altura;
    public:
       int area();
       int perimetro();
};

int inline Retangulo::area() { // força o método a ser inline
     return largura * altura;
}

int Retangulo::perimetro() {
     return 2 *(largura + altura);
}
```

Construtor

- É um método especial que é chamado quando criamos um novo objeto
- Deve possuir o mesmo nome da classe
- Não possui retorno
- É utilizado para inicializar os atributos da classe

Destrutor

- Método especial que é chamado automaticamente quando um objeto está prestes a ser apagado da memória
- Deve ter o mesmo nome da classe mas precedido por um ~
- Assim como o construtor ele não possui retorno
- Além disso, ele não pode ter parâmetros

Exemplo

```
class Retangulo {
    private:
        int largura;
        int altura;

    public:
     Retangulo(int a, int l);
     ~Retangulo() { } // destrutor padrão
}

Retangulo::Retangulo(int a, int l) {
     altura = a;
     largura = l;
};
```

Alocação de Memória

- new
  - Aloca memória para um objeto
  - Retorna um ponteiro para a posição alocada
  - Exemplo:

```
Retangulo *r = new Retangulo(10, 15);
Retangulo *array = new Retangulo[10];
```

- delete
  - Libera um região de memória alocada previamente
  - Exemplo:

```
delete r;
delete[] array;
```

Erros Comuns

```
// ...

Retangulo r = new Retangulo(10, 15);
// Errado: r não é um ponteiro!!!

delete r; // Errado: r não é um ponteiro!!!

// ...
```

Estruturas

- C++ permite a criação de estruturas com métodos
- Estas são praticamente idênticas às classes
- Porém todos os seu membros são public por default

Exemplo

```
struct Retangulo {
    int getAltura() { return altura; }; // public por default

    void setAltura(int a) { // public por default
        if (a > 0) altura = a;
    }

    private:
        int largura;
        int altura;
};
```

Modularizando o seu programa

- Quando queremos criar um projeto com diversas classes fazemos uso de algumas convenções:
  - Criamos um arquivo “.h” só com a definição da classe e os métodos inline
  - E um arquivo “.cpp” só com a implementação dos seus métodos

Exemplos

- *Retangulo.h*
- *Retangulo.cpp*
- *main.cpp*

Herança

- É um mecanismo utilizado para permitir o reuso de código
- Quando uma classe B herda de uma outra classe A dizemos que B é uma sub-classe de A
- Em C++ todos os membros public e protected da classe base são herdados
- Construtores, Destrutores, o operador = e os friends não são herdados

Herança

- Sintaxe
  - *class nomeDaClasse : tipoDaHeranca classeBase*
  - *struct nomeDaClasse : tipoDaHeranca classeBase*
  - *tipoDaHeranca:*
    - public – Os membros public e protected da classe base são por default public e protected respectivamente
    - private – Os membros public e protected da classe base são por default private
    - protected – Os membros public e protected da classe por default são protected
    - Se *tipoDaHeranca* não for especificado, private é assumido para as classes e public para as estruturas

Herança

- Apesar dos construtores e destrutores não serem herdados, quando criamos um objeto ele chama o construtor padrão de sua classe base e também o destrutor padrão quando ele está prestes a ser desalocado da memória
- Também podemos re-utilizar o código do construtor da classe base:
  - *construtorClasse(parametros) : construtorDaSuperClasse(parametros) { ... }*

Ponteiros e Herança

- Podemos criar ponteiros para classes
- A inicialização é feita através do operador new ou do &
- Quando usarmos o new também temos que usar o delete para liberar memória
- Restrições:
  - Um ponteiro para uma classe base só pode chamar métodos desta
  - Mas podemos usar ***casts*** para ter acesso aos demais métodos

Exemplo

- *exemploHeranca.cpp*

Herança Múltipla

- C++ permite que uma classe herde de várias outras
- Sintaxe:

```
class nomeDaClasse : tipoHeranca classeBase1,
                     // ...
                     tipoHeranca classeBaseN
```

- Mas isso traz problemas:
  - A e B possuem o método m
  - C herda de A e de B
  - O que acontece quando chamamos m de C?

Exemplo

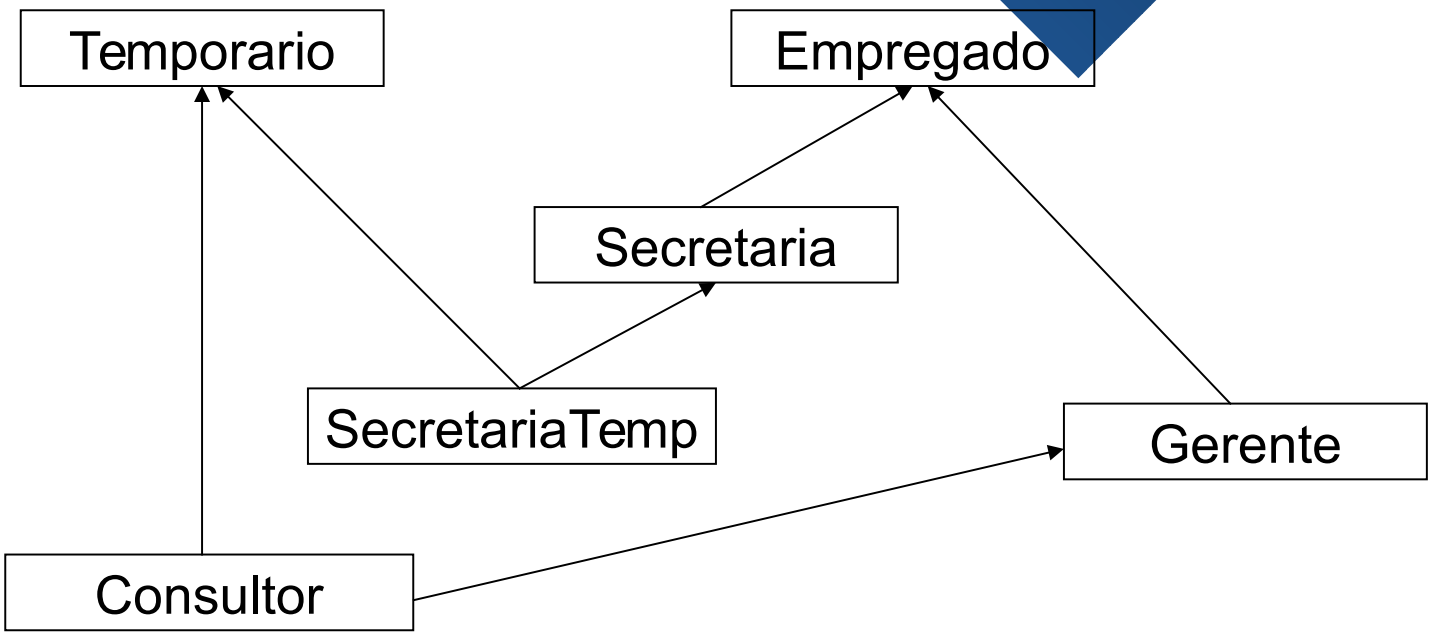
- *exemploHerancaMultipla.cpp*

Dicas Importantes

- Sempre crie um construtor vazio para sua classe
- Sempre utilize herança pública, a menos que tenha um bom motivo para fazer o contrário

Exercícios

1)Implemente a seguinte hierarquia de classes:



Classe Empregado

- Atributos:
  - Nome
  - E-mail
  - Celular
  - Salário
  - Departamento
- Métodos
  - gets e sets
  - Construtor com os seu atributos

Classe Gerente

- Atributos
  - Os empregados que gerenciam
  - O número empregados que gerencia
- Métodos
  - Gets e sets
  - Construtor com os seus atributos

Classe Secretaria

- Atributos
  - O empregado do qual é secretaria
- Métodos
  - Gets e sets
  - Construtor com os seus atributos

Classe Temporario

- Atributos
  - A data de início do contrato
  - O número de meses do contrato
- Métodos
  - Gets e Sets
  - Construtor com os seus atributos

SecretariaTemporaria

- Métodos:
  - Construtor com os seu atributos

Referências

- Slides da cadeira de Introdução à Programação:
  - http://www.cin.ufpe.br/~phmb/ip
- Arnaut: Oficina de Programação
  - http://www.arnaut.eti.br/op/index.html
- Programação orientada a objectos em PHP
  - http://www.tutoriaismania.com.br/imprime.php?id=396
- Slides de Gustavo (ghpc@cin.ufpe.br) do Curso de C/C++