Introdução ao C++

Prof. Leandro Buss Becker
mailto:lbecker@inf.pucrs.br

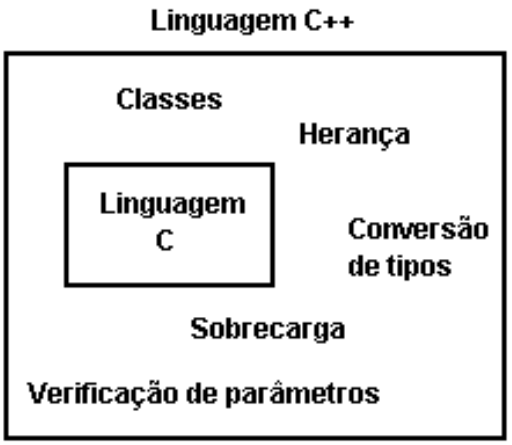
O que C++ oferece ?

- encapsulamento
  (dados + funções juntos)
- instâncias
- classes
- overloading
- polimorfismo
- dynamic binding

C++ - Historico

- criado por Stroustroup em 1980)
- um superset da linguagem C
- um C " melhorado" (a better C)
- C com conceitos de orientação a objetos ("C with Classes")
- AT&T Standard (ANSI e ISO "a caminho")

C++ e C



Qualidades de C

- inerente da linguagem
  - tipos de dados
  - 'pointers'
  - variáveis estáticas
  - manipulação de bits
- experiência
  - separação entre definição tipos de dados + protótipos de funcao (.h) e implementação (.c)

Problemas com C (ou problemas de linguagens procedurais):

- localidade (encapsulamento)
- criação de várias cópias de programas (instâncias)
- Alocação e liberação de memória (pointers)

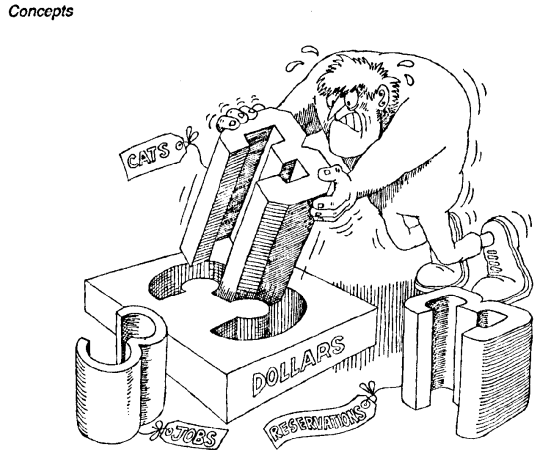
Características "únicas" de C++

- funções inline
  - toda vez que a função é chamada seu código é expandido
  - reduz 'custo' associado ao empilhamento de parâmetros e do endereço de retorno
  - checagem da sintaxe pelo compilador
- Toda e qualquer operação é uma chamada de função
  - float a,c; ... =(a, float()); =(c,float());
  - double b=5.6; ... =(b, double(5.6));
  - a+b ... +(a,b)
  - (int) (expressão) ... int (expressão)
  - b = c; ... =(b,c);
  - b=c*2; ... =(a, double (*(c, float(2))));

Características "únicas" de C++

- sobrecarga de funções ('overloading')
  - quanto ao tipo de parâmetros, quanto a quantidade dos mesmos ou ambos

Linguagem fortemente Tipada



Características "únicas" de C++

- argumentos padrão ('default')
- exemplo:

```
void teste(int a,int b,int op=1) {
   int aux;
    aux = a*b;
    if (op)
      if (aux < 0) aux *= -1;
    printf("%d\n",aux);
 }
```

```
void main() {
   teste(-10,20);
  teste(-10,20,0);
              }
```

Compatibilidade entre C e C++

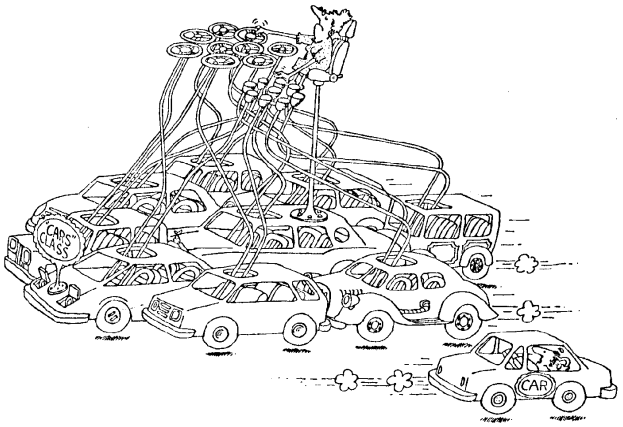
- prototipação das funções: obrigatória em C++
- declaração de variáveis: ordem de declaração é mais flexível em C++
- escopo das variáveis
- comentários: // ou /* ... */
- 'casts': int(2.5)
- argumentos de funções: padrão ANSI em C++

Classes e Objetos

Prof. Leandro Buss Becker
mailto:lbecker@inf.pucrs.br

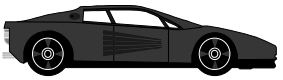
Classes

Concepts



A class represents a set of objects that share a common structure and a common behavior.

Classes e Objetos



Mundo Real

---

classe

Veículo
marca *placa*
lerPlaca

ferrari
IDN8810

fiatUno
IAS3333

instâncias

Classes e Objetos em C++

- "pseudo-classe" em C: struct + funções de acesso
  - vinculação está mais 'na cabeça do programador' do que na sintaxe da linguagem
- class
  - elementos de uma classe são, por definição, privados (‘private’). Podem ser tornados públicos (‘public’) ou protegidos (‘protected’)

Sintaxe da declaração de classes

```
class <NomeClasse>{
      <tipo>  <NomeAtributo1>;
     <tipo>   <NomeAtributo2>;

     <tipo_retorno> <NomeFunção1>(<parâmetros>);
    <tipo_retorno> <NomeFunção2>(<parâmetros>);
public:
    <tipo_retorno> <NomeFunção3>(<parâmetros>);
};
```

Exemplo classe Carro

```
//Arquivo Carro.h
class Carro{
      char marca[20];
      char placa[8];
public:
     char* lerPlaca();
};
```

Definição do código dos métodos

- Em geral separa-se a definição da classe (arquivo.h) da definição do código (arquivo.cpp)

```
<tipo_retorno>
   <NomeClasse>::<NomeFunção>(<parâmetros>){
  código
};
```

Exemplo classe Carro

```
//Arquivo Carro.cpp
#include “Carro.h”
char* Carro::lerPlaca(){
        return placa;
}
```

Construtores e Destrutores

- função construtora:
  - possui o mesmo nome da classe
  - executada automaticamente na criação de instâncias
  - aloca área para a instância
- função destrutora
  - libera a área de memória ocupada pelo objeto
  - ~<nome_classe>
  - chamada implícita e explícita

Exemplo de construtor e destrutor

```
//Arquivo Carro.h
class Carro{
      char marca[20];
      char placa[8];
public:
     Carro();  //construtor da classe Carro
     ~Carro(); //destrutor da classe Carro
     char* lerPlaca();
};
```

Exemplo de construtor e destrutor

```
#include “Carro.h”

Carro::Carro(){
   strcpy(marca, “none”);
   strcpy(placa, “xxx0000”);
}

Carro::~Carro(){
}
```

Objetos

- Objetos são gerados a partir de classes
- Todo objeto é uma instância de uma classe

Declaração - Instância

- Declaração: Não aloca memória
- Instância: Só aloca a memória necessária

```
#include “Carro.h”;
int main(){
        Carro ferrari;
        Carro fiatUno;
}
```

Construtor Parametrizado

- Permite a atribuição de valores aos atributos
- Valem as mesmas regras que para funções (parâmetros default, sobrecarga, etc.)
- Compilador assume a existência de um construtor ‘default’ sem parâmetros caso nenhum construtor seja definido
- Caso um construtor parametrizado seja definido, nenhum construtor ‘default’ é assumido (programador deve declará-lo)

Exemplo de construtor parametrizado

```
//Arquivo Carro.h
class Carro{
      char marca[20];
      char placa[8];
public:
     Carro(char m[20], char placa[8]={“xxx-
   000”);
};
```

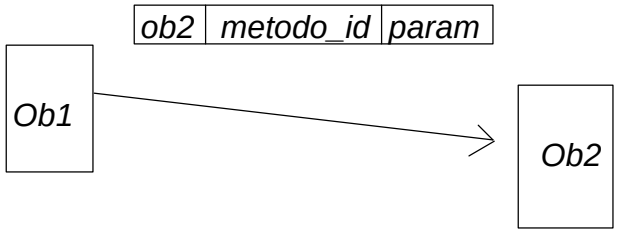
Declaração - Instância

```
#include “Carro.h”;
int main(){
        Carro ferrari(“Fiat”, “IGS );
        Carro fiatUno(“Fiat”);
}
```

Mensagens e métodos

- Métodos (operações) podem alterar ou consultar o estado do objeto, podem solicitar a execução de um serviço, podem informar ocorrência de eventos, etc...
- A definição de um método assemelha-se à definição de um procedimento ou função (ou seja, método pode ter tipo de retorno, lista de parâmetros, variáveis locais, ...)

Comunicação: Figura



Formas de implementar a comunicação

- Contexto/classe que envia a mensagem (ou que chama o método) deve 'conhecer' a instância que receberá a mensagem
- 2 possibilidades
  1) instâncias que se comunicam pertencem ao mesmo contexto
     => ex: controlador-sensor-atuador
  2) uso de atributos
  2.1.) instância como atributo (agregação)
  2.2.) referência a outra instância

Exemplo de comunicação

```
#include “Carro.h”;
int main(){
        Carro ferrari(“Fiat”, “IGS );
        Carro fiatUno(“Fiat”);
        cout << “Minha placa: “ << ferrari.lerPlaca();
}
```

Regras de Escopo

- Bloco de Comandos: {int a;...}
- Instância: [struct|union|class] {int a;...}
- Classe: [struct|union|class] {static int a;...}
- Arquivo: static int a;...
- Global: int a; // extern int a;

Operador de Resolução de Escopo: **::**

- <escopo>::
  - Global ao escopo atual: ::<var>
  - "Definido" <escopo>::<var> (namespaces)

Referências Bibliográficas

- Schield, H. “*Expert C++*”.
- Pereira et al. Apostila de C++.