Crash Course C/C++

Roberto Parra

C/C++

Aula 3

Objetos & Classes
1° semestre de 2004

C++ : + (pelo menos) 13 palavras

- class

C++ : + (pelo menos) 13 palavras

- class
- public private protected

C++ : + (pelo menos) 13 palavras

- class
- public private protected
- inline

C++ : + (pelo menos) 13 palavras

- class
- public private protected
- inline
- pure, virtual

C++ : + (pelo menos) 13 palavras

- class
- public private protected
- inline
- pure, virtual
- mutable, const

C++ : + (pelo menos) 13 palavras

- class
- public private protected
- inline
- pure, virtual
- mutable, const
- try, catch, throw (maravilha do século)

C++ : + (pelo menos) 13 palavras

- class
- public private protected
- inline
- pure, virtual
- mutable, const
- try, catch, throw (maravilha do século)
- (extern, ...)

C++ : + (pelo menos) 13 palavras

- class
- public private protected
- inline
- pure, virtual
- mutable, const
- try, catch, throw (maravilha do século)
- (extern, ...)
- + 10 diretivas do preprocessor

Objeto: quae ecce sperma est ?

- conceito abstrato

Objeto: quae ecce sperma est ?

- conceito abstrato
- objeto não é o código

Objeto: quae ecce sperma est ?

- conceito abstrato
- objeto não é o código
- código é uma *representação*

Objeto: quae ecce sperma est ?

- conceito abstrato
- objeto não é o código
- código é uma *representação*
- *ENCAPSULAMENTO*

Objeto: quae ecce sperma est ?

- conceito abstrato
- objeto não é o código
- código é uma *representação*
- *ENCAPSULAMENTO*
- *MODULARIDADE*

Objeto: quae ecce sperma est ?

- conteúdo isolado

Objeto: quae ecce sperma est ?

- conteúdo isolado
- acesso apenas via método

Objeto: quae ecce sperma est ?

- conteúdo isolado
- acesso apenas via método
- Conjunto formado pelo conteúdo e métodos de acesso

Objeto: quae ecce sperma est ?

- conteúdo isolado
- acesso apenas via método
- Conjunto formado pelo conteúdo e métodos de acesso
- implementados na linguagem desejada

Objeto: quae ecce sperma est ?

- conteúdo isolado
- acesso apenas via método
- Conjunto formado pelo conteúdo e métodos de acesso
- implementados na linguagem desejada
- *esta é a representação do objeto*

Objeto: quae ecce sperma est ?

- conteúdo isolado
- acesso apenas via método
- Conjunto formado pelo conteúdo e métodos de acesso
- implementados na linguagem desejada
- *esta é a representação do objeto*
- vamos nos referir ao código implementado como 'objeto' por facilitar o diálogo mas sempre tendo em mente que tal prática é um abuso da linguagem.

Implementando um objeto

- Característica de Acesso

Implementando um objeto

- Característica de Acesso
- Construção / Destruição

Implementando um objeto

- Característica de Acesso
- Construção / Destruição
- Escopo : Implementação

Componentes da Classe

- **public** acesso externo permitido

Componentes da Classe

- **public** acesso externo permitido
- **private** acesso externo proibido

Componentes da Classe

- **public** acesso externo permitido
- **private** acesso externo proibido
- **protected** acesso herdado proibido

Componentes da Classe

construtor nome_da_classe()
destrutor ~nome_da_classe()

```
class StTeste
{
   public:
      StTeste();
      ~StTeste();
};
```

Esta classe não faz nada, mas note que contém os elementos mínimos para a sua definição. Tanto o Construtor quanto o Destrutor **precisam** ser declarados **public**. A definição de uma classe normalmente fica num arquivo header (.h)

Componentes da Classe

Escopo: *define de qual classe pertence a implementação da função*

```
StTeste::StTeste()
{
   //construtor default
}

StTeste::~StTeste()
{
   //destrutor default
}
```

Esta classe não faz nada, mas note que contém os elementos mínimos para a sua definição. Tanto o Construtor quanto o Destrutor **precisam** ser implementados. As implementações dos métodos de uma classe normalmente residem num arquivo de extensões: C, cpp ou cxx

Estruturas de Dados

```
struct __congelador
{
   float rTemperatura;
   float rVolume;
};
struct __torneira
{
   float rVazao;
   float rTemperatura;
};
struct __prateleira
{
   int zMesh;
   float rArea;
};
```

```
typedef struct __prateleira _prateleira;
typedef struct __torneira   _torneira;
typedef struct __congelador _congelador;

struct __geladeira
{
   _congelador congelador;
   _torneira   torneira;
   _prateleira prateleiras[5];
};

typedef struct __geladeira _geladeira;
```

Estruturas de Dados

```
class StCongelador
{
   public:
      StCongelador();
      StCongelador( float rTemp );
      ~StCongelador();

   private:
      float rTemperatura;
      float rVolume;

   public:
      float TermoStato( float rT );
      float SetVolume( float rV );
};
```

```
class StPrateleira
{
   public:
      StPrateleira();
      ~StPrateleira();

   private:
      int zMesh;
      real rArea;

   public:
      int SetMesh( int zM );
      real SetArea( int zA );
};
```

Classes

```
classe StGeladeira
{
   public:
      StGeladeira();
      ~StGeladeira();

   private:
      StCongelador congelador;
      StTorneira torneira;
      StPrateleira prateleiras[5];

   public:
      void degela();
      int enchecopinho();
      int liga();
      int desliga();
};
```

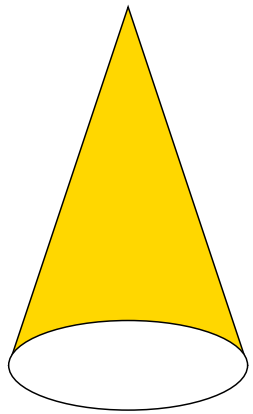
```
int main( void )
{
   StGeladeira oFrigidaire;
   StGeladeira* poClimax =
      new StGeladeira;

   printf
   (
      "Magna: %d\n",
      sizeof( oFrigidaire )
   );

   oFrigidaire.degela();
   poClimax->degela();
}
```

Conceito Typedef

DEFINIÇÃO DA ESTRUTURA DE DADOS

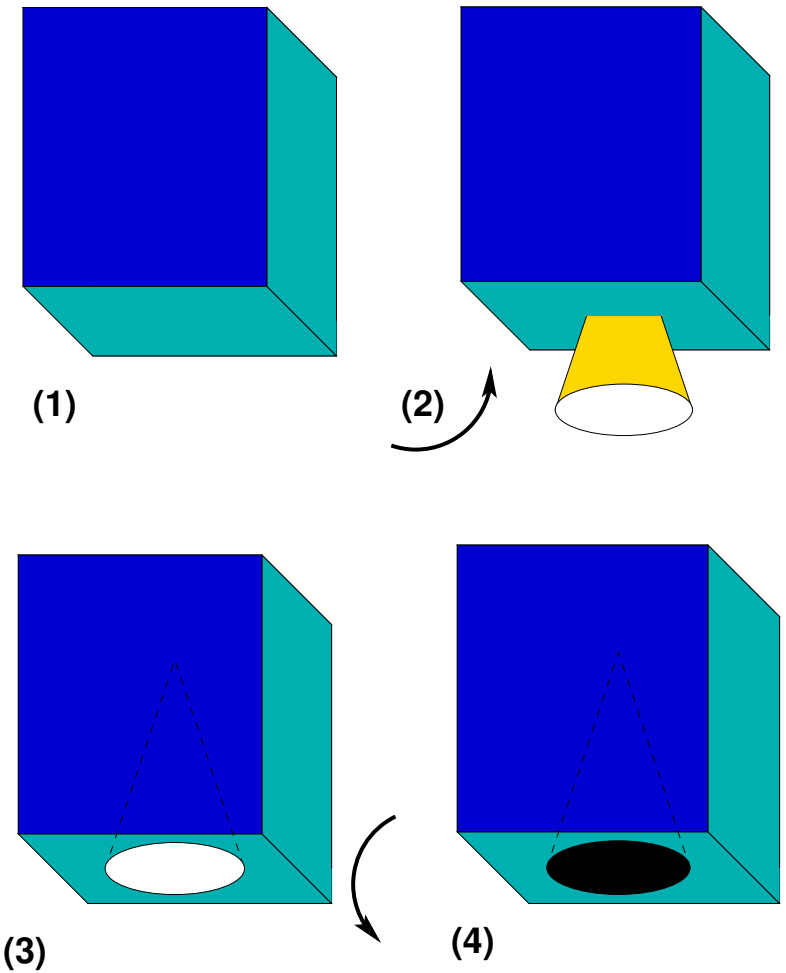
o cone ao lado simboliza a definição da estrutura.

"__cone"é o nome da estrutura:



```
struct __cone
{
   int cor_lateral;
   int cor_base;
   float altura;
   float raio;
};
```

Conceito Typedef

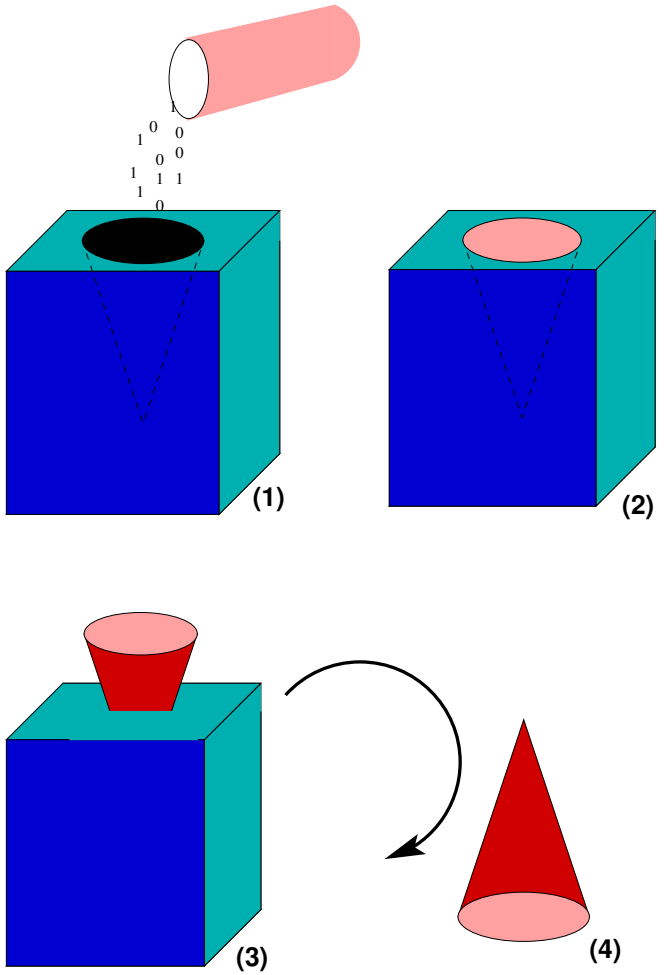


Operador TYPEDEF

(1) o operador typedef é um bloco potencialmente maleável para conter qq forma de dados
(2) **baseado** na estrutura __cone,
(3) crio um **molde** da estrutura
(4) a esse molde dou o nome de **_cone**

```
typedef struct __cone _cone;
```

Conceito Typedef



INSTANCIANDO CONES

(1) usando o molde **_cone**,
(2) "encho-o"de memória,
(3) criando uma estrutura
(4) que pode ser utilizada agora no programa. essa estrutura, cujo nome é **cone**, existe na memória tendo endereço específico de acesso.

```
int main( void )
{
   int contador;
   float area;
   _cone cone;

   .....
}
```

Acessando Membros

NO STACK (locais)
(operador ponto '.')

```
.........................
int zHits;

StTrack oTrack;
StThreeVectorF oVector;

zHits = oTrack.numberOfHits();
oVector = oTrack.momentum();
```

NO HEAP (remotos)
(operador vetor '->')

```
.........................
int zHits;

StTrack* poTrack;
StThreeVectorF oMomentum, oOrigin;

zHits = poTrack->numberOfHits();
oMomentum = poTrack->geometry()->momentum();
oOrigin = poTrack->geometry()->origin();
```

Acessando Membros

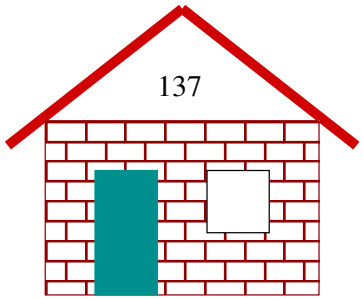
NO STACK (locais)
(operador reference '&')

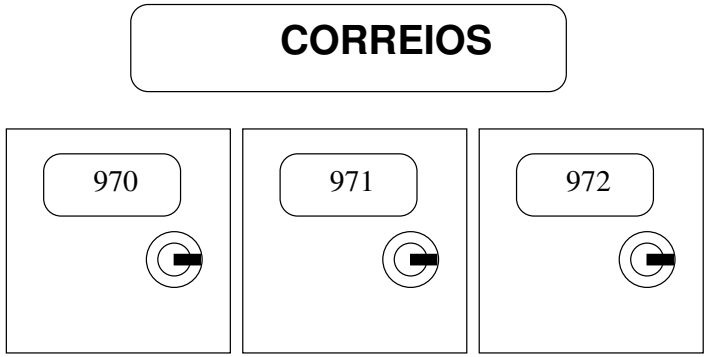
```
.........................
int zHits;

StTrack oTrack;
StThreeVectorF oVector;

zHits = oTrack.numberOfHits();
oVector = oTrack.momentum();
usatrack( &oTrack );

.................

void usatrack( StTrack* poTrilha )
{
...
```

NO HEAP (remotos)
(operador pointer '*')

```
.........................
int zHits;

StTrack* poTrack;
StThreeVectorF* poMomentum, poOrigin;
usatrack( poTrack );
.................

void usatrack( StTrack* poTrilha )
{
...
```

Acessando Membros






```
float carta = 1.0;
float casa;
float* cxpostal;

&casa;  //  == 137

cxpostal = &casa;  //(o cara alugou uma caixa postal)

&cxpostal;  // == 970

*cxpostal = carta;  // (137) == 1 !
// no exemplo acima, a carta foi pra caixa postal no correio
// que a despachou para casa correta.
```

Polimorfismo

Muitas Formas para um objeto

É possível declarar várias funções com o mesmo nome, porém com argumentos de tipos diferentes, executando operações diferentes. Um exemplo simples é pensar na função *valor absoluto*.
Porque existem diversos tipos de variáveis, deve existir diversos tipos de função *abs* pois devem retornar o valor absoluto do tipo correspondente.

Polimorfismo

em C:

```
int iabs( int ival );

float fabs( float fval );

double lfabs( double dval );

long double labs( long double lval );


modulo2( _Vector2f* vetor_plano );

modulo3( _Vector3f* vetor_espaco );
```

em C++:

```
int abs( int val );

float abs( float val );

double abs( double val );

long double abs( long double val );


modulo( StVector2F* vetor_plano );

modulo( StVector3F* vetor_spaco );
```

Polimorfismo

```
#include <stdio.h>

class teste
{
   public:
      teste();
      ~teste();

   public:
      char  colher;
      short xicara;
      long  jarra;

   public:
      char  entra( char c );
      short entra( short s );
      long  entra( long l );
};
```

```
teste::teste()
{
   colher = 0 ,xicara = 0, jarra  = 0L;
}

teste::~teste() {}

char teste::entra( char c )
{
   return( colher = c );
}

short teste::entra( short s )
{
   return( xicara = s );
}

long teste::entra( long l )
{
   return( jarra = l );
}
```

Polimorfismo

```
#include <teste.h>

void pc( teste* oObj );

int main( int zArgc, char** pasArgv )
{
   char cCh=0x41;
   short zSt=32000;
   long zLg=130000;
   teste t;

   t.entra( (char)4 );
   pc( &t );
   t.entra( (short)21 );
   pc( &t );
   t.entra( (long)42 );
   pc( &t );
   t.entra( cCh );
   pc( &t );
   t.entra( zSt );
   pc( &t );
   t.entra( zLg );
   pc( &t );
   return 0;
}
```

```
void pc( teste* oObj )
{
   printf
   (
      "1: %d\t2: %d\t4: %ld\n",
      oObj->colher,
      oObj->xicara,
      oObj->jarra
    );
}
```

EXERCÍCIO:

o que aconteceria se as variáveis es-

tivessem num bloco "private"?

Mecanismo de Herança

def: uma classe derivada (advinda) 'herda' todos os membros de dados e todos os métodos da classe base.

Os dados declarados como **protected** são acessíveis mas não modificáveis (via de regra) pela classe derivada

Finalmentes

Roberto Parra nUSP 1694869

esta apresentação foi totalmente elaborada em **LaTeX**, no **slackware** linux ftrn

- 100% Free Software
- 100% MS free !
- 100% elétrons recicláveis

Heisenberg **pode** ter estado aqui !