Introdução ao C++

Ambientes de Desenvolvimento Avançados

2 de Outubro de 2002

C++

- Extensão ao c
- Facilita a representação de tipos de dados abstractos (tipo classe)
- Suporte de programação orientada ao objecto
- Tratamento de erros (`try ... catch`)
- Comentários começados por `//` (vão até ao fim da linha)
- Funções `inline` (dispensa macros)
  - Combina a eficiência das macros com a segurança das funções
- Alocação e libertação de memória dinâmica com `new` e `delete` mais controlada do que com `malloc` e `free` (nomeadamente quando se usam classes)

C++

- Declaração de variáveis podem ocorrer em qualquer lado (p.e: na parte de iniciação da instrução `for`)
- Funções e operadores sobrecarregados (*overloading)*
- Valores por omissão para os parâmetros das funções
- Referências e passagem de parâmetros por referência (&)
- Variáveis qualificadas com `const` podem ser usadas onde é esperada uma expressão constante (dispensa macros)
- Novo tipo `bool` (pode conter `true` ou `false`)
- Biblioteca alternativa de entrada e saída de dados baseada em *streams*, mais segura e mais simples (cout, cin, ...)

C++

- Os nomes `struct`, `classe`, `union` e `enum` são um tipo de dado (dispensam o `typedef`)
- Espaços de nomes (*namespace*) como módulos lógicos mais flexíveis do que os módulos físicos (ficheiros com código fonte)
  - Permite restringir a abrangência de classes, funções e objectos globais
  - **namespace** [*identifier*] **{** *namespace-body* **}**
- Operadores de conversão de tipos mais seguros
- Padrões (*templates*) de funções (dispensa macros)

Classes c++

Declaração de Classes

```
Class nome_da_classe
{
private:
    membros/métodos privados
protected:
    membros/métodos protegidos
public:
    membros/métodos públicos
};
```

Declaração de métodos fora da classe

```
[tipo_dado_retorno] nome_classe :: nome_método ( [parâmetros formais] )
{ ... código do método ... }
```

Exemplo de uma Classe C++

Interface (quadrado.h)

```
#include <iostream.h>

class quadrado
{
private:
        int ix, iy;
        float flado;
        char * cor;

public:
        quadrado();
        quadrado(int x, int y, float l, char *c);
        quadrado(const quadrado &q);
        ~quadrado();
        quadrado operator+(const quadrado&);
        void listar();
        float GetArea() const;
};
```

Exemplo de uma Classe C++

Implementação (quadrado.cpp)

```
quadrado::quadrado()
{  ix=0; iy=0; flado=0.0; cor=NULL; };
quadrado::quadrado(int x,int y,float l,char *c)
{  ix=x; iy=y; flado=l;
   cor=new char[strlen(c) +1];
   strcpy(cor , c);
};
quadrado::~quadrado()
{  if(cor)  delete [] cor; };
void quadrado::listar()
{ cout<<"cordenada x="<<ix<<'\n';
   cout<<"cordenada y="<<iy<<'\n';
   cout<<"lado="<<flado<<'\n';
   cout<<"cor="<< (cor?cor:"indefinida") <<'\n';
};
float quadrado::GetArea() const
{  return flado*flado; }
```

Invocação

```
#include “quadrado.h"

int main()
{
  quadrado q1;
  quadrado q2(1,1,2,"branco");

  q1.listar();
  cout<<"--------------"<<endl;
  q2.listar();

  return 0;
}
```

Passagem de Parâmetros

- **Parâmetros por valor -** A passagem de parâmetros por valor invoca o construtor cópia do tipo de dados do parâmetro formal. Quando uma função termina, os destrutores dos tipos de dados dos parâmetros formais *destroem* os valores desses parâmetros
  - **int funcA(int a);** // a - parâmetro por valor
- **Parâmetros por referência -** Os parâmetros actuais são ligados aos parâmetros formais, não existe cópia dos parâmetros. O nome dos parâmetros formais substitui o nome dos parâmetros actuais. Qualquer alteração do parâmetro formal, altera o parâmetro actual (*Alias*)
  - **int funcB(int& b);** // b - parâmetro por referência
- **Parâmetros por referência constante -** Neste caso o parâmetro formal substitui o actual, mas como é constante, não admite alteração do seu valor
  - **int funcC(const int& c)** // c - parâmetro por referência constante

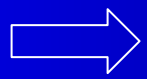
Construtores

- Têm o mesmo nome da classe
- O construtor serve normalmente para iniciar os membros-dados
- O construtor é invocado automaticamente sempre que é criado um objecto da classe
- Construtores também podem ser invocados explicitamente

```
quadrado();
quadrado(int x,int y,float l,char *c);
quadrado(int x=0, int y=0, float l=2, char * c="branco" );
```

Construtor cópia

- não devolve valores
- apenas possui um parâmetro



```
quadrado::quadrado(const quadrado &q)
{
  flado=q.flado;
  ix=q.ix; iy=q.iy;
  cor=NULL;
  if(q.cor) { cor=new char[ strlen(q.cor)+1]  ;
              strcpy(cor,q.cor); }
};
```

Destrutores

- Têm a mesmo nome da classe precedido do símbolo ~
- Não têm parâmetros e não devolvem quaisquer valores
- Não são invocados explicitamente
- São invocados automaticamente sempre o objecto deixa de existir
- Destrutores servem normalmente para libertar recursos (memória dinâmica, etc.) associados ao objecto

```
quadrado::~quadrado()
{
  if(cor)
      delete [] cor;
};
```

Sobrecarga de operadores

- Permite rescrever código para operadores de modo a manipularem operandos de classe diferente dos que estão predefinidos
- Não é possível alterar a precedência nem o número de operandos

```
quadrado quadrado::operator+(const quadrado& q)
{
        quadrado qtemp;
        qtemp.ix  = this->ix;
        qtemp.iy  = this->iy;
        qtemp.flado = (float)sqrt( this->GetArea()+q.GetArea() ) ;
        return qtemp;
}
```

Relação "IS-A" e Herança

- **Muitas vezes, conceptualmente, um objecto de uma classe também é um objecto de uma outra classe (relação IS-A)**

Exemplo:

```
clase Pessoa {/*propriedades comuns a todas as pessoas*/};
class Aluno : public Pessoa {/*propriedades específicas*/};
Aluno Obj;
```

- **Obj** é um aluno e é uma pessoa (objecto da classe Aluno e da classe Pessoa)
- Aluno herda as propriedades definidas para a Pessoa (relação de herança)
- Pessoa é uma classe base (superclasse) para Aluno
- Aluno é uma classe derivada (subclasse ou subtipo) de Pessoa
- Objectos da classe derivada podem ser tratados como objectos da classe base (desde que manipulados através de apontadores ou referências)

Exemplo de classe derivada

```
class Pessoa
{
protected:
 string nome;
 Data dataNasc;

public:
 Pessoa(string, Data);
 string getNome() const;
 Data getDataNasc()const;
 int getIdade() const;
 void setNome(string);
 void setDataNasc(Data);
 void imprime() const;
};
```

```
class Aluno : public Pessoa
{
protected:
  string curso;

public:
  Aluno(string, Data ,string);
  string getCurso() const;
  void setCurso(string);
  void imprime() const;
};
```

Resumo Programação OO em C++

- A classe é a unidade de **ocultação de dados** e de **encapsulamento**
- Classe como **tipo abstracto de dados**: a classe é o mecanismo que suporta **abstracção de dados**, ao permitir que os detalhes de representação sejam escondidos e acedidos exclusivamente através de um conjunto de operações definidas como parte da classe
- A classe proporciona uma unidade de **modularidade**
- **Reutilização de código** promovida pelo mecanismo de **herança**
- **Polimorfismo** (a capacidade de objectos de diferentes classes relacionadas por herança responderem diferentemente à chamada da mesma função-membro) é suportado através de classes com funções virtuais

Microsoft Visual C++

Criar um projecto do tipo *Console Application*

```
#include <stdio.h>

int main( void )
{
     printf("Olá Mundo!\n");
     return 0;
}
```

Questões ...