

Introdução a C++

Conceitos Básicos

Jorge Almeida

Departamento de Engenharia Mecânica
Universidade de Aveiro
almeida.j@ua.pt

13 de Fevereiro de 2015

Bibliografia

- <http://www.cplusplus.com/doc/tutorial/>
- <https://class.coursera.org/cplusplus4c-002/lecture>

O que é C++

- “C com classes”, designação comum mas insuficiente
- Informação e operações agregadas
- Orientada a objetos, uma metodologia diferente de programar
- Type-safe, um método de evitar erros típicos em conversões entre tipos de dados
- Organizada para projetos grandes

Tópicos abordados

- Namespaces
- Referências
- Overload de funções
- Templates
- Classes
- Overload de operadores
- Biblioteca STL
- Ponteiros Partilhados/Inteligentes

Primeiro programa

CMakeLists.txt

```
1 | cmake_minimum_required (VERSION 2.8)
2 | project (Example)
3 | add_executable (main main.cpp)
```

main.cpp

```
1 | #include <iostream>
2 |
3 | int main()
4 | {
5 |     std::cout << "Hello World!" << std::endl;
6 | }
```

Compiler

```
$ cd ~/Example_1
$ cmake .
$ make
```

Output

```
Hello World!
```

Namespaces

- Ferramenta de contextualização
- Agrupamento e segmentação de código
- A::B, B pertence a A

```
1 namespace myNamespace
2 {
3     const double pi = 3.1416;
4
5     double value()
6     {
7         return 2*pi;
8     }
9 }
10
11 using namespace std;
12
13 int main()
14 {
15
16     cout << "Pi: " << myNamespace::pi << endl;
17     cout << "Value: " << myNamespace::value() << endl;
18 }
```

```
Pi: 3.1416
Value: 6.2832
```

Referências

- Uma referência de uma variável pode ser tratada como se fosse a própria variável
- Fundamentalmente é um sinónimo
- O símbolo & neste contexto têm uma interpretação diferente do contexto de ponteiro
- Podem ser utilizadas num scope diferente da variável original, por exemplo como argumentos de entrada de funções

```
1 int main()
2 {
3     int x;
4     int& ref_x = x;
5
6     ref_x = 14;
7
8     cout << x << endl;
9 }
```

14

Exercício

1 Criar a função `swap()` que substitua o valor de dois inteiros

- Utilizar referências

```
1 int main()
2 {
3     int x = 5;
4     int y = 10;
5
6     cout << "Antes: " << endl;
7     cout << x << endl;
8     cout << y << endl;
9
10    swap(x,y);
11
12    cout << "Depois: " << endl;
13    cout << x << endl;
14    cout << y << endl;
15 }
```

Antes :

5

10

Depois:

10

5

Overload de funções

- Múltiplas funções com o mesmo nome
- Diferentes parâmetros em número ou tipo
- Os overloads não podem diferenciar apenas no valor de retorno, os parâmetros têm de ser diferentes

```

1 int op(int a, int b)
2 {
3     return a+b;
4 }
5
6 int op(int a, int b, int c)
7 {
8     return a+b+c;
9 }
10
11 string op(string a, string b)
12 {
13     return a+b;
14 }
15
16 int main()
17 {
18     cout << op(1,2) << endl;
19     cout << op(1,2,3) << endl;
20     cout << op("hello", "world") << endl;
21 }

```

```

3
6
helloworld

```

Templates

- Tipos genéricos indefinidos
- Múltiplos templates são possíveis
- Permitem uma função seja utilizada com tipos de dados diferentes

```

1 template <typename T>
2 T op(T a, T b)
3 {
4     return a+b;
5 }
6
7 template <typename T, typename B>
8 T op(T a,B b)
9 {
10    return a+b;
11 }
12
13 int main()
14 {
15     cout << op<int>(1,2) << endl;
16     cout << op<double,double>(1.7,2.5) << endl;
17     cout << op<int,double>(2,5.5) << endl;
18 }

```

3
4.2
7

Exercício

- 2 Escreva a função `getMax()` que obtenha o valor máximo e a sua posição nos três arrays definidos

- Utilize templates

```

1 | int i[]={10,5,6,823,10,156,3};
2 | double d[]={.2,0.8,1.2,5.6,2.2};
3 | string s[]{"santos", "soares", "pereira", "fonseca", "
   |           castro"};

```

- 3 Leia 10 valores inteiros introduzidos pelo utilizador e efetue o mesmo cálculo

- Dica, utilize

```
1 | cin >> array[i];
```

para ler valores do teclado

```

3, 823
3, 5.6
1, soares

```

Classes

- Tipo composto
- Estrutura com funções
- Funções membros chamam-se métodos
- Controlo de acesso
 - private, default, apenas membros e amigos
 - protected, membros, amigos ou derivados
 - public, todos
- Uma classe é um tipo de dados
- Uma variável do tipo da classe é chamada de instancia

```

1 class Vector
2 {
3     private:
4         double x, y;
5
6     public:
7         void setValues(double x, double y)
8         {
9             this->x = x;
10            this->y = y;
11        }
12
13        double norm()
14        {
15            return sqrt(x*x+y*y);
16        }
17    };
18
19 int main ()
20 {
21     Vector v;
22     v.setValues(1,1);
23     cout << v.norm() << endl;
24 }

```

1.41421

Exercícios

- 4** Adicione à classe anterior um método para somar vetores

```
v.sum();
```

- 5** e também um método para calcular o produto interno

```
v.dot();
```

```
1 | Vector va;  
2 | Vector vb;  
3 |  
4 | va.setValues(1,1);  
5 | vb.setValues(2,2);  
6 |  
7 | va.sum(vb);  
8 | va.dot(vb);
```

```
va: (3, 3)  
vb: (2, 2)  
dot: 12
```

Construtores e destrutores

- Construtor
 - Definir valores iniciais
- Destrutor
 - Operações de limpeza
- Existem versões de default
- O construtor é chamado sempre que se cria uma instancia da classe
- O destrutor é chamado sempre que a instancia sai de scope

```

1 | class Vector
2 | {
3 |     private:
4 |         double x, y;
5 |     public:
6 |         Vector(double x, double y)
7 |         {
8 |             this->x = x; //Utilize "this->" para
9 |                 referenciar o membro da classe
10 |             this->y = y;
11 |         }
12 |         ~Vector()
13 |         {
14 |             cout<<"destrutor"<<endl;
15 |         }
16 |         ...
17 | };
18 |
19 | int main()
20 | {
21 |     Vector v(1,1);
22 |     cout << v.norm() << endl;
23 | }

```

1.41421
destrutor

Overload de operadores

- Redefinir operadores
- É possível em C++ usar operadores simples, + - / *, para efetuar operações em classes
- Apenas alguns dos operadores passíveis de redefinir:
 - + - * / = < > +=
 - = /= << >> == !=
 - ++ -- % & ^ ! | ~
 - && || [] () , ->
 - new delete
- Apenas devem ser utilizados quando a tarefa efetuada for óbvia e não contra-intuitiva; por exemplo: subtrair em vez de somar com o operador +

```

1 | class Vector
2 | {
3 |     ...
4 |
5 |     public:
6 |         Vector operator+(const Vector v)
7 |         {
8 |             Vector temp;
9 |             temp.x = x + v.x;
10 |            temp.y = y + v.y;
11 |            return temp;
12 |        }
13 | };
14 |
15 | int main()
16 | {
17 |     Vector va(1,1);
18 |     Vector vb(2,2);
19 |     va = va + vb;
20 |     cout << va.norm() << endl;
21 | }

```

4.24264

Exercícios

6 Defina overloads para as operações de adição/subtração escalar

7 e multiplicação/divisão escalar

8 Colocar a classe `Vector` num ficheiro separado. Deverá existir um ficheiro `Vector.h` com a declaração da classe e um ficheiro `Vector.cpp` com a implementação.

```

1 | Vector v(1,0);
2 |
3 | v = v + 5;
4 | v = v + 5.2;
5 | v+= 3;
6 | v-= 2;
7 | v = v*3;
8 | v = v/10;

```

v: (3.66, 3.36)

Extensões de classes, hereditariedade

- Uma classe pode ser criada expandindo outra
- A nova classe derivada vai herdar todos os membros da classe base
- Múltiplas classes podem ser herdadas

```

1 class Point
2 {
3     public:
4         double x,y;
5
6 };
7
8 class Vector: public Point
9 {
10     private:
11         double direction,magnitude;
12
13     public:
14         Vector() = default;
15
16         Vector(double x, double y)
17         {
18             this->x = x;
19             this->y = y;
20
21             direction = atan2(this->y,this->x);
22             magnitude = norm();
23         }
24 };

```

STL Containers

- STL, Standard Template Library
- Containers, contentores para coleções de objetos relacionados
- São definidos usando templates o que permite uma grande flexibilidade
- Gerem a memória associada aos seus elementos
- Podem ser:
 - Sequenciais: vector, array, deque, list, forward_list
 - Associativos: map, set, multimap, multiset
 - Adaptadores: queue, priority_queue, stack
- Apenas vão ser dados exemplos de vectors e maps

Vector

- Array de tamanho variável
- Alocação dinâmica automática
- Tipo de dados indefinido, template
- Iteradores para percorrer elementos
 - `my_map.begin()`
 - `my_map.end()`
- **Atenção!** o método: `.end()` devolve um iterador para a posição depois da última válida

```

1 #include <vector>
2
3 int main()
4 {
5     vector<int> my_vector;
6
7     for(int i = 10; i >= 0; i--)
8         my_vector.push_back(i);
9
10    cout<<"vector: ";
11
12    for(vector<int>::iterator it=my_vector.begin(); it
13        != my_vector.end(); ++it)
14        cout << ' ' << *it;
15
16    cout << endl;

```

- Alguns dos métodos da classe

vector

- `push_back()`
- `size()`
- `resize()`
- `clear()`
- `erase()`

vector: 10 9 8 7 6 5 4 3 2 1 0

Map

- Contendor associativo
- Composto por uma chave e um valor
- Ambos os campos são tipos indefinidos
- Alguns dos métodos da classe map
 - find()
 - size()
 - clear()
 - erase()

```

1 #include <map>
2
3 int main()
4 {
5     map<char, string> my_map;
6
7     my_map['a'] = "um elemento";
8     my_map['b'] = "outro elemento";
9     my_map['c'] = my_map['b'];
10
11     cout << "my_map['a']: " << my_map['a'] << endl;
12     cout << "my_map['b']: " << my_map['b'] << endl;
13     cout << "my_map['c']: " << my_map['c'] << endl;
14     cout << "my_map['d']: " << my_map['d'] << endl;
15
16     cout << "my_map contem agora " << my_map.size() <<
17         " elementos." << endl;
18 }

```

```

my_map['a']: um elemento
my_map['b']: outro elemento
my_map['c']: outro elemento
my_map['d']:
my_map contem agora 4 elementos.

```

Exercícios

- 9 Criar `std::vector` com 100 valores inteiros aleatórios entre 1 e 100
- 10 Criar um overload para mostrar o vector usando `cout<<my_vec<<endl;`
- 11 Obter o máximo usando a biblioteca `<algorithm>`
- 12 Inverter todos os elementos do vector, usar `reverse()`
- 13 Remover todos os elementos que **não** são números primos, usar `remove_if()`

shared_ptr

- Ponteiro inteligente
- Não é necessário desalocar a memória associada a ele
- Possui uma contagem interna do número de cópias existentes, quando esse contador chega a zero, o destrutor da classe associada é invocado
- O template permite o uso qualquer tipo de dados
- Útil para utilizar dentro dos contenedores STL, visto não ser necessário garantir que a nossa classe é copy-constructible e copy-assignable

```

1 | #include <boost/shared_ptr.hpp>
2 |
3 | namespace geometry
4 | {
5 |     class Vector
6 |     {
7 |         ...
8 |
9 |     public:
10 |         typedef boost::shared_ptr<Vector> Ptr;
11 |
12 |         friend ostream& operator<<(ostream& o, const Vector& i);
13 |     };
14 |
15 | ostream& operator<<(ostream& o, const Vector& i)
16 | {
17 |     o<<" (<<i.x<<", "<<i.y<<")";
18 |     return o;
19 | }
20 | }
21 |
22 | int main()
23 | {
24 |     vector<geometry::Vector::Ptr> vec;
25 |     vector<geometry::Vector::Ptr>::iterator it;
26 |
27 |     for (uint i=0; i<10; i++)
28 |     {
29 |         geometry::Vector::Ptr p(new geometry::Vector);
30 |         p->setValues(i, i);
31 |         vec.push_back(p);
32 |     }
33 |
34 |     for (it=vec.begin(); it!=vec.end(); it++)
35 |         cout << *it << endl;
36 | }

```

Exercícios

- 14 Criar uma lista de 100 vectores aleatórios
- 15 Utilizar a função `std::max_element()` para obter o vector de maior norma
- 16 Utilizar a função `std::sort()` para ordenar os vectores por norma
- 17 Calcular a média e desvio padrão das normas dos vectores
- 18 Obter o rácio de vectores com norma superior a 1
- 19 Defina uma class `PolyLine` usando a classe `Point` criada anteriormente
 - A classe `PolyLine` deveser poder conter um número indeterminado de pontos
 - Deveser possível adicionar pontos com o `operator+`
 - Criar o método `draw()` que deveser desenhar numa `cv::Mat` a linha completa
 - Criar métodos auxiliares para definir a cor e espessura da linha

```
1 geometry::PolyLine line;  
2 line = line + geometry::Point(1,1) + geometry::Point(280,1) + geometry::Point(280,280);  
3 line.setColor(CV_RGB(255,0,0));  
4 line.setThickness(2);  
5  
6 cv::Mat image(300,300, CV_8UC3, cv::Scalar(255,255,255));  
7 line.draw(image);  
8  
9 cv::imshow("test", image);
```

Exercícios

- 20 Usando `opencv` ler imagens de uma webcam, PC ou exterior, e mostrar as imagens obtidas
- 21 Aplicar operações básicas de processamento de imagem `GaussianBlur()` e `Canny()`, mostrar o resultado

Exercícios para resolver na aula

- No contexto do futuro desafio em ROS
- 22** Criar um tabuleiro de dimensões variáveis
 - 23** Têm de ser possível adicionar minas em posições específicas no tabuleiro
 - 24** Têm de ser possível adicionar naves do utilizador e movimenta-las
 - 25** Têm de ser possível desenhar o tabuleiro no terminal
- Utilizar os conceitos apresentados anteriormente
 - Criar a classe `Map`
 - Criar a classe `Ship`
 - Utilizar a biblioteca `Eigen3` para criar a matriz do mapa