Física Computacional 5
Namespaces, estruturas

1. Estruturas (revisão)
2. classes
3. namespaces

fc.trabalhosalunos@gmail.com

Revisão Estruturas

- As estruturas em C são tipos agregados, que contêm outros tipos: inteiros, floats, ponteiros, outras estruturas, etc.
  - **struct** particle{
  - **float** px, py, pz, e;
  - std::string name;
  - **int** pdgid;
  - };
- A sintaxe de utilização das estruturas é a mesma de qualquer tipo.
  - particle protao;
- Após inicialização podemos por exemplo ler cada campo da estrutura
  - std::cout <<“o momento x do protao e” << protao.px;
- Recomendamos o estudo do código exemplo diponibilizado na página da cadeira, **structurewithstrings.cpp**

Classes

- Históricamente, o conceito de classe foi introduzido para simular embarcações, nas linguagens Simula I e Simula 67, desenvlvidas nos anos 1960s no Norwegian Computing Center por Ole-Johan Dahl and Kristen Nygaard.

- O criador do C++ (1983), Bjarne Stroustrup, reconheceu a influência do Simula 67 na génese do C++, concebido (1979) para ter a velocidade do C e a produtividade das classes.

Classes

- Em C++ a estruturas são alargadas para possibilitar a programação por objetos:
  - Encapsulamento
  - Herança
  - Polimorfismo

- Cria-se assim um novo conceito, o de classe. É possível declarar novas classes em C++, usando as *keywords* **class** e **struct**

- Uma classe tal como uma estrutura do C é um tipo agregado que para além de possuir campos de dados, possui também funções — *métodos*

- Designaremos por *classe* o tipo em si e por *instância da classe,* ou *objeto,* as variáveis desse tipo.

Classes

- Exemplo:

```
#include <iostream>

struct A { //Porquê struct?
   int x;
   void print_x() {
      std::cout << x << std::endl;
   }
};

int main() {
   A a;
   a.x = 5;
   a.print_x(); //imprime 5
}
```

Classes

- Notamos que, presentemente, ao contrário do C, em C++ a estrutura definida com **struct** já pode incluir métodos.
  - A diferença entre keyword **struct** e a keyword **class** é que a **struct** é pública por defeito enquanto que a **class** é privada por defeito.
  - A keyword **class** é comum a diversas linguagens, sempre que se utilizam classes.
  - A keyword **public** indica que os métodos e campos da classe são acessíveis exteriormente à classe.
  - A keyword **private** indica que os mesmos membros (métodos e campos) da classe só são acessíveis internamente (pelos métodos da classe) e pelos amigos (outras classes ou funções) da classe.
  - Os amigos têm de ser declarados com a keyword **friend** no interior da classe.

Classes

- Podemos também ter funções (métodos) e variáveis (campos) estáticas, as quais não estão associadas a nenhuma instância da classe, e podem ser acedidas usando o operador de *scope* ::
  - Ex de um campo estático:

```
struct B {
   static int x;
   int y;
};

int B::x = 0; //inicialização de B::x  — fora do corpo da classe

int main() {
   B::x = 4; //OK
   B::y = 5; //ERRO: variável y não é estática
   B b;
   b.y = 0; //Ok
}
```

- Os métodos estáticos só podem aceder a variáveis estáticas. Os métodos não estáticos tanto podem aceder às variáveis estáticas como às não estáticas
- Na verdade, a cada método não estático é passado o ponteiro para a instância da classe/estrutura — o **this** :

```
#include <iostream>

struct C {
    void printThis() {                  // por defeito não é estático
        std::cout << this << std::endl;
    }
};

int main() {
    C c;
    std::cout << &c << std::endl; //imprime endereço da instância c
    c.printThis(); //idem
}
```

Classes

Classes

- Os métodos de uma classe podem ser implementados fora do corpo da classe
  - Para isso utiliza-se o operador ::
  - Exemplo:

```
//Ficheiro c.h
   struct C {
      void printThis();
   };

   //Ficheiro c.cpp
   #include “c.h”
   #include <iostream>

   void C::printThis() {
      std::cout << this << std::endl;
   }
```

- na prática é como se estivessemos ainda dentro da classe

Namespaces

- Os namespaces são semelhantes às classes, no sentido em que são agregados de várias variáveis
  - Ao contrário das classes não podem ser instanciados (é como uma classe com todos os membros estáticos)
  - Enquanto que as classes são usadas para programação por objetos, os namespaces funcionam como uma forma de agrupar vários tipos e objetos, evitando que se polua o programa com inúmeros nomes
  - Recomendamos o estudo do código exemplo disponibilizado na página da cadeira, **namespace.cpp**

Namespaces

- Criação de namespaces:

```
namespace N {
  struct S { int x, y; };
  int a;
  float zero() { return 0; }
} // Não é necessário o ;
```

- O conteúdo de um namespace pode ser acedido, utilizando o operador **::**

```
int main() {
  N::S s;
  N::a = 4;
  float z = N::zero();
}
```

Namespaces

- Podemos evitar o uso do operador :: através da keyword **using**

```
int main() {
    using namespace N;
    S s;
    a = 4;
    float z = zero();
}
```

- O uso do operador :: torna-se desnecessário dentro do bloco onde o **using** é utilizado
  - Se for colocado dentro de uma função é válido nessa função
  - Se for colocado fora de qualquer função, será válido em todo o ficheiro

Namespaces

- A biblioteca standard do C++ está contida no **namespace *std***
- Podemos utilizar a sintaxe “**using namespace** std;” para evitar escrever sempre o std::
  Assim:
  - ```
    #include <iostream>
    using namespace std;
    int main() {
       cout << “Olá Mundo” << endl; //Sem ::
    }
    ```
  - De notar que o uso sistemático de **using**, elimina uma das vantagens dos namespaces
    ```

Namespaces

- Contrariamente às classes, a declaração de um **namespace** pode ser feita em vários ficheiros

```
//file1.cpp
namespace N {
   int a = 0;
}
//file2.cpp
namespace N {
   int b = 0;
}
```

  - Assim, ambas as variáveis — a e b — ficam contidas no namespace N

Namespaces

- Podemos aceder ao namespace global usando o operador :: sem nada à esquerda

```
void coisa() {
    std::cout << "::coisa()" << std::endl;
}

namespace meuespaco {

void coisa() {
    std::cout << "meuespaco::coisa()" << std::endl;
}

 void test() {
        coisa();   //chama meuespaco::coisa
        ::coisa(); //chama coisa global
        }
}
```