




Problemas de nomenclatura 

- Nomes de classes repetidos

Projeto 1

```
class Usuario {
public:
  int a;
}
```

Projeto 2

```
class Usuario {
public:
  char b;
}
```


Projeto 3

```
class Usuario {
public:
  float c;
}
```

Projeto 4

```
#include "projeto1/usuario.h"
#include "projeto2/usuario.h"
#include "projeto3/usuario.h"

Usuario *u; // Erro: várias definições da classe Usuario
```

Namespaces 

- Organizar os arquivos por diretórios não resolve a definição de nomes de classes repetidos
- Renomear os nomes das classes pode ser inviável
- Nomes complicados não são amigáveis, tais como **RHProjFolhaPagVersaoDoisUsuario;**
- **Namespaces:** diretiva para organizar em fichários as classes sem alterar seus endereços no sistema de arquivos

Exemplo de namespaces



- Utilizando namespaces simples

Projeto 1

```
namespace Locadora {
    class Usuario {
    public:
        int a;
    };
}
```

Projeto 2

```
namespace RH {
    class Usuario {
    public:
        char b;
    };
}
```

Projeto 3


```
namespace RioNegro {
    class Usuario {
    public:
        float c;
    };
}
```

Projeto 4

```
#include "projeto1/usuario.h"
#include "projeto2/usuario.h"
#include "projeto3/usuario.h"


Locadora::Usuario *u;
```

Características



- Organiza as classes em uma nova hierarquia, similar ao **package** do Java
- Não afeta a hierarquia familiar das classes, nem suas heranças e/ou relacionamentos
- Não altera o caminho no sistema de arquivos
- É possível aninhar namespaces, criando diferentes níveis e agrupamentos:
Softblue::RH::FolhaDePagamento::Usuario

Aninhamento de namespaces



```
namespace Softblue {
    namespace RH {
        namespace FolhaDePagamento {
            class Usuario {
            public:
                int a;
            };
        }
    }
}
```

```
namespace Softblue {
    namespace RH {
        namespace Treinamentos {
            class Usuario {
            public:
                char b;
            };
        }
    }
}
```

```
Softblue::RH::FolhaDePagamento::Usuario *u1;
Softblue::RH::Treinamentos::Usuario *u2;
```

Código amigável



- Sentenças não amigáveis
`Softblue::RH::FolhaDePagamento::Usuario *u1;`
- Criando apelidos (alias) para namespaces
`namespace Softblue::RH::FolhaDePagamento::Usuario = UsuarioFP;
UsuarioFP *u;`
- Diretiva **using**
`using Softblue::RH::FolhaDePagamento;
Usuario *u;`
- Uso com cautela se houver classes de mesmo nome

Considerações



- Não se deve instanciar objetos (**new**) ou deletar objetos (**delete**) dentro de namespaces
- É um dos recursos mais recentes do C++
- Ainda não são todos os compiladores que já implementam namespaces
- O namespace padrão (standart), conhecido por sua sigla **std**, acompanha os principais compiladores

Namespace STD



- Biblioteca padrão que disponibiliza funções de uso geral, úteis para o dia a dia
- Automações e funcionalidades prontas para muito do que era feito no braço no C padrão
- Algumas funcionalidades já conhecidas:
 - **cin**: Entrada e captura de informações
 - **cout**: Exibição e impressão de informações

```
using namespace std;  
using namespace std::cin;  
using namespace std::cout;
```

Classe string



- Manipulação de strings
 - Classe **string**
 - `string nomeVar(suaString)`
 - `string nomeVar(suaString, início)`
 - `string nomeVar(suaString, início, tamanho)`
 - `append(outraString)`
 - `length()`
 - `compare(outraString)`
 - `find(stringParaBuscar)`
 - `find(stringParaBuscar, posição)`
 - `substr(posição, tamanho)`
 - `c_str()`
 - Documentação
 - <http://www.cplusplus.com/reference/string/string/>

Particularidades do cin



- Realiza a leitura de números facilmente
- Realiza a leitura de textos até o primeiro espaço em branco
- Para leitura de textos completos utilize a função **getline**, também da **std**
- Limpeza de buffer: método **get**

```
int i;
cin >> i;

string s;
cin >> s;           // Hello World
cin.get();
getline(cin, s);
```

Classe vector



- Manipulação de arrays
 - Classe **vector**
 - `vector<tipo> nomeVar()`
 - `vector<tipo> nomeVar(tamanho)`
 - `vector<tipo> nomeVar(array, tamanho)`
 - `vector<tipo> nomeVar(quantidade, valor)`
 - `iterator: begin(), end()`
 - `insert(iterator, valor)`
 - `insert(iterator, quantidade, valor)`
 - `insert(iterator, array, tamanho)`
 - `size()`
 - `clear()`
 - Documentação
 - <http://www.cplusplus.com/reference/vector/vector/>

Classe stack



- Manipulação de pilhas (LIFO)
 - Classe **stack**
 - stack<tipo> nomeVar;
 - push(elemento);
 - top();
 - pop();
 - size();
 - Documentação
 - <http://www.cplusplus.com/reference/stack/stack/>

Classe queue



- Manipulação de filas (FIFO)
 - Classe **queue**
 - queue<tipo> nomeVar;
 - push(elemento);
 - front();
 - pop();
 - size();
 - Documentação
 - <http://www.cplusplus.com/reference/queue/queue/>

Função qsort



- Ordenação
 - Quicksort (**qsort**)
 - Array de elementos
 - Posições totais do array
 - Tamanho de cada elemento do array
 - Função comparadora
 - Função comparadora
 - Recebe dois objetos
 - Retorna -1 se o primeiro for menor
 - Retorna 0 se forem iguais
 - Retorna 1 se o primeiro for maior
 - <http://www.cplusplus.com/reference/cstdlib/qsort/>

Função bsearch



- Busca
 - Busca binária (*bsearch*)
 - Elemento a ser buscado
 - Array de elementos
 - Posições totais do array
 - Tamanho de cada elemento do array
 - Função comparadora
 - Documentação
 - <http://www.cplusplus.com/reference/cstdlib/bsearch/>

Aulas práticas e manuais on-line



Assista agora as aulas práticas.

[Clique aqui](#) para visualizar as aulas práticas disponíveis
