# C++

## Básico ao Avançado

### O primeiro tipo do bebê

Heitor Rodrigues Savegnago

UFABC Rocket Design

2017.3

# Listando opções

- Qual previsão do tempo pra hoje?
  - Chuva?
  - Sol?
  - Nublado?
  - Neve?
  - Tufão?
  - Tsunami?
  - Chuva de meteoros?
  - Lagarto gigante?
- Montar um programa que mostra o que fazer conforme previsão do tempo
- Mas como fazer a escolha? Talvez um código para cada clima...
  1. Chuva → pegar guarda-chuva
  2. Sol → pegar protetor solar
  3. Nublado → pegar um casaco
  4. Neve → pegar um chocolate quente
  5. Tufão → fugir para o porão (?)
  6. Tsunami → fugir para as montanhas (?)
  7. Chuva de meteoros → fugir pra lua (!)
  8. Lagarto gigante → chamar os King Kong (!)

# Legal, e?

- Ajudaria muito usar um `switch`
- Qual código atribuir para cada situação?
  - Ordem lexicográfica? (alfabética)
    - Tufão, tornado, furacão, ciclone...
  - Nível de periculosidade?
    - Um meteoro pode gerar uma tsunami...
  - Tamanho da palavra?
- Basicamente precisamos enumerar estes itens
- E se existisse um modo de colocar códigos de maneira mais simples?
- E se exitisse um tipo que só pode receber valores pré determinados?
- E se este tipo só pudesse receber um valor entre os descritos?
- Tudo isso é possivel com o poder da **imaginação**
- É o que?

# Ilustrando

- Primeiro vamos ver o código sem estas vantagens
- Lembrando que o que define comando é o ponto-e-vírgula

```
//...
int clima = 0;
//...
switch(clima)
{
  case 1: /*Chuva*/ /*...*/ break;
  case 2: /*Sol*/ /*...*/ break;
  case 3: /*Nublado*/ /*...*/ break;
  case 4: /*Neve*/ /*...*/ break;
  case 5: /*Tufão*/ /*...*/ break;
  case 6: /*Tsunami*/ /*...*/ break;
  case 7: /*Meteoros*/ /*...*/ break;
  case 9: /*Lagarto gigante*/ /*...*/ break;
}
//...
```

- Parece O.K.
- Mas e se a variável não estiver dentro dos valores válidos?

# Enumerando

- Podemos definir um novo tipo
  - Apenas recebe valores específicos
  - Apenas recebe valores por palavras-chave
  - O programador escolhe as palavras-chave
  - Este tipo precisa ter um nome
- A palavra-chave `enum` é utilizada para criar estes novos tipos
- Operadores são inconsistentes mas podemos mudar isso utilizando polimorfismo
- A sintaxe lembra vetores

```
enum <nomeTipo> {<val1>, ..., <valN>};
//...
<nomeTipo> <nomeVar>;
//...
```

# Voltando ao clima

```
//...
enum clima {Chuva,Sol,Nublado,Neve,Tufao,Tsunami,ChuvaMeteoro
    ,LagartoGigante};
//...
clima Hoje = Nublado;
//...
switch(Hoje)
{
  case Chuva: /*...*/ break;
  case Sol: /*...*/ break;
  case Nublado: /*...*/ break;
  case Neve: /*...*/ break;
  case Tufao: /*...*/ break;
  case Tsunami: /*...*/ break;
  case ChuvaMeteoro: /*...*/ break;
  case LagartoGigante: /*...*/ break;

}
//...
```

# Abstraia

- Pensa numa construção
- Ela pode ter portas, quantas?
- Ela pode ter janelas, quantas?
- Ela pode ter cômodos, quantos?
- Ela pode andares, quantos?
- A faichada é pintada de alguma cor, qual?
- Podemos criar variáveis para armazenar cada valor
- Mas isso não garante que estes valores estarão associados
- E se existir outra construção? Mais variáveis?

# Estruture

- É possível criar um tipo que armazene mais de um valor? Vetores!
- Mas vetores só funcionam para valores do mesmo tipo
- Será que existe algo que armazene vários valores de tipos diferentes?
- Sim! A `struct`
- Semelhante ao `union`
- Dentro dela, colocamos várias variáveis
- E o mais legal: ela é um tipo, e podemos declarar variáveis dela

```
struct <nomeTipo>
{
  <tipo1> <nome1>;
  ...
  <tipoN> <nomeN>;
};
//...
<nomeTipo> <nomeVar>;
//...
<nomeVar>.<nome1> = <val1>;
```

# Horário

```
//...
struct relogio
{
  int hora;
  int minuto;
  int segundo;
};
//...
relogio tictac;
tictac.hora = 0;
tictac.minuto = 0;
tictac.segundo = 0;
//...
```

# Unidos somos um

- Irmão estranho da `struct`
- Só uma das variáveis pode ser utilizada
- Como assim mano?
- Ele só reserva memória pra salvar um dos valores
- A memória reservada é a com o tamanho da maior variável
- Usado para casos de valor exclusivo

```
union <nomeTipo>
{
  <tipo1> <nome1>;
  ...
  <tipoN> <nomeN>;
};
//...
<nomeTipo> <nomeVar>;
//...
<nomeVar>.<nome1> = <val1>;
```

# Mostra ae

```
//...
union numero
{
  int inteiro;
  double flutuante;
};
//
numero X;
X.inteiro = 10;
//...
X.flutuante = 15.883;
//...
```

# Vamos testar!