C++
Básico ao Avançado
Faça o que eu digo

Heitor Rodrigues Savegnago

UFABC Rocket Design

2017.3

Dificuldade

- Algum fluxos são muito complexos
- Poucos são os casos onde as estruturas básicas não são suficientes
- *Vocês lembram das estruturas básicas?*
- `if...else` e `while`
- Vamos pensar num caso onde fica complicado... na lousa

Indo direto

- Existe um controle de fluxo especial
- Tão antigo quanto a própria programação
- Criado pela mãe da programação
- Utilizado em *Assembly*
- Serve para *pular* para outras partes do código
- Normalmente é associado a um `if`

Vá para

- Sua sintaxe é herdada do *Assembly*
- A palavra-chave é `goto`
- É necessário definir uma *etiqueta* (*label*)
  - Para definir uma *etiqueta*, utilizamos dois pontos (`:`)
- Pode ser utilizado antes da definição da *etiqueta*

```
//...
<etiqueta>:
//...
goto <etiqueta>;
//...
```

- Não faz sentido usar sem `if`

Dois casos

```
//...
inicio:
if(!<cond>) goto fim;
{
  <comand1>;
  //...
  <comandN>;
}
goto inicio;
fim:
//...
```

```
//...
inicio:
{
  <comand1>;
  //...
  <comandN>;
}
if(<cond>) goto inicio;
//...
```

Dois casos

```
//...
while(<cond>)
{
  <comand1>;
  //...
  <comandN>;
}
//...
```

```
//...
do
{
  <comand1>;
  //...
  <comandN>;
}
while(<cond>);
//...
```

Criando problemas

- Existe uma palavra-chave especial que generaliza tipos
- Com ela, não é mais necessário escolher o tipo da variável
- O compilador analisará o uso da variável no programa e escohará seu tipo
- E isso nos dá uma lista de problemas:
  - Inconsistência de tipo
  - Exige definição do uso para escolha
  - *Casting* implícito desnecessário
  - O compilador pode escolher um `double` onde um `float` era suficientes
- É recomendado quando:
  - Tem experiência com tipos
  - Não se conhece o tipo de retorno de uma função
  - Apenas interessa utilizar uma varável auxiliar
- A palavra-chave é `auto`, que substitui o tipo na sintaxe padrão

Generalizando

- Uma função que soma dois valores depende de um tipo
- Este tipo define o retorno da função e os argumentos de entrada
- Nem sempre um `auto` serve
- Um exemplo é uma variável que somente é declarada
- É possivel declarar uma função que é generaliza
- Servindo até para tipos abstratos
- Especialmente **eu** prefiro criar classes genericas, ou classes *template*
- A sintaxe é igual para classes ou funções/procedimentos

Colocando templates

```
class alpha
{
private:
  int N;
  int *P;
public:
  alpha(int n)
  {
    N = n;
    P = new int[N];
  }
  ~alpha()
  {
    delete[] P;
  }
```

Colocando templates

```
template <class T> class alpha
{
private:
  int N;
  T *P;
public:
  alpha(int n)
  {
    N = n;
    P = new T[N];
  }
  ~alpha()
  {
    delete[] P;
  }
```

Funções/procedimentos

```
template <class T> class alpha
{
  //...
public:
  //...
  T operator[](int i)
  {
    return P[i];
  }
  void setItem(T item);
};

template <class T> void alpha::setItem(T item, int i)
{
  P[i] = T;
}
```

Declarando

- A alteração fica na instanciação do objeto da classe
- Ela recebe o tipo selecionado

```
alpha <int> primeiro(10);
alpha <char> segundo(20);
alpha <float> terceiro(30);
```

- Quando a função/procedimento não é membro de classe, não é necessário definir o tipo
- Mais de um *template* pode ser utilizado simutaneamente

Vamos testar!