Programação Genérica em C++

DEC0006

Prof. Martín Vigil

Revisão

C/C++ é uma linguagem com tipagem estática

- Programador precisa definir o tipo de cada variável
- Compilador verifica tipos em tempo de compilação

```
1   #include <string>
2
3   int main(){
4       float sum = 0;
5
6       sum = std::string("hello");
7
8       return 0
9   }
```

```
main.cpp:6:11: error: assigning to 'float' from incompatible type 'std::string' (aka 'basic_string<char,
      char_traits<char>, allocator<char> >')
    sum = std::string("hello");
          ^~~~~~~~~~~~~~~~~~~~
1 error generated.
```

Problemática

Nem sempre o programador sabe os tipos *a priori*

- Necessidade generalizar tipos em alguns algoritmos
- Exemplos:
  - Encontrar máximo entre: inteiros, floats, doubles
  - Ordenar: inteiros, floats, doubles, strings

Solução de repetir código (baixa manutenibilidade)

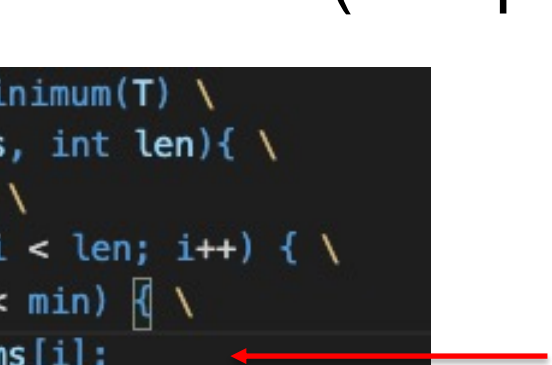
```
17  int get_minimum(int* nums, int len){
18      int min = nums[0];
19      for (int i = 1; i < len; i++) {
20          if (nums[i] < min) {
21              min = nums[i];
22          }
23      }
24      return min;
25  }
26
27  int get_minimum(float* nums, int len){
28      float min = nums[0];
29      for (int i = 1; i < len; i++) {
30          if (nums[i] < min) {
31              min = nums[i];
32          }
33      }
34      return min;
35  }
```

Solução de utilizar macros (complexa)

```
3   #define define_get_minimum(T) \
4   T get_minimum(T* nums, int len){ \
5       T min = nums[0]; \
6       for (int i = 1; i < len; i++) { \
7           if (nums[i] < min) { \
8               min = nums[i]; \
9           } \
10      } \
11      return min; \
12  }
13
14  define_get_minimum(int)
15  define_get_minimum(float)
16
17  int main(){
18      int intArray[5] = {0,1,2,3,4};
19      float floatAarray[5] = {0.2f, 0.4f, 1.2f, 0.7f, 1.9f};
20      std::cout << get_minimum(intArray,5) << std::endl;
21      std::cout << get_minimum(floatAarray,5) << std::endl;
22  }
```

Solução de utilizar macros (complexa)

```
3   #define define_get_minimum(T) \
4   T get_minimum(T* nums, int len){ \
5       T min = nums[0]; \
6       for (int i = 1; i < len; i++) { \
7           if (nums[i] < min) { \
8               min = nums[i];
9           } \
10      } \
11      return min; \
12  }
```



```
main3-macros.cpp:22:2: error: expected '}'
}
 ^
main3-macros.cpp:14:1: note: to match this '{'
define_get_minimum(int)
^
main3-macros.cpp:6:35: note: expanded from macro 'define_get_minimum'
    for (int i = 1; i < len; i++) { \
                                  ^
main3-macros.cpp:22:2: error: expected '}'
}
 ^
main3-macros.cpp:14:1: note: to match this '{'
define_get_minimum(int)
^
main3-macros.cpp:4:32: note: expanded from macro 'define_get_minimum'
T get_minimum(T* nums, int len){ \
                               ^
8 errors generated.
```

Solução usando C++

Programação em duas etapas

1 Declarar função ou classe com tipos genéricos (*templates*)

2 Usar função ou classe usando tipos específicos

Tipos genéricos: duas alternativas equivalentes

1. template <class NomeDoTipo>
2. template <typename NomeDoTipo>

Declarando e usando funções template (altern #1)

```
4   template<typename T>
5   T get_minimum(T* nums, int len){
6       T min = nums[0];
7       for (int i = 1; i < len; i++) {
8           if (nums[i] < min) {
9               min = nums[i];
10          }
11      }
12      return min;
13  }
14
15  int main(){
16      int intArray[5] = {0,1,2,3,4};
17      float floatAarray[5] = {0.2f, 0.4f, 1.2f, 0.7f, 1.9f};
18      std::cout << get_minimum<int>(intArray,5) << std::endl;
19      std::cout << get_minimum<float>(floatAarray,5) << std::endl;
20  }
```

Declarando e usando funções template (altern #2)

```
4   template<class T>
5   T get_minimum(T* nums, int len){
6       T min = nums[0];
7       for (int i = 1; i < len; i++) {
8           if (nums[i] < min) {
9               min = nums[i];
10          }
11      }
12      return min;
13  }
14
15  int main(){
16      int intArray[5] = {0,1,2,3,4};
17      float floatAarray[5] = {0.2f, 0.4f, 1.2f, 0.7f, 1.9f};
18      std::cout << get_minimum<int>(intArray,5) << std::endl;
19      std::cout << get_minimum<float>(floatAarray,5) << std::endl;
20  }
```

Declarando e usando classes template

```
4   template<typename T>
5   class MeuVetor {
6       protected :
7           T* _array;
8           int _len;
9
10      public :
11          MeuVetor(T* array, int len){
12              this->_array = array;
13              this->_len = len;
14          }
15
16          T get_minimum(){
17              T min = this->_array[0];
18              for (int i = 1; i < this->_len; i++)
19                  if (this->_array[i] < min)
20                      min = this->_array[i];
21              return min;
22          }
23  };
24
25  int main(){
26      int intArray[5] = {0,1,2,3,4};
27      float floatArray[5] = {0.2f, 0.4f, 1.2f, 0.7f, 1.9f};
28
29      MeuVetor<int> meuVetorInt(intArray,5);
30      MeuVetor<float> meuVetorFloat(floatArray,5);
31
32      std::cout << meuVetorInt.get_minimum() << std::endl;
33      std::cout << meuVetorFloat.get_minimum() << std::endl;
34  }
```

Uma função/classe pode utilizar múltiplos templates

```
4   template<typename T, typename U>
5   class MeuVetor {
6       protected :
7           T* _array1;
8           int _len1;
9           U* _array2;
10          int _len2;
11
12      public :
13          MeuVetor(T* array1, int len1, U* array2, int len2){
14              this->_array1 = array1;
15              this->_len1 = len1;
16
17              this->_array2 = array2;
18              this->_len2 = len2;
19          }
20          T get_minimum1(){
21              T min = this->_array1[0];
22              for (int i = 1; i < this->_len1; i++)
23                  if (this->_array1[i] < min)
24                      min = this->_array1[i];
25              return min;
26          }
27          U get_minimum2(){
28              U min = this->_array2[0];
29              for (int i = 1; i < this->_len2; i++)
30                  if (this->_array2[i] < min)
31                      min = this->_array2[i];
32              return min;
33          }
34  };
35
36  int main(){
37      int intArray[5] = {0,1,2,3,4};
38      float floatArray[5] = {0.2f, 0.4f, 1.2f, 0.7f, 1.9f};
39      MeuVetor<int, float> meuVetorIntFloat(intArray,5, floatArray, 5);
40      std::cout << meuVetorIntFloat.get_minimum2() << std::endl;
41  }
```

Limitação: não separar código em arquivos .h e .cpp

MeuVetor.h > MeuVetor<T>

```
1   template<typename T>
2   class MeuVetor {
3       protected :
4           T* _array;
5           int _len;
6
7       public :
8           MeuVetor(T* array, int len);
9
10          T get_minimum();
11  };
```

MeuVetor.cpp

```
1   #include "MeuVetor.h"
2
3   template<typename T>
4   MeuVetor<T>::MeuVetor(T* array, int len){
5       this->_array = array;
6       this->_len = len;
7   }
8
9   template<typename T>
10  T MeuVetor<T>::get_minimum(){
11      T min = this->_array[0];
12      for (int i = 1; i < this->_len; i++)
13          if (this->_array[i] < min)
14              min = this->_array[i];
15      return min;
16  }
```

```
Undefined symbols for architecture x86_64:
  "MeuVetor<float>::get_minimum()", referenced from:
      _main in main4-df0173.o
  "MeuVetor<float>::MeuVetor(float*, int)", referenced from:
      _main in main4-df0173.o
  "MeuVetor<int>::get_minimum()", referenced from:
      _main in main4-df0173.o
  "MeuVetor<int>::MeuVetor(int*, int)", referenced from:
      _main in main4-df0173.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

Síntese da aula

- Templates permitem generalizar tipos em C++
- Forma de utilizar templates:
  - Escreva função/classe generalizando tipos
  - Utilize (instancie) função/classe especificando tipos
- Duas formas de generalizar com o mesmo efeito:
  - template <typename NomeDoTipo>
  - template <class NomeDoTipo>