Curso de C/C++ Avançado

Aula 7 – Templates, namespaces, Exceções, Casts

Allan Lima – http://allanlima.wordpress.com

- *Você pode:*
  - *copiar, distribuir, exibir e executar a obra*
  - *criar obras derivadas*
  - *fazer uso comercial da obra*
- *Sob as seguintes condições:*
  - ***Atribuição**. Você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante.*
  - ***Compartilhamento pela mesma Licença**. Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.*
  - *Para cada novo uso ou distribuição, você deve deixar claro para outros os termos da licença desta obra.*
  - *Qualquer uma destas condições podem ser renunciadas, desde que Você obtenha permissão do autor.*
- ***Veja aqui a licença completa***

Templates

- *São mecanismos que permitem a definição genérica de funções e classes sem especificar os tipos de implementação*
- *Ajudam a reutilizar o código*
- *São muito bons para a construção de bibliotecas*

Templates de Funções

- *Definem um algoritmo genérico e independente de tipo*
- *Exemplo:*
  - *Busca em um array*
  - *Maximo entre elementos*
  - *Ordenação de um array*
  - *Mínimo entre elementos*

Templates de Funções

- *Permitem a criação de funções genéricas*
  - *Recebendo qualquer tipo de dado como parâmetro*
  - *Retornando qualquer tipo de dado*
- *Uma única função criada pode ser aplicada a qualquer tipo*
- *Sintaxe:*
  - *template <class identificador> função;*
  - *template <tipo identificador> função;*

Exemplo

```
template<class T>
T maximo(T a, T b) {
   return (a > b) ? a : b;
}
```

```
int a = maximo(10, 45) ;
double d = maximo(10.5, 5.06);
// como a e b devem ter o mesmo tipo devemos usar o  <double>
double c = maximo <double> (5, 5.06);
```

Exemplo

- *exemploTemplatesFuncoes.cpp*

Templates de Classes

- *Problema:*
  - *Queremos criar uma classe Stack que pode ser de inteiros, floats, strings, ..., ou qualquer outro tipo definido pelo programador*
  - *Queremos criar uma classe Stack* ***genérica***
- *Solução:*
  - *Utilizar templates de classes*

```
template <class T>
class Stack { ... };
```

Exemplo

- *exemploTemplatesClasses.cpp*

Especialização de Templates

- *Muitas vezes o comportamento genérico não é capaz resolver todos os problemas*
- *Exemplo:*
  - *Stack<char *> sChar(10);*
    - E se liberarem a memória da string?
    - Este objeto não cria cópias das strings, faz apenas cada elemento seu apontar para o mesmo endereço da string que foi passada como parâmetro
  - *Solução:*
    - Especialização de Templates

Exemplo

```
// não precisamos mais do template <class T>
void Stack<char*>::push(char* &element) {
    if (this->nextIndex != -1) {
        this->elements[this->nextIndex]
            = new char[strlen(element) + 1];
        strcpy(this->elements[this->nextIndex], element);
        this->nextIndex--;
    }
}
```

Especialização de Templates

- *Também podemos especializar um Template de Classe ou uma função qualquer*
- *Assim garantimos o comportamento correto para tipo*

Exemplo

- *exemploEspecializacaoTemplates.cpp*

Especialização de Templates

- *Podemos especializar Templates de classes*
- *É útil quando os comportamentos são muito diferentes*
- *Estas classes devem redefinir todos os membros para o tipo específico*
- *Também podemos criar novos membros*
- *Sintaxe:*

```
template <>
class nomedaClasse <Tipo> { ... };
```

Exemplo

- *exemploEspecializacaoClasses.cpp*

namespaces

- *Um namespace é um mecanismo para expressar um agrupamento lógico*
- *Sintaxe:*

```
namespace nomeDoNamespace {
        corpoDoNamespace
}
```

namespaces

- *Podemos utilizar um namespace para agrupar diversas funções, classes, variáveis ....*
- *Por exemplo, se tivermos muitas funções para realização de operações matemáticas podemos criar um namespace para todas*

Exemplo:

- *exemploNamespace.h*

namespaces

- *Podemos acessar os membros de um namespace de duas maneiras diferentes:*
  - *Usando o operador* ***::***
  - *Através do comando using namespace*

Exemplo

```
#include <iostream>
#include "exemploNamespace.h"
using namespace Mat;

int main() {
    std::cout << maximo(10, 56) << std::endl;
    std::cout << minimo(10, 56) << std::endl;
    std::cout << PI << std::endl;
    NumeroComplexo c;
    BigInteger b;
    return 0;
}
```

namespaces

- *namespaces podem ser utilizados quando tivermos mais de uma função com o mesmo protótipo.*
- *Exemplo:*
  - *exemploNamespaceFuncoes.cpp*

Exceções

- *Como podemos descobrir quando um erro ocorre no nosso programa?*
- *Podemos utilizar o conceito de exceção*
- *Exceções são erros que ocorrem em tempo de execução*
- *Lançando uma exceção:*
  - *throw nomeDaExceção;*

Exceções

- *Tratando uma exceção:*

```
try {
    // comandos
} catch (TipoDaExceção) {
    // código executado quando ocorre uma exceção
}
```

- *Obs.: Com o comando throw podemos lançar qualquer coisa. E com o try podemos tratar qualquer coisa que foi lançada.*

Exceções

- *C++ permite o uso de try-catch's aninhados*
- *Também permite o uso de vários catch's para um único try*
- *Quando queremos tratar uma exceção qualquer podemos fazer catch (...)*

Exemplos

- *exemploExcecoes1.cpp*
- *exemploExcecoes2.cpp*

Exceções

- *A biblioteca padrão de C++ contém um conjunto de exceções predefinido*
- *Estas exceções herdam da classe std::exception que foi definida no header <exception>*

Exceções

Exceções da Biblioteca Padrão de C++

```
exception
  bad_alloc           (thrown by new)
  bad_cast            (thrown by dynamic_cast when fails with a referenced type)
  bad_exception       (thrown when an exception doesn't match any catch)
  bad_typeid          (thrown by typeid)
  logic_error
    domain_error
    invalid_argument
    length_error
    out_of_range
  runtime_error
    overflow_error
    range_error
    underflow_error
  ios_base::failure   (thrown by ios::clear)
```

Exemplo

```
class A {
    virtual void a() {};
};
int main() {
    try {
        A *a = NULL;
        typeid(*a);
    } catch (std::exception &e) {
        cout << e.what() << endl;
    }
    return 0;
}
```

Cast

- *O operador de cast tradicional pode ser aplicado de forma indiscriminada para fazer conversões entre tipos*
- *Exemplo:*
  - *exemploCast.cpp*

Cast

- *Em C++ o cast no estilo antigo é obsoleto (deprecated)*
- *Em compensação C++ possui novos operadores de cast:*
  - *static_cast <novoTipo> (expressão)*
  - *const_cast <novoTipo> (expressão)*
  - *reinterpret_cast <novoTipo> (expressão)*
  - *dynamic_cast <novoTipo> (expressão)*
- *Com eles podemos obter os mesmos resultados do estilo antigo, porém de forma mais segura*

static_cast

- *Realiza casts mais seguros e portáveis em comparação com o método antigo.*
- *Verifica se algumas conversões de tipos são compatíveis*
- *Pode ser utilizado para realizar conversões entre ponteiros e tipos básicos*
- *Exemplo:*
  - *exemploStaticCast.cpp*

const_cast

- *Utilizado para as conversões:*
  - *Constante ⇔ Variável*
  - *Variável volátil ⇔ Variável não volátil*
- *Exemplo:*
  - *exemploConstCast.cpp*

reinterpret_cast

- *Utilizado para as conversões:*
  - *Ponteiro ⇔ Ponteiro*
  - *Inteiro ⇔ Ponteiro*
- *Não faz qualquer verificação de tipo*
- *É o mais perigoso dos operadores de cast*
- *Exemplo:*
  - *exemploReinterpretCast.cpp*

dynamic_cast

- *Utilizado para a conversão entre ponteiros e referências para objetos*
- *Verifica se a operação é válida em tempo de execução.*
  - *Se não for retorna NULL*
- *Exemplo:*
  - *exemploDynamicCast.cpp*

Violando o encapsulamento

- *Podemos usar o reinterpret_cast para violar o encapsulamento dos atributos de uma classe*
- *É só criar uma classe com os mesmos atributos da classe original, mas com o modificador de acesso public*
- *Exemplo:*
  - *exemploEncapsulamento.cpp*

typeid

- *É utilizado para saber informações sobre os tipos*
- *Syntaxe:*
  - *type_info typeid(variavel);*
- *A classe type_info possui o método name() que retorna o nome do tipo*
- *A implementação deste método não é padronizada e depende do compilador*
- *A classe type_info também sobrecarrega os operadores == e !=*
- *Para poder utilizar a classe type_info precisamos incluir o header <typeinfo>*

Exemplo

- *exemploTypeid1.cpp*
- *exemploTypeid2.cpp*

Exercícios

1) *Crie um template chamado Vetor*
   - *Esta classe irá guardar um array de elementos genéricos*
   - *Os seu métodos são:*
     - void inserir(T &elemento);
     - int procurarIndice(T &elemento);
     - bool contem(T &elemento);
     - void remover(T &elemento);
     - Faça a sua classe funcionar corretamente com char *
     - Implemente o operador []
   - *Declare a sua classe dentro de um namespace*
   - *Lance uma exceção quando o método inserir for chamado e o array já estiver cheio*

Referências

- *Stroustrup, Bjarne. The C++ Programming Language, Special Edition*
- *Eckel, Bruce. Thinking in C++, 2nd ed. Volume 1*
- *Slides de Gustavo (ghcp@cin.ufpe.br) do curso de C/C++*