Templates

Prof. Valdir Pedrinho de Tomin Junior, Eng.

Universidade Federal de Santa Catarina
Centro Tecnológico de Joinville
EMB5631 – Programação III

e-mail: valdir.pedrinho@ufsc.br

9 de outubro de 2018

Introdução

Os templates

Os templates de função e os templates de classe permitem aos programadores especificar, com um único segmento de código, uma série inteira de funções relacionadas (sobrecarregadas) — chamadas especializações de template de função — ou uma série inteira de classes relacionadas — chamadas especializações de template de classe. Essa técnica é chamada programação genérica.

Especializações

Poderíamos escrever um único template de classe para uma classe de pilha e, então, fazer o C++ gerar especializações separadas de template de classe, como uma pilha de classe int , uma de float , uma de string e assim por diante.

Templates de Funções

Templates vs sobrecargas

- As funções sobrecarregadas normalmente realizam operações semelhantes ou idênticas sobre tipos de dados diferentes;
- Se forem idênticas para cada tipo, as operações podem ser expressas de maneira mais compacta e conveniente utilizando templates de função;
- Com base nos tipos de argumentos fornecidos explicitamente ou inferidos de chamadas para essa função, o compilador gera funções de código-objeto separadas (isto é, especializações de template de função) para tratar cada chamada de função de modo apropriado.

Resolvidos em tempo de compilação

O tipo de dado que a função / classe template irá manipular é definido pelo compilador ainda em tempo de compilação.

Templates de Funções

Como usar?

Todas as definições de template de função começam com a palavra-chave template seguida por uma lista de parâmetros de template para o template de função entre colchetes angulares (< e >).

Sintaxe

```
template <typename T, ..., typename Z>
 tipo funcao(lista de parâmetros)
```

Chamada:

```
funcao<T> (parâmetros); // quando retorna um tipo T
funcao (parâmetros);
```

Exemplo - printArray

Programa 14.1 do livro *DEITEL. C++, Como programar. 5ª ed.*

Sobrecarregando Funções Template

Sobrecarregar templates segue as mesmas regras já vistas

- Podemos fornecer outros templates de função que especificam o mesmo nome de função mas parâmetros de função diferentes;
  - Por exemplo, o template de função printArray do exemplo anterior poderia ser sobrecarregado com outro template de função printArray com os parâmetros adicionais lowSubscript e highSubscript para especificar a parte do array a ser gerada para a saída;
- Um template de função também pode ser sobrecarregado fornecendo funções não-template com o mesmo nome de função, mas com diferentes argumentos de função;
  - Por exemplo, o template de função printArray da Figura 14.1 poderia ser sobrecarregado com uma versão não-template que imprime especificamente um array de strings;

Sobrecarregando Funções Template

Exercício

Sobrecarregue o template de função printArray do exemplo anterior de modo que ele aceite dois argumentos do tipo inteiro adicionais, a saber, int lowSubscript e int highSubscript . Uma chamada para essa função imprimirá somente a parte designada do array. Valide lowSubscript e highSubscript; se qualquer uma estiver fora do intervalo ou se highSubscript for menor que ou igual a lowSubscript , a função sobrecarregada printArray deve retornar 0; caso contrário, printArray deve retornar o número de elementos impressos.

Template de Classes

Tipos Parametrizados

- Os templates de classe são chamados de tipos parametrizados, porque requerem um ou mais parâmetros de tipo para especificar como personalizar um template de 'classe genérica' para formar uma especialização de template de classe;
- O programador que quiser produzir uma variedade de especializações de template de classe escreverá somente uma definição de template de classe;
- Toda vez que uma especialização de template de classe adicional for necessária, o programador utilizará uma notação simples e concisa, e o compilador escreverá o código-fonte para a especialização requerida pelo programador.

Template de Classes

Sintaxe

```
template <typename T, ..., typename Z>
class NomeDaClasse
{
   ...
}
```

Chamada:

```
NomeDaClasse<T> objeto; // para um tipo T
```

Exemplo - Stack

Programa 14.2-3 do livro *DEITEL. C++, Como programar. 5ª ed.*

Exemplo - Stack e templates de função

Programa 14.4 do livro *DEITEL. C++, Como programar. 5ª ed.*

Parâmetros sem tipo e tipos-padrão para templates de classes

Parâmetros não-tipo

É possível utilizar os parâmetros de template não-tipo ou parâmetros não-tipo, que podem ter argumentos-padrão e são tratados como consts.

Exemplo – Stack

```
template< typename T, int elements >
class Stack {...};

Stack< double, 100 > mostRecentSalesFigures;
```

poderia instanciar uma Stack de 100 elementos se a classe possuir um array como atributo private:

```
T stackHolder[ elements ];
```

Parâmetros sem tipo e tipos-padrão para templates de classes

Parâmetros tipo-padrão

Um parâmetro de tipo pode especificar um tipo-padrão

Exemplo

```
template< typename T = string > // por padrão,
    assume o tipo string
```

Especialização Explícita

Se um tipo qualquer não funcionar com o template ou exigir processamento personalizado, pode-se definir uma especialização explícita do template para um tipo particular

Parâmetros sem tipo e tipos-padrão para templates de classes

Especialização Explícita

Exemplo

Vamos supor que quiséssemos criar uma especialização explícita de Stack para os objetos Employee . Para fazer isso, forme uma nova classe com o nome Stack< Employee > como segue:

```
template<>
class Stack< Employee >
{// corpo da definição de classe
};
```

Exemplo

Função f

Formas de Relacionamento

- Um template de classe pode ser derivado de uma especialização de template de classe;
- Um template de classe pode ser derivado de uma classe não-template;
- Uma especialização de template de classe pode ser derivada de uma especialização de template de classe;
- Uma classe não-template pode ser derivada de uma especialização de template de classe.

Funções Friends e Classes Templates

Define-se o template de classe:

```
template<typename T> class X {...};
```

- É possível tornar uma função f1 friend de cada especialização de template de classe instanciada:

  ```
  friend void f1();
  ```

  Por exemplo, a função f1 é friend de X<double> e X<string>.
- É possível tornar uma função f2 friend apenas de uma especialização de template de classe com o mesmo argumento de tipo:

  ```
  friend void f2(X<T> &);
  ```

  Por exemplo, se T for um float , a função f2(X<float> &) é friend da especialização de template de classe X<float>.

Funções Friends e Classes Templates

- É possível declarar um método de outra classe como friend de qualquer especialização de template de classe gerada a partir do template de classe.

```
friend void A::f3();
```

  A declaração torna o método f3 da classe A friend de cada especialização de template de classe instanciada a partir do template de classe precedente. Por exemplo, a função f3 é friend de X<double> e X<string>.

- Um método de outra classe só pode ser friend de uma especialização de template de classe com o mesmo argumento de tipo:

```
friend void C<T>::f4(X<T> & );
friend void C<float>::f4(X<float> & );
```

Classes Friends e Classes Templates

- É possível tornar o conjunto de uma classe inteira de métodos friend de um template de classe:

```
friend class Y;
```

- É possível tornar todas os métodos de uma especialização de template de classes friends de outra especialização de template de classe com o mesmo argumento de tipo:

```
friend class Z<T>;
```

Templates e Membros Static

E quanto aos membros de dados static?

- Toda especialização de template de classe instanciada de um template de classe tem sua própria cópia de cada membro de dados static do template de classe;
- Todos os objetos dessa especialização compartilham esse único membro de dados static;
- Como ocorre com membros de dados static de classes não-template, os membros de dados static de especializações de template de classe devem ser definidos e, se necessário, inicializados no escopo de arquivo;
- Toda especialização de template de classe obtém sua própria cópia das funções-membro static do template de classe.

Fila Duplamente Encadeada com Prioridade

Exercício

Implemente uma fila duplamente encadeada template, com prioridade na inserção dos elementos. A definição da classe Fila é dada a seguir:

```
template <typename tipo> class Fila {
    NoDERank<tipo> * ptr_cabeca, * ptr_cauda;
    unsigned num_elementos;
public:
    Fila(void); ~Fila(void);
    const tipo & frente(void);
    const tipo & cauda(void);
    void enfileira(const tipo & elemento, unsigned
        rank = numeric_limits<unsigned>::max());
    void desenfileira(void);
    bool vazia(void);
    unsigned tamanho(void);};
```

Quanto menor o **rank** maior a prioridade. Inclua <**limits**>.

Referências e Exercícios

Livro Texto

Paul Deitel e Harvey Deitel.**C++: Como programar**. 5ªed. Ed. Prentice Hall Brasil. 2006.

Conteúdo dessa Aula

- Capítulo 14.3 ao 14.23

Exercícios do Livro

Exercícios do capítulo 14.