Universidade Federal de Mato Grosso do Sul
Faculdade de Computação

TÓPICOS EM COMPUTAÇÃO II 2022

Paulo A. Pagliosa

Prova Substitutiva
07/07/2022

Responda as questões abaixo em um arquivo texto com extensão `.txt`, identificado com seu nome. Codifique a questão de implementação em um arquivo chamado `myprintf.h`, contendo comentários com seu nome. Escreva o restante de código em um arquivo com extensão `.cpp`, identificado com seu nome. Compacte todos os arquivos em um arquivo `.zip`, identificado com seu nome, e o submeta via AVA.

QUESTÃO DE PROGRAMAÇÃO (3.5)

Escreva código para que o `template` de função

```
template <typename... Args>
void myprintf(const char* format, Args&&... args);
```

instancie uma função `myprintf` para escrita na saída padrão que seja "segura". O parâmetro `format` é uma cadeia de caracteres (exceto `%`) que são copiados na saída padrão (por simplicidade, não são considerados caracteres de controle tais como `\n`). Quando um `%` é encontrado, este deve ser seguido por um caractere *especificador de conversão*. Se não houver caractere após `%`, então a cadeia de formato está mal formada e uma exceção deve ser lançada. Se o especificador de conversão for `%`, então `%` é copiado para a saída padrão. Para qualquer outro especificador de conversão espera-se um argumento em `args` de um tipo válido para o especificador. Se não houver um argumento para um especificador de conversão, ou se o argumento for de um tipo inválido, ou se o especificador de conversão for inválido, uma exceção deve ser lançada. Os especificadores de conversão válidos são:

Caractere	Tipos esperados
`c`	`char`
`s`	`const char*` `const std::string&`

`d`	`int`
	`long`
	`short`

`u`	`unsigned int`
	`unsigned long`
	`unsigned short`

`f`	`float` `double`
`b`	`bool`

Ao final, `\n` é copiado na saída padrão. Escreva uma função de testes para seu `template` (com tratamento de exceções).

QUESTÃO 1 (2.5)

Descreva quais são os (seis) métodos definidos implicitamente pelo compilador para uma classe `X`, quando são gerados e qual a funcionalidade **`default`** de cada um.

QUESTÃO 2 (1.0)

Quais as funcionalidades e diferenças de `std::move` e `std::forward`?

QUESTÃO 3 (2.0)

Considere o seguinte código:

```
template <typename...  Args>
void log(std::ostream& os, Args&&...  args)
{
  (os << ...  << std::forward<Args>(args)) << '\n';
}

using strings = std::vector<std::string>;
using logfunc = std::function<void(const strings&)>;

inline void logText(logfunc f) { f({"UFMS", "FACOM", "TCCII-2022"}); }

inline void testLog(std::ostream& os = std::cout)
{
  logText([&os](const strings& s)
    {
      for (size_t n = s.size(), i = 0; i < n; ++i)
        log(os, i + 1, ':', s[i]);
    });
}
```

Reescreva a função `testLog` usando um *functor* no lugar da expressão lambda.

QUESTÃO 4 (1.0)

Explique a diferença entre **`struct`**s e **`union`**s.

Boa prova!