UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

BCC6002 - Aspectos de Linguagens de Programação

Prof. Dr. Rodrigo Hübner

Aula 10: Pré-processamento de linguagens: *pragmas*, *annotations* e *decorators*.

Introdução

- O pré-processamento de uma linguagem é útil quando queremos modificar o comportamento de um programa já existente.
- É possível por meio de bibliotecas ou diretivas da própria linguagem
- Podem ser “invocadas” por **chamada de função** ou **decoração de código**

`C/C++`: diretivas de pré-compilação

- Algumas diretivas do compilador ou funções de biblioteca, podem serutilizadas para modificar o fluxo de execução do programa.
- Um exemplo é a utilização de atributos do `GCC` tais como: `constructor` e `destructor`.
- Também podemos utilizar funções de bibliotecas, tal como `atexit` da biblioteca `stdlib.h`.
- Ver exemplo em `precomp_startup_exit.c`

`C/C++`: diretiva `pragma`

- Os pragmas em `C` podem variar entre compiladores, pois eles são específicos para a implementação do compilador. Alguns compiladores têm pragmas próprios para fornecer informações adicionais ao compilador ou controlar o comportamento do código gerado.

`C/C++` : diretiva `pragma`

- `#pragma GCC optimize` : Permite otimizar funções específicas com diferentes níveis de otimização.
- `#pragma GCC diagnostic` : Controla mensagens de diagnóstico do compilador.
- `#pragma clang loop` : Controla otimizações específicas para loops.
- `#pragma clang diagnostic` : Controla mensagens de diagnóstico do compilador.
- Ver `pragma_exemplos.c`

`C/C++`: diretiva `pragma`

- Pragmas do `Microsoft Visual C++`:
  - `#pragma warn -rvl`: Oculta os avisos que são gerados quando uma função que deveria retornar um valor não o retorna
  - `#pragma warn -par`: Oculta aqueles avisos que são gerados quando uma função não usa os parâmetros passados para ela
  - `#pragma warn -rch`: Oculta os avisos que são gerados quando um código está inacessível.
- Ver `pragma_warn.c`

`C/C++` : exemplo de diretiva `pragma`

```c
#include <stdio.h>
#include <omp.h>

int main()
{
  #pragma omp parallel num_threads(4)
  {
    int i = omp_get_thread_num();
    printf_s("Thread %d executando!\n", i);
  }
}
```

`C/C++`: diretivas `pragma` e `define`

- `#pragma GCC poison`: Esta diretiva é suportada pelo compilador GCC e é usada para remover completamente um identificador do programa
  - Ver `pragma_poison.c`
- Diretivas `define`: define um código **estático** para o programa

```
#define MAX(a,b) ((a)>(b) ? (a): (b))
```

`Java` : `annotations`

- `@Override` : sobrescreve um método da Superclasse.
- `@Deprecated` : necessário para que o compilador saiba que um método está obsoleto.
- `@SuppressWarnings` : diz ao compilador para ignorar avisos específicos que eles produzem.
- Ver `BuiltinAnnotations.java`

`Java` : `annotations`

- `@FunctionalInterface`: esta anotação foi introduzida no `Java 8` para indicar que a interface deve ser uma interface funcional.
- `@SafeVarargs`: uma afirmação do programador de que o corpo do método ou construtor anotado não executa operações potencialmente inseguras em seu parâmetro `varargs`.

`Python` : `decorators`

- `Decorators` nos permitem envolver uma função para estender o comportamento da função envolvida, sem modificá-la permanentemente.
- Ver `simple_decor.py`

Próxima aula

- Paralelismo e concorrência