Programação Estruturada

Aula 18 - Pré-processador

Yuri Malheiros (yuri@ci.ufpb.br)

Introdução

- O pré-processador edita um programa antes da sua compilação
- `#define` e `#include` são tratadas pelo pré-processador

Como funciona

- `#define` representa uma macro (um nome que representa alguma outra coisa)
- Ao encontrar essa diretiva, o pré-processador guarda o nome da macro e o seu valor
- Quando a macro é encontrada no programa, o pré-processador substitui a macro pelo valor

Como funciona

- `#include` faz com que o pré-processador abra um arquivo e inclua o seu conteúdo como parte do código que vai ser compilado
- Por exemplo, `#include <stdio.h>` instrui o pré-processador a abrir o arquivo `stdio.h` e trazer o seu conteúdo para o programa
    - `stdio.h` tem os protótipos das funções de input e output

Como funciona

- A entrada do pré-processador é um programa em C
- O pré-processador executa as diretivas e as remove
- A saída é outro programa em C (uma versão editada da inicial)
- O compilador recebe a saída do pré-processador

Como funciona

```c
#include <stdio.h>

#define FREEZING_PT 32.0
#define SCALE_FACTOR (5.0 / 9.0)

int main(void) {
  float fahrenheit, celsius;

  printf("Digite uma temperatura em Fahrenheit: ");
  scanf("%f", &fahrenheit);

  celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;
  printf("A temperatura em Celsius é: %f\n", celsius);

  return 0;
}
```

Diretivas

- A maioria das diretivas são de uma dessas três categorias:
- Definição de macro
  - `#define` define uma macro
  - `#undef` remove uma macro
- Inclusão de arquivo
  - `#include` inclui o conteúdo de um arquivo
- Compilação condicional
  - `#if`, `#ifdef`, `#ifndef`, `#elif`, `#else`, `#endif` permitem que partes do código sejam incluídas ou não na compilação
- Ainda existem as diretivas: `#error`, `#line` e `#pragma`

Compilação Condicional

- Suponha que estamos testando um programa para descobrir um erro (debug)
- Vamos chamar a função `printf` para exibir valores em partes críticas do programa
- Não queremos exibir os valores usando `printf` após corrigir o erro
- Entretanto, pode ser uma boa ideia deixar os `printf`s caso necessário futuramente

Compilação Condicional

```
#define DEBUG 1

#if DEBUG
printf("Valor de x: %d\n", x);
printf("Valor de y: %d\n", y);
#endif
```

- Se `DEBUG` for um valor diferente de 0, então os `printf`s são executados

Compilação Condicional

- Com `#ifdef` testamos se um identificador foi definido como uma macro
- `#ifdef DEBUG` testa se `DEBUG` foi definido
- Com `#ifndef` testamos se um identificador não foi definido como uma macro
- `#ifndef DEBUG` testa se `DEBUG` não foi definido

Compilação Condicional

- `#elif` e `else` podem ser usadas com o `#if`, `#ifdef` e `ifndef` para testar uma série de condições

```
#if expressao1
...
#elif expressao2
...
#else
...
#endif
```

Compilação Condicional

- Existem várias macros pré-definidas com informações importantes, por exemplo, qual o sistema operacional
    - https://sourceforge.net/p/predef/wiki/Home/

```c
int main(void ) {
    #ifdef __APPLE__
    printf("mac\n");
    #elif __WINDOWS__
    printf("windows\n");
    #elif __linux__
    printf("linux\n");
    #else
    printf("outro\n");
    #endif

    return 0;
}
```

Macros pré-definidas

- Outras macros pré-definidas são:
- `__LINE__` - número da linha
- `__FILE__` - nome do arquivo
- `__DATE__` - data de compilação (no formato mês dia ano)
- `__TIME__` - hora de compilação (no formato hora:minuto:segundos)

Diretiva #error

- A diretiva `#error` tem essa forma:
  - `#error mensagem`
- Quando o pré-processador encontra a diretiva `#error` ele exibe um erro com a mensagem especificada
- Essa diretiva é usada com frequência junto com a compilação condicional
- Por exemplo, se quisermos retornar um erro se um programa está sendo compilado em um sistema operacional não suportado