Programação de Computadores

Aula #10

O Preprocessador da Linguagem C

Qual a função do preprocessador C? Como o preprocessador C se enquadra no processo de transformação de um código fonte em um executável?

Ciência da Computação – BCC e IBM – 2023/02
Prof. Vinícius Fülber Garcia

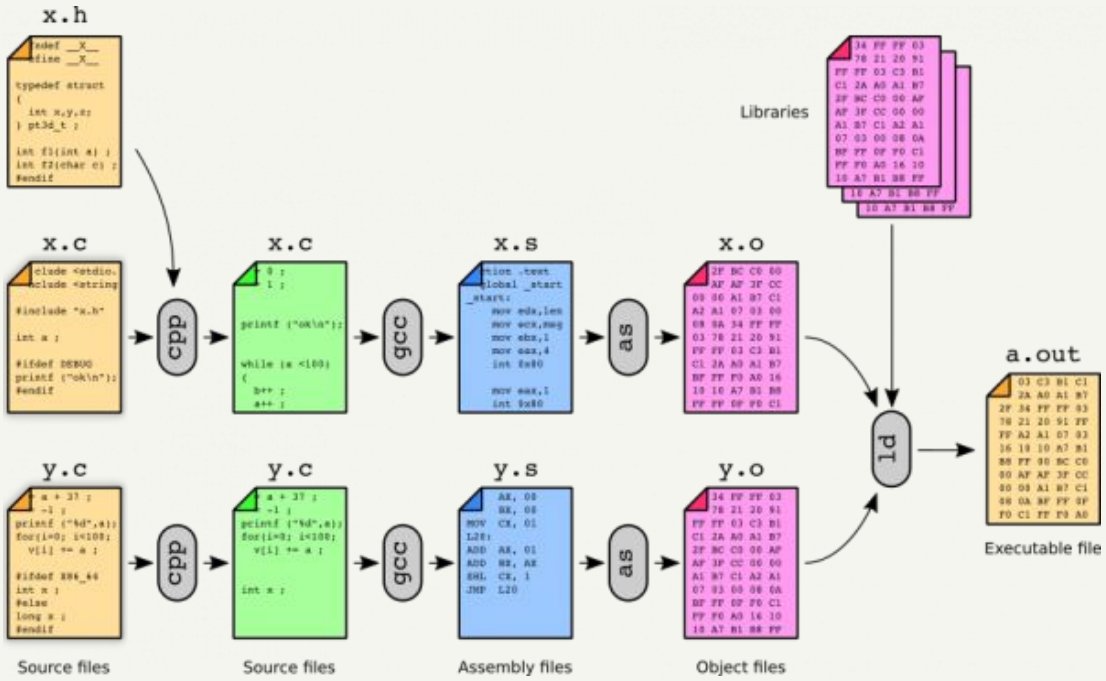
Do Fonte ao Executável

Escrever o **código fonte é apenas o processo inicial** para se alcançar um programa capaz de ser executado em um computador.

Existem muitas etapas a serem cumpridas entre a geração do fonte até a obtenção de um executável. As principais etapas são:

- Preprocessamento
- Compilação
- Montagem
- Ligação

Do Fonte ao Executável



O Preprocessador

Em resumo, o preprocessador é responsável pelo tratamento de diretivas (sim, aquelas que começam com #) incluídas pelo programador em um código fonte.

Exemplos dessas diretivas?

#include
#define
#if
#elif

O Preprocessador

Na prática, o preprocessador nada mais é do que uma **ferramenta para substituição de texto** em código fonte.

Esta ferramenta é invocada de forma automática pelo compilador C no início de sua execução, consumindo a saída do mesmo para o processo de compilação do programa.

```
gcc -E arquivo.c
```

O Preprocessador

De maneira bem geral, podemos resumir as atividades de um preprocessador em:

1. Inclusão de arquivos
2. Definição e uso de constantes
3. Tratamento de constantes predefinidas
4. Tratamento de compilação condicional
5. Definição de avisos e erros
6. Definição e uso de macros

Inclusão de Arquivos

A inclusão (**diretivas #include**) permite trazer a um arquivo de código fonte trechos de código providos em arquivos externos.

Ou seja, durante o pré-processamento, os dados contidos no arquivo indicado em uma diretiva de inclusão são copiados para a posição do código fonte cuja diretiva foi inserida.

Inclusão de Arquivos

escreva.h

```
#ifndef __ESCREVA__
#define __ESCREVA__
void escreva (char *msg);
#endif
```

escreva.c

```
#include <stdio.h>
void escreva (char *msg)
{
  printf ("%s", msg);
}
```

main.c

```
#include "escreva.h"
int main ()
{
 escreva ("Hello, world!\n") ;
 return (0) ;
}
```

Exemplo!

Vamos considerar a inclusão de “escreva.h” em “main.c”

Inclusão de Arquivos

main.c
(antes do pré processamento)

```
#include "escreva.h"
int main ()
{
 escreva ("Hello, world!\n") ;
 return (0) ;
}
```

main.c
(após o pré processamento)

```
void escreva (char *msg) ;
int main ()
{
 escreva ("Hello, world!\n") ;
 return (0) ;
}
```

Exemplo!

Vamos considerar a inclusão de “escreva.h” em “main.c”

Inclusão de Arquivos

Cabe ressaltar que existem duas notações de inclusão de arquivos, cada uma com suas peculiaridades:

- #include <...>: o arquivo indicado será buscado nos diretórios padrão do compilador, geralmente /usr/include/* nos sistemas Unix.
- #include "...": o arquivo indicado será buscado primeiro no diretório corrente (onde está o arquivo que está sendo compilado), e depois nos diretórios padrão do compilador.

Definição e Uso de Constantes

A **diretiva #define** é frequentemente utilizada para indicar uma definição de constante ao preprocessador.

De maneira bastante imediata, o preprocessador substitui todas as ocorrências de uma dada constante pelo seu respectivo valor.

Definição e Uso de Constantes

main.c
(antes do pré processamento)

```
#define VETSIZE 128
int main ()
{
 char string[VETSIZE];
 ...
 return (0) ;
}
```

main.c
(após o pré processamento)

```
int main ()
{
 char string[128];
 ...
 return (0) ;
}
```

Exemplo!

Vamos considerar a definição da constante chamada VETSIZE

Tratamento de Constantes Predefinidas

O tratamento de constantes predefinidas acontece exatamente igual ao de constantes definidas pelo programador; a única diferença é que **o valor das primeiras é dado pelo sistema**. Alguns exemplos são:

- __DATE__: data atual (formato “MMM DD YYYY”)
- __TIME__: horário atual (formato “HH:MM:SS”)
- __FILE__: nome do arquivo corrente
- __LINE__: número da linha corrente do código-fonte
- __func__: nome da função corrente

Compilação Condicional

Uma constante sem valor específico atua como uma *flag*, sendo verdadeira quando definida, e falsa quando não definida.

Note que uma constante pode ser definida em dois momentos:

- No código-fonte, utilizando a diretiva #define
- No momento de executar o compilador, utilizando a flag -D seguida (sem espaço) do nome da constante

Compilação Condicional

Então, podemos utilizar algumas diretivas para testar a definição ou não de uma constante:

- #ifdef CONSTANTE: verifica se a constante já está definida
- #ifndef CONSTANTE: verifica se a constante não está definida

Essas diretivas criam blocos condicionais de execução os quais tem seu final marcado pela diretiva #endif.

Compilação Condicional

main.c
(antes do pré processamento)

```
#include <stdio.h>
int main (){
 #ifdef PRINT
 printf ("A constante PRINT foi definida!") ;
 #endif
 return (0);
}
```

main.c
(após pré processamento;
PRINT não definida)

```
#include <stdio.h>
int main (){
 return (0);
}
```

main.c
(após pré processamento;
PRINT definida)

```
#include <stdio.h>
int main (){
 printf ("A constante PRINT foi definida!") ;
 return (0);
}
```

Exemplo!

Vamos considerar a inclusão de “escreva.h” em “main.c”

Compilação Condicional

A estratégia de compilação condicional é muito utilizada também para a criação de guardas em arquivos de código fonte. Guardas servem principalmente para:

- Evitar reinclusões de arquivos
- Evitar redefinição de variáveis

O modo de uso das diretivas condicionais é dado exatamente como apresentado anteriormente.

Compilação Condicional

main.c
(guardas de inclusão)

```
#ifndef STDIO
#include <stdio.h>
#define STDIO
#endif
int main (){
 printf ("Um programa qualquer!") ;
 return (0);
}
```

main.c
(guardas de constante)

```
#include <stdio.h>
#ifndef CONSTANTE
#define CONSTANTE 10
#endif
int main (){
 printf ("Um programa qualquer!") ;
 return (0);
}
```

Exemplo!

Da esquerda para a direita, o primeiro caso apresenta guardas de inclusão, já o segundo, guardas de constante.

Definição de Avisos e Erros

Outra utilidade do preprocessador é definir avisos e erros a serem exibidos ao programador dada determinadas condições definidas em código. As diretivas para definir esses recursos são:

- #warning: define um aviso; mesmo que este ocorra, o processo de geração do executável continua
- #error: define um erro; o processo de geração do executável é abortado

Definição de Avisos e Erros

main.c
(aviso)

```
#include <stdio.h>
#ifndef CONSTANTE
#warning “CONSTANTE definida como 10”
#define CONSTANTE 10
#endif
int main (){
 printf ("Um programa qualquer!") ;
 return (0);
}
```

main.c
(erro)

```
#include <stdio.h>
#ifndef CONSTANTE
#error “A CONSTANTE precisa ser
definida”
#endif
int main (){
 printf ("Um programa qualquer!") ;
 return (0);
}
```

Exemplo!

Da esquerda para a direita, o primeiro caso apresenta a definição de um aviso, já o segundo, a definição de um erro.

Definição e Uso de Macros

Por fim, o preprocessador pode ser utilizado para **definir macros que consistem em funções simples com parâmetros**.

A cada utilização do macro, o preprocessador substitui sua ocorrência por toda a função definida.

A diretiva #define é utilizada para definir um macro, sendo seguida de um nome com parâmetros e da operação desejada.

Definição e Uso de Macros

main.c
(definição de um macro)

```
#include <stdio.h>
#define SQUARE(v) v*v
int main (){
 printf ("2 ao quadrado: %d",
SQUARE(2)) ;
 return (0);
}
```

main.c
(macro após o pré processamento)

```
int main (){
 printf ("2 ao quadrado: %d", 2*2) ;
 return (0);
}
```

Exemplo!

Da esquerda para a direita, o primeiro caso apresenta a definição de um macro, já o segundo, o código gerado após o preprocessamento do macro.

Definição e Uso de Macros

Porém, macros são bastante suscetíveis a erros:
É NECESSÁRIO TER CUIDADO!

Esses erros tipicamente ocorrem entre a definição de parâmetros e a passagem de argumentos, gerando resultados indesejados.

O uso de parênteses na utilização dos parâmetros nas expressões pode ser uma alternativa viável para evitar tais erros.

Definição e Uso de Macros

main.c
(utilização de um macro - ERRO)

```
#include <stdio.h>
#define SQUARE(v) v*v
int main (){
 printf ("Expressão: %d",
SQUARE(2+1)) ;
 return (0);
}
```

main.c
(utilização de um macro - CORRETO)

```
#include <stdio.h>
#define SQUARE(v) (v)*(v)
int main (){
 printf ("Expressão: %d",
SQUARE(2+1)) ;
 return (0);
}
```

Exemplo!

Da esquerda para a direita, o primeiro caso apresenta um potencial erro causado pelo uso de um macro, já o segundo, a utilização correta do macro.

Exercício #10

Considere um código fonte qualquer e desenvolva três níveis de mensagens de sistema para ele através da definição de uma constante chamada DETAIL (utilize o preprocessador para isso):

- No DETAIL 0, nenhuma mensagem é mostrada
- No DETAIL 1, mensagens com o nome da função sendo executada são exibidas
- No DETAIL 2, além das mensagens do DETAIL 1, exiba o tempo de execução de cada função

Obrigado!

Vinícius Fülber Garcia
inf.ufpr.br/vinicius
vinicius@inf.ufpr.br