[MAC0211] Laboratório de Programação I
Aula 17
Processadores de Macros
M4

Alair Pereira do Lago

DCC–IME–USP

7 de maio de 2015

Pré-processamento

Pré-processador

- É um programa que processa seus dados de entrada para produzir dados de saída que serão usados como entrada para outros programas como, por exemplo, compiladores
- Fazem diferentes tipos de processamento: uns são capazes de realizar substituições textuais simples e expansões de macros, enquanto outros têm o poder de uma linguagem de programação
- Exemplo de pré-processador clássico: o que processa o código-fonte de um programa, preparando-o para a compilação

Pré-processamento

Pré-processador léxico

- É o tipo de pré-processador de mais baixo nível (que é executado antes de qualquer outro processamento no código)
- Faz uma **análise léxica** do código-fonte
- Ações comuns:
  - Substituição de cadeias de caracteres de acordo com regras (= macros) definidas por usuários
  - Inclusão textual de outros arquivos
  - Compilação ou inclusão condicional
- Exemplo mais conhecido: o pré-processador do C (CPP)

Macro (= Macroinstrução)

- É um conjunto de instruções que podem ser reutilizadas em diferentes contextos
- Pode ser parametrizável

Processamento de macros aparece em programas variados:

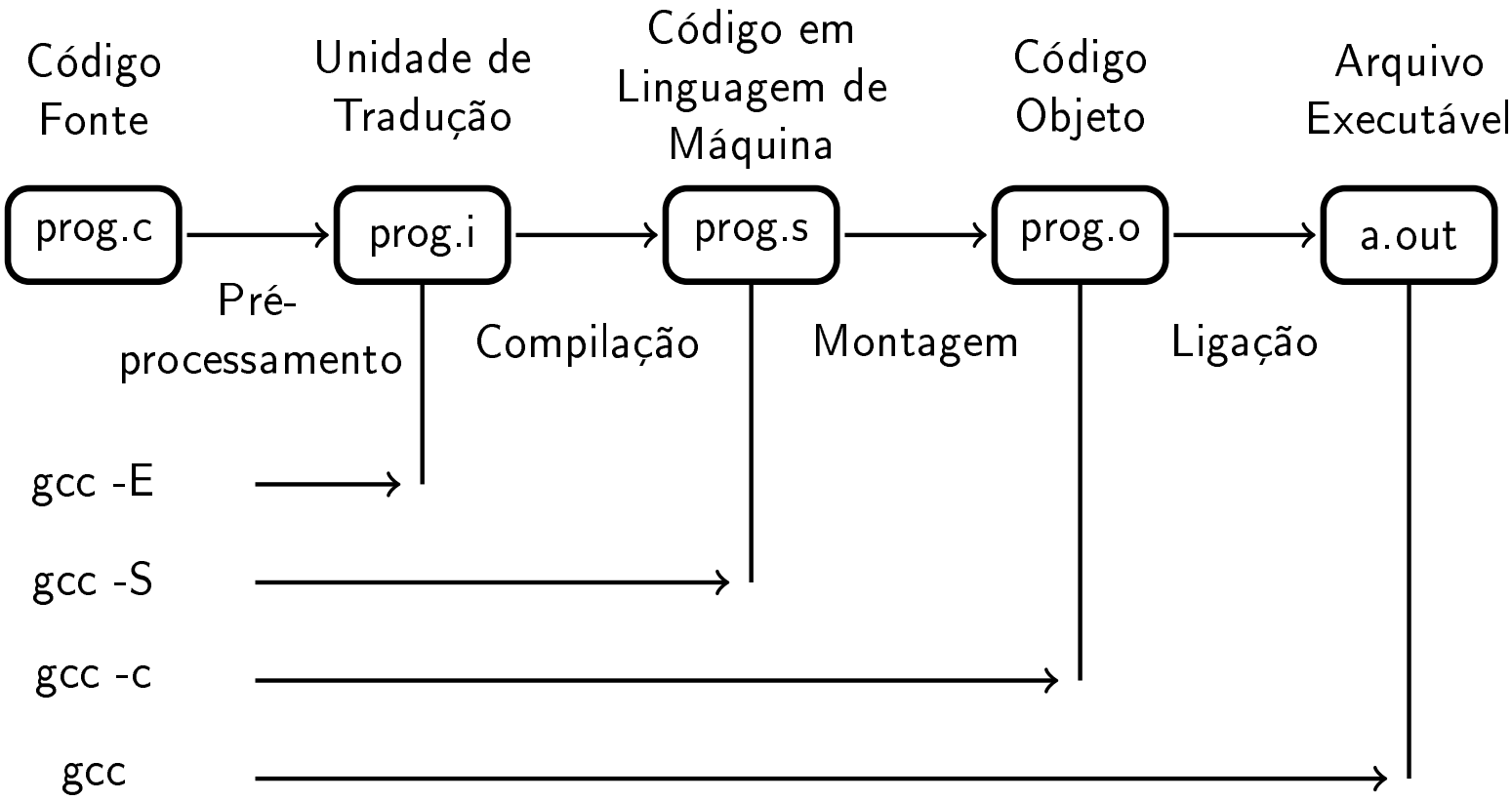
- editores de texto
- planilhas eletrônicas
- interpretadores e compiladores

[Relembrando] Do código fonte de alto nível ao executável

GCC – Gnu C Compiler

- A compilação feita pelo GCC tem vários estágios:

```
alair@linux$ gcc prog.c
```



O pré-processador do C (CPP)

- É um processador de macros (porque permite que macros sejam definidas nos programas)
- Usado em programas C, C++ e Objective-C antes que eles sejam compilados
- Diretivas:
  - `#include` – copia o conteúdo de um arquivo no arquivo corrente, no ponto em que a diretiva aparece (objetivo: reúso de código)
  - `#define` – define macros
  - `#ifndef`, `#endif` – fazem compilação condicional de código

Macros como funções

Em C, macros podem ser usadas como substitutas para funções que implementam uma computação curta e que será executada frequentemente

- Isso é uma tendência dos programadores mais velhos
- Motivo: a macro evita o custo adicional de uma chamada de função
- Nos computadores e compiladores modernos, as desvantagens do uso de macros como funções superam os seus benefícios

Macros como funções

Problemas

- Preocupação com a substituição textual

Ex.: macro que calcula o quadrado de um número (versão 1)

```
#define QUADRADO(x) x * x
...

c = QUADRADO(a+b);
...
```

Problema: a expansão da macro resultará a expressão $a + b * a + b$, que não é o quadrado de $a + b$

Macros como funções

Problemas

- Preocupação com a substituição textual

Ex.: macro que calcula o quadrado de um número (versão 2)

```
#define QUADRADO(x) (x) * (x)
...

c = QUADRADO(a+b);        // funciona corretamente!
c = 1 / QUADRADO(c);      // xiii...
...
```

Problema: a expansão da macro resultará a expressão $1/(c) * (c)$,
enquanto que o esperado é $1/((c) * (c))$

Macros como funções

Problemas

- Preocupação com a substituição textual

Ex.: macro que calcula o quadrado de um número (versão 3)

```
#define QUADRADO(x) ((x) * (x))
...

c = QUADRADO(a+b);      // funciona corretamente!
c = 1 / QUADRADO(c);    // funciona corretamente!
...
```

Para este exemplo, a parentização feita resolveu o problema. Mas nem sempre é possível evitar todos os erros de avaliação de uma expressão que podem ser causados por macros.

Macros como funções

Problemas

- Parâmetros que aparecem mais de uma vez na definição da macro são avaliados mais de uma vez
- Possível resultado 1: erros sutis

Ex.: macro que testa se um caractere é uma letra maiúscula

```
#define MAIUSCULA(c) ((c) >= 'A' && (c) <= 'Z')
...

while ( MAIUSCULA(c = getchar()) )
...
```

Problema: o parâmetro *c* aparece 2 vezes no corpo do macro. Na chamada feita no *while*, a macro executará duas vezes o *getchar* – uma em *(c = getchar()) >= 'A'* e outra em *(c = getchar()) >= 'Z'*

Macros como funções

Problemas

- Parâmetros que aparecem mais de uma vez na definição da macro são avaliados mais de uma vez
- Possível resultado 2: pior desempenho

Ex.: macro que arredonda um número para inteiro

```
#define ARREDONDA(x) ((int) (((x)+(((x)>0)?0.5:-0.5)))
...

tamanho = ARREDONDA(sqrt(dx*dx + dy*dy))
...
```

Problema: a função *sqrt* será executada o dobro de vezes da quantidade necessária

Processadores de macros genéricos

- Não estão associados a nenhuma linguagem ou software em particular
- São programas que copiam um *stream* de texto de um lugar para outro, aplicando no texto um conjunto sistemático de substituições
- Exemplos de programas desse tipo:
  - M4 – criado em 1977 por Brian Kernighan e Dennis Ritchie, criadores da linguagem C
    implementação do projeto GNU:
    `http://www.gnu.org/software/m4/m4.html`
  - gema – `http://gema.sourceforge.net/`
  - GPP – `http://en.nothingisreal.com/wiki/GPP`

M4

- Foi criado como uma ferramenta básica do sistema UNIX em 1977 e passou a ser incorporado em várias ferramentas de mais alto nível em ambiente UNIX
- O CPP (pré-processador do C) se baseia nele
- Mais recentemente, o GNU `autoconf` passou a utilizá-lo fortemente
- Outras ferramentas como `sendmail` também o utilizam em seus arquivos de configuração

M4

A entrada do M4 é um texto onde se encontram definições, texto comum e macros que devem ser expandidas.

O M4 possui funcionalidades para:

- expansão de macros
- inclusão de arquivos
- execução de comandos *shell*
- manipulações de texto variadas
- aritmética de números inteiros
- recursão

M4

Exemplo

Entrada	**Saída**
define(`foo', `Hello world.')	
define(`troca', `$2, $1')	
define(`eu', `Este sou `eu'.')	
foo	Hello world.
troca(papai,oi)	oi, papai
eu	Este sou eu.
foo # comentario foo	Hello world. # comentario foo

M4

Características

- Os caracteres ` e ' são usados para suspender uma expansão
- Comentários de linha começam com #. Um comentário para o M4 apenas delimita uma sequência de caracteres que não será candidata à expansão. Mas comentários são jogados para a saída
- Parâmetros passados para a macro são indicados por $número (por exemplo, $1, $2, $3,...)
- O número de argumentos é $#
- A lista de argumentos é $* ou $@. Em $@, os argumentos não são expandidos
- Na definição de uma macro, é preciso que o caracter (apareça colado à palavra-chave `define`, como em:
  `define(`foo', `Hello world.')`

M4

Algumas macros e comandos especiais

- *undefine(NOME)* – remove a macro
- *defn(NOME)* – expande a definição da macro NOME, assim é possível fazer:
  ```
  define(`batiza',defn(define))
  ```
  que cria uma macro chamada *batiza* que faz o mesmo que a *define*
- *pushdef(NOME,DEF)* e *popdef(NOME)* – permitem redefinir macros temporariamente
- *ifdef(NOME,STRING1[,STRING2])* – se uma macro com nome *NOME* estiver definida, expande para *STRING1*. [Opcional: Expande para *STRING2* no caso contrário]. Exemplo:
  ```
  ifdef(`foo', ``foo' ja esta definido',
          `define(`foo', `Hello world.')')
  ```

M4

Algumas macros e comandos especiais

- *ifelse(STRING 1, STRING 2, IGUAL, DIFERENTE)*

  Com mais de quatro argumentos e se a comparação for falsa, repete a comparação a partir do quarto argumento. Exemplo:
  `ifelse(A,B,sim,C,C,agora sim, nunca)`
  vira
  `agora sim`

- shift – joga o primeiro argumento fora e devolve os seguintes. Exemplo:
  ```
  define(`inverte', `ifelse($#, 0, , $#, 1, ``$1'',
  `inverte(shift($@)), `$1'')')
  ```

- *traceon(NOME)* e *traceoff(NOME)* – para acompanhar a expansão da macro NOME (usado para depuração)

- *dnl* – "consome" o resto da linha, resultando em nenhuma saída

Bibliografia e materiais recomendados

- Manual do pré-processador do C
  `http://gcc.gnu.org/onlinedocs/cpp/index.html`
- Manual do M4
  `http://www.gnu.org/software/m4/manual/`
- Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon
  `http://www.ime.usp.br/~kon/MAC211`

Cenas dos próximos capítulos...

Nas próximas aula:

- Um programa que faz uso do M4: o Autoconf
- Expressões regulares