Bibliotecas e Makefile

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados I (AE22CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

Sumário

- Biblioteca
- Makefile

Introdução

- Até o momento, desenvolvemos e reutilizamos vários TADs
- A partir de um conjunto de TADs, podemos criar pelo menos uma biblioteca
- A criação de bibliotecas possibilita o reuso de funções
- A biblioteca padrão ANSI C contém 24 cabeçalhos (TADs), tais como *stdio.h*, *stdlib.h*, *math.h*, entre outros
- Para criarmos bibliotecas, precisamos de arquivos .h e .c

Introdução

- Para a criação de bibliotecas, também podemos precisar de arquivos de outra(s) biblioteca(s), cuja extensão utilizada no *gcc* (compilador da linguagem C) é *.a* (Unix e MacOS) ou .lib (Windows)
  - Bibliotecas estáticas: código é ligado ao programa durante a sua construção

Sumário

Biblioteca

- Bibliotecas são conjuntos de funções empacotadas na forma de um arquivo
- Para usarmos tais bibliotecas, basta "chamarmos" os seus respectivos arquivos .h através do comando *#include*
- O GCC (GNU *compiler collection*) é um conjunto de ferramentas utilizadas para a compilação de programas

Biblioteca

- A compilação de um código-fonte C é executada em etapas
  - Pré-processamento
  - Compilação
  - Montagem
  - Ligação
- Ao final da compilação, casa não ocorra erros, um arquivo executável é gerado
  - No Windows, por exemplo, é gerado um arquivo .exe

Biblioteca

- Pré-processamento
  - Pode ser realizado separadamente através do comando *cpp* (C *preprocessor*) no terminal
  - Etapa inicial da compilação que trata as diretrizes *#include*, *#define*, *#ifdef*, *#ifndef*, etc
  - Veja o que ocorre ao executarmos o comando "cpp teste.c", onde o arquivo .c contém apenas o conteúdo abaixo

```
int main{
  printf("Oi!  Eu sou o Goku!\n!");
  return 0;
}
```

  - Lógico que o arquivo fonte acima precisa do comando *#include <stdio.h>*
    - Inclua instrução acima no código fonte execute novamente o comando "cpp teste.c"

Biblioteca

- Algumas diretivas de compilação utilizadas pelo pré-processador
  - *#include*: inclusão de um arquivo específico
    - *#include <nome_arquivo>*
    - *#include "nome_arquivo"*
  - *#define*: além de ser utilizado para definirmos uma constante, também podemos utilizar para determinarmos um símbolo (ou um macro) para ser utilizado durante o pré-processamento e a compilação
    - *#define nome_simbolo*
    - *#define nome_constante valor*
    - *#define nome_macro(parâmetro) expressão_substituta*

Biblioteca

- Algumas diretivas de compilação utilizadas pelo pré-processador
  - *#undef*: faz que uma macro seja esquecida, ou seja, a partir do ponto em que o compilador leu esse comando, passa a não conhecer mais o respectivo macro
    - *#undef nome_ macro*
  - *#ifdef*: operação condicional similar ao comando *if*

```
#ifdef nome_simbolo
   ...
   Implementações
   ...
#endif
```

Biblioteca

- Algumas diretivas de compilação utilizadas pelo pré-processador
  - *#ifndef* : operação condicional similar ao comando *if*, mas para o código em que o símbolo não foi definido
    - O símbolo não foi definido através da diretiva *#define*

```
#ifndef nome_simbolo
  #define nome_simbolo
#endif
```

  - Exemplos de outras diretivas: *#if*, *#else* e *#elif* (*else if*)

Biblioteca

- Exemplo

```
#include <stdio.h>
#define max(A, B) ((A > B) ? (A) : (B))

#ifdef MENSAGEM
void mostra(){
  printf("Me chamo Jefferson!\n");
}
#else
void mostra(){
  printf("Eu nao!\n");
}
#endif

int main(){
  printf("Oi!  Eu sou o Gokku!  E voce?\n");
  mostra();

  return 0;
}
```

Biblioteca

- Compilação
  - Pode ser realizado através do comando *gcc* no terminal
  - Análise sintática e semântica
  - Transforma as instruções C em instruções assembly, tipicamente gerando arquivo .s
    - Experimente utilizar o comando *gcc teste.c -S* ou *gcc -S teste.c*
    - Ao executar o comando sugerido acima, o arquivo.s será gerado

Biblioteca

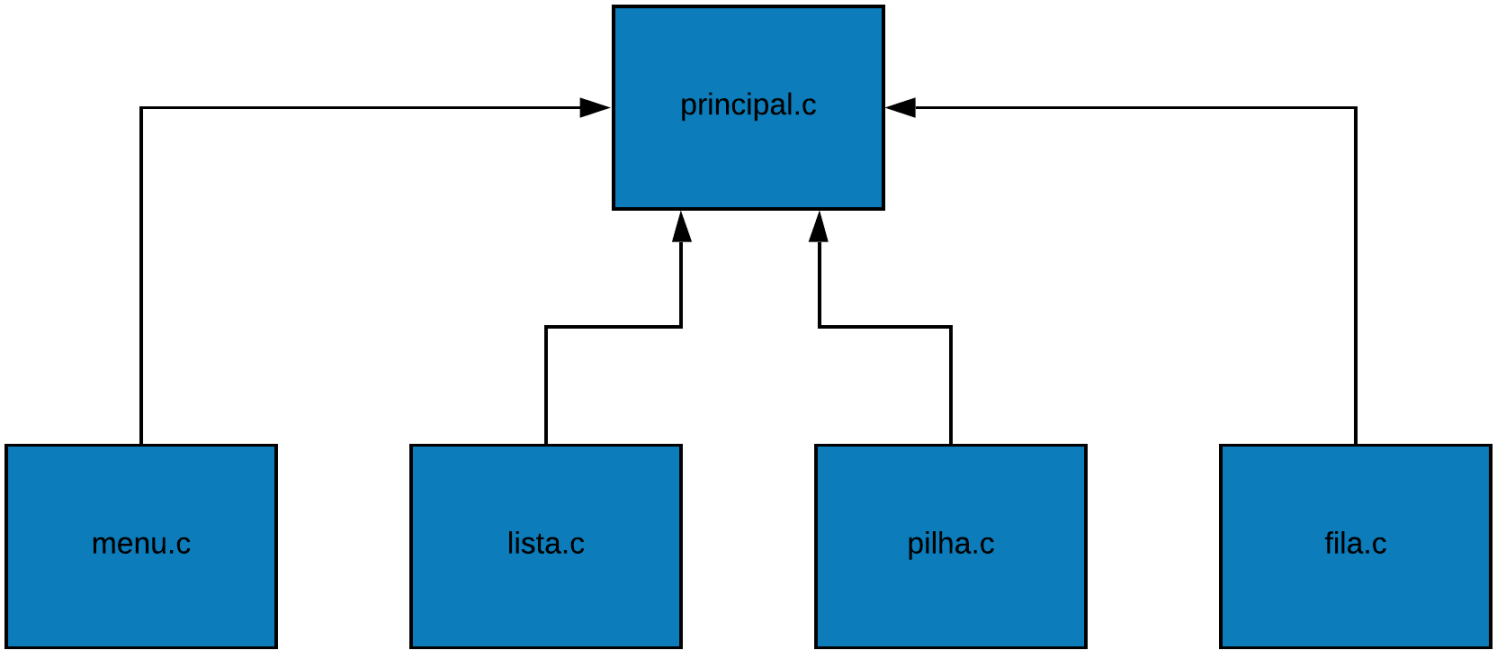
- Montagem
  - Também realizada através do *gcc*
  - Transforma o arquivo com instruções assembly em um arquivo compilado .o, conhecido como arquivo "objeto" (ou módulo), que são os arquivos arquivos binários
  - Experimente utilizar o comando *gcc -c teste.c*
  - Nessa etapa, também pode ser utilizado o arquivo .s para fazer a montagem: *gcc -c teste.s*

Biblioteca

- Ligação
  - Outra etapa realizada por meio do *gcc*
  - Os módulos .o gerados durante a montagem são agrupados para gerar um arquivo executável
  - Nessa etapa, as bibliotecas são adicionadas
  - Experimente utilizar o comando *gcc teste.c -o teste* ou *gcc teste.o -o teste*
  - Para o uso do comando *gcc* nessa etapa, o nome do arquivo de saída deve ser fornecido, conforme a instrução acima

Biblioteca

- Trabalhando com múltiplos arquivos .c



Biblioteca

- Principal.c

```
#include "lista.h"
#include "fila.h"
#include "pilha.h"
#include "menu.h"

int main(){
  Lista *l;
  Pilha *p;
  Fila *f;

  l = criar_lista();
  p = criar_pilha();
  f = criar_fila();

  exibir_menu();

  return 0;
}
```

Biblioteca

- Menu.h

```
#ifndef _MENU_
#define _MENU_

void exibir_menu();

#endif
```

- Menu.c

```
#include "menu.h"
#include <stdio.h>

void exibir_menu(){
  printf("Menu\n");
}
```

Biblioteca

- Os demais arquivos (TAD de listas, pilhas e filas) estão no repositório da disciplina no GitHub

- Compilação de arquivos .c e geração de arquivos .o:

```
gcc -c lista.c
gcc -c pilha.c
gcc -c fila.c
gcc -c menu.c
gcc -c principal.c

// Junção de todos os arquivos objetos em um executável
gcc -o saida menu.o principal.o lista.o fila.o pilha.o
```

Biblioteca

- Criando bibliotecas

```
// Criando uma biblioteca com o nome biblioteca.a.  Caso
estiver usando Windows, em vez de utilizar a extensão .a,
use .lib
ar rs biblioteca.a lista.o pilha.o fila.o

// mostra quais arquivos objeto existem dentro de uma
// biblioteca
ar -t biblioteca.a

// compila o programa principal, junta com a biblioteca e
// gera o executável
gcc principal.c biblioteca.a menu.o -o saida

// A linha acima pode acarretar em erro porque as
// bibliotecas são procuradas em um diretório padrão
// Use:  gcc principal.c -L./ -biblioteca menu.o -o saida
```

Sumário

Makefile

- O *makefile* é um arquivo texto com instruções de como um conjunto de arquivos fonte deve ser compilado

- O arquivo *makefile* é lido por um programa denominado make, que deve ser digitado na linha de comando
  - Caso o arquivo *makefile* exista no diretório dos arquivos fonte, basta digitar *make* e pressionar a tecla *enter*

  - Perceba que não é passado qual arquivo deve ser lido, pois apenas o arquivo com o nome *makefile* será lido

- Um *makefile* é utilizado para compilação, ligação e montagem

Makefile

- No *makefile* também pode ser incluídas instruções para limpeza de arquivos temporários, execução de comandos, entre outros

- Vantagens
  - Evita a compilação de arquivos desnecessários
  - Automatiza tarefas, como limpeza de arquivos temporários
  - Por mais que seja mais aplicado para compilação de arquivos, também pode ser utilizado como uma linguagem geral de *script*

Makefile

- O *makefile* consiste em regras definidas na seguinte sintaxe:

  alvo : pré-requisitosreceita

  - Alvo é o nome da ação que será executada ou o nome do arquivo que deve ser produzido

  - Pré-requisitos são os arquivos utilizados como entrada do *nome_da_ação*

  - Receita é a ação realizada pelo comando *make*

  - A receita pode ter mais de um comando

  - Também, os comandos podem ser atribuídos a uma variável, por exemplo: *COMPILADOR = gcc*

  - Para utilizar a variável que contém o nome do comando, basta utilizar *$(NOME_VAR)*

Makefile

- Exemplo de *makefile*:

```
COMPILADOR=gcc
APAGA=rm -f
saida:  principal.o menu.o lista.o pilha.o fila.o
        $(COMPILADOR) -o saida principal.o menu.o lista.o pilha.o fila.o

principal.o:  principal.c menu.h lista.h pilha.h fila.h
        $(COMPILADOR) -c principal.c

menu.o:  menu.h menu.c
        $(COMPILADOR) -c menu.c

lista.o:  lista.h lista.c
        $(COMPILADOR) -c lista.c

pilha.o:  pilha.h pilha.c
        $(COMPILADOR) -c pilha.c

fila.o:  fila.h fila.c
        $(COMPILADOR) -c fila.c

clean:
        $(APAGA) saida *.o
```

- Após a definir e salvar o *makefile*, basta executar o comando *make* no diretório em que o arquivo encontra-se salvo

Referências I

de la Rocha, F. R.
Bibliotecas.
Slides. Engenharia de Computação. Dainf/UTFPR/Pato Branco, 2011.

Geeks for geeks.
Gcc command in Linux with examples.
https://www.geeksforgeeks.org/gcc-command-in-linux-with-examples/.

Ponti, M. P.
Uma breve introdução à criação de bibliotecas e makefiles em C/C++.
http://wiki.icmc.usp.br/images/0/0a/ApostilaMakefiles2011.pdf.

Referências II

Wikibooks.
Programar em C/Pré-processador.
https://pt.wikibooks.org/wiki/Programar_em_C/Pr%C3%A9-processador.

Wikibooks.
Programar em C/Makefiles.
https://pt.wikibooks.org/wiki/Programar_em_C/Makefiles.