Engenharia de Computação

Fundamentos de Programação

Aula 23 – Bibliotecas e Macros

Prof. Muriel de Souza Godoi
muriel@utfpr.edu.br

Criação de Cabeçalhos

- Quando o compilador vê uma chamada de função que ele não conhece
  - Ele anota e espera que a definição venha depois
  - Mas ele não sabe o tipo de retorno da função
    - Supõe que é um inteiro (int)
  - Quando ele acha a função e ela retorna float
    - Conflito de tipos: inteiro ou real?
  - Solução
    - Declarar todas as funções antes de chamá-las
    - Mas, nem sempre é possível.
  - Como resolver?
    - Separando a declaração da implementação!

Criação de Cabeçalhos

- Coloque as assinaturas de todas as funções no início do seu código
  - Implementações das funções podem vir em qualquer ordem

```
float soma( float valor1, float valor1);
float quadrado( float valor);
bool ePar( int numero);
```

- Podemos colocar esses cabeçalhos em um arquivo separado com a extensão .h (header)
  - Depois incluímos o arquivo no código original

```
#include "main.h"
```

Bibliotecas - Problema

- **Nossos programas estão crescendo….**
  - Até agora nossos programas são relativamente pequenos
  - Mas, com o tempo percebemos que um conjunto de funções pode ser **reaproveitado** em diversos programas
  - Como reaproveitar essas funções sem **duplicar o código**?
  - Como manter um programa real com **milhares** de linhas de código?
- **Solução?**
  - Dividir o código fonte em diversos arquivos menores e gerenciáveis
  - Depois compilaremos tudo em um **grande programa**!

Biblioteca - Exemplo

- Arquivo: **cripto.h** (header)

```
void criptografa(char* msg);
```

- Arquivo: **cripto.c**

```
#include "cripto.h"

void criptografa(char* msg){
  while(*msg != '\0'){
    *msg = (*msg) + 2;
    msg++;
  }// while
}//criptografa
```

Biblioteca - Exemplo

- Arquivo **main.c**

```c
#include "cripto.h"

int main(){

  char mensagem[35] = "Minha mensagem secreta";

  printf("Mensagem original: %s \n", mensagem);
  criptografa(mensagem);
  printf("Mensagem criptografada: %s \n", mensagem);

  return 0;
}//main
```

Biblioteca

- **Como compilar?**
  - Basta incluir cada um dos arquivos de código fonte do comando de compilação

```
gcc cripto.c main.c -o main
```

  - Ou incluí-los todos de uma vez usando ***.c**:

```
gcc *.c -o main
```

  - **Saída gerada**

```
$./main
Mensagem original: Minha mensagem secreta
Mensagem criptografada: Okpjc"ogpucigo"ugetgvc
```

Diretivas de Compilação

- Instruções incluídas no código mas que **não são compiladas**

Lista de diretivas de compilação			
#include	#define	#undef	#ifdef
#ifndef	#if	#endif	#else
#elif	#line	#error	#pragma

  - Faz alterações no código fonte antes de enviá-lo ao compilador
- Exemplo: **#include**
  - Diz ao compilador para tratar o conteúdo do arquivo incluído como se tivesse sido digitado no arquivo onde está o include.
  - Utilizado para declarar as bibliotecas que serão utilizadas no no programa

Diretivas de Compilação

- **Macros:** `#define` **e** `#undef`
  - Já utilizamos para declarar constantes no começo da disciplina
  - Procura todas as ocorrências de uma expressão e substitui por outra antes da compilação
  - Permite três sintaxes:

```
//Apenas cria o simbolo
#define nome

//Cria o simbolo e define o valor de substituição
#define nome_da_constante valor_da_constante

//Cria uma função macro
#define nome_da_funcao(parametros) funcao
```

Diretivas de Compilação

- **Funções macro com #define**

```c
#define maior(x,y)x>y?x:y //função macro

int main(){
  int a = 5, b = 8, valor;

  valor = maior(a,b);
  printf("O maior valor é %d\n",valor);

  //É equivalente a
  valor = a>b?a:b;
  printf("O maior valor é %d\n",valor);

  return 0;
}//main
```

Diretivas Condicionais

- **Macro:** `#ifdef` e `#ifndef`
  - Permite verificar se uma macro foi definida anteriormente ou não

```c
#define macro

int main(){

  #ifdef macro
    printf("A macro já foi definida\n");
  #else
    printf("A macro não foi definida\n");
  #endif

  return 0;
}//main
```

Include Guard

- Quando trabalhamos com várias bibliotecas, e includes, corremos o risco de incluir duas vezes a mesma biblioteca
  - Isso gera um **erro!**
- Para evitar isso utilizamos a técnica de **Include Guard** que testa se a inclusão já foi realizada

```
#ifndef BIBLIOTECA_H
#define BIBLIOTECA_H

float soma( float valor1, float valor1);
float quadrado( float valor);
bool ePar( int numero);

#endif /* BIBLIOTECA_H */
```

Compilando Projetos Maiores

- Os comandos e parâmetros necessários para compilar um projeto **cresce conforme o tamanho** do projeto
- Não é necessário compilar todos os arquivos fontes, todas vezes.
  - Podemos compilar apenas os que foram modificados e ganhar tempo de compilação
- É necessária uma ferramenta para **automatizar** esse processo e facilitar o trabalho
- Solução
  - **Make**

Compilando Projetos Maiores

- Criar um arquivo chamado “`makefile`”
  - Contém conjunto de regras com alvos serem compilados
- **Sintaxe geral:**

```
alvo: dependencias
   receita
```

- Cada regra contém
  - **Alvo**: Arquivo a ser gerado
  - **Dependências**: Quais arquivos que irão gerar o **alvo**
  - **Receita**: Conjunto de comandos para transformar as **dependências** no **alvo**

Cada linha de receita **precisa começar** com um caractere de **tabulação**,
Se iniciar com espaços não ira funcionar

Makefile

- Exemplo de makefile para o projeto de criptografia

```
# Comentário do makefile de exemplo

main: main.o
    @echo "Gerando arquivo executável"
    gcc  *.o -o main

main.o: main.c
    @echo "Compilando e gerando os objetos"
    gcc -c *.c

clean:
    @echo "Apagando objetos e executáveis antigos..."
    rm -f *.o main

run:
    ./main
```

Usando o Make file

- Para compilar seu projeto

```
make
```

- Para executar seu projeto

```
make run
```

- Para limpar as compilações anteriores

```
make clean
```