Programação de Microcontroladores

Aula 04

Evandro J.R. Silva[1]

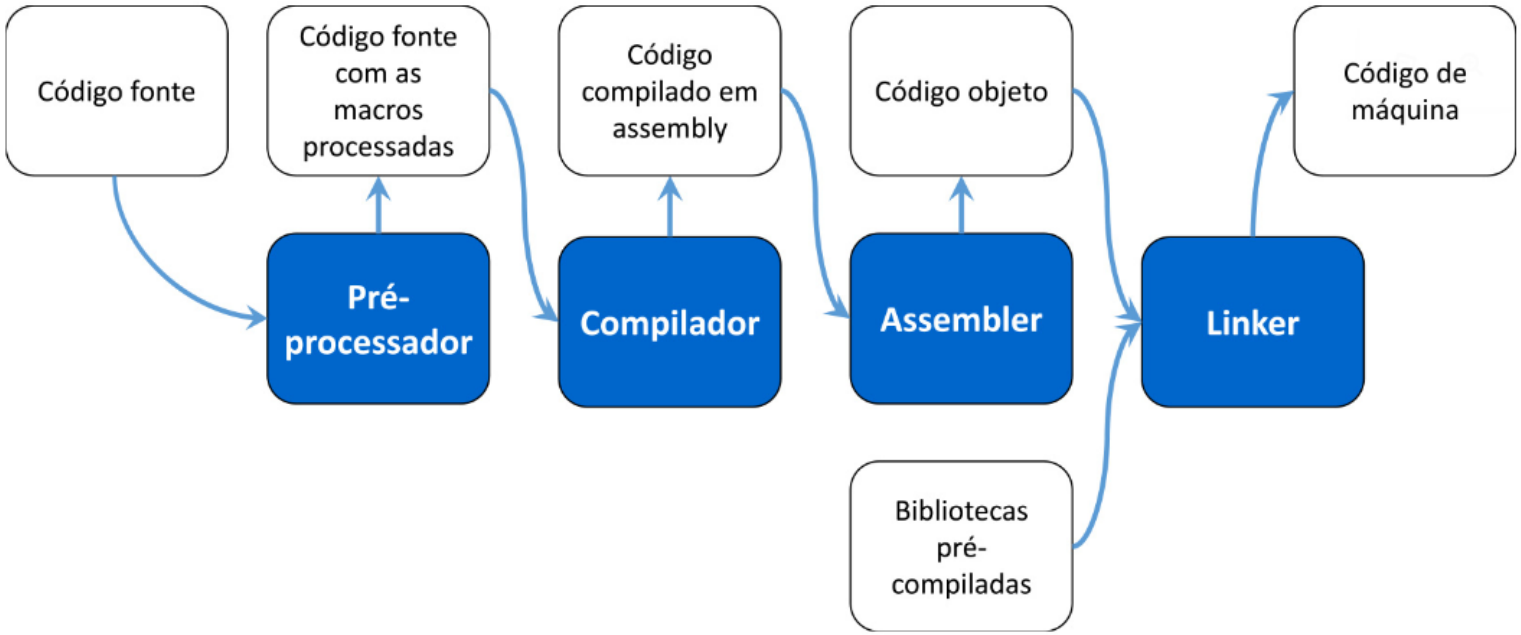
[1] Bacharelado em Ciência da Computação
Estácio Teresina

Sumário

Introdução

Introdução

- Vejamos alguns conceitos pouco explorados da linguagem C nos cursos de graduação.
- Porém, antes dos conceitos, vamos entender bem o processo de compilação



Diretivas de pré-compilação

Diretivas de pré-compilação

- São instruções inseridas nos arquivos de código que visam alterar o programa antes do processo de compilação.
- Começam com o símbolo **#** (pt-br: jogo da velha, en: hash).

Diretivas de pré-compilação
Inclusão de arquivos

Inclusão de arquivos

- A diretiva `#include` é a responsável por permitir a utilização de funções que foram implementadas em outros arquivos. Ex.:

```
#include <nome_arquivo.h> // arquivo do path do compilador
#include "nome_arquivo.h" // arquivo no mesmo diretório
```

- O agrupamento de funções em um arquivo para uso posterior é chamado de **biblioteca**.

Inclusão de arquivos

- O agrupamento de funções em um arquivo para uso posterior é chamado de **biblioteca**.
- Um arquivo com extensão `.h` (arquivo *header*) geralmente possui apenas os protótipos de funções ou definição de tipos de variável disponibilizados pela biblioteca.

Inclusão de arquivos

- O agrupamento de funções em um arquivo para uso posterior é chamado de **biblioteca**.
- Um arquivo com extensão `.h` (arquivo *header*) geralmente possui apenas os protótipos de funções ou definição de tipos de variável disponibilizados pela biblioteca.
- Se o código fonte for distribuído junto com uma biblioteca ele possui, em geral, a extensão `.c`. Nesse caso, o programador não precisa distribuir o código fonte, podento optar por entregar apenas o **código objeto** (extesão `.o`).

Diretivas de pré-compilação
Definição e expansão de macros

Definição e expansão de macros

- Outra funcionalidade do processo de pré-compilação é poder efetuar a troca de símbolos, valores ou textos.
- Diretiva: `#define nome <lista de comandos>`. Ex.:

```
#define SEGUNDOS_MINUTOS 60
#define MINUTOS_HORAS 60
#define HORAS_DIAS 24

int segundos_para_dias (int segundos) {
    return (SEGUNDOS_MINUTOS * MINUTOS_HORAS * HORAS_DIAS) /
     segundos;
}
```

Definição e expansão de macros

- Outra funcionalidade do processo de pré-compilação é poder efetuar a troca de símbolos, valores ou textos.
- Diretiva: `#define nome <lista de comandos>`. Ex.:

```
#define SEGUNDOS_MINUTOS 60
#define MINUTOS_HORAS 60
#define HORAS_DIAS 24

int segundos_para_dias (int segundos) {
    return (SEGUNDOS_MINUTOS * MINUTOS_HORAS * HORAS_DIAS) /
     segundos;
}
```

- Após a substituição dos valores, a função estará escrita como:

```
int segundos_para_dias (int segundos) {
    return (60 * 60 * 24) / segundos;
}
```

Definição e expansão de macros

- Outra funcionalidade do processo de pré-compilação é poder efetuar a troca de símbolos, valores ou textos.
- Diretiva: `#define nome <lista de comandos>`. Ex.:

```
#define SEGUNDOS_MINUTOS 60
#define MINUTOS_HORAS 60
#define HORAS_DIAS 24

int segundos_para_dias (int segundos) {
    return (SEGUNDOS_MINUTOS * MINUTOS_HORAS * HORAS_DIAS) /
     segundos;
}
```

- Após a substituição dos valores, a função estará escrita como:

```
int segundos_para_dias (int segundos) {
    return (60 * 60 * 24) / segundos;
}
```

- Como os valores são constantes, o compilador poderá otimizar o código da seguinte forma:

```
int segundos_para_dias (int segundos) {
    return 86400 / segundos;
}
```

Definição e expansão de macros

- Outra forma de utilizar a macro:
  `#define <nome>(lista de parâmetros) <lista de comandos>`

Definição e expansão de macros

- Outra forma de utilizar a macro:
  `#define <nome>(lista de parâmetros) <lista de comandos>`
- Ex.:

```
#define Media(a,b,c) ((a+b+c)/3)

void main() {
    int x=10, y=20, z=30, resultado;
    resultado = Media(x,y,z);
}
```

Definição e expansão de macros

- A diretiva `#define` é comumente utilizada também para criar um conjunto de referências que serão utilizadas ao longo do programa, incluindo definições de hardware do dispositivo (ex.: endereços dos periféricos ou definições padrão).
- Caso você queira redefinir uma macro já existente, você pode utilizar a diretiva `#undef`. Assim a definição será removida, possibilitando ao programador definir outra coisa.

Diretivas de pré-compilação
Compilação condicional

Compilação condicional

- São seis as diretivas que delimitam blocos de texto que devem ser compilados sob uma determinada condição:
  - **`#if:`** verifica se uma expressão constante é verdadeira, ou seja, um valor diferente de 0 (*nonzero*).
  - **`#ifdef:`** verifica se um dado identificador está definido.
  - **`#ifndef:`** verifica se um dado identificador não está definido.
  - **`#else:`** código alternativo para o caso das condições de `#if`, `#ifdef` ou `#ifndef` forem falsas.
  - **`#elif:`** o correspondente a `else-if`.
  - **`#endif:`** diretiva para finalizar o bloco.

Compilação condicional

- Exemplo:

```
#include <stdio.h>
#define a 10

void main() {
   #ifdef a
      printf("Alguma coisa");
   #endif
   #ifndef a
      printf("Não definido");
   #else
      printf("Estou aqui!");
   #endif
}
```

- Saída:
  Alguma coisa
  Estou aqui!

Compilação condicional

- Operador `defined` é útil quando se quer checar as definições de várias macros em uma linha, em vez de usar vários `#ifdef` ou `#ifndef`. Ex.:

```
#if define (macro1) || !define(macro2) || defined(macro3)
   printf("Olá!\n");
#endif
```

Diretivas de pré-compilação
Diretiva pragma

Diretiva pragma

- Funciona de modo diferente, pois não altera o código fonte, mas fornece instruções especiais ao compilador alterando o processo de compilação ou especificando detalhes extra-código.
- A diretiva depende da implementação do compilador, bem como as opções disponíveis, as quais dependem do fabricante do compilador e do processador para o qual o código está sendo compilado.
- Sintaxe: **`#pragma`** `<definição> <opções>`
- Documentação da diretiva pragma para o compilador C/C++ da Microsoft
- Ex.:

```
#pragma config OSC=HS // oscilador a cristal externo
```

Enumeradores

- Enumeradores são definições da linguagem C que visam facilitar a criação de referências visuais.
- Sintaxe:
  `enum{LABEL1 = VALOR1, LABEL2 = VALOR2, ..., LABELN = VALORN}.`
- Se algum valor é omitido, ele receberá automaticamente o incremento do valor anterior. Ex.:
  `enum{AZUL = 1, VERDE, VERMELHO = 4};`
  Neste caso VERDE será AZUL+1.

Manipulando bits

Manipulando bits

- **Bit fields**
  - Uma necessidade comum em sistemas embarcados é a capacidade de manipular apenas uma certa quantidade de bits ou até mesmo um único bit.
  - Isto pode ser feito através de **operações bitwise** ou da criação de estruturas com variáveis cujo tamanho seja determinado pelo programados.
  - Sintaxe das estruturas:

```
struct{
    tipo nome :  tamanho;
};
```

  - Vejamos exemplo01.c

Manipulando bits
Operações bitwise

Operações bitwise

- As operações são baseadas na **Álgebra Booleana**
  - Então você terá de ver novamente sobre as operações **AND**, **OR**, **NOT**, **XOR**, etc.
- Possuem efeito sobre cada bit de uma variável.
- Operações binárias
  - **NÃO**: troca o bit.
    - Sintaxe: `resultado = ~ variavel;`
    - Ex.:
      `char A = 12;` (0b00001100)
      `char r;`
      `r = ~A;` // r → 0b11110011 (243 em decimal)
  - **E**: operação AND.
    - Sintaxe: `resultado = variavel1 & variavel2;`
    - Ex.:
      `char A = 8;` (0b00001000)
      `char B = 5;` (0b00000101)
      `char r;`
      `r = A & B;` (0b00000000)

Operações bitwise

- Operações binárias
  - **OU**: operação OR.
    - Sintaxe: `resultado = variavel1 | variavel2;`
    - Ex.:
      `char A = 9;` (0b00001001)
      `char B = 3;` (0b00000011)
      `char r;`
      `r = A | B;` (0b00001011) $\rightarrow$ 11 em decimal.
  - **OU EXCLUSIVO**: operação XOR.
    - Sintaxe: `resultado = variavel1 ^ variavel2;`
    - Ex.:
      `char A = 12;` (0b00001100)
      `char B = 5;` (0b00000101)
      `char r;`
      `r = A ^ B;` (0b00001001) $\rightarrow$ 9 em decimal.

Manipulando bits
Operações de deslocamento

Operações de deslocamento

- Operação nativa dos processadores.
- Sintaxe:
  `resultado = variavel` $\gg$ `vezes;` $\rightarrow$ para a direita
  `resultado = variavel` $\ll$ `vezes;` $\rightarrow$ para a esquerda
- Exemplo:
  $10111001 \ll 1 \rightarrow 01110010$
  $10111001 \gg 1 \rightarrow 01011100$
- O exemplo acima funciona com tipos **unsigned**. Porém, quando o tipo é **signed**, ou seja, com sinal, pode acontecer o *shift aritmético*:
  $\mathbf{1}0111001 \gg 1 \rightarrow \mathbf{1}1011100$

Aula baseada no livro:
Almeida, Rodrigo Maximiniano A. **Programação de Sistemas Embarcados - Desenvolvendo Software para Microcontroladores em Linguagem C**. São Paulo: Grupo GEN, 2016.

LER capítulos 3 a 9!

FIM