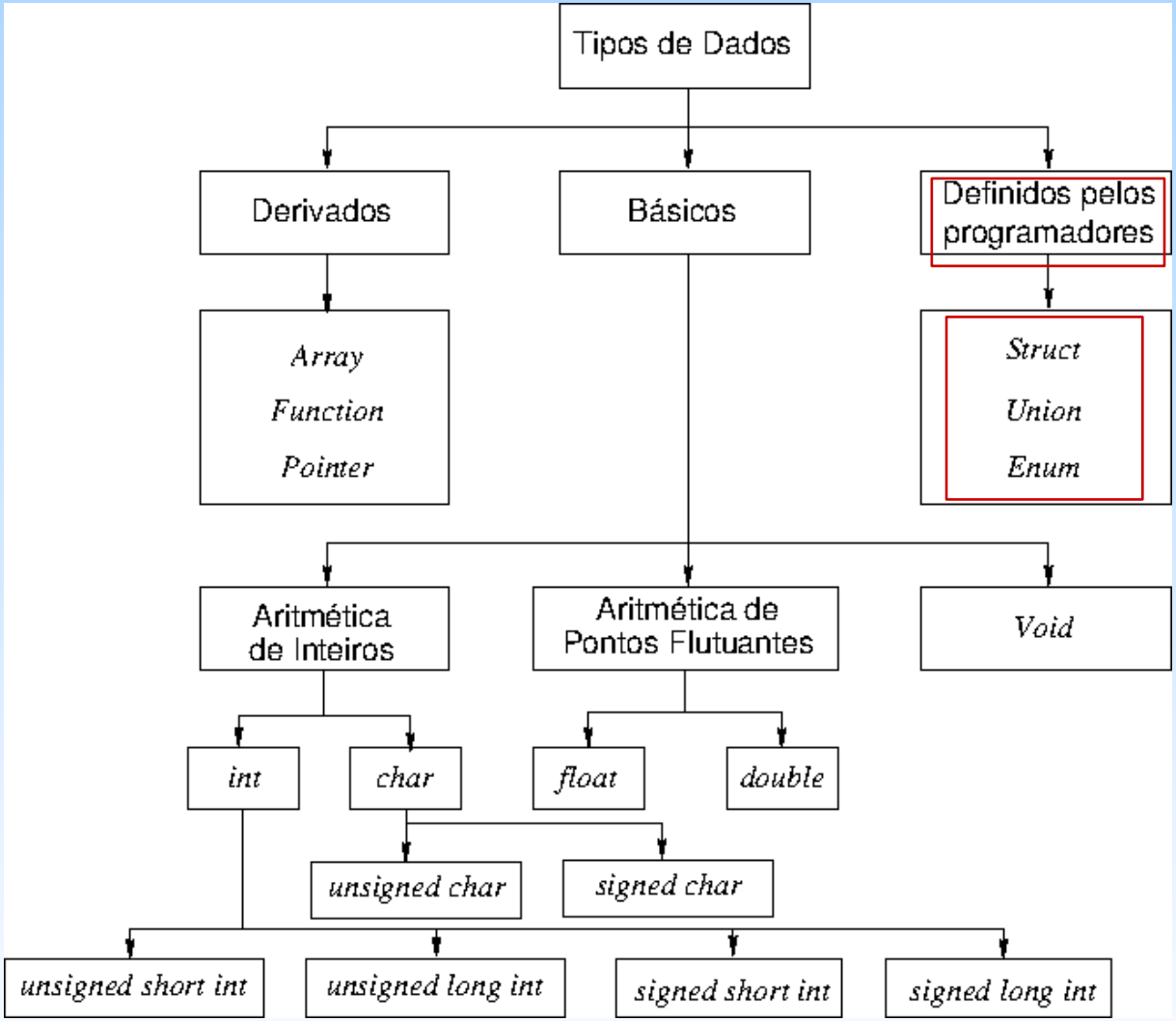
EA871

Primeira Biblioteca

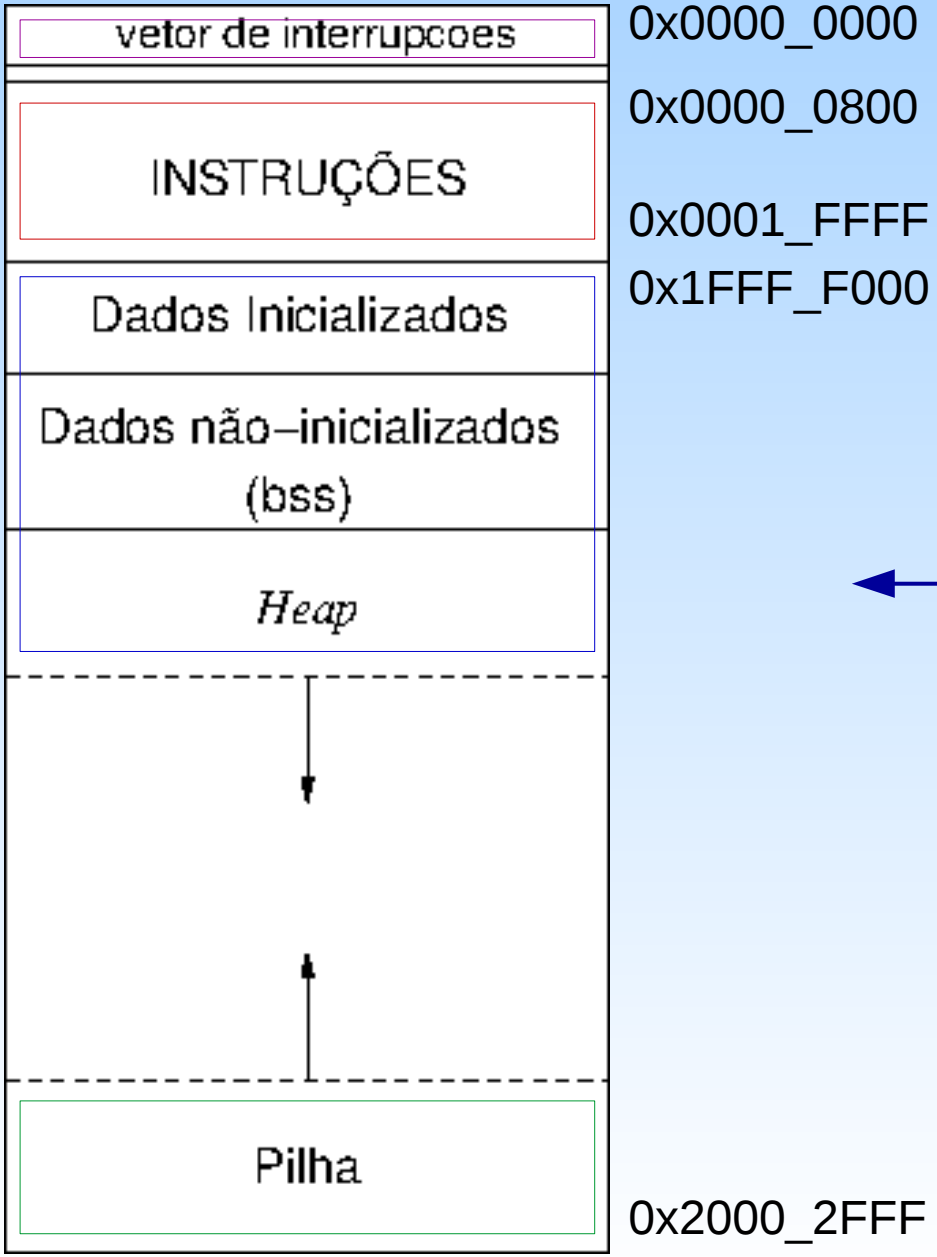
Tipos de dados programados: enum, union, struct

Wu Shin – Ting
DCA – FEEC - Unicamp
Segundo Semestre de 2020

Tipos de Dados Programados



Conceitos



Declaração de um tipo de dado programado: informar o compilador sobre o nome, os seus elementos, o padrão de organização desses elementos no espaço de memória alocado às suas variáveis.

Conceitos

- **enum** (Enumeração): é um tipo de dado em que o programador consegue atribuir nomes aos **valores inteiros**, facilitando a interpretação destes valores.
  - **Declaração**: **enum [nome] {<lista de nomes>} <variáveis>;**

    enum Booleana {Verdadeiro=1,Falso=0} checa;

    enum Semana{Seg=1000,Ter,Qua,Qui,Sex,Sab,Dom} dia;

    enum Naipes{Paus,Ouros,Copas,Espadas} carta, cp_carta;

- Observações:
  - Pode-se declarar a cada nome um valor inteiro, não necessariamente único e na ordem crescente.
  - Quando não declarado explicitamente, o primeiro nome da lista assume o valor inteiro 0 e o restante dos nomes, o valor inteiro do elemento precedente incrementado de 1.

Conceitos

- **enum** (Enumeração)
  - **Declaração:**

    enum Booleana {Verdadeiro=1,Falso=0} checa;

    enum Semana{Seg=1000,Ter,Qua,Qui,Sex,Sab,Dom} dia;

    enum Naipes{Paus,Ouros,Copas,Espadas} carta, cp_carta;

  - **Definição**: é **alocado à variável declarada** um espaço de tamanho do maior valor inteiro associado aos nomes da lista.

    checa {0,1} → 1 *byte*

    dia {1000,1001,1002,1003,1004,1005,1006} → 2 *bytes*

    carta, cp_carta {0,1,2,3} → 1 *byte*

Conceitos

- **union** (União): é um tipo de dado em que diferentes tipos de dados podem ser armazenados num mesmo endereço da memória.
  - **Declaração**: **union [nome] {<uma lista de membros>} <variáveis>;**

    union Identidade {

        unsigned int CPF;

        char nome[20];

        } pessoa;
  - **Definição:** é **alocado à variável declarada** um espaço de memória do tamanho do maior espaço ocupado pelos seus membros.

    pessoa (4 *bytes*/20 *bytes*) → 20 *bytes*

Conceitos

- **struct**: é um tipo de dado que permite armazenar todos os elementos de diferentes tipos de dados num espaço contíguo da memória.
  - **Declaração**: **struct [nome] {<uma lista de membros>} <variáveis>;**

    ```
    struct Identidade {
        unsigned int CPF;
        char nome[20];
        } pessoa;
    ```

  - **Definição:** é **alocado à variável declarada** um espaço de memória correspondente à soma dos espaços de memória demandados pelos seus membros.

    pessoa (4 *bytes*+20 *bytes*) → 24 *bytes*

Acessos a membros

```
union Identidade {
    unsigned int CPF;
    char nome[20];
    } pessoa, *ptr_pessoa;
```

```
struct Identidade {
    unsigned int CPF;
    char nome[20];
    } pessoaS,
    *ptr_pessoaS;
```

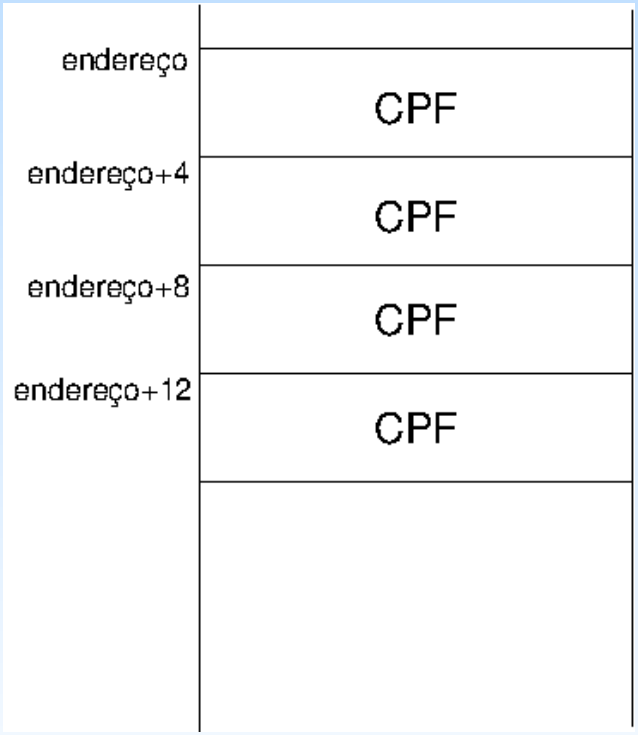
- Por variável

  pessoa.CPF, pessoa.nome, pessoaS.CPF
- Por ponteiro

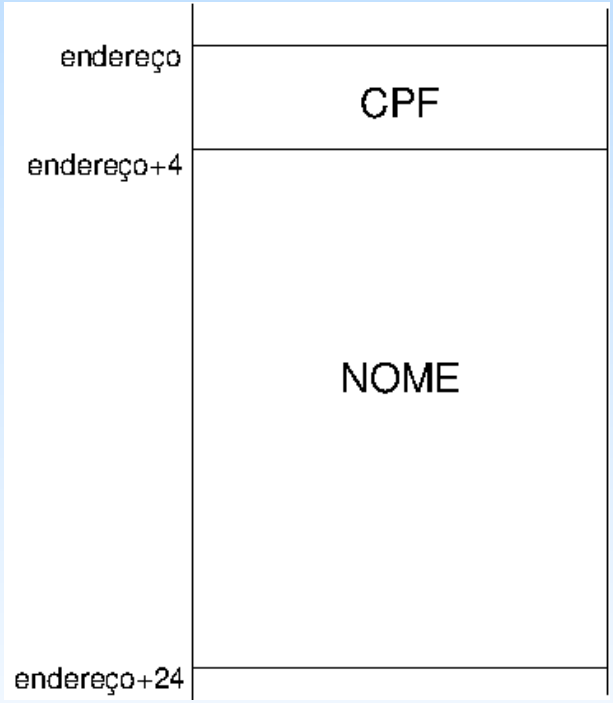
  ptr_pessoa→CPF, ptr_pessoaS→ nome

Visão Comparativa

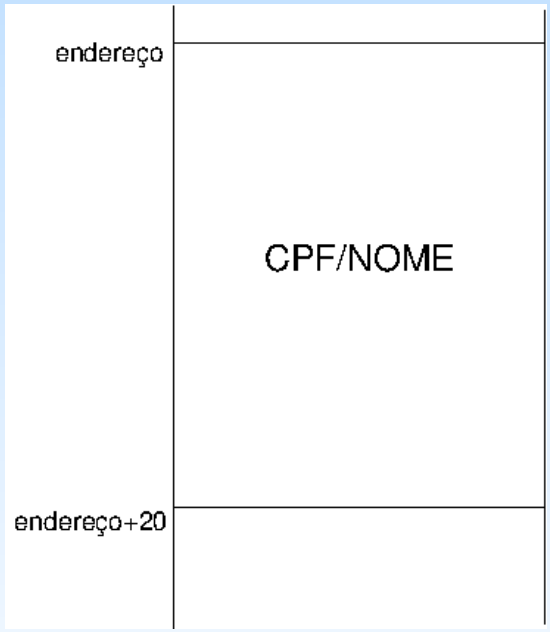
arranjo: lista de elementos de mesmo tipo



struct: lista de elementos de tipos diferentes



union: um elemento cujo tipo pode variar entre os tipos declarados para seus membros



Através da declaração de um tipo struct consegue-se associar nomes aos espaços de tamanhos diferentes de um bloco de memória.

Uma Aplicação

- Como endereçar os registradores já mapeados num espaço contíguo de memória por nomes mais fáceis de entender?

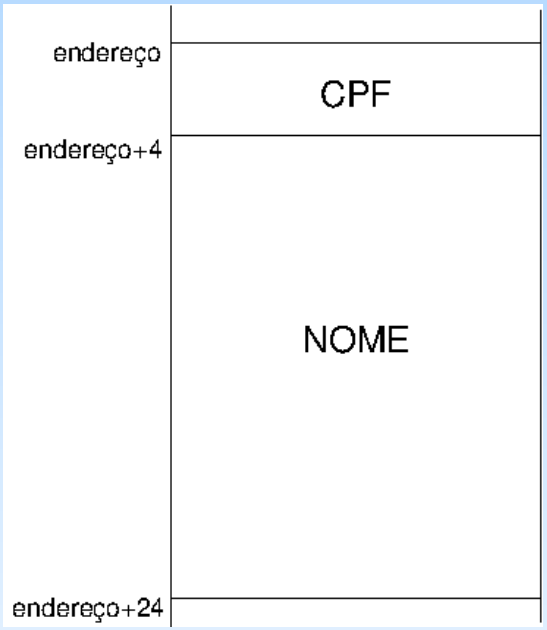
Absolute address (hex)	Register name	Width (in bits)	Access	Reset value
400F_F100	Port Data Output Register (GPIOE_PDOR)	32	R/W	0000_0000h
400F_F104	Port Set Output Register (GPIOE_PSOR)	32	W (always reads 0)	0000_0000h
400F_F108	Port Clear Output Register (GPIOE_PCOR)	32	W (always reads 0)	0000_0000h
400F_F10C	Port Toggle Output Register (GPIOE_PTOR)	32	W (always reads 0)	0000_0000h
400F_F110	Port Data Input Register (GPIOE_PDIR)	32	R	0000_0000h
400F_F114	Port Data Direction Register (GPIOE_PDDR)	32	R/W	0000_0000h

Uma Aplicação

Declaração

```
struct Identidade {
        unsigned int CPF;
        char nome[20];
} pessoa;
```

compilador



Mapeamento inverso

?

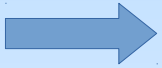
programador

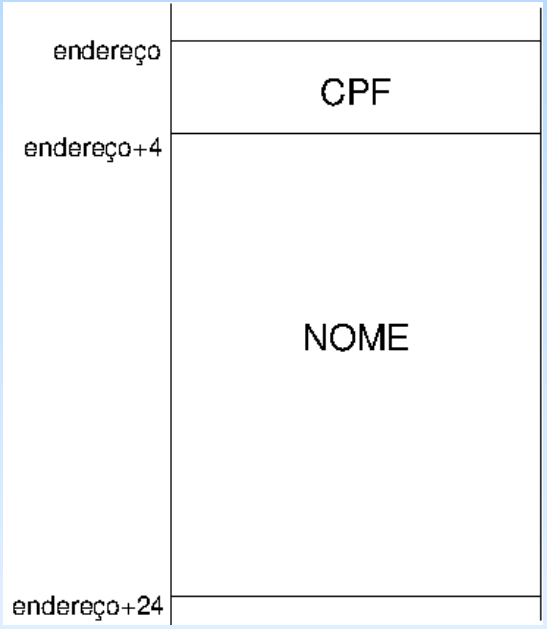
Uma Aplicação

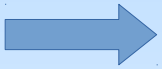
Declaração

```
struct Identidade {
        unsigned int CPF;
        char nome[20];
} pessoa;
```

compilador





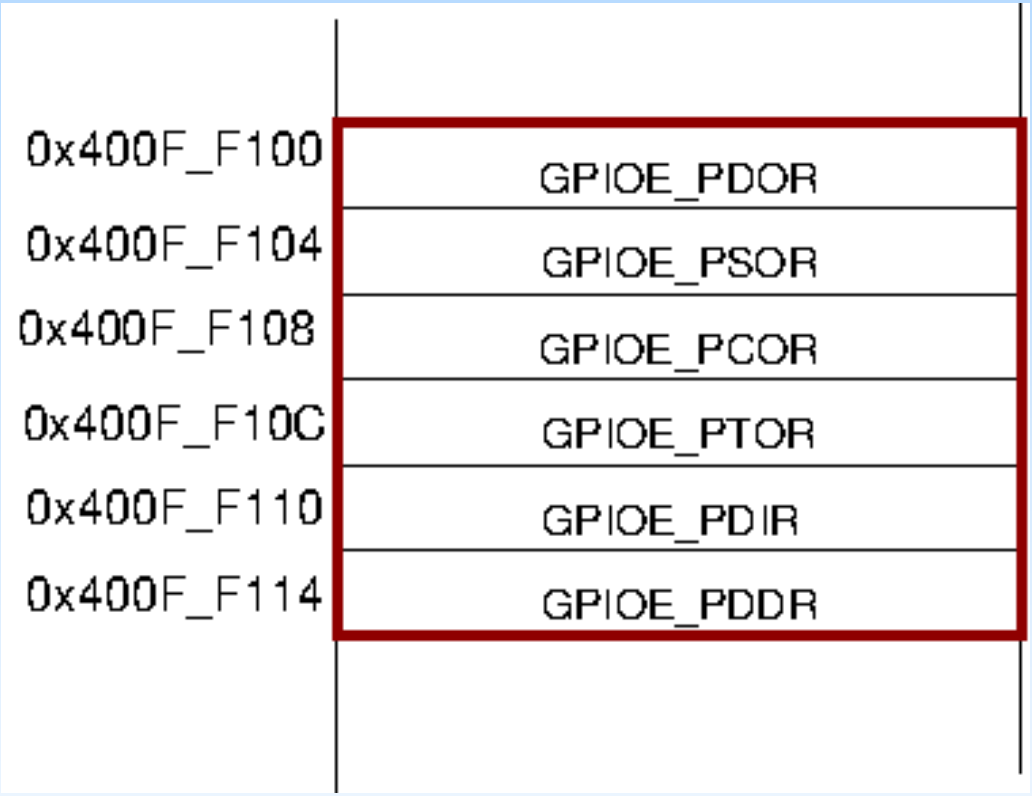


programador

Mapeamento inverso

```
(struct Identidade {
        unsigned int CPF;
        char nome[20];
}) endereço;
```

Uma Aplicação



- **Desenho do novo tipo de dado**

  struct GPIO_MemMap{

  unsigned int PDOR;

  unsigned int PSOR;

  unsigned int PCOR;

  unsigned int PTOR;

  unsigned int PDIR;

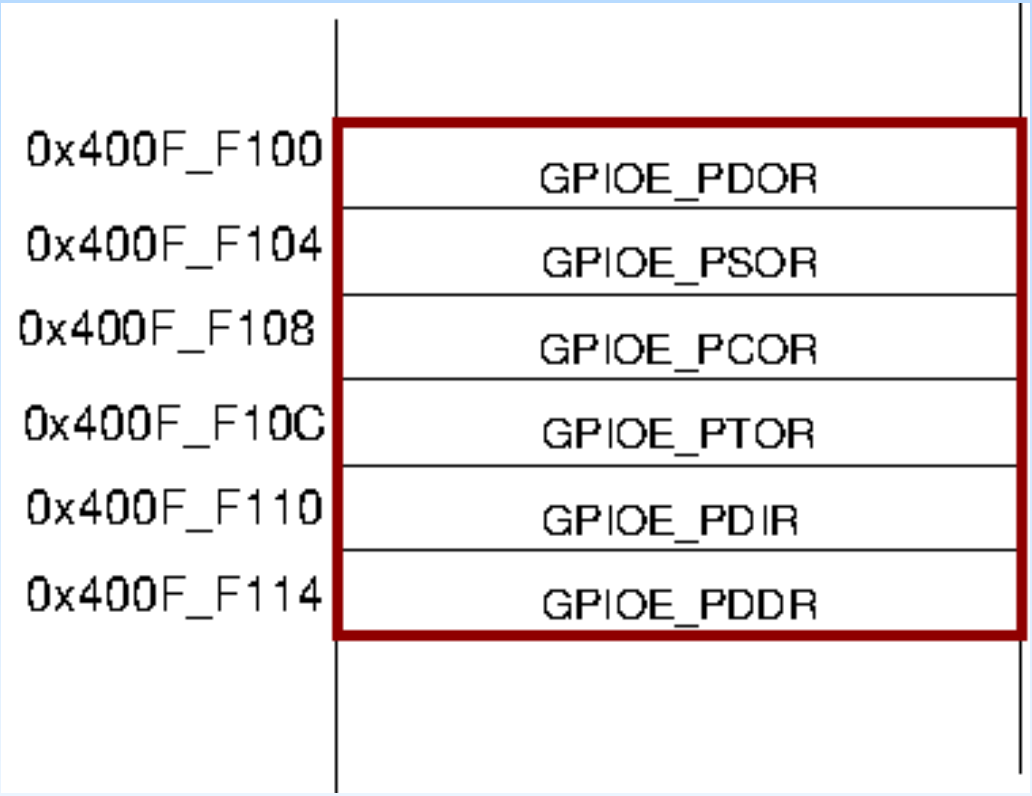
  unsigned int PDDR;

  };

- **Conversão explícita do endereço**

  (struct GPIO_MemMap volatile *) (0x400FF100u)

Uma Aplicação



- **Acessos por ponteiros**

struct GPIO_MemMap
volatile *end_gpioe;

end_gpioe = (struct GPIO_MemMap volatile *) (0x400FF100u);

end_gpioe→PDOR;

end_gpioe→PSOR;

end_gpioe→PCOR;

end_gpioe→PTOR;

end_gpioe→PDIR;

end_gpioe→PDDR;

CodeWarrior IDE Development Suite

Informações Adicionais

- Enumeration (or enum) in C

  https://www.geeksforgeeks.org/enumeration-enum-c/
- Unions in C

  https://www.geeksforgeeks.org/union-c/
- Structures in C

  https://www.geeksforgeeks.org/structures-c/?ref=lbp
- Linguagem C: Representação de Dados

  ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/RepresentacaoDados.pdf

EA871

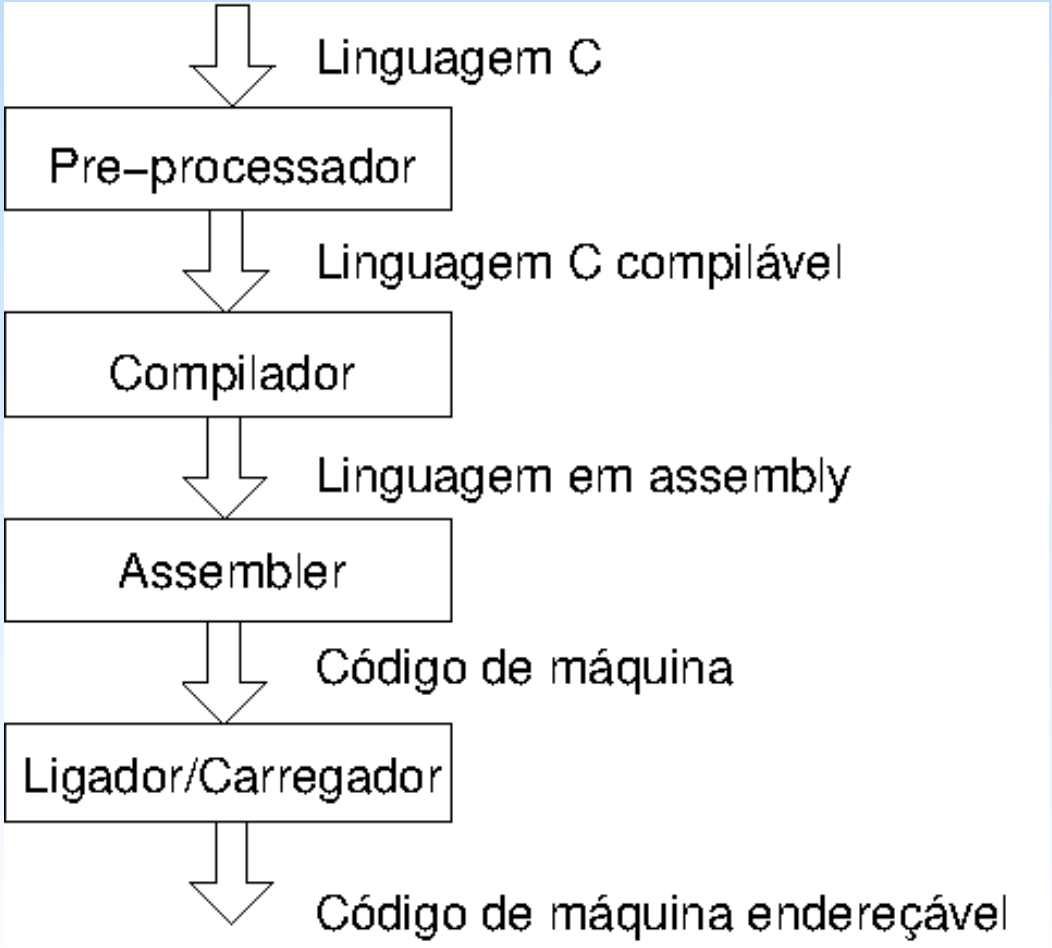
Primeira Biblioteca

Pré-processamento

Wu Shin – Ting
DCA – FEEC - Unicamp
Segundo Semestre de 2020

Conceitos

- **Pré-processador em C**: é um aplicativo que pré-porcessa as diretivas em linguagem de alto nível, transformando os códigos-fonte em linguagem C compilável.



Conceitos

- **Diretivas em C:** são expressões em linguagem C que “dirigem” as ações do compilador. A sintaxe de uma diretiva é usualmente

  #<diretiva> [macro] [definição da macro]

  Exemplos:

  #include “derivative.h”

  #define SIM_SCGC5 (*(unsigned int volatile *) 0x40048038u)

  #define GPIO_PIN(x) (1 << x)

  #define max (a,b) \

  (a < b ? \

  b: \

  a)

Diretivas em C

#include: inclusão do conteúdo de um arquivo.	#if: condicional sobre uma expressão
#define: definição de uma macro	#else: alternativa de um condicional
#undef: cancelamento de uma definição	#elif: um novo condicional como alternativa
#ifdef: condicional (positivo) de uma dada definição	#endif: fim do escopo de um condicional
#ifndef: condiconal (negativo) de uma dada definição	#error: indicação de erro
#pragma: usada para inclusão ou exclusão das funcionalidades do compilador.	

Conceitos

- **Tarefas de um pré-processador**
  - Remover comentários dos códigos
  - Inclusão de arquivos
    - <arquivo.h>: busca de arquivo.h no caminho padrão do compilador.
    - “arquivo.h”: busca de arquivo.h também na pasta corrente.
  - Expansão de macros
  - e outras.

Conceitos

- **Macro:** é uma sequência de códigos referenciada por um nome ao longo de um programa em C. Este nome é substituído automaticamente antes da compilação do programa.
  - **Substituição similar a um objeto**

```
#define SIM_SCGC5 (*(unsigned int volatile *) 0x40048038u)

#define SIM_SCGC5_PORTE_MASK 0x2000u

SIM_SCGC5 = SIM_SCGC5 | SIM_SCGC5_PORTE_MASK;

(*(unsigned int volatile *) 0x40048038u) =  (*(unsigned int volatile *) 0x40048038u) | 0x2000u;
```

  - **Substituição similar a uma função**

```
#define PIN(x) (1<<x)

SIM_SCGC5 |= PIN(13);

(*(unsigned int volatile *) 0x40048038u) |= (1<<13);
```

Boas práticas para uso de macros

- Use maíusculas ao definir macros.
- Substitua números “mágicos” (ex. pi) pela macro.
- Use macros para funções simples.
- Evite ambiguidades na precedência de operações.
- Evite ambiguidade na aplicação dos operadores.
- Evite ambiguidade na interpretação do escopo de uma macro.

Exemplos

```
#define bloco1 (a,b) \
    a++; \
    b--;

    main () {
    int x,y;
    :
    if ( x > 0)
    bloco1(x,y);
    :
    :
```

```
If (x > 0)  x++;

y--;
```

```
#define fPTR float *

fPTR a, b;
```

```
float *a, b;
```

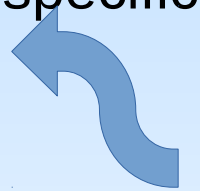
- a é um ponteiro ao tipo float.
- b é uma variável do tipo float.

As definições das macros substituem LITERALMENTE as macros.

Um Exemplo: Derivative.h

```
/*
 * Note: This file is recreated by the project wizard whenever the MCU is
 *       changed and should not be edited by hand
 */


/* Include the derivative-specific header file */
#include <MKL25Z4.h>
```



Contém a redefinição dos endereços dos registradores mapeados no espaço da memória em nomes usados no manual de referência e todas as máscaras para acessar diferentes campos dos registradores. Cada família de microcontroladores tem um arquivo próprio.

MKL25Z4.h

- Inclusão do arquivo **stdint.h** que contém as redefinições dos tipos de dados para os especificados pelo padrão ISO/IEC 9899:1999

  uint*N*_**t**: inteiro sem sinal com *N* bits.

  int*N*_**t**: inteiro com sinal com *N bits*.

  #define uint32_t unsigned int

  #define int32_t signed int/int

- Convenção de nomes de macros

  <Módulo>_<Registrador>_<Campo>_<função>

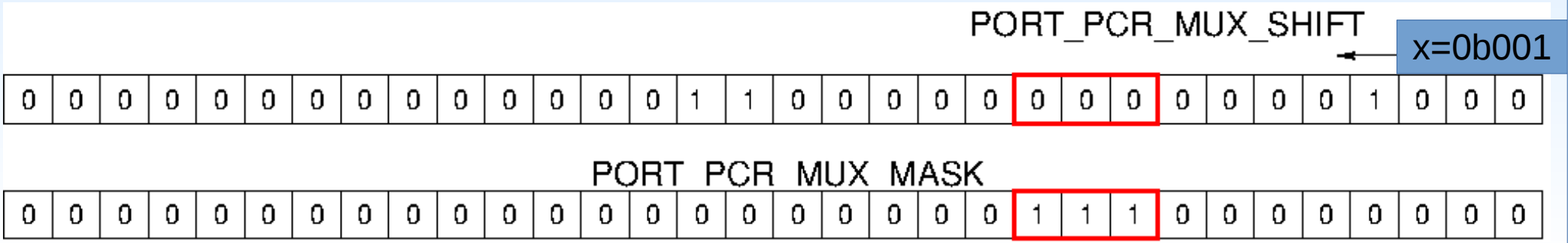
  Exemplos:

  - GPIOE_PDDR
  - PORTE_PCR23_MUX
  - SIM_SCGC5_PORTE_MASK

MKL25Z4.h

- Macros simples

```
#define SIM_SCGC5_PORTE_MASK 0x2000u
#define SIM_SCGC5_PORTE_SHIFT 13
#define PORT_PCR_MUX_MASK          0x700u
#define PORT_PCR_MUX_SHIFT         8
#define PORT_PCR_MUX(x)      (\
    ((uint32_t)(((uint32_t (x))<<PORT_PCR_MUX_SHIFT)) \
    & PORT_PCR_MUX_MASK)
```



MKL25Z4.h

- Declaração de struct que mapeia os endereços a nomes de registradores adotados nos manuais de referência

```
/** PORT - Peripheral register structure */
typedef struct PORT_MemMap {
  uint32_t PCR[32];
  uint32_t GPCLR;
  uint32_t GPCHR;
  uint8_t RESERVED_0[24];
  uint32_t ISFR;
} volatile *PORT_MemMapPtr;
```

MKL25Z4.h

```
/** SIM - Peripheral register structure */
typedef struct SIM_MemMap {
  uint32_t SOPT1;
  uint32_t SOPT1CFG;
  uint8_t RESERVED_0[4092];
  uint32_t SOPT2;
  uint8_t RESERVED_1[4];
  uint32_t SOPT4;
  uint32_t SOPT5;
  uint8_t RESERVED_2[4];
  uint32_t SOPT7;
  uint8_t RESERVED_3[8];
  uint32_t SDID;
  uint8_t RESERVED_4[12];
  uint32_t SCGC4;
  uint32_t SCGC5;
  uint32_t SCGC6;
  uint32_t SCGC7;
  uint32_t CLKDIV1;
  uint8_t RESERVED_5[4];
  uint32_t FCFG1;
  uint32_t FCFG2;
  uint8_t RESERVED_6[4];
  uint32_t UIDMH;
  uint32_t UIDML;
  uint8_t RESERVED_7[156];
  uint32_t COPC;
  uint32_t SRVCOP;
} volatile *SIM_MemMapPtr;
```

MKL25Z4.h

- Declaração de struct para mapear os endereços dos registradores a nomes usados nos manuais de referência

```
/** GPIO - Peripheral register structure */
typedef struct GPIO_MemMap {
  uint32_t PDOR;
  uint32_t PSOR;
  uint32_t PCOR;
  uint32_t PTOR;
  uint32_t PDIR;
  uint32_t PDDR;
} volatile *GPIO_MemMapPtr;
```

MKL25Z4.h

- Macros mais elaborados

  /** Peripheral PORTE base pointer */

  - Endereço-base do bloco de memória onde os registradores do módulo PORT são mapeados

  #define **PORTE_BASE_PTR** ((PORT_MemMapPtr)0x4004D000u)

  - Acesso ao registrador mapeado dentro do bloco

  #define **PORT_PCR_REG(base,index)** ((base)→PCR[index])

  - Redefinição em nome usado no manual de referência

  #define **PORTE_PCR23** PORT_PCR_REG(PORTE_BASE_PTR,23)

MKL25Z4.h

- Macros mais elaborados

/** Peripheral SIM base pointer */

Endereço-base do bloco de memória onde os registradores do módulo SIM são mapeados

#define **SIM_BASE_PTR** ((SIM_MemMapPtr)0x40047000u)

- Acesso ao registrador mapeado dentro do bloco

#define **SIM_SCGC5_REG(base)** ((base)→SCGC5)

- Redefinição em nome usado no manual de referência

#define **SIM_SCGC5** SIM_SCGC5_REG(SIM_BASE_PTR)

MKL25Z4.h

- Macros mais elaborados

  /** Peripheral PTE base pointer */

  - Endereço-base do bloco de memória onde os registradores do módulo GPIOE são mapeados

  #define **PTE_BASE_PTR** ((GPIO_MemMapPtr)0x400FF100u)

  - Acesso aos registradores mapeados dentro do bloco

  #define **GPIO_PSOR_REG(base)** ((base)->PSOR)

  #define **GPIO_PCOR_REG(base)** ((base)->PCOR)

  #define **GPIO_PDDR_REG(base)** ((base)→PDDR)

  - Redefinição em nomes usados no manual de referência

  #define **GPIOE_PSOR** GPIO_PSOR_REG(PTE_BASE_PTR)

  #define **GPIOE_PCOR** GPIO_PCOR_REG(PTE_BASE_PTR)

  #define **GPIOE_PDDR** GPIO_PDDR_REG(PTE_BASE_PTR)

CodeWarrior IDE Development Suite

Informações Adicionais

- Preprocessor Directives – C Programming

  https://developerinsider.co/preprocessor-directives-c-programming/
- Simplifique: use Macros (ou não)

  https://sergioprado.org/simplifique-use-macros-ou-nao/
- O pré-processador C

  http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node150.html
- Lista de palavras reservadas em C

  http://linguagemc.com.br/lista-de-palavras-reservadas-em-c/
- Linguagem C: Representação de Dados

  ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/RepresentacaoDados.pdf

Informações Adicionais

- Macros (avançado)

  https://gcc.gnu.org/onlinedocs/cpp/Macros.html

- stdint.h

  https://pubs.opengroup.org/onlinepubs/009695399/basedefs/stdint.h.html

- KL25 Sub-Family Reference Manual

  ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf

EA871

Primeira Biblioteca

Construção e Uso de Biblioteca em CodeWarrior

Wu Shin – Ting
DCA – FEEC - Unicamp
Segundo Semestre de 2020

EA871

Primeira Biblioteca

Construção e Uso de Biblioteca em CodeWarrior

Wu Shin – Ting
DCA – FEEC - Unicamp
Segundo Semestre de 2020

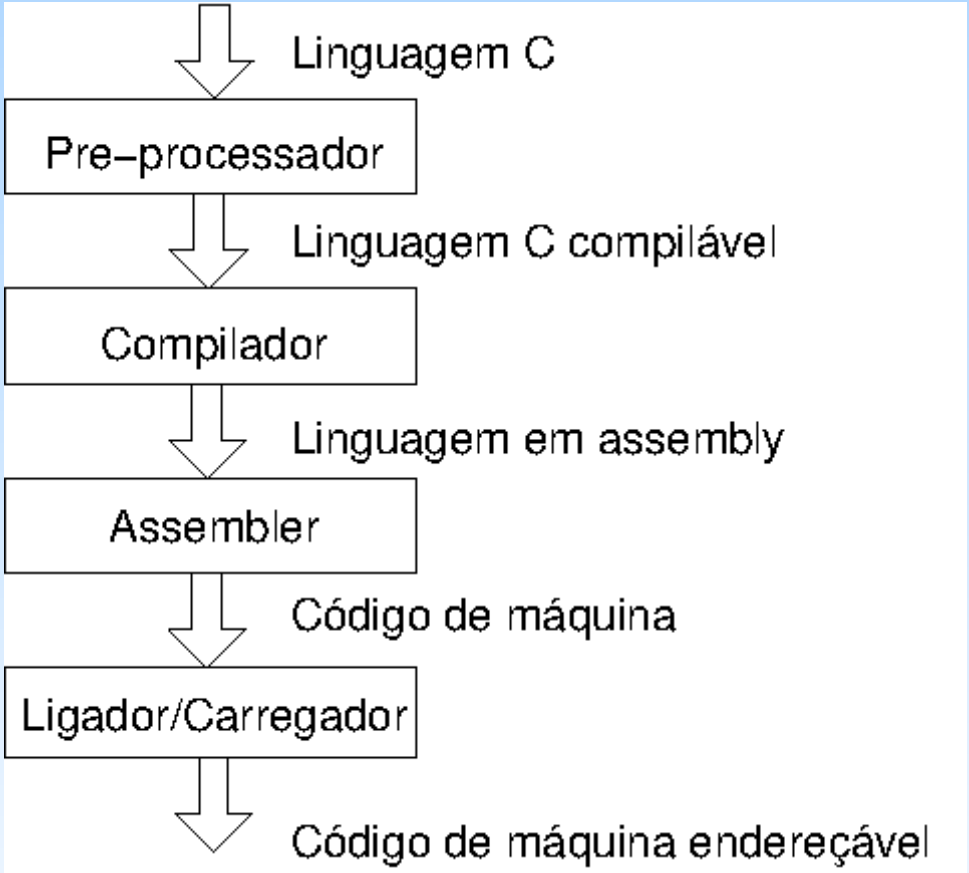
Conceitos

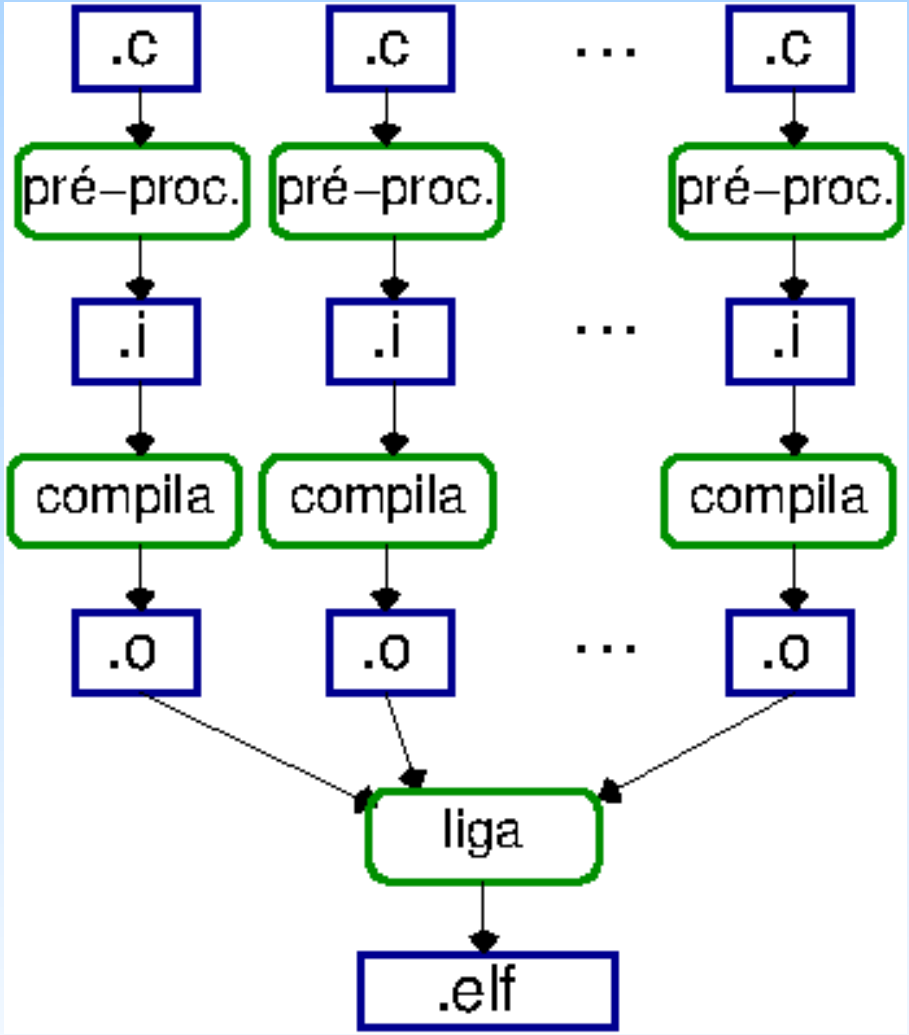
- **Programação modular:** é um paradigma em que um programa é dividido em diferentes funções/rotinas agrupadas em componentes de *software*.
- **Componentes de *software*:** são unidades independentes de programas que encapsulam uma série de funcionalidades relacionadas com uma tarefa específica.

- **Vantagens**
  - Reusabilidade
  - Fácil manuseio/uso
  - Fácil manutenção

- **Desafios**
  - Estratégia de particionamento
  - Escopo de cada módulo
  - Interface de cada módulo

Programação Modular





Conceitos

- **Interface** ou **protótipo de uma função**: especifica o nome da função, os tipos de dados dos seus argumentos e o tipo de dado do valor retornado.
  - Sintaxe:

  <tipo_de_dado_retornado><nome>(<lista_de_argumentos>)
- **Implementação de uma função**: é o conjunto de instruções necessárias para realização da tarefa atribuída à função.
  - Sintaxe:

  <tipo_de_dado_retornado><nome>(<lista_de_argumentos>) {

  :

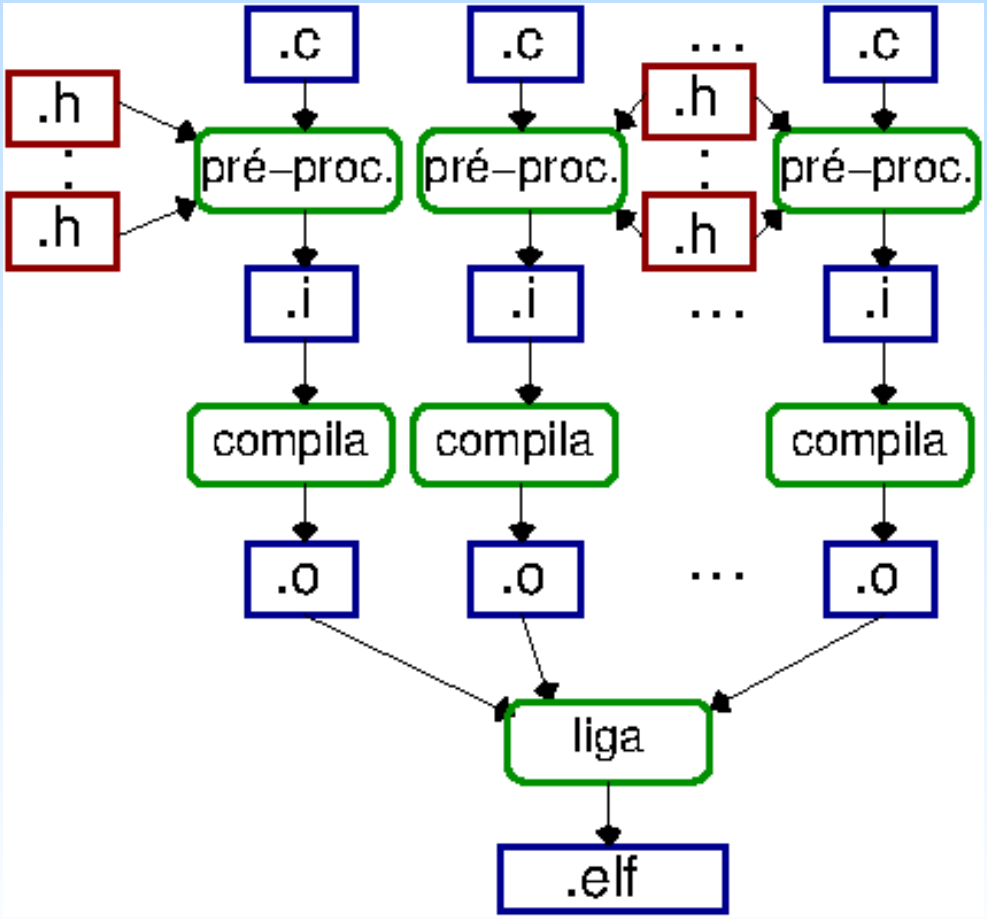
  :

  return <valor>;

  }

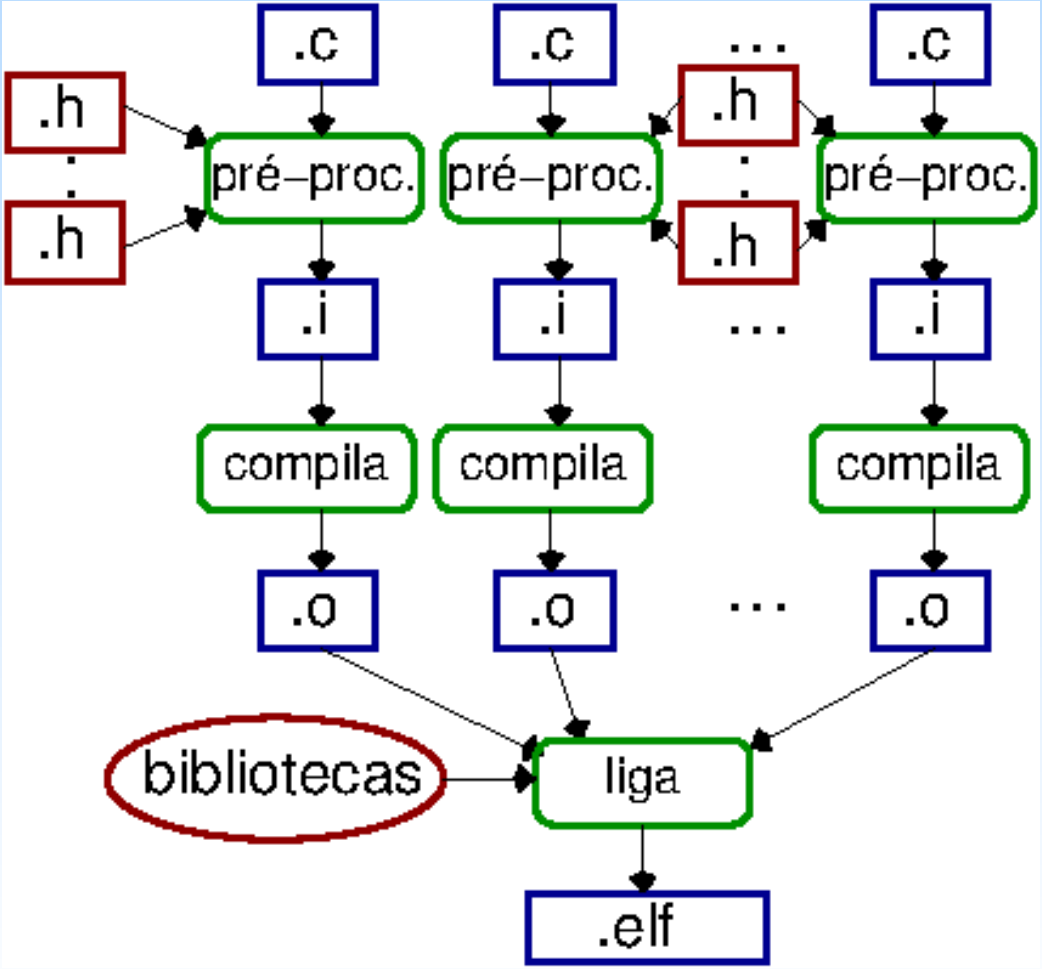
Conceitos

- **Arquivos de cabeçalho (*headers*, .h):** são arquivos que contêm as declarações das funções, das variáveis e dos tipos de dados compartilháveis de um componente de *software*.



Conceitos

- **Bibliotecas (.a)**: são arquivos que contêm os códigos compilados dos componentes de *software*.



Componente GPIO_ledRGB
Controle do led RGB pelo módulo GPIO

- Escopo do módulo
  - Inicialização dos módulos que controlam sinais digitais das três cores do led
  - Controle do estado {HABILITA,DESABILITA} de cada *led* individualmente
- Interface do módulo (.h)
  - InitLedRGB(), initLedR(), initLedG(), initLedB()
  - LedR(estado)
  - LedG(estado)
  - LedB(estado)
  - LedRGB(estado)
- Implementação do módulo (.c) na biblioteca lib<RA>.a

Componente util
Funções Utilitárias

- Escopo do módulo
  - Funções auxiliares
- Interface do módulo (.h) (no momento)
  - delay (iteracoes)
- Implementação do módulo (.c) na biblioteca lib<RA>.a

Construção de uma biblioteca

- Criar uma biblioteca <RA>
- Remover Lib/mylibrary.c
- Adicionar arquivos de implementação das funções do componente na pasta Lib: util.c, GPIO_ledRGB.c
- Para compilar os componentes:
  - Adicionar derivative.h e MKL25Z4.h na pasta Project Headers (se usar as macros)
  - Adicionar arquivos de declaração das funções dos componentes na pasta Project Headers: util.h GPIO_ledRGB.h
- Construir a biblioteca

Uso de uma biblioteca

- Adicionar no projeto os arquivos de interface (.h) OU o caminho onde se encontra a pasta de arquivos de interface (.h)
- Adicionar no projeto o arquivo e o caminho onde se encontra o arquivo que contém as implementaçõe (biblioteca)
- Construir o projeto

CodeWarrior IDE Development Suite

Informações Adicionais

- Modular Approach in Programming

  https://www.geeksforgeeks.org/modular-approach-in-programming/

- Creating and using Libraries with ARM gcc and Eclipse

  https://mcuoneclipse.com/2013/02/12/creating-and-using-libraries-with-arm-gcc-and-eclipse/

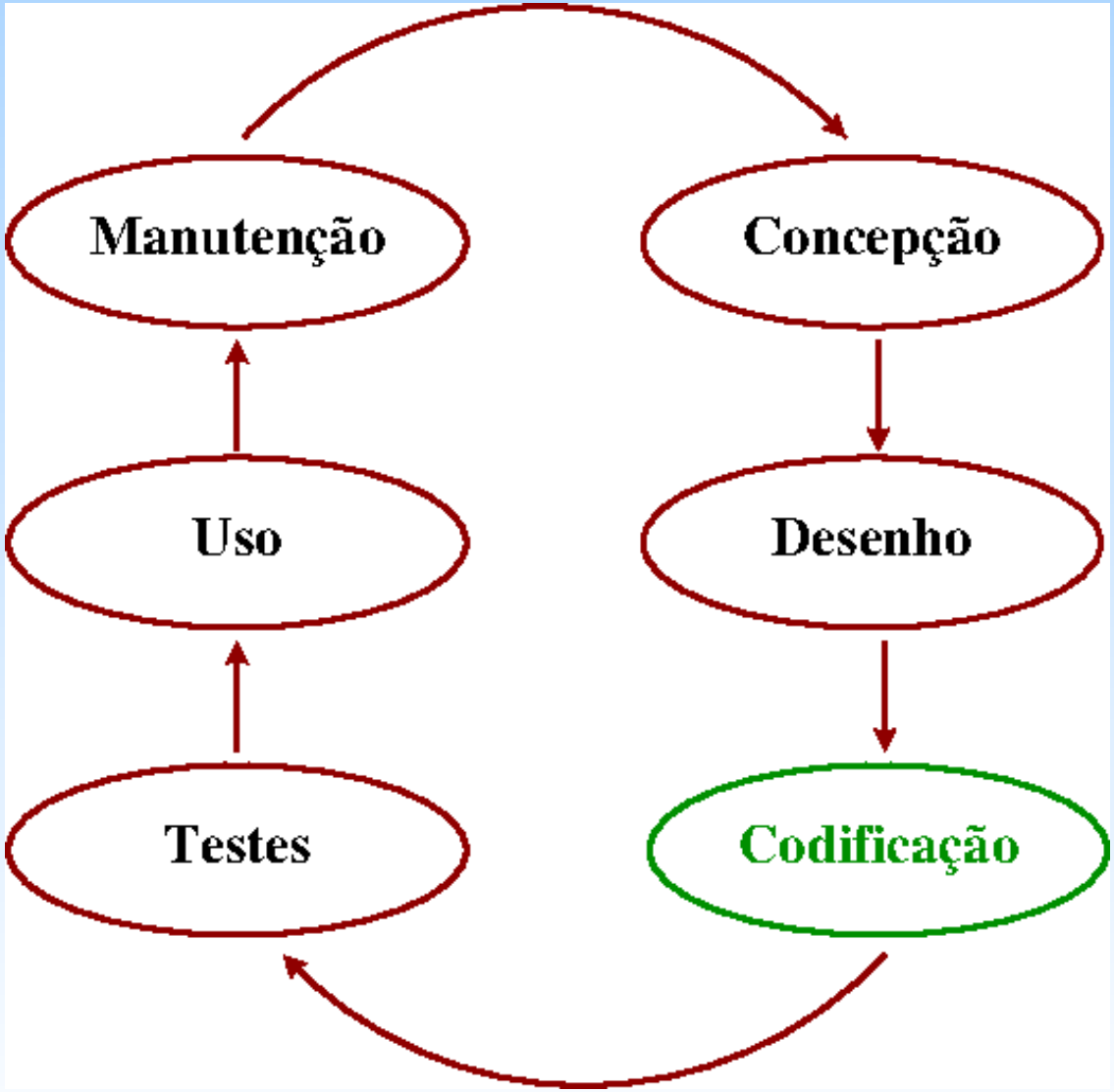
EA871

Primeira Biblioteca

Documentação

Wu Shin – Ting
DCA – FEEC - Unicamp
Segundo Semestre de 2020

Ciclo de Vida de Desenvolvimento



Documentação

- Pré-desenvolvimento (concepção e desenho)
  - Registro das funcionalidades desejadas e concepção de uma arquitetura que as atenda (como), para facilitar a análise de custos e benefícios e servir de modelo de referência ao longo da vida do *software*.
- Desenvolvimento (codificação e testes)
  - Justificativa (o por quê) das instruções implementadas (o quê), para facilitar a comunicação entre os desenvolvedores e a manutenção dos códigos.
- Pós-desenvolvimento (uso e manutenção)
  - Apresentação dos componentes e das interrelações entre eles para facilitar o uso (abstração de “o quê” numa interface mais amigável).

Conceitos

- **Documentação em programas**: “é um texto escrito ou uma ilustração que acompanha um programa ou é inserido nos códigos-fonte. Ela explica tanto como um programa funciona quanto como usá-lo. Tem valores distintos para profissionais diferentes.” (Wikipedia)
  - **Documentação interna**: são textos (comentários) que justificam as implementações nos códigos-fonte.
  - **Documentação externa**: são textos que descrevem a interface de programação de aplicativos (API, quando se trata de uma biblioteca) ou a forma de uso das funções disponíveis (quando se trata de um aplicativo).

Regras Básicas

- Documente os trechos de código somente quando não são auto-explicativos.
- Documente os códigos assim que são adicionados ao programa. Mantenha-os sempre atualizados.
- Documente a evolução das modificações nos seus códigos.
- Use uma linguagem simples, preferivelmente em voz ativa.
- Faça uso de ferramentas de suporte à documentação.

Adaptado de: https://easternpeak.com/blog/source-code-documentation-best-practices/

Ferramentas de Suporte

- Há ferramentas apropriadas para geração automática de documentos em cada fase do ciclo de vida de desenvolvimento de um projeto de software.
- Fase de Codificação:
  - Doxygen (livre)**:** extração automática da interface de programação dos aplicativos (API) a partir dos comentários nos códigos-fonte. É integrado ao ambiente IDE pelo *plugin* eclox e usa os aplicativos:
    - Graphviz: geração dos grafos de dependência entre as funções
    - Mscgen: geração sequência de mensagens

Doxygen

- Criar um *script* de geração de documento (extensão .doxyfile).
- Editar o *script* de geração
  - Inserir o nome e a versão do projeto
  - Incluir no diretório as pastas dos arquivos que contêm informações de documentação → Project Headers (não precisa ser varredura recursiva)
  - Selecionar o modo → somente os componentes documentados
  - Selecionar o formato de saída → html
  - Incluir no *script* o caminho do aplicativo graphviz que gera gráficos de dependência entre as funções (Advanced >: HAVE DOT YES; DOT Path: *C:\Programas_Aux\Graphviz2.38\bin*)
  - Incluir no script o caminho do aplicativo mscgen que gera diagrama de interações entre os componentes de *software (Advanced > msgen Path: C:\Programas_Aux\Mscgen)*
  - “Use dot tool from ...”
- Executar o *script*
- Abrir com um *browser* o arquivo index.html na pasta html gerada

CodeWarrior IDE Development Suite

Informações Adicionais

- Documenting your Code – Is it important?

  https://www.rapidvaluesolutions.com/documenting-your-code-is-it-important/

- Importance of software documentation

  https://deepsource.io/blog/importance-of-software-documentation/

- Documentation

  http://www.cs.ecu.edu/karl/3300/spr14/Notes/Documentation/index.html

- Every Stage of Project Implementation

  https://stepshot.net/21-software-documentation-tools-for-every-stage-of-project-implementation/#section6

Informações Adicionais

- Doxygen

  http://www.stack.nl/~dimitri/doxygen/

- 5 Best Eclipse Plugins: #1 (Eclox, with Doxygen, Graphviz and Mscgen)

  https://mcuoneclipse.com/2012/06/25/5-best-eclipse-plugins-1-eclox-with-doxygen-graphviz-and-mscgen/