Modularização e TADs

Algoritmos e Estruturas de Dados 2
2017-1
Flavio Figueiredo (http://flaviovdf.github.io)

Você foi contratado para desenvolver o sistema do DCC. Como você organizaria seu programa?

Tudo em um mega arquivo C?

- Transações
- Dados de correntistas
- Dados de FGTS
- Operações com outros bancos
- Operações com o banco de dados
- Comunicação entre processos
- Interface gráfica
- ...
- ...
- ...

Modularização

Modularização

- Organizar seu código em módulos

- O que é um módulo?
  - Conjunto de funções com um propósito comum
  - Vocês utilizam módulos direto

- Qual a diferença entre?
  - #include <stdio.h>
  - #include <stdlib.h>
  - #include <math.h>

Vamos pensar em um meu_math.h

- Quais funções vocês colocariam aqui?

Vamos pensar em um meu_math.h

- Quais funções vocês colocariam aqui?

- Quais constantes vocês colocariam aqui?

Vamos pensar em um meu_math.h

- Quais funções vocês colocariam aqui?
- Quais constantes vocês colocariam aqui?
- Quais structs vocês colocariam aqui?

Vamos pensar em um meu_math.h

- Quais funções vocês colocariam aqui?
- Quais constantes vocês colocariam aqui?
- Quais structs vocês colocariam aqui?
- Quais macros vocês colocariam aqui?

meu_math.h

- O arquivo .h é um cabeçalho
- Define funções/tipos/constantes que serão exportadas
- Quando você importa:
  - #include <stdio.h>
  - De onde vem a constante global EOF?
- Pense no cabeçalho como um contrato
  - Alguém vai ter que implementar o mesmo

```c
#ifndef MATH_H
#define MATH_H

#define M_E 2.71
#define M_PI 3.1415

//...

/*
 * Computa a raiz quadrada de um número
 */
double sqrt(double x);

/*
 * Computa a potência de um número
 */
double pow(double x, double y);

//...

#endif
```

meu_complexo.h

- Módulo de números complexos

- 2 funções
- 1 tipo
  - complex_t

- Precisamos separar o complexo_h do meu_math.h?
  - Esse é o problema de modularização
  - Alto nível
  - Módulos com coeso
- Não faz sentido ter 1 módulo por função

```
#ifndef COMPLEXO_H
#define COMPLEXO_H

struct ComplexoStruct {
      double real;
      double imaginario;
};

/*
 * Adiciona 2 números complexos.
 * Definido como a soma do real e a soma do imag.
 */
struct ComplexoStruct add(struct ComplexoStruct x, \
    struct ComplexoStruct y);

/*
 * Multiplicação 2 números complexos
 * usando: (rx + ix) * (ry + iy) =
 *          rx * ry + rx * ix + ix * ry + ix * iy
 */
struct ComplexoStruct mult(struct ComplexoStruct x, \
    struct ComplexoStruct y);

#endif
```

Aonde fica o código?!

Para cada .h criamos um .c

```c
#include <stdlib.h>
#include "complexo.h"

struct ComplexoStruct add(struct ComplexoStruct x, \
     struct ComplexoStruct y) {

  struct ComplexoStruct resultado = {.real=0.0, .imaginario=0.0};
  resultado.real = x.real + y.real;
  resultado.real = x.imaginario + y.imaginario;
  return resultado;

}

struct ComplexoStruct mult(struct ComplexoStruct x, \
     struct ComplexoStruct y) {

  struct ComplexoStruct resultado;
  //... código aqui
  return resultado;
}
```

O main

- Inclui cada módulo

- Uso de "" ao invés de <>
  - Módulos com caminho relativo ao código
  - Se for linkar de outros locais, usamos <>

- Um include para cada módulo

```c
#include "complexo.h"
#include "meu_math.h"

int main(void) {
   //..código aqui
   return 0;
}
```

Compilando

- gcc main.c meu_math.c complexo.c
- Não precisamos inserir os .h
- [Espero] O codeblocks deve cuidar disso para vocês
- Um bom Makefile resolve tudo

```c
#include "complexo.h"
#include "meu_math.h"

int main(void) {
   //..código aqui
   return 0;
}
```

Compilando

- Um bom Makefile resolve tudo
- Makefile?
  - Arquivo com definições para compilar um programa
  - Usa o comando make
  - Linha de comando
- Exemplo de um make simples ao
- Dentro da pasta do projeto bastar teclar make

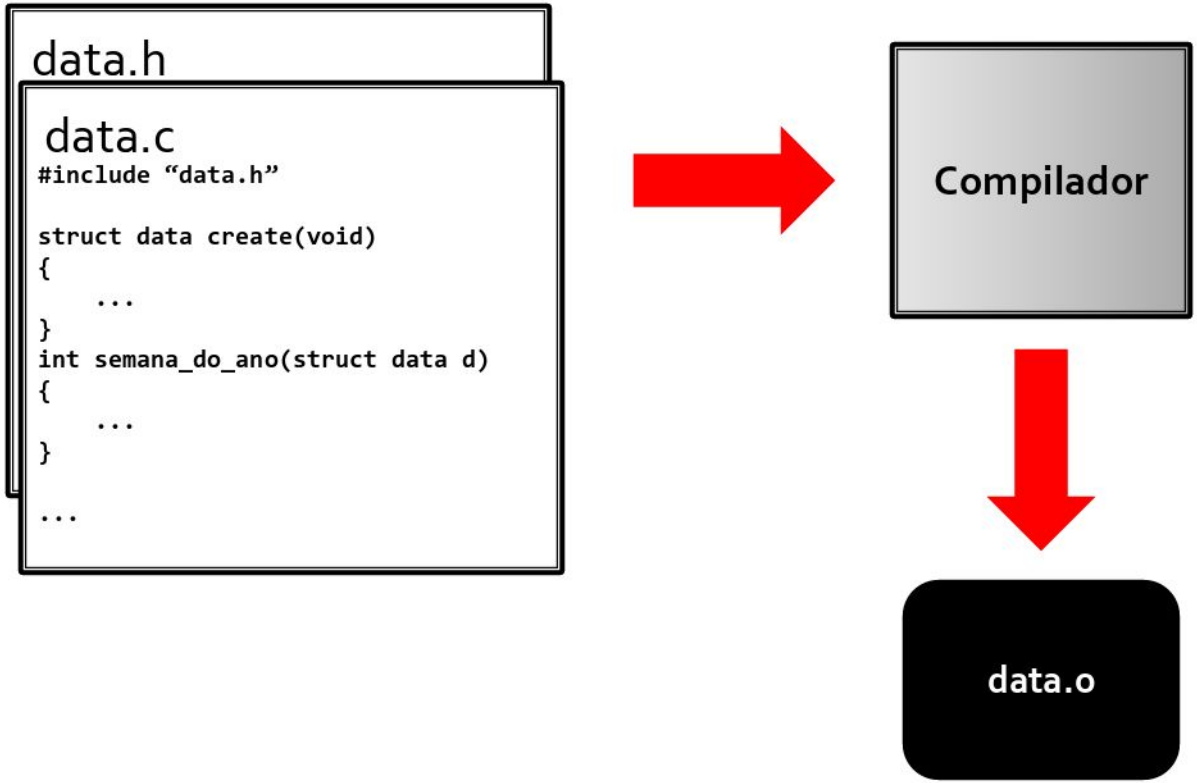
```
CC=gcc
CFLAGS=-Wall

all: meu_math.o complexo.o
    gcc main.c -o main
```

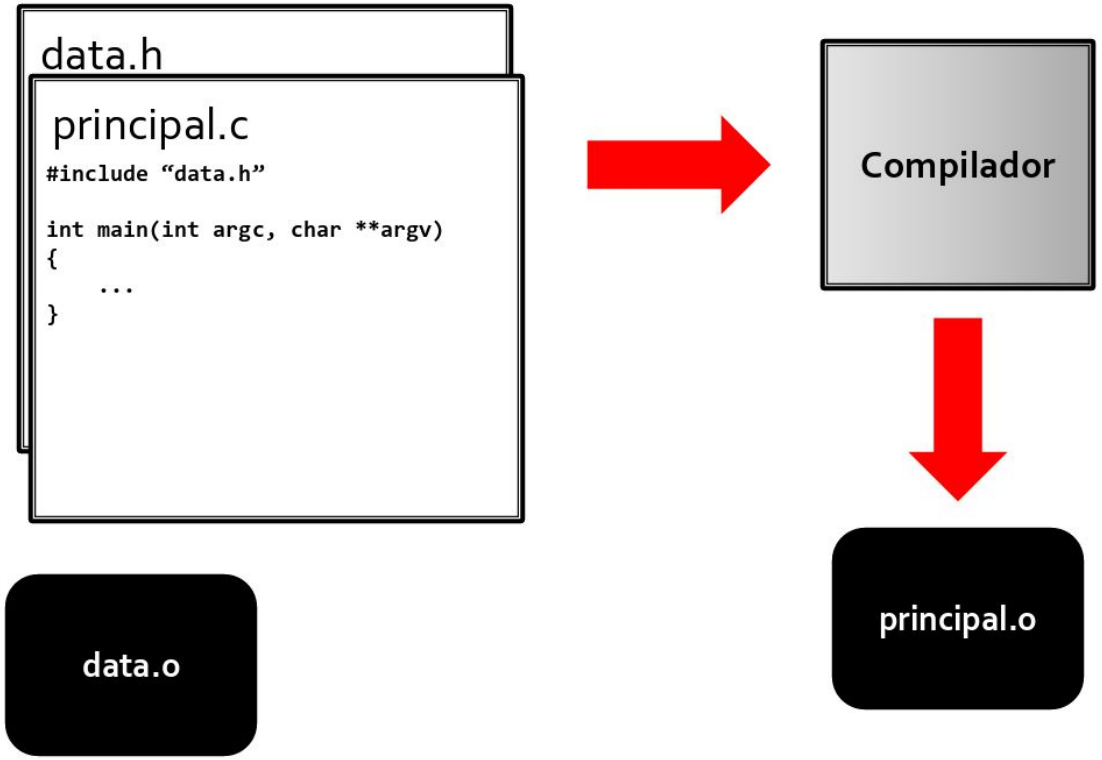
Observações

- Modularize seu código para organizar o mesmo
- Criar bibliotecas que serão utilizadas por outros códigos
- Somos preguiçosos:
  - Re-utilizamos por bibliotecas é um conceito importante em programação
  - Por isso linguagens têm bibliotecas padrão
  - Nosso exemplo de math já existe no math.h por exemplo
- Aonde fica o código da biblioteca padrão?
  - Veja a pasta do seu MinGW (lib e include)
  - /lib, /include. /usr/lib, /usr/include no linux

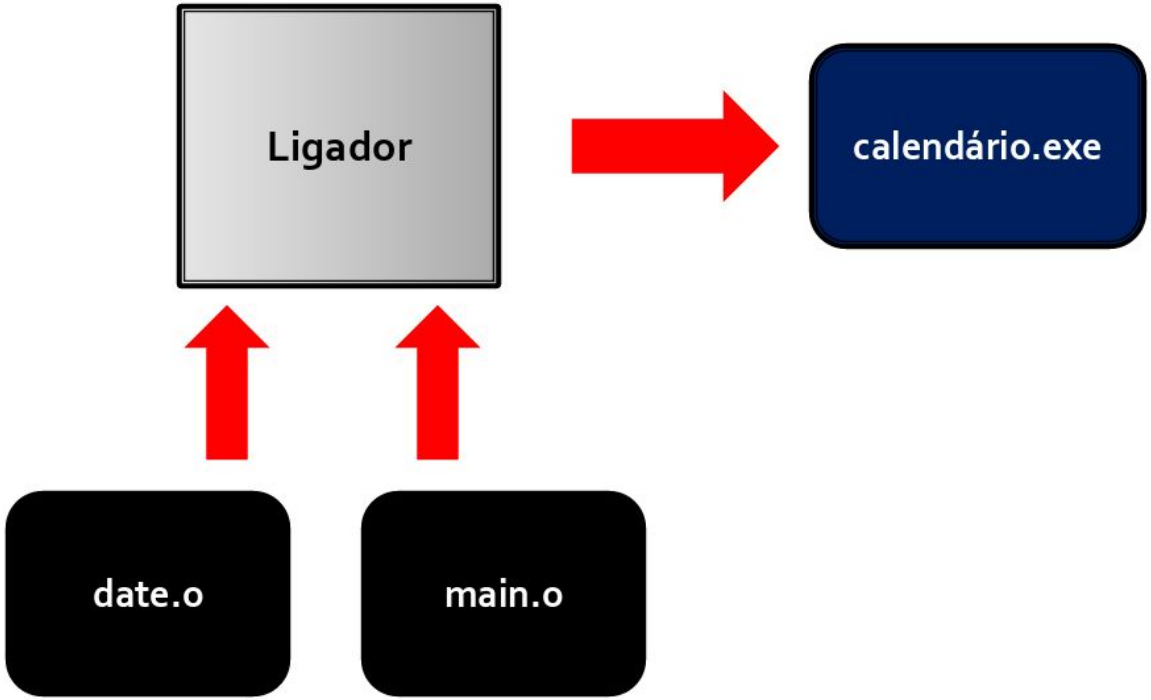
Compilação


data.h
data.c
#include “data.h”
struct data create(void)
{
...
}
int semana_do_ano(struct data d)
{
...
}
...
Compilador
data.o

Compilação


data.h
principal.c
#include “data.h”
int main(int argc, char **argv)
{
...
}
Compilador
data.o
principal.o

Compilação



Tipos Abstrato de Dados

Algoritmos e Estruturas de Dados

- Algoritmo:
  - Sequência de passos/ações
  - Trabalham em cima das estruturas de dados

- Estruturas de dados:
  - Abstração de uma situação real
  - “Dão suporte” aos algoritmos

Estruturas de Dados

- Dados podem ser representados de diversas formas

- A forma que representamos os dados é guiado pela operações que vamos fazer em cima deles
  - Quais operações fazemos em números complexos?
  - Falta alguma coisa no nosso struct de exemplo?

Como podemos representar o tempo/datas?

Tempo

- Datas:
  - Dia
  - Mês
  - Ano
  - Hora
  - Minuto
  - Segundo
  - Fuso

- Intervalos de tempo (deltas):
  - Doubles
    - Quantos segundos passaram

- https://github.com/flaviovdf/AEDS2-2017-1/tree/master/exemplos/tads

Encapsulamento

- Encapsulamento é um conceito importante em TADs e modularização

- Um TAD junto com um módulo bem documentado
  - Não preciso saber do código
  - Vocês não leem o código de <stdio.h>
    - Até podem

- Usuário:
  - Enxerga a interface
  - Structs e Funções
  - Não se preocupa, em primeiro momento, como é o TAD por baixo

Contrato

- TADs são contratos

- Tal função vai realizar tais operações em cima dos dados
  - Geralmente structs
  - Mallocs
  - Vetores

- O usuário lê o contrato e entende

- TADs andam juntos de módulos
  - Um TAD é um bom candidato para virar um módulo

Isolamento e Reuso

- TADs (e módulos) bem feitos
  - São re-utilizados
  - Não preciso fazer tudo do 0

- Isolam o comportamento
  - Em um bom programa podemos resolver o bug localmente
    - No código do TAD

  - Isto é, o TAD é bem isolado do resto do código
  - Um bug no meu TAD não deveria afetar o meu main

- Bons TADS vão fazer parte de diversos programas seus

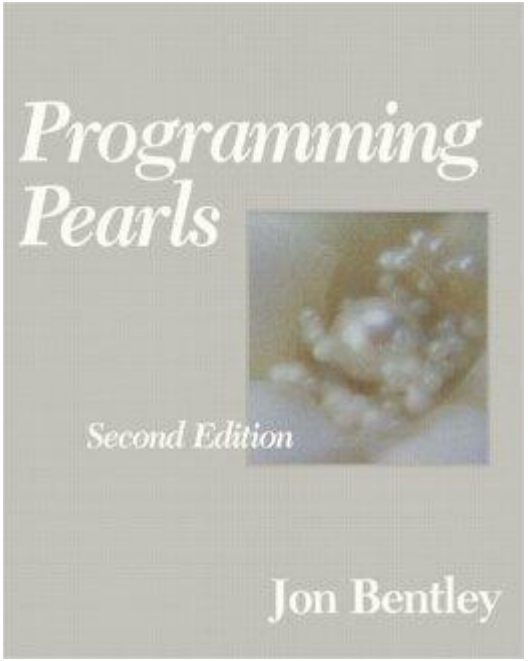
Os dados guiam o seu código

Do Capítulo 3

1. Encapsule estruturas complexas:
    a. Se você precisa de algo sofisticado abstraia

2. Deixe os dados estruturar o programa:
    a. Código complicado fica mais simples com estruturas boas (TADs)

    b. Pense bem em como representar seus dados



Em C

- TADs são implementados com
  - Definições de Tipo
    - Structs
    - Vetores
    - ...
  - Funções que operam em tais definições
    - Cabeçalhos
    - Funções
- Java, Python, Go
  - Interfaces
  - Classes
  - Assunto de outras matérias

Cuidado com os Typedefs

- Não exagere
- Tipos complexos viram structs e typedefs
- Tipos simples como int, double, vetores
  - Podem virar como o time_delta_t
  - Tem que ter um uso claro

```
typedef struct {
   char nome[48];
   int matricula;
   char conceito;
} Aluno;
typedef struct {
   int dia;
   int mes;
   int ano;
} data_t;
```

```
// Utilidade duvidosa:
typedef int[10] Vetor;
typedef char[48] Nome;
```

Modularização

- TADs são bons candidatos para virar módulos
- [Relembrando] Código no .c
- [Relembrando] Cabeçalho no .h
- [Se for bem feito] Usuários precisam conhecer apenas o cabeçalho
    - Trabalho de vocês garantir isto

Exemplo

- Vamos pensar em um TAD para uma conta bancária

- O Banco Aeds é simples.
  - Cada conta tem:
    - Número da conta
    - Saldo em conta

- Queremos implementar as operações:
  - Criar uma nova conta
  - Depositar um valor
  - Sacar um valor
  - Imprimir o saldo

ContaBancaria.h

- Perguntas:
  - Quais as vantagens de utilizar ponteiros?
  - Inicializa recebe **
    - Qual o motivo?
- Podemos definir o cabeçalho sem nomes das variáveis

```
#ifndef CONTA_BANCARIA_H
#define CONTA_BANCARIA_H

// definição do tipo
typedef struct {
    int numero;
    double saldo;
} ContaBancaria;

// cabeçalho das funções
ContaBancaria* NovaConta(int, double);
void Deposito(ContaBancaria*, double);
void Saque(ContaBancaria*, double);
void Imprime(ContaBancaria*);

#endif
```

ContaBancaria.c

- Uso de ponteiros ajuda
- Atualizar valores sem cópias

```c
#include <stdio.h>
#include "ContaBancaria.h"

ContaBancaria *NovaConta(int num, double saldo) {
    ContaBancaria *conta = malloc(sizeof(ContaBancaria));
    conta->numero = num;
    conta->saldo = saldo;
    return conta;
}

void Deposito(ContaBancaria *conta, double valor) {
    conta->saldo += valor;
}

void Saque(ContaBancaria *conta, double valor) {
    conta->saldo -= valor;
}

void Imprime(ContaBancaria *conta) {
    printf("Numero: %d\n", conta->numero);
    printf("Saldo: %f\n", conta->saldo);
}
```

Main.c

https://goo.gl/0FAK2a

Passo a passo no CTutor

Sem módulos
Limitação do CTutor

```c
#include <stdio.h>
#include <stdlib.h>
#include "ContaBancaria.h"

int main(void) {
    ContaBancaria* conta1 = NovaConta(918556, 300.00);

    printf("\nAntes da movimentacao:\n ");
    Imprime(conta1);

    Deposito(conta1, 50.00);
    Saque(conta1, 70.00);

    printf("\nDepois da movimentacao:\n ");
    Imprime(conta1);

    return 0;
}
```