Estruturas de Dados Avançadas (INF1010)

Introdução

Sobre o Curso

Objetivo do Curso

- apresentar estruturas de dados de uso comum
- estratégias básicas de organização de dados na construção de algoritmos

Programa do Curso

- Parte 1: árvores binárias de busca, árvores AVL, custo computacional
- Parte 2: árvores rubro-negras, árvores B, mapas de bits, heaps
- Parte 3: tabelas de dispersão (hash), grafos

Material Básico de Referência

Introdução a Estruturas de Dados – com técnicas de programação em C (2016)

Waldemar Celes, Renato Cerqueira, José Lucas Rangel

Data Structures and Program Design in C (2007)

Robert Kruse, C.L.Tondo, Bruce Leung, Shashi Mogalla

Critério de Avaliação

Cada grau é a média geométrica de uma prova (peso 2) e exercícios de laboratório

$$G_n = \sqrt[3]{(P_n^2 \times L_n)}$$

Média = (G1 + G2 + G3) / 3

- se **G1, G2 e G3 ≥ 3.0** e **M ≥ 5.0**, Média é a nota final (NF)
- caso contrário, o aluno faz G4
  - se **G4 ≥ 3.0** então **NF = (Gn + Gm + G4) / 3**
  - senão **NF = (G1 + G2 + G3 + G4*3) / 6**

Geração de um Executável

code/intro/hello.c

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("hello, world\n");
6 }
```

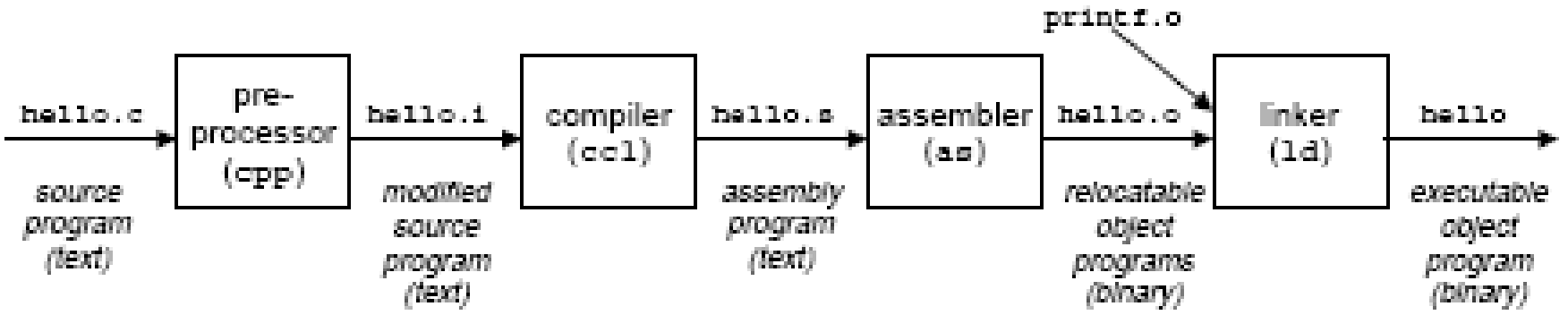
code/intro/hello.c

O programa **fonte** (arquivo texto) deve ser "traduzido" para uma sequência de instruções de linguagem de máquina, que é armazenada em um arquivo binário (**executável**)

➡ essa tradução é realizada em 4 passos

Passos para a geração do executável

`gcc -o hello hello.c`



- Passo 1: pré-processamento
- Passo 2: compilação
- Passo 3: montagem
- Passo 4: linkedição (amarração, ligação)

Compilação em separado

Um programa C pode ser dividido em vários arquivos

- módulos: “agrupamento” de funções afins
- cada módulo é compilado separadamente, gerando um objeto
- o ligador é responsável por unir os objetos, gerando o executável

Interfaces de módulos

- um módulo que utilize funções de outro módulo precisa conhecer os “protótipos” dessas funções
- cada módulo costuma ter associado um arquivo que define sua “interface”: protótipos, definições, tipos de dados
- arquivos de “cabeçalho” (*header*): extensão **.h**

Estruturas e Tipos Abstratos de Dados

Estruturas de Dados

- estratégias básicas de organização de dados
- arrays (vetores), listas encadeadas, árvores, ...

Tipos Abstratos de Dados

- conjuntos de valores e operações sobre esses valores
- a representação (**implementação**) é “abstrata”
  - “usuário” conhece apenas a **interface** do TAD
  - tipicamente baseada em uma estrutura de dados específica
  - diferentes implementações são possíves: tradeoffs

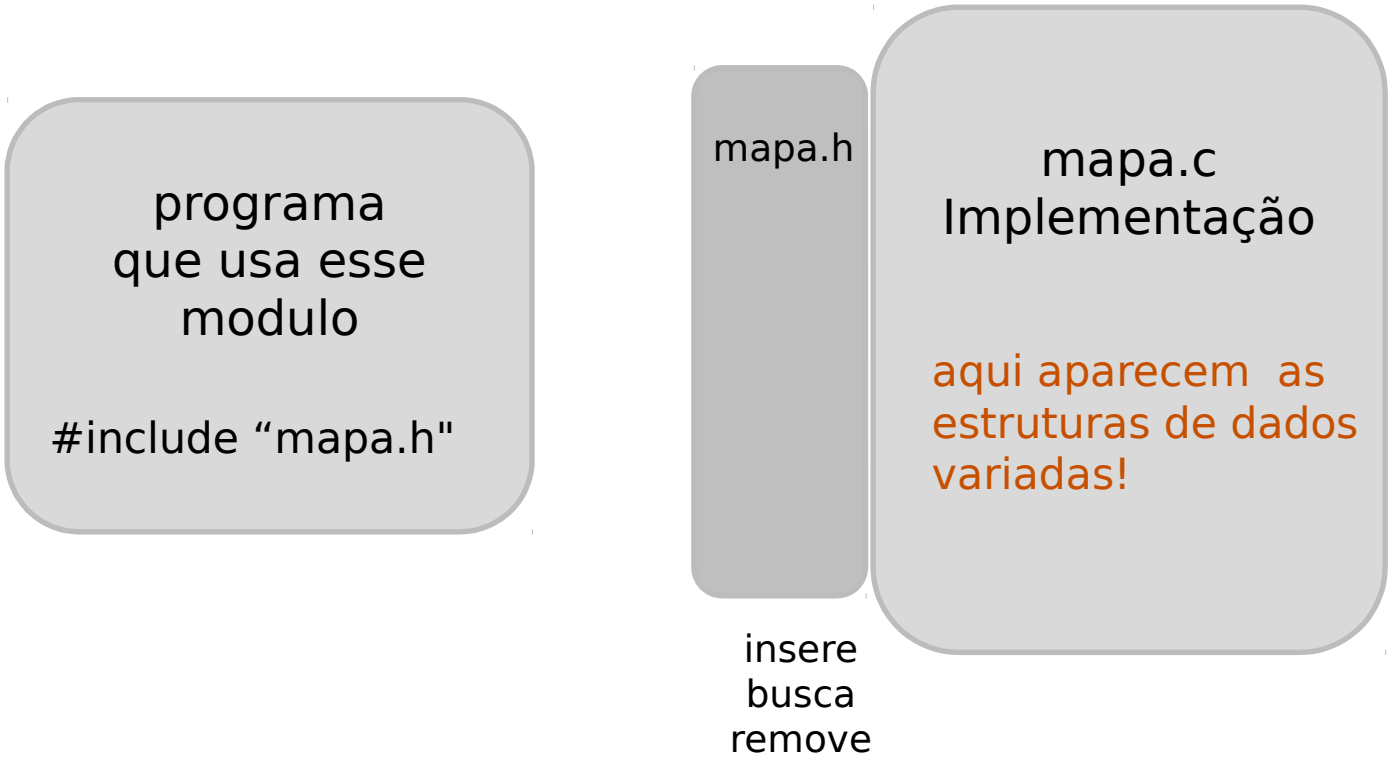
TAD Mapa

Coleção de pares chave x valor

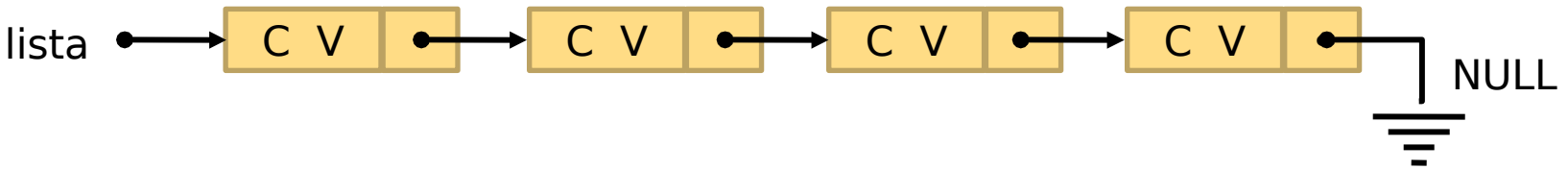
- chaves são únicas
- operações típicas: adição, remoção, busca

```
typedef struct smapa Mapa;

Mapa* cria (void);

Mapa* insere (Mapa* m, TC chave, TV valor);
TV    busca (Mapa* m, TC chave);
Mapa* retira (Mapa* m, TC chave);

void  destroi (Mapa* m);
```

Implementações de Mapa

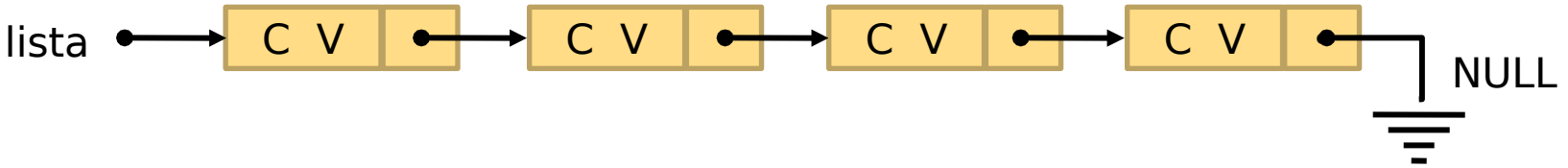


Implementando Mapa com Lista




```
typedef struct smapa Mapa;
```

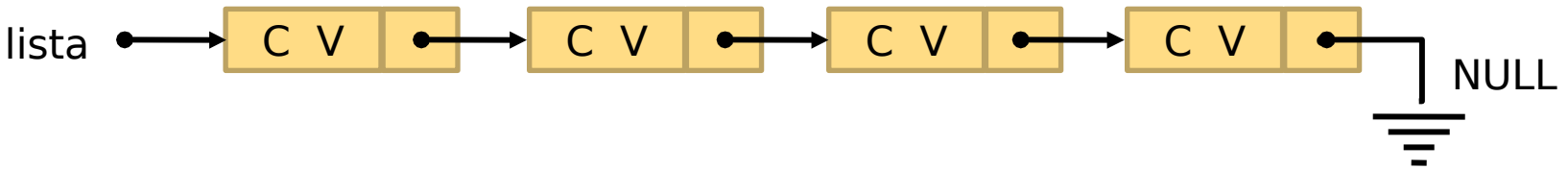
Implementando Mapa com Lista




```
typedef struct smapa Mapa;
```

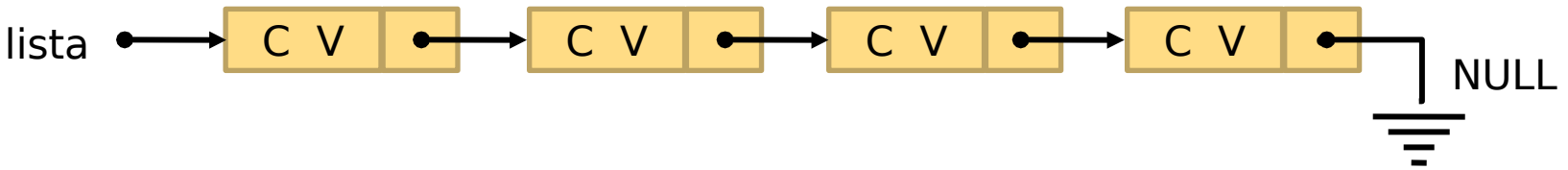
```
#include <stdlib.h>
#include "mapa.h"

struct smapa {
  int chave;
  int valor;
  struct smapa *prox;
};
```

Implementando Mapa com Lista



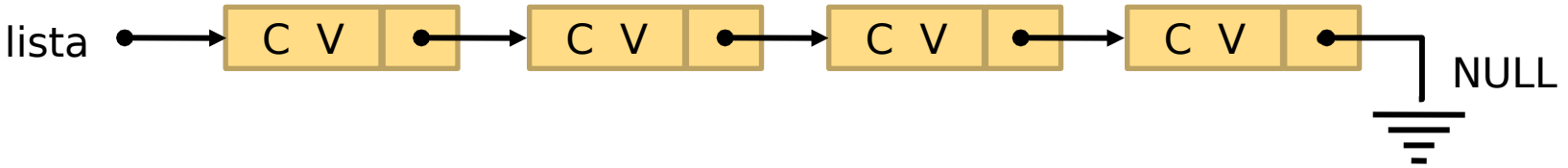

```
Mapa* cria(void) {
  return NULL;
}

Mapa* insere (Mapa* m, int chave, int valor) {
  Mapa *novo = (Mapa*) malloc(sizeof(Mapa));
  if (novo != NULL) {




  }
  return m;
}
```

Implementando Mapa com Lista




```
Mapa* cria(void) {
  return NULL;
}

Mapa* insere (Mapa* m, int chave, int valor) {
  Mapa *novo = (Mapa*) malloc(sizeof(Mapa));
  if (novo != NULL) {
    novo->chave = chave;
    novo->valor = valor;
    novo->prox = m;
    return novo;
  }
  return m;
}
```

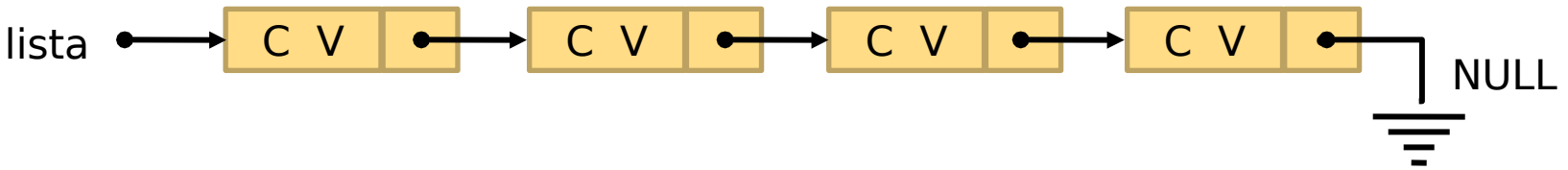
Implementando Mapa com Lista




```
int busca (Mapa* m, int chave) {
  while (m) {




  }
  return INT_MIN;
}
```

limits.h

Implementando Mapa com Lista




```
int busca (Mapa* m, int chave) {
  while (m) {
    if (m->chave == chave) {
      return m->valor;
    }
    m = m->prox;
  }
  return INT_MIN;
}
```

limits.h