

The background of the slide is a grayscale image of a circuit board. It features a complex network of black lines representing traces, with several circular pads and vias. The pattern is symmetrical and occupies the entire frame, with a dark horizontal band across the middle where the text is located.

# Estruturas Genéricas em C

Disciplina: Laboratório de Programação II

Professor Luciano Brum  
Email: [lucianobrum18@gmail.com](mailto:lucianobrum18@gmail.com)

**Assunto da aula de hoje:**

# **Estruturas Genéricas**

# Tópicos

- Motivação.
- Objetivos.
- Funções de um TAD genérico e cliente de um TAD Genérico.
- Ponteiro Genérico.
- Lista Genérica.
- Callback.

# Estruturas Genéricas: Motivação

- Estruturas que foram vistas até agora são específicas para o tipo de informação que manipulam.
- Por exemplo: listas encadeadas de inteiros, de caracteres e de estruturas compostas.
- Para manipular cada um destes tipos, algumas funções do TAD devem ser reimplementadas.

# Estruturas Genéricas: Motivação

- Exemplo 1, a função Pertence:
  - Lista de caracteres (compara caracteres).
  - Lista de inteiros (compara inteiros).
- Exemplo 2, a função Imprime:
  - Lista de caracteres (imprime caracter: “%c”).
  - Lista de inteiros (imprime inteiro: “%d”).

# Estruturas Genéricas: Objetivos

- TAD de tipo Genérico: Uma estrutura genérica deve armazenar qualquer tipo de informação.
- Para isso, um TAD genérico deve desconhecer o tipo da informação.
- As funções do TAD genérico não podem manipular diretamente as informações.
- As funções são responsáveis pela manutenção e encadeamento das informações na estrutura.

# Estruturas Genéricas: Cliente do TAD genérico

- O cliente de um TAD Genérico fica responsável pelas operações que envolvem acesso direto à informação.
- O cliente de um TAD Genérico deve também converter o ponteiro genérico para um ponteiro específico para acessar a informação

# Estruturas Genéricas: Cliente do TAD genérico

- Exemplo: o cliente do TAD lista genérica:
  - Se o cliente deseja manipular inteiros, precisa implementar operações para manipular inteiros.
  - Se o cliente deseja manipular caracteres, precisa implementar operações para manipular caracteres.



# Estruturas Genéricas: Ponteiro Genérico

- O `void*` é um ponteiro genérico que pode armazenar qualquer tipo de informação:
  - Informações de tipos primitivos: `int`, `char`, `float`,...;
  - Informações de tipos estruturados;
  - Informações de ponteiros;

# Estruturas Genéricas: Ponteiro Genérico

- Porém, para acessar a informação do ponteiro void\*, devemos obrigatoriamente fazer a operação CAST para convertermos a informação.

- Ex:

```
void *pp;
```

```
int p2= 10;
```

```
pp = &p2;
```

```
printf("conteúdo: %d\n", *pp); //INCORRETO
```

```
printf("conteúdo: %d\n", *(int*)pp);//CORRETO
```

# Estruturas Genéricas: Lista Genérica

- Uma célula da lista genérica guarda um ponteiro para informação que é genérico (void\*).

```
struct listagen{  
    void* info;  
    struct listagen* prox;  
};  
typedef struct listagen ListaGen;
```

# Estruturas Genéricas: Lista Genérica

- As funções do TAD lista que não manipulam informações (cria lista, verifica se está vazia) são implementadas da mesma maneira.
- Teremos funções que manipulam informações como objeto opaco:
  - Ex: Função que insere uma nova célula na lista;
  - Opaco porque não precisa acessar as informações da lista.

```
ListaGen* lgen_inserere (ListaGen* l, void* p) {  
    ListaGen* n = (ListaGen*) malloc(sizeof(ListaGen));  
    n->info = p;  
    n->prox = l;  
    return n; }
```

# Estruturas Genéricas: Callbacks

- Estratégia:
  - Separação entre:
    - Função que percorre os elementos da estrutura de dados;
    - Operação realizada sobre cada elemento;
  - A função de percorrimento é única e pode ser usada para vários fins;
  - Operação a ser executada é passada como parâmetro para a função de percorrimento;

# Estruturas Genéricas: Callbacks

- Callback:
  - É a Função do cliente (quem usa a função de percorrimento).
  - A Função “callback” é uma função do cliente que é “chamada de volta” para cada elemento encontrado na estrutura de dados.
  - Usualmente recebe como parâmetro a informação associada ao elemento encontrado na estrutura.

# Estruturas Genéricas: Callbacks

- Ponteiro para função:
  - Nome de uma função representa o endereço da função.
- Exemplo:
  - Assinatura da função callback: `void callback (void* info);`
  - Declaração de variável ponteiro para armazenar o endereço da função:  
`void (*cb) (void*);`
  - `cb`: variável do tipo ponteiro para funções com a mesma assinatura da função callback;

# Estruturas Genéricas: Callbacks

- Exemplo: uma função genérica para percorrer elementos da lista:

```
void lgen_percorre(ListaGen* l, void (*cb)(void*)){\n    ListaGen *p;\n    for(p=l;p!=NULL;p=p->prox){\n        cb(p->info);\n    }\n}
```

Parâmetros da  
função do cliente.

Variável ponteiro  
que armazena o  
endereço da função.

Chamada à função  
cliente, passando como  
parâmetro a informação.

Tipo de retorno da  
função.



- TAD de tipo genérico:

```
ListaGen* lgen_inserere (ListaGen* l, void* p) {  
    ListaGen* n = (ListaGen*) malloc(sizeof(ListaGen));  
    n->info = p;  
    n->prox = l;  
    return n; }
```

```
struct ponto{  
    float x, y;  
};  
typedef struct ponto Ponto;
```

- Cliente do TAD:

```
static ListaGen* insere_ponto(ListaGen* l, float x, float y){  
    Ponto *p = (Ponto*)malloc(sizeof(Ponto));  
    p->x = x;  
    p->y = y;  
    return lgen_inserere(l,p);  
}
```

- Para imprimir pontos, podemos fazer uso da função *percorre* vista anteriormente:
- TAD de tipo genérico:

```
void lgen_percorre(ListaGen* l, void (*cb)(void*)) {
    ListaGen *p;
    for(p=l; p!=NULL; p=p->prox) {
        cb(p->info);
    }
}
```

```
struct ponto {
    float x, y;
};
typedef struct ponto Ponto;
```

- Cliente do TAD:

```
static void imprime(void* info) {
    Ponto *p = (Ponto*)info;
    printf("%f %f\n", p->x, p->y);
}
```

- No programa principal, para chamar a função:

```
lgen_percorre(l, imprime);
```

- Para descobrir o centro geométrico dos pontos, podemos usar a função **percorrer**:

- TAD de tipo genérico:

```
void lgen_percorre(ListaGen* l, void (*cb)(void*)){\n    ListaGen *p;\n    for(p=l;p!=NULL;p=p->prox){\n        cb(p->info);\n    }\n}
```

```
struct ponto{\n    float x, y;\n};\ntypedef struct ponto Ponto;
```

- Cliente do TAD:

```
static void centro_geom(void* info){\n    Ponto *p = (Ponto*)info;\n    CG.x = CG.x + p->x;\n    CG.y = CG.y + p->y;\n    NP++;}
```

Variáveis Globais!!

Variáveis Globais!!

- No programa principal:

```
NP = 0;\nCG.x = CG.y = 0.0f;\nlgen_percorre(l,imprime);\nCG.x /= NP;\nCG.y /= NP;
```

# Estruturas Genéricas: Callbacks

- É possível implementar o callback sem o uso de variáveis globais? SIM.
- **Vamos passar dados para nossa função callback** e alterar a função percorre.
- Também vamos definir uma estrutura nova para armazenar as informações necessárias para calcular o centro geométrico.

# Estruturas Genéricas: Callbacks

```
void lgen_percorre(ListaGen* l, void (*cb)(void*,void*),void* dado){  
    ListaGen *p;  
    for(p=l;p!=NULL;p=p->prox){  
        cb(p->info, dado);  
    }  
}
```

```
struct ponto{  
    float x, y;  
};  
typedef struct ponto Ponto;  
struct cg{  
    int n;  
    Ponto p;  
};  
typedef struct cg CG;
```

```
static void centro_geom(void* info, void* dado){  
    Ponto *p = (Ponto*)info;  
    CG *cg = (CG*)dado;  
    cg->p.x = cg->p.x + p->x;  
    cg->p.y = cg->p.y + p->y;  
    cg->n++;}
```

- No programa principal, para chamar a função:

```
...  
CG cg = {0,{0.0f,0.0f}};  
lgen_percorre(l,centro_geom,&cg);  
cg.p.x = cg.p.x / cg.n;  
cg.p.y = cg.p.y / cg.n;  
...
```

# Estruturas Genéricas: Callbacks

- Agora vamos analisar um exemplo de retorno de valores de uma callback:
- Exemplo: verificar a ocorrência de um determinado ponto de coordenadas (x,y) na estrutura da lista.

```
void lgen_percorre(ListaGen* l, void (*cb)(void*,void*),void* dado){  
    ListaGen *p;  
    for(p=l;p!=NULL;p=p->prox){  
        cb(p->info, dado);  
    }  
}
```

**Qual seria o problema desta implementação ?**

**Mesmo encontrando o ponto, a busca continua.**

# Estruturas Genéricas: Callbacks

- Vamos modificar nossa função de percorre, fazendo que a call-back retorne 0 se não encontrou e 1 se encontrou o ponto.
- Portanto, a assinatura da nossa função percorre também mudará, pois agora ela tem retorno: 0 se não houve interrupção do cliente e != de zero se houve.

# Estruturas Genéricas: Callbacks

```
int lgen_percorre(ListaGen* l, int (*cb)(void*,void*),void* dado){
    ListaGen *p;
    for(p=l;p!=NULL;p=p->prox){
        int r = cb(p->info, dado);
        if (r!=0){
            return r;
        }
    }
    return 0;
}

static int igualdade(void* info, void* dado){
    Ponto *p = (Ponto*)info;
    Ponto *q = (Ponto*)dado;
    if(fabs(p->x-q->x)<TOL && p->y-q->y<TOL)
        return 1;
    else
        return 0;
}

static int pertence(Listagen* l, float x, float y){
    Ponto q;
    q.x = x; q.y = y;
    return lgen_percorre(l,igualdade,&q);
}
```



# Estruturas Genéricas: Resumo

## Técnicas de programação utilizando callbacks e ponteiros genéricos:

- Muito empregadas em programação;
- Permitem esconder do cliente a forma como os elementos armazenados estão estruturados internamente;
- Cliente pode visitar e manipular todas as informações armazenadas, independentemente da estrutura de dados utilizada.

# Exercício

- Implemente todas as funções básicas de um TAD de lista genérica:
  - Inserção;
  - Remoção;
  - Pertence (usando a função percorre);
  - Cria;
  - Libera;
  - Imprime (usando a função percorre);
- Considere que os dados sejam do tipo **Ponto**, com atributos **x** e **y** do tipo **int**.

# Bibliografia

- CELES, W.; CERQUEIRA, R.; RANGEL, J. L. **Introdução a Estruturas de Dados**. 1ª. Ed. Rio de Janeiro: Campus, 2004.
- **Slides baseados no material da PUC-Rio**. Disponível em: <http://www.inf.puc-rio.br/~inf1620/material/slides/capitulo14.PDF> . Acesso em: 23/02/2017.

# Dúvidas?

Professor Luciano Brum  
email: [lucianobrum18@gmail.com](mailto:lucianobrum18@gmail.com)  
<https://sites.google.com/view/brumluciano>