

Estruturas de Dados

Módulo 14 - Estruturas genéricas

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO



Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel,
Introdução a Estruturas de Dados, Editora Campus
(2004)

Capítulo 14 – Estruturas genéricas

Tópicos

- Introdução
- Lista genérica
- Uso de callbacks
 - Um exemplo de cliente
 - Passando dados para a callback
 - Retornando valores de callbacks

Introdução

- Estrutura de dados genérica
 - armazena qualquer tipo de informação
- TAD de tipo genérico
 - desconhece a natureza das informações associadas aos elementos da estrutura
 - responsável apenas pela manutenção da estrutura
 - guarda ponteiro genérico para a informação (do tipo void*)
 - não pode acessar a memória através do ponteiro
- Cliente de um TAD de tipo genérico
 - responsável pelas operações sobre as informações associadas aos elementos da estrutura
 - converte o ponteiro genérico em um ponteiro específico e acessa a informação

Introdução

- Implementação de funções opacas:
 - não precisam acessar a informação
 - implementação semelhante aos casos já discutidos
- Implementação de funções não opacas:
 - TAD deve prover uma função genérica para percorrer os elementos da estrutura
 - para cada elemento visitado, TAD implementa um mecanismo para chamar o cliente passando a informação associada
 - cliente processa a informação

Lista genérica

- Lista genérica:
 - estrutura do elemento de uma lista genérica
 - guarda ponteiro para a informação e ponteiro para o próximo nó

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

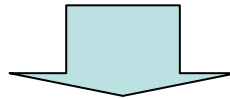
Lista genérica

- função lgen_inserere
 - manipula informação como um objeto opaco (não precisa acessar a informação)
 - cliente é responsável por passar para a função o ponteiro da informação que será armazenada nesse novo nó

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

Uso de callbacks

- Estratégia:
 - separação entre:
 - função que percorre os elementos da estrutura de dados
 - operação realizada sobre cada elemento



- 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

Uso de callbacks

- Callback:
 - função do cliente (quem usa a função de percorrimento)
 - função “chamada de volta” a cada elemento encontrado na estrutura de dados
 - usualmente recebe como parâmetro a informação associada ao elemento encontrado na estrutura

Uso de 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

Uso de callbacks

- Exemplo - função genérica para percorrer os elementos da lista
 - para cada elemento visitado, chama a função do cliente passando como parâmetro a informação associada

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

Uso de callbacks - Um exemplo de cliente

- Aplicação cliente:
 - armazena pontos (x,y) em uma lista genérica
 - usa tipo Ponto

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

Uso de callbacks - Um exemplo de cliente

- Inserção de pontos na lista genérica:
 - cliente aloca dinamicamente uma estrutura do tipo Ponto
 - passa seu ponteiro para a função de inserção
 - cliente implementa função auxiliar para inserir pontos (x,y) na estrutura da lista genérica

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

Uso de callbacks - Um exemplo de cliente

- Callback para cálculo do centro geométrico dos pontos armazenados na lista:

– atualiza variáveis globais a cada chamada da *callback*:

NP	tipo int	representa o número de elementos visitados
CG	tipo Ponto	representa o somatório das coordenadas

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

Uso de callbacks - Um exemplo de cliente

- Cálculo do centro geométrico dos pontos pelo cliente:

```
...  
NP = 0;  
CG.x = CG.y = 0.0f;  
lgen_percorre(l,centro_geom);  
CG.x /= NP;  
CG.y /= NP;  
...
```

Uso de callbacks

Passando dados para a callback

- Mecanismo para transferir dados do cliente para a função *callback*:
 - utiliza parâmetros da callback:
 - informação do elemento sendo visitado
 - ponteiro genérico com um dado qualquer
 - cliente chama a função de percorrimento passando como parâmetros
 - a função *callback*
 - o ponteiro a ser repassado para a *callback* a cada elemento visitado

Uso de callbacks

Passando dados para a callback

- Exemplo - função genérica para percorrer os elementos da lista
 - utiliza assinatura da função *callback* com dois parâmetros

```
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);
    }
}
```

Uso de callbacks

Passando dados para a callback

- Exemplo - função para calcular o centro geométrico dos pontos, sem usar variáveis globais
 - passo 1: criação de um tipo que agrupa os dados para calcular o centro geométrico:
 - número de pontos
 - coordenadas acumuladas

```
struct cg
{
    int n;
    Ponto p;
};

typedef struct cg Cg;
```

Uso de callbacks

Passando dados para a callback

- Exemplo - função para calcular o centro geométrico dos pontos, sem usar variáveis globais
 - passo 2: re-definição da *callback* para receber um ponteiro para um tipo Cg que representa a estrutura

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

Uso de callbacks

Passando dados para a callback

- Exemplo - chamada por parte do cliente

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

Uso de callbacks

Retornando valores de callbacks

- Mecanismo para permitir ao cliente interromper a visitação aos elementos
 - utiliza função callback com valor de retorno
 - =0 função deve prosseguir visitando o próximo elemento
 - ≠0 função deve interromper a visitação aos elementos

Uso de callbacks

Retornando valores de callbacks

- Exemplo - função genérica para percorrer uma lista
 - passa a ter um valor de retorno:
 - =0 função deve prosseguir visitando o próximo elemento
 - ≠0 função deve interromper a visitação aos elementos

```
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;
}
```

Uso de callbacks

Retornando valores de callbacks

- Exemplo - função *callback* igualdade
 - recebe como entrada as coordenadas do ponto a pesquisar
 - retorna 1, se o ponto visitado tem as coordenadas do ponto a pesquisar (dentro de um intervalo de tolerância $TOL=1e-5$)

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

Uso de callbacks

Retornando valores de callbacks

- Exemplo - função do cliente para verificar a ocorrência das coordenadas (x,y) na estrutura

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

Resumo

Técnicas de programação utilizando *callbacks*

- 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