Estrutura de Dados I

Luciana Lee

Tópicos da Aula

Estruturas Genéricas - Motivação

- Estruturas que vimos até agora são específicas para o tipo de informação que manipulam
- Por exemplo, vimos listas de inteiros, de caracteres e de estruturas compostas
- Para manipular cada um destes tipos, algumas funções do TAD devem ser reimplementadas
- Por exemplo, a função Pertence
  - Lista de caracteres (compara caracteres)
  - Lista de inteiros (compara inteiros)
- Função Imprime
  - Lista de caracteres (imprime caracter: “%c”)
  - Lista de inteiros (imprime inteiro: “%d”)

Objetivos

- Uma estrutura genérica deve ser capaz de 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

Cliente do TAD

- O cliente de um TAD Genérico fica responsável pelas operações que envolvem acesso direto à informação
- Por 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

Estrutura de uma Lista Genérica

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

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

Funções de uma Lista Genérica

- As funções do TAD lista que não manipulam informações são implementadas da mesma maneira. Por exemplo: função para criar um elemento da lista, função para verificar se a lista está vazia
- Funções com objeto opaco:
  - Função que insere um novo elemento na lista
  - Cliente passa para função um ponteiro para a informação

Exemplos

```
ListaGen *cria_listagen (void *v) {
  ListaGen *novo = (ListaGen *)calloc(1,sizeof(ListaGen));
  if (novo == NULL) {
    printf("ERRO: nao foi possivel alocar estrutura.\n");
    exit(1);
  }
  novo->info = v;
  return novo;
}

ListaGen *insere_listagen (ListaGen *L, void *v) {
  ListaGen *novo = cria_listagen(v);
  novo->prox = L;
  return novo;
}
```

Como acessar os dados?

- Problema surge nas funções que precisam manipular as informações contidas em uma célula
  - Excluir um elemento da TAD: o cliente fica responsável por liberar as estruturas de informação.
  - Verificar se um elemento está na lista: o TAD não é capaz de comparar informações.
- Solução: TAD deve prover uma função genérica para percorrer todos os nós da estrutura.
- Precisamos ter um mecanismo que permita, a partir de uma função do TAD, chamar o cliente => **Callback ("chamada de volta")**

Callback

- Função genérica do TAD lista é a função que percorre e visita as células
- A operação específica a ser executada na célula (comparação, impressão, etc) deve ser passada como parâmetro (Ponteiro para Função!)
- O nome de uma função representa o endereço dessa função

Callback - Exemplo

- Assinatura da função de 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

Exemplos

- Função que percorre uma lista genérica.
- Entrada:
  - Ponteiro para o início da lista
  - Ponteiro da função que irá tratar o campo genérico de um elemento da lista
- Saída: `void`

```
void percorre_listagen (ListaGen *L, void (*cb) (void*)) {
  ListaGen *aux = L;
  while (aux != NULL) {
    cb(aux->info);
    aux = aux->prox;
  }
}
```

- Chamada da função pelo cliente:

```
percorre_listagen(L, imprime_int);
```

Exemplo de Aplicação Cliente

- Suponha que a aplicação cliente trabalha com pontos do plano cartesiano.

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

- Para inserir um elemento na lista genérica, o cliente deve:
  - alocar dinamicamente uma estrutura do tipo Ponto
  - Passar o ponteiro para a função de inserção

```
ListaGen *insere_ponto (ListaGen *L, float x, float y) {
  Ponto *xpto = (Ponto *)malloc(sizeof(Ponto));
  if (!xpto){ printf("Erro.\n"); exit(1); }
  xpto->x = x;
  xpto->y = y;
  return insere_listagen(L, xpto);
}
```

Exemplo de Aplicação Cliente

- Para imprimir um ponto da lista genérica
  - Cliente converte ponteiro genérico (void*) em Ponto (type cast)
  - Imprime informação

```
void imprime_ponto (void *p) {
  Ponto *xpto = (Ponto *)p;
  printf("[%.1f,%.1f]\n",xpto->x,xpto->y);
}
```

- Para imprimir todos os pontos da lista
  - Usa a função percorre_listagen
  - Passando o lista e a função para imprimir um ponto

```
percorre_listagen(L, imprime_ponto);
```

Exemplo de Aplicação Cliente

- Callback para cálculo do centro geométrico dos pontos armazenados na lista:
  - atualiza variáveis globais a cada chamada da callback:
    - *NumPontos*: tipo int representa o número de elementos visitados
    - *SomaCoord*: tipo Ponto representa o somatório das coordenadas

```
void soma_ponto (void *p) {
  Ponto *xpto = (Ponto *)p;
  SomaCoord.x += xpto->x;
  SomaCoord.y += xpto->y;
  NumPontos += 1;
}
```

Exemplo de Aplicação Cliente

- Cálculo do centro geométrico dos pontos pelo cliente:
  - Usa a função `percorre_listagen` passando o endereço da função `soma_ponto` como parâmetro

```
void centro_geometrico (ListaGen *L) {
  ListaGen *aux = L;
  NumPontos = 0;
  SomaCoord.x = 0.0;
  SomaCoord.y = 0.0;

  percorre_listagen(L,soma_ponto);

  SomaCoord.x /= NumPontos;
  SomaCoord.y /= NumPontos;
}
```

Exemplo de Aplicação Cliente

- Devemos evitar varáveis globais
  - Pode tornar o programa difícil de ler e difícil de manter
- Para evitar o uso de variáveis globais, precisamos de mecanismos que permitam passagem de informações do cliente para a função de callback
  - utiliza parâmetros da callback:
    - ★ informação do elemento sendo visitado
    - ★ ponteiro genérico com um dado qualquer
  - cliente chama a função de percorrer passando como parâmetros
    - ★ a função callback
    - ★ o ponteiro a ser repassado para a callback a cada elemento visitado

Passando dados para o callback

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

```
void percorre_listagen (ListaGen *L, void (*cb) (void*, void*),
                        void *dado) {
  ListaGen *aux = L;
  while (aux != NULL) {
    cb(aux->info,dado);
    aux = aux->prox;
  }
}
```

Passando dados para o callback

- Modificando a função para calcular o centro geométrico dos pontos (não precisa de variáveis globais)
  - passo 1: criação de um tipo que agrupa os dados para calcular o centro geométrico:
    - ★ número de pontos
    - ★ coordenadas acumuladas

```
typedef struct centroGeo {
   int numP; // numero de pontos
   Ponto p; // soma das coordenadas dos pontos
}CentroGeo;
```

Passando dados para o callback

- passo 2: re-definição da callback para receber um ponteiro para um tipo CentroGeo que representa a estrutura

```
void soma_ponto (void *p, void *dado) {
  Ponto *xpto = (Ponto *)p;
  CentroGeo *centro = (CentroGeo *)dado;
  centro->p.x += xpto->x;
  centro->p.y += xpto->y;
  centro->numP += 1;
}
```

Passando dados para o callback

- passo 3: modificando a função para calcular o centro geométrico dos pontos (não precisa de variáveis globais)

```
void centro_geometrico (ListaGen *L) {
  ListaGen *aux = L;
  CentroGeo *cg = (CentroGeo *)calloc(1,sizeof(CentroGeo));
  percorre_listagen(L,soma_ponto,cg);
  cg->p.x /= cg->numP;
  cg->p.y /= cg->numP;
  printf("centro geometrico: [%.1f,%.1f]\n",
         cg->p.x,cg->p.y);
  free(cg);
}
```

Retornando valores de callback

- Função pertence
  - Retorna 1 se o ponto existe na lista e 0 caso contrário

```
int pertence_listagen (ListaGen *L, int (*cb) (void*, void*),
  ListaGen *aux = L;
  while (aux != NULL) {
    if (cb(aux->info, proc)) return 1;
    aux = aux->prox;
  }
  return 0;
}
```

Aplicação Cliente

- A aplicação cliente deve implementar a função callback que compara dois pontos.

```
int igual (void *a, void *b) {
  Ponto *p1 = (Ponto *) a;
  Ponto *p2 = (Ponto *) b;
  return (p1->x == p2->x && p1->y == p2->y);
}
```

- A chamada no código cliente:

```
void pertence (ListaGen *L, float x, float y) {
  Ponto xpto;
  xpto.x = x;
  xpto.y = y;
  if (pertence_listagen(L, igual, &xpto))
    printf("O ponto ja existe na lista.\n");
  else printf("O ponto nao existe na lista.\n");
}
```

Dúvidas?