Estruturas de Dados

Tipos Abstratos de Dados e Formas de Alocação de Memória

Prof. Nilton Luiz Queiroz Junior

Estruturas de dados

- Estruturas de dados e algoritmos estão intimamente ligados;
  - O estudo de estrutura de dados envolve os algoritmos associados a elas;
  - A escolha dos algoritmos depende da representação e das estruturas de dados;
- As estruturas de dados são uma forma de representar abstrações da realidade no computador;

Estruturas de dados

- Programar é basicamente estruturar dados e construir algoritmos.
  - Programas são formulações concretas de algoritmos abstratos e estruturas específicas de dados;
  - Programas são uma classe especial de algoritmos que podem ser executados por computadores;

Estruturas de dados

- Para resolver um problema, é necessário escolher uma abstração da realidade, em geral mediante a definição de um conjunto de dados que representa a situação real;
- A seguir deve ser escolhida a forma de representar esses dados (estado) e as operações sobre estes(comportamento);
- A escolha da representação dos dados é determinada, entre outras, pelas operações a serem realizadas sobre os dados e pela Linguagem de Programação escolhida;

Tipos de dados

- Fundamentalmente um tipo de dado significa um conjunto de valores e uma sequência de operações para os valores;
- Os valores e as operações sobre tais valores formam uma construção que pode ser implementada tanto em software quanto em hardware;

Tipos de dados

- Os tipos de dados definem a forma como um dado deve ser armazenado ou recuperado;
- Variáveis, constantes, expressões podem assumir tipos, e funções podem gerar tipos;
    - O tipo de dado de um elemento define quais valores podem ser assumidos ou então gerados, além de como será feita cada operação sobre ele;

Tipos de dados

- Exemplos de Tipos de dados em C:
  - int: permite valores inteiros e operações de adição, multiplicação, subtração, divisão, resto de divisão, atribuição, etc...
  - float: permite valores reais e operações de adição, multiplicação, subtração, divisão, atribuição, etc...
  - ponteiros: permite valores de endereços de memória e operações de soma com valores inteiros, subtração com valores inteiros;
    - As operações de soma e subtração com ponteiros são diferentes das operações de soma e subtração entre inteiros;

Representação de dados

- Os dados podem estar representados de diferentes maneiras;
- Normalmente, a escolha da representação é determinada pelas operações que serão utilizadas sobre eles;
- Exemplo: números inteiros
  - Representação por “palitinhos”: II + IIII = IIIIII
    - Boa para pequenos números (operação simples)
  - Representação decimal: 1278 + 321 = 1599
    - Boa para números maiores (operação complexa)

Tipos abstratos de dados

- A expressão tipo abstrato de dado (TAD) refere-se ao conceito matemático que define o tipo de dado;
- São ferramentas para especificar as propriedades lógicas de um tipo de dado;
- Alguns TADs não podem ser implementados em determinado software ou hardware;

Tipos abstratos de dados

- Tipos abstratos de dados (TADs) podem ser definidos como modelos matemáticos com operações definidas sobre o modelo;
  - Não se preocupa com eficiência do tipo de dado;
    - Qual o consumo de espaço;
    - Qual o tempo consumido pelas operações;
  - A eficiência é uma questão de implementação;

Tipos abstratos de dados

- A definição de um TAD consiste em duas partes:
  - Definição de valores;
    - Quais valores são aceitos para o TAD;
  - Definição de operadores;
    - Quais operações podem ser realizadas no TAD
- Para serem usados em computadores os TADs precisam ser transformados em um Tipo de Dado Concreto (ou Tipo de dado)
  - Implementação
    - Finita;
    - Leva em conta o consumo de recursos;

Exemplo de implementação de um TAD

- Implementando um TAD ponto no R^2
  - Assim devemos definir um conjunto de operações e valores que o ponto pode receber
    - Valores:
      - Qualquer par ordenado *(x,y)* onde $x \in R$ e $y \in R$.
    - Operações:
      - cria: cria um ponto ordenado com coordenadas x e y;
      - acessa: devolve as coordenadas de um ponto;
      - atribui: atribui novas coordenadas ao ponto;
      - distancia: operação que calcula a distância entre dois pontos;

Alocação de memória

- Existem dois principais tipos de alocação de memória:
  - Alocação estática;
    - Espaço de memória alocado durante a execução de seu escopo;
    - É perdido somente quando seu escopo é destruído;
    - Todo o espaço a ser usado é alocado de uma vez;
      - Desconsidera a necessidade do programa;
  - Alocação dinâmica:
    - Espaço de memória alocado durante tempo de execução;
      - Em geral usa-se um ponteiro para manipular os dados do espaço;
    - Os espaços podem ser alocados, liberados e realocados;
    - Podem ser liberados quando o programa é encerrado ou de maneira explícita;

Ponteiros

- Ponteiros são variáveis que armazenam endereços de memória de outras variáveis;
  - Quando um ponteiro armazena um endereço de memória dizemos que ele aponta para aquele endereço;
- Eles tornam possíveis o acesso aos dados armazenados nos conteúdos por eles apontados;
- Cada tipo de dado precisa de seu tipo específico de ponteiro;

Alocação de memória

- Com o uso de alocação dinâmica de memória é possível implementar estruturas que crescem e diminuem conforme a necessidade de armazenar dados;
- Com a alocação estática, tudo o que se prevê que será usado deve ser adotado de uma vez;

Alocação de memória

- Suponha que se deseja armazenar os nomes de 1000 pessoas;
- Com alocação estática, deve-se definir um limite máximo para o tamanho de cada nome e alocar 1000 vezes o tamanho decidido;
- Com alocação dinâmica é possível ler cada nome e depois alocar o espaço necessário para armazena-lo;

Alocação de memória em C

- A alocação de memória em C é feita por meio da função malloc;
- A função malloc aloca dados em uma área de memória chamada de heap, que é gerenciada pelo sistema operacional;
- O tamanho da área a ser alocado é passado como parâmetro para a função malloc;
- A função malloc recebe como argumento a quantidade de bytes que será alocado e devolve o endereço (ou seja, um ponteiro) inicial da memória alocada;
    - O tipo de dado retornado é void *, assim, quando se usa a função malloc é uma boa prática de programação fazer um cast do endereço retornado para o tipo de variável que se está atribuindo;

Exemplo malloc

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int *pont_int;
    float *pont_float;
    char *pont_char;
    pont_int = (int*)malloc(sizeof(int));
    pont_float = (float*)malloc(sizeof(float));
    pont_char = (char*)malloc(sizeof(char));
    (*pont_int)=10;
    (*pont_float)=13.3;
    (*pont_char)='x';
    printf("valor inteiro:%d\n",(*pont_int));
    printf("valor float  :%f\n",(*pont_float));
    printf("valor char   :%c\n",(*pont_char));
}
```

Alocação de memória em C

- Para desalocar memória utiliza-se o procedimento free;
- O procedimento free tem uma sintaxe bem mais simples:
  - Não retorna nenhum valor;
  - Recebe somente um argumento;
    - O ponteiro que aponta para o endereço inicial a ser desalocado;

Exemplo free

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int *pont_int;
    float *pont_float;
    char *pont_char;
    pont_int = (int*)malloc(sizeof(int));
    pont_float = (float*)malloc(sizeof(float));
    pont_char = (char*)malloc(sizeof(char));
    (*pont_int)=10;
    (*pont_float)=13.3;
    (*pont_char)='x';
    printf("valor inteiro:%d\n",(*pont_int));
    printf("valor float  :%f\n",(*pont_float));
    printf("valor char   :%c\n",(*pont_char));
    free(pont_char);
    free(pont_float);
    free(pont_int);
}
```

Alocação de memória em C

- Desalocar o endereço apontado por um ponteiro não altera o conteúdo do ponteiro;
  - O ponteiro passa a apontar para um endereço inválido;
- Para contornar esse problema, é comum “aterrar” o ponteiro;
  - Por convenção, um ponteiro aterrado aponta para o valor NULL;

Não se deve acessar o conteúdo de um ponteiro aterrado;

Exercícios

1. Faça uma função que aloque um agregado heterogêneo chamado agregado_circular, contendo um valor inteiro e um ponteiro que aponta para um agregado_circular. Em seguida faça uma função que receba um valor N, aloque um agregado_circular , preencha o valor do inteiro com N e faça o agregado apontar para ele mesmo.
2. Faça um agregado heterogêneo chamado elemento que contenha um valor inteiro e um ponteiro para elemento (próximo). Em seguida leia 3 números e aloque um agregado do tipo elemento para cada número digitado. Faça o primeiro elemento ter seu ponteiro apontando para o segundo, o segundo ter seu ponteiro apontando para o terceiro e o terceiro apontando para NULL. Por fim, escreva os 3 valores usando uma variável auxiliar.