Professor: Danilo Silva dos Santos UFBA
MATA40 Estrutura de Dados

Lista - 6

Na linguagem C, existem os tipos básicos (char, int, float, etc.) e seus respectivos ponteiros que podem ser usados na declaração de variáveis. Para estruturar dados complexos, nos quais as informações são compostas por diversos campos, necessitamos de mecanismos que nos permitam agrupar tipos distintos. Para ilustrar, vamos considerar o desenvolvimento de programas que manipulam pontos no plano cartesiano. Cada ponto pode ser representado por suas coordenadas x e y, ambas dadas por valores reais. Sem um mecanismo para agrupar os dois componentes, teríamos que representar cada ponto por duas variáveis independentes:

float x;
float y;

No entanto, deste modo, os dois valores ficam dissociados e, no caso do programa manipular vários pontos, cabe ao programador não misturar a coordenada x de um ponto com a coordenada y de outro. Para facilitar este trabalho, a linguagem C oferece recursos para agruparmos dados. Uma estrutura, em C, serve basicamente para agrupar diversas variáveis dentro de um único contexto. No nosso exemplo, podemos definir uma estrutura ponto que contenha as duas variáveis. A sintaxe para a definição de uma estrutura é mostrada abaixo:

```
struct ponto {
   float x;
   float y;
};
```

Desta forma ponto passa a ser um tipo.

1- Revisando estruturas em C:

a. Declare uma variável do tipo ponto.
b. Use scanf para solicitar valores para a coordenada x e y da variável que criou e imprima na tela. (obs: Observe que o operador de acesso ao campo da estrutura tem precedência sobre o operador “endereço de”).
c. Separando o código, crie duas funções. Uma para capturar os valores de (x,y) e outra para imprimir. (obs: Passe a estrutura por valor).
d. Crie um ponteiro para estrutura.
e. Faça o ponteiro apontar para estrutura declarada em a.
f. Crie duas funções. Uma para capturar os valores de (x,y) e outra para imprimir. (obs: Passe a estrutura por referência).
g. Comente sobre as funções em c. e f.

h. Crie uma função para determinar a distância entre dois pontos. Considere a implementação de uma função que tenha como valor de retorno a distância entre dois pontos. O protótipo da função deve ser:

```
float distancia (struct ponto *p, struct ponto *q);
```

Nota: A distância entre dois pontos é dada por: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Da mesma forma que os vetores, as estruturas podem ser alocadas dinamicamente. Por exemplo, é válido escrever:

```
struct ponto* p;
p = (struct ponto*) malloc (sizeof(struct ponto));
```

Neste fragmento de código, o tamanho do espaço de memória alocado dinamicamente é dado pelo operador sizeof aplicado sobre o tipo estrutura (sizeof(struct ponto)). A função malloc retorna o endereço do espaço alocado, que é então convertido para o tipo ponteiro da estrutura ponto. Após uma alocação dinâmica, podemos acessar normalmente os campos da estrutura, através da variável ponteiro que armazena seu endereço:

```
...
p->x = 12.0;
...
```

Da mesma forma que fazemos com tipos comuns (int, float) podemos declarar vetores de estruturas e podemos também declarar vetores de ponteiros para estruturas. Considere:

```
struct aluno {
   char nome[81];
   int mat;
   char end[121];
   char tel[21];
};

typedef struct aluno Aluno;
```

Vamos montar a tabela de alunos usando um vetor global com um número máximo de alunos. Uma primeira opção é declarar um vetor de estruturas:

```
#define MAX 100
Aluno tab[MAX];
```

Desta forma, podemos armazenar nos elementos do vetor os dados dos alunos que queremos organizar. Seria válido, por exemplo, uma atribuição do tipo:

```
...
tab[i].mat = 9912222;
...
```

No entanto, o uso de vetores de estruturas tem, neste caso, uma grande desvantagem. O tipo Aluno definido acima ocupa pelo menos 227 (=81+4+121+21) bytes. A declaração de um vetor desta estrutura representa um desperdício significativo de memória, pois provavelmente estaremos armazenando de fato um número de alunos bem inferior ao máximo estimado. Para contornar este problema, podemos trabalhar com um vetor de ponteiros.

```
typedef struct aluno *PAluno;

#define MAX 100
PAluno tab[MAX];
```

3- Considerando o vetor de ponteiros declarado acima como uma variável global.

a. Crie uma função de inicialização que atribui NULL a todos os elementos da tabela, significando que temos, a princípio, uma tabela vazia.
b. Crie uma função que armazena os dados de um novo aluno numa posição do vetor. Vamos considerar que os dados serão fornecidos via teclado e que uma posição onde os dados serão armazenados será passada para a função. Se a posição da tabela estiver vazia, devemos alocar uma nova estrutura; caso contrário, atualizamos a estrutura já apontada pelo ponteiro.
c. Crie uma função para remover os dados de um aluno da tabela. Considere que a posição da tabela a ser liberada será passada para a função.
d. Crie uma função para consultar os dados. Considere que a posição da tabela será passada.
e. Crie uma função que imprima os dados de todos os alunos da tabela.
f. Faça um programa que utilize as funções da tabela de alunos escritas acima.
g. Re-escreva as funções acima sem usar uma variável global. (obs: Crie um tipo Tabela e faça as funções receberem este tipo como primeiro parâmetro.)