Programação 2024/25

LEI, LEI-PL, LEI-CE

Aula Laboratorial 6

Bibliografia:
K. N. King. *C programming: A Modern Approach* (2nd Edition). W. W. Norton: capítulos 15 e 16.

Código de apoio para a aula:
https://gist.github.com/FranciscoBPereira

Arrays de Estruturas

Exercícios Obrigatórios

1. Pretende-se criar um programa para armazenar um conjunto de retângulos e efetuar algumas operações sobre eles. Ao responder a esta questão deve utilizar as estruturas definidas na aula laboratorial anterior, incorporando o novo código no projeto já existente. Sempre que possível, as operações sobre retângulos individuais devem ser feitas através de chamadas às funções que já estão implementadas.

a) Declare uma tabela que tenha capacidade para armazenar 10 retângulos. A tabela deve ser uma variável local da função `main()`. Deverá igualmente declarar uma variável inteira que sirva como contador do número de retângulos armazenados na tabela.

`tab`										`total`
R1	R2	R3								3

b) Escreva uma função em C que imprima a informação relativa a todos os retângulos armazenados na tabela. Para cada retângulo devem ser indicadas as coordenadas dos seus 4 cantos. A função recebe o endereço inicial da tabela e o número de retângulos aí armazenados.

c) Escreva uma função em C que adicione um novo retângulo à tabela. Os dados necessários são indicados pelo utilizador. A função recebe o endereço inicial da tabela e o endereço de uma variável inteira contendo o número de retângulos aí armazenados. A função deve verificar se existe espaço na tabela para adicionar o novo retângulo. Caso a adição seja efetuada, a função deve atualizar a variável inteira que contém o número de retângulos armazenados. A função devolve 1 se a adição for sem sucedida, ou 0, caso contrário.

d) Escreva uma função em C que imprima a informação relativa a todos os retângulos armazenados na tabela que sejam quadrados. Para cada retângulo devem ser indicadas as coordenadas dos seus 4 cantos. A função recebe o endereço inicial da tabela e o número de retângulos aí armazenados.

e) Escreva uma função em C que duplique a altura e largura de todos os retângulos com área par que estão armazenados na tabela. A função recebe o endereço inicial da tabela e o número de retângulos aí armazenados.

f) Escreva uma função em C que verifique quantos retângulos armazenados na tabela têm o seu canto inferior esquerdo no primeiro quadrante do plano cartesiano. A função recebe o endereço inicial da tabela e o número de retângulos aí armazenados como parâmetros e devolve o valor contabilizado.

g) Escreva uma função em C que elimine da tabela o retângulo com menor área que aí estiver armazenado. A função recebe o endereço inicial da tabela e o endereço de uma variável inteira contendo o número de retângulos aí armazenados. Caso a eliminação seja bem sucedida, a função deve atualizar a variável inteira que contém o número de retângulos armazenados.

h) Escreva uma função em C que inverta a ordem pela qual os retângulos estão armazenados na tabela. A função recebe o endereço inicial da tabela e o número de retângulos aí armazenados.

i) Escreva uma função em C que elimine da tabela todos os retângulos com área inferior a um determinado limite. A função recebe o endereço inicial da tabela, o endereço de uma variável inteira contendo o número de retângulos aí armazenados e o valor limite a considerar. Esta função deve atualizar a variável inteira que contém o número de retângulos armazenados.

Exercícios Complementares

2. O aeródromo de Coimbra pretende um programa para gerir a informação relativa às partidas de voos. Cada voo é identificado por: número do voo, nome da companhia, cidade destino e hora de partida (horas e minutos).

a) Crie um tipo estruturado chamado *struct tempo* que permita armazenar as componentes de uma determinada hora de um dia (horas e minutos).

b) Crie um tipo estruturado chamado *struct voo* que permita armazenar a informação relativa a um voo. O campo que armazena a hora de partida deve ser do tipo *struct tempo.*

c) Escreva uma função em C que mostre o conteúdo dos campos de uma variável estruturada do tipo *struct voo*. A função recebe como argumento a variável já preenchida.

d) Escreva uma função em C que permita introduzir informação relativa a um novo voo. É passado como argumento um ponteiro para a estrutura a inicializar.

e) Escreva uma função em C que altere a hora de partida de um voo. Recebe como argumento um ponteiro para uma estrutura já preenchida e solicita ao utilizador as novas componentes da hora para efetuar a alteração.

f) Escreva uma função em C que verifique se um determinado voo já partiu. Recebe como argumentos uma estrutura do tipo *struct voo* onde está armazenada a informação do voo e uma estrutura do tipo *struct tempo* onde está armazenada a hora atual. Devolve 1 se o voo já tiver partido, ou 0, se isso ainda não tiver acontecido.

g) Declare uma tabela que lhe permita armazenar informação relativa a 300 voos. A tabela deve ser uma variável local da função `main()`.

h) Escreva uma função em C que permita adicionar novos voos à tabela. A função recebe como argumentos um ponteiro para o início da tabela e um ponteiro para o inteiro que indica quantos voos existem. É o utilizador que indica quantas novas entradas quer inserir e que especifica a informação relativa a cada um dos novos voos. A função deve atualizar a tabela e o inteiro referenciado pelo segundo argumento, indicando quantos voos passam a existir.

i) Escreva uma função em C que liste a informação completa de todos os voos armazenados na tabela. A função recebe como argumentos um ponteiro para o início da tabela e o número de voos que esta contém.

j) Escreva uma função em C que verifique quais os voos que partem nos próximos 30 minutos. Deve ser escrito o número, o destino e o tempo que falta para a partida de cada um desses voos. Um ponteiro para o início da tabela de voos, o número de elementos que esta contém e a hora atual são passados como argumentos.

k) Escreva uma função em C que retire da tabela todos os voos que já partiram. São passados como argumentos um ponteiro para o início da tabela, um ponteiro para um inteiro que indica quantos voos esta contém e a hora atual. A função deve atualizar a tabela e o contador de voos.

l) Crie um programa que construa um menu com o seguinte formato:

1. Introduzir novos voos
2. Listar todos os voos
3. Listar proximos voos
4. Atualizar tabela de voos
5. Terminar

O programa deve fazer a gestão das seleções que o utilizador for efetuando. Termina a execução quando for selecionada a opção 5.

Sugestão: Utilize a seguinte função para obter a hora atual do sistema. O resultado é devolvido numa estrutura do tipo `struct tempo`.

```
#include <stdio.h>
#include <time.h>

// Estrutura auxiliar para guardar as componentes da hora
struct tempo{ int h, m;};

struct tempo hora_atual(){
      time_t a;
      struct tm* b;
      struct tempo atual;

      time(&a);
      b = localtime(&a);
      atual.h = b->tm_hour;
      atual.m = b->tm_min;
      return atual;
}
```

```
// Exemplo de utilização da função hora_atual()

int main(){
      struct tempo t = hora_atual();
      printf(" São %2.2d:%2.2d\n", t.h, t.m);
      return 0;
}
```

Trabalho Autónomo

3. Escreva uma função em C que ordene os retângulos da tabela criada no exercício 1 pela distância Euclidiana do seu canto inferior esquerdo à origem das coordenadas (do mais próximo para o mais afastado). A função recebe o endereço inicial da tabela e o número de retângulos aí armazenados.

Algoritmos de Ordenamento

Os algoritmos de ordenamento são estratégias que permitem organizar os elementos de um conjunto numa determinada ordem. O ordenamento eficiente é extremamente importante, uma vez que é uma tarefa realizada muitas vezes e com grandes volumes de dados.

Existem inúmeros algoritmos de ordenamento, com diferenças importantes na estratégia, eficiência e complexidade. Os alunos interessados podem consultar o livro *Algorithms in C* de Robert Sedgewick (disponível na biblioteca do ISEC), para uma visão mais detalhada das possíveis estratégias e implementação.

Neste trabalho sugerem-se 2 possibilidades para resolver o problema:

a) Implementar o algoritmo *Bubble Sort*. É um dos algoritmos de ordenamento mais simples e mais fáceis de implementar, embora seja pouco eficiente. Procure uma explicação desta estratégia e implemente a função correspondente.

b) Recorrer à função *qsort()* da biblioteca <*stdlib.h*> para resolver o problema. Esta é uma função pertencente às bibliotecas standard da linguagem C que consegue ordenar um array de acordo com um determinado critério. Procure exemplos de utilização e veja como implementar a função correspondente. Ao utilizar esta possibilidade terá de lidar com ponteiros para funções, pelo que deverá procurar alguma informação sobre o assunto. A secção 17.7 do livro *C Programming: A Modern Approach* contém detalhes sobre estes tópicos.