UAlg FCT

UNIVERSIDADE DO ALGARVE

FACULDADE DE CIÊNCIAS E TECNOLOGIA

Laboratório de Programação

Lição n.º 3

Estruturas e arrays de estruturas

Estruturas e arrays de estruturas

- Estruturas.
- Typedefs.
- Arrays de estruturas.
- Buscas em arrays de estruturas.
- Argumentos na linha de comando.

Estruturas

- *A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling*. (K&R, p. 127).
- Ao usar estruturas, em cada caso definiremos um tipo, por meio de um typedef.

Exemplo: Pontos

- Para representar pontos de coordenadas inteiras, usamos o seguinte tipo Point:

```
typedef struct {
  int x;
  int y;
} Point;
```

Não confunda com pontos no plano, em que as coordenadas seriam double. Este tipo Point serve, por exemplo, para identificar pixéis numa janela ou quadrículas numa grelha.

- Dizemos que o tipo Point é um tipo struct e que cada ponto é uma *estrutura*.
- Cada variável na estrutura é um *membro*; logo, cada ponto tem dois membros, x e y, ambos de tipo int.

Operações com pontos

- Distância

```
//Euclidean distance
double distance(Point p, Point q)
{
  return sqrt((p.x-q.x)*(p.x-q.x) + (p.y-q.y)*(p.y-q.y));
}
```

- Colinearidade

```
//q.y - p.y    r.y - q.y
//--------- == ---------
//q.x - p.x    r.x - q.x

int collinear(Point p, Point q, Point r)
{
  return (q.x-p.x)*(r.y-p.y) == (r.x-p.x)*(q.y-p.y);
}
```

Construtor

- Um construtor é uma função que devolve uma estrutura, a partir dos valores de cada um dos membros:

```
Point point(int x, int y)
{
  Point result;
  result.x = x;
  result.y = y;
  return result;
}
```

Estilo: os nomes dos tipos struct escrevem-se com maiúscula inicial (Point) e o construtor é o mesmo nome, mas escrito todo em minúsculas (point).

Testes unitários

- Eis uma função de teste unitário para cada uma das funções distance e collinear:

```
void unit_test_distance(void)
{
  Point p1 = point(2,4);
  Point p2 = point(5,8);
  assert(distance(p1, p2) == 5);
  Point p3 = point(0,0);
  Point p4 = point(1,1);
  assert(distance(p3, p4) == sqrt(2));
  assert(distance(p1, p3) == sqrt(20));
  assert(distance(p1, p1) == 0);
  assert(distance(p2, p4) == sqrt(65));
}
```

```
void unit_test_collinear(void)
{
  Point p1 = point(2,4);
  Point p2 = point(5,8);
  Point p3 = point(8,12);
  assert(collinear(p1, p2, p3));
  Point p4 = point(0,0);
  Point p5 = point(1,1);
  Point p6 = point(4,4);
  assert(collinear(p4, p5, p6));
  Point p7 = point(1000, 4);
  assert(collinear(p1, p6, p7));
  assert(!collinear(p4, p1, p2));
}
```

Teste na consola

```
void test_points(void)
{
  int x1, y1, x2, y2, x3, y3;
  while (scanf("%d%d%d%d%d%d", &x1, &y1, &x2, &y2, &x3, &y3) != EOF)
  {
    Point p1 = point(x1, y1);
    Point p2 = point(x2, y2);
    Point p3 = point(x3, y3);
    double d12 = distance(p1, p2);
    double d23 = distance(p2, p3);
    double d31 = distance(p3, p1);
    int c = collinear(p1, p2, p3);
    printf("%.4f %.4f %.4f\n", d12, d23, d31);
    printf("%d\n", c);
  }
}
```

```
$ ./a.out
1 3 2 4 3 5
1.4142 1.4142 2.8284
1
0 0 4 0 0 3
4.0000 5.0000 3.0000
0
2 1 6 1 -5 1
4.0000 11.0000 7.0000
1
$
```

Distância ao quadrado

- Em muitos problemas, queremos comparar distâncias, duas a duas, não queremos propriamente conhecer as distâncias uma a uma.
- Por exemplo, num array de pontos, determinar os dois pontos mais próximos um do outro.
- Ora, em vez de comparar distâncias, é mais simples e económico comparar os *quadrados* das distâncias:

```
int distance_squared(Point p, Point q)
{
  return (p.x-q.x)*(p.x-q.x) + (p.y-q.y)*(p.y-q.y);
}
```

- Evitamos assim o custo de calcular a raiz quadrada.

Segundo exemplo: Voos

- Temos ficheiros com a lista de partidas nos aeroportos de Faro, Lisboa, Porto, com o formato que o exemplo ilustra.

```
0605 TP1900 Lisboa
0945 FR1767 East_Midlands
0945 U21880 Manchester
0950 HV6094 Rotterdam
0950 U28914 London,_Gatwick
1005 FR8249 Bristol
1035 U22016 London,_Luton
1040 U26794 Belfast
1100 U26006 Bristol
1105 TP1908 Lisboa
1130 FR7481 Memmingen
1145 BA2693 London,_Gatwick
1235 EC4486 Lyon,_St._Exupery
1250 LS1440 London,_Stansted
1330 LS876  Manchester
1340 FR7033 Dublin
1345 FR4032 Liverpool
1410 LS1222 Birmingham
1500 FR4051 Manchester
1500 FR7411 Eindhoven
1550 FR9283 London,_Stansted
1640 TP1902 Lisboa
1700 EI495  Dublin
1705 U27362 London,_Southend
1715 FR6165 Dusseldorf_Weeze
1735 FR4170 Frankfurt
1755 OE2365 Vienna
2030 FR5487 Porto
2030 FR652  Prestwick
2045 U28918 London,_Gatwick
```

- Queremos realizar diversas operações sobre estes dados:
  - Que voos têm um dado destino?
  - Que voos partem entre as tantas e as tantas?
  - Listar os voos por ordem alfabética de destino.
  - Quais são os voos de uma dada companhia?
  - Etc.

Estes são os valores reais das partidas do aeroporto de Faro no dia 12 de dezembro de 2019.

Tipo Flight

- Caraterizamos um voo pela hora de partida, número de voo e destino:

```
typedef struct {
  const char *code;
  const char *destination;
  int departure;
} Flight;
```

A utilização de const char * indica que através do endereço registado na variável podemos consultar a cadeira referenciada, mas não podemos modificá-la.

- O número de voo e o destino serão cadeias de carateres no heap, representadas pelo seu endereço.
- O construtor é simples:

```
Flight flight(const char *code, const char *destination, int departure)
{
  Flight result;
  result.code = code;
  result.destination = destination;
  result.departure = departure;
  return result;
}
```

Lendo para array

- O processamento será feito sobre os dados em memória, após terem sido lidos do ficheiro para um array.
- Para ler, usamos a seguinte função:

```
int flights_read(FILE *f, Flight *a)
{
  int result = 0;
  char code[16];
  char destination[64];
  int departure;
  while (fscanf(f, "%d%s%s", &departure, code, destination) != EOF)
    a[result++] =
      flight(str_dup(code), str_dup(destination), departure);
  return result;
}
```

As dimensões das variáveis locais code e destination são arbitrárias, mas suficientes para os valores em jogo.

Note bem: as cadeias lidas são duplicadas para o heap e os endereços são guardados na estrutura.

Escrevendo o array

- Para controlar que a leitura ficou bem feita, escrevemos o array, com um formato apropriado:

```
void flights_write(FILE *f, Flight *a, int n)
{
  for (int i = 0; i < n; i++)
    fprintf(f, "[%d][%s][%s]\n",
      a[i].departure, a[i].code, a[i].destination);
}
```

Escrevemos os valores dos membros entre parêntesis retos, para melhor controlo.

Função de teste, leitura-escrita

- Eis uma função de teste, que lê o ficheiro com as partidas de Faro e que depois escreve o array na consola:

```
#define MAX_FLIGHTS 10000

void test_flights_read_write(void)
{
  FILE *f = fopen("partidas_faro.txt", "r");
  assert(f != NULL);
  Flight flights[MAX_FLIGHTS];
  int n_flights = flights_read(f, flights);
  flights_write(stdout, flights, n_flights);
}
```

```
$ ./a.out
[605][TP1900][Lisboa]
[945][FR1767][East_Midlands]
[945][U21880][Manchester]
[950][HV6094][Rotterdam]
[950][U28914][London,_Gatwick]
[1005][FR8249][Bristol]
[1035][U22016][London,_Luton]
[1040][U26794][Belfast]
[1100][U26006][Bristol]
[1105][TP1908][Lisboa]
[1130][FR7481][Memmingen]
[1145][BA2693][London,_Gatwick]
[1235][EC4486][Lyon,_St._Exupery]
[1250][LS1440][London,_Stansted]
[1330][LS876][Manchester]
[1340][FR7033][Dublin]
[1345][FR4032][Liverpool]
[1410][LS1222][Birmingham]
[1500][FR4051][Manchester]
[1500][FR7411][Eindhoven]
[1550][FR9283][London,_Stansted]
[1640][TP1902][Lisboa]
[1700][EI495][Dublin]
[1705][U27362][London,_Southend]
[1715][FR6165][Dusseldorf_Weeze]
[1735][FR4170][Frankfurt]
[1755][OE2365][Vienna]
[2030][FR5487][Porto]
[2030][FR652][Prestwick]
[2045][U28918][London,_Gatwick]
$
```

Note bem: estamos a supor, simplificando, que o executável a.out está colocado na diretoria de trabalho onde residem os ficheiros de dados. Nem sempre será assim.

Voos para um dado destino

- Queremos uma função para encontrar todos os voos que têm um dado destino.
- O resultado será o array dos índices dos voos cujo membro destination é igual ao destino dado:

```
int flights_select_by_destination(const Flight *a, int n,
                                  char *destination, int *b)
{
  int result = 0;
  for (int i = 0; i < n; i++)
    if (strcmp(a[i].destination, destination) == 0)
      b[result++] = i;
  return result;
}
```

Note bem: o array b é um array de int. Cada valor representa uma posição no array a onde o membro destination tem valor igual ao valor do argumento destination.

Função de teste, voos para destino dado

- Primeiro, uma função para escrever os voos de um array cujos índices vêm noutro array:

```
void flights_write_selection(FILE *f, const Flight *a, int *b, int n)
{
  for (int i = 0; i < n; i++)
    fprintf(f, "[%d][%s][%s]\n",
      a[b[i]].departure, a[b[i]].code, a[b[i]].destination);
}
```

- Agora a função de teste:

```
void test_flights_select_destination(void)
{
  FILE *f = fopen("partidas_lisboa.txt", "r");
  assert(f != NULL);
  Flight flights[MAX_FLIGHTS];
  int n_flights = flights_read(f, flights);
  char line[MAX_LINE_LENGTH];
  while (scanf("%s", line) != EOF)
  {
    int b[n_flights];
    int n = flights_select_by_destination(flights, n_flights, line, b);
    flights_write_some(stdout, flights, b, n);
  }
}
```

```
$ ./a.out
Porto
[705][TP1928][Porto]
[800][FR2094][Porto]
[935][TP1926][Porto]
[1335][TP1930][Porto]
[1600][TP1936][Porto]
[1835][TP1934][Porto]
[2010][TP1940][Porto]
[2025][FR2096][Porto]
[2210][TP1922][Porto]
Faro
[950][TP1907][Faro]
[1525][TP1909][Faro]
[2210][TP1901][Faro]
Luanda
[1000][DT651][Luanda]
[2325][TP289][Luanda]
```

Argumentos na linha de comando

- Para selecionar uma ou outra função de teste, podemos usar "argumentos na linha de comando".
- De facto, temos acesso, no nosso programa, a cada uma das cadeias de carateres presentes na linha de comando usada para invocar o nosso programa.
- Observe:

```
int main(int argc, char **argv)
{
  strings_printfln(argv, argc, "{%s}");
  return 0;
}
```

```
$ ./a.out
{./a.out}
$ ./a.out um dois tres
{./a.out}{um}{dois}{tres}
$ ./a.out 89 23 abcd 3.141592
{./a.out}{89}{23}{abcd}{3.141592}
$
```

Quer dizer: programando a função main desta maneira, temos no array argv cada uma das "palavras" da linha de comando. A primeira dessas palavras é a cadeia que invoca o programa; as outras são o que nós quisermos.

Selecionando, na função main

- Tipicamente, para nós o primeiro argumento será uma letra maiúscula, com a qual selecionamos a função de teste:

```
int main(int argc, const char **argv)
{
  char x = 'A';
  if (argc > 1)
    x = *argv[1];
  if (x == 'A')
    test_flights_read_write();
  else if (x == 'B')
    test_flights_select_by_destination();
  else
    printf("%c Invalid option.\n", x);
  return 0;
}
```

Note bem, argv é um array de cadeias de caracteres, argv[1] é uma cadeia de caracteres e *argv[1] é o primeiro caractere da cadeia de caracteres argv[1].

Nomes de ficheiro na linha de comando

- Para escolher o ficheiro de dados, entre os vários ficheiros disponíveis, podemos também usar argumentos na linha de comando.
- Mas antes temos de retocar as nossas funções de teste, de modo a aceitarem como argumento o nome do ficheiro que devem usar (o qual deixa de ser fixo):

```
void test_flights_read_write_better(const char *filename)
{
  FILE *f = fopen(filename, "r");
  assert(f != NULL);
  ...
}

void test_flights_select_by_destination_better(const char *filename)
{
  FILE *f = fopen(filename, "r");
  assert(f != NULL);
  ...
}
```

Função main, nova versão

- Temos agora quatro funções de teste:

```
int main(int argc, const char **argv)
{
  char x = 'A';
  const char *filename = "partidas_faro.txt";
  if (argc > 1)
    x = *argv[1];
  if (x == 'A')
    test_flights_read_write();
  else if (x == 'B')
    test_flight_flights_select_destination();
  else if (x == 'C')
    test_flights_read_write_better(argc > 2 ? argv[2] : filename);
  else if (x == 'D')
    test_flights_select_by_destination_better
      (argc > 2 ? argv[2] : filename);
  else
    printf("%c Invalid option.\n", x);
  return 0;
}
```

Não havendo argumentos na linha de comando (para além da invocação do programa) usa-se a opção 'A'. No caso das opções 'C' e 'D', se faltar o nome do ficheiro, usa-se "partidas_faro.txt".