Ficha de Trabalho n.º 4

Estruturas Compostas – *Struct*

Uma ***struct*** permite guardar numa mesma estrutura diferentes valores de diferentes tipos (incluindo outras *struct*). É uma estrutura composta por diversos campos.

Declaração

```
struct <nome da estrutura>
{
      <tipo 1> <nome do campo 1>;
      <tipo 2> <nome do campo 2>;
      ...
      <tipo n> <nome do campo n>;
};
```

```
struct dados_aluno
{
      char curso[5]; // campo curso é uma string
      int numero; // campo numero é um inteiro
      char nome[50]; // campo nome é uma string
      float media; // campo média é um real
}; //definição de uma estrutura dados_aluno com 4 campos
```

As definições de estruturas devem aparecer no início dos programas antes dos subprogramas.

A declaração de uma variável deste tipo faz-se do mesmo modo que qualquer outra variável (no *main* e outros subprogramas).

```
<estrutura definida> <nome da variável>;
```

```
struct dados_aluno al1; // variável al1 é uma estrutura dados_aluno
```

Para facilitar a declaração de variáveis, muitas vezes definem-se as estruturas como um tipo, usando a partícula `typedef`[1]:

```
struct dados_aluno
{
      char curso[5];
      int numero;
      char nome[50];
      float media;
};
typedef struct dados_aluno aluno;// aluno é um tipo
```

---

[1] **typedef** pode ser usado para qualquer tipo e deve aparecer antes dos subprogramas porque esta definição é válida em todo o programa.

typedef <tipo> <nome a atribuir ao tipo>;

Para definir-se o tipo *real* para *float* basta

```
typedef float real;
```

Para declarar e inicializar uma variável x do tipo real

```
real x = 1.2;
```

Ou, de forma equivalente, pode definir-se a estrutura e o tipo ao "mesmo" tempo:

```
typedef struct dados_aluno
{
        char curso[5];
        int numero;
        char nome[50];
        float media;
} aluno;
```

Assim, para declarar a variável al1 como uma estrutura dados_aluno, basta

```
aluno al1;
```

Atribuição

Tal como nas tabelas, a atribuição deve ser feita elemento a elemento. Neste caso, campo a campo.

```
<nome da variável>.<nome do campo> = <valor>;
```

```
strcpy(al1.curso,"lei");
// por ser uma string a atribuição deve ser feita usando strcpy
al1.numero = 123;
strcpy(al1.nome,"ze");
al1.media = 12.0f;
```

Leitura

A leitura depende do tipo do campo a ler.

```
printf("curso: ");
fgets(al1.curso,5,stdin);
printf("nome: ");
fgets(al1.nome,50,stdin);
printf("numero: ");
scanf("%i",&al1.numero);
printf("media: ");
scanf("%f",&al1.media);
```

(se um dos campos fosse uma tabela de reais, por exemplo, os seus elementos deveriam ser lidos individualmente como nas restantes tabelas de reais:

```
scanf("%f",&al1.tabela[0]),
scanf("%f",&al1.tabela[1]),...).
```

Escrita

A escrita, tal como a leitura e a atribuição, deve ser feita campo a campo e tendo em conta os seus tipos.

```
printf("nome: %s",al1.nome);
printf("curso: %s",al1.curso);
printf("numero: %i\n",al1.numero);
printf("media: %f\n",al1.media);
```

(se um dos campos fosse uma tabela de reais, por exemplo, os seus elementos deveriam ser escritos individualmente como nas restantes tabelas de reais:

```
printf("%f",al1.tabela[0]),
printf("%f",al1.tabela[1]),...).
```

1. Complete o seguinte programa tendo em conta os comentários

```
//...
struct estrutura_data
{
	int dia;
	int mes; // tambem poderia ser uma string
	int ano;
};

typedef struct estrutura_data data;

/* le uma data da consola (standard input) na forma dia-mes-ano
	devolve a estrutura que descreve a data */
data le_data()
{/*...*/}

// escreve uma data na consola (standard output) na forma dia-mes-ano
void escreve_data(data d)
{/*...*/}

/* compara duas datas
	se a 1a data e posterior a 2a data devolve '>'
	se a 1a data e anterior a 2a data devolve '<'
	se as duas datas sao iguais devolve '=' */
char compara_datas(data d1, data d2)
{/*...*/}

// determina a "maior" data entre duas datas d1 e d2 */
data maior_data(data d1, data d2)
{/*...*/}

// testa as funcoes anteriores
int main()
{
	data dat1, dat2;

	dat1 = le_data();
	dat2 = le_data();

	escreve_data(dat1);
	escreve_data(dat2);

	// usar compara_datas para obter a data mais recente entre dat1 e dat2
	/*...*/

	// usar maior_data para encontrar a "maior" data entre dat1 e dat2
	/*...*/

	return 0;
}
```

2. Considere o seguinte programa.

```
/*...*/

struct coordenadas_ponto{
  float x;
  float y;
};
typedef struct coordenadas_ponto ponto;

// subprograma escreve_ponto
// escreve um ponto com as coordenadas x e y na forma (x,y)
/*...*/

// subprograma funcao
void funcao(ponto a, ponto b)
{
  char h[15], v[15];

  if(a.x < b.x)
    strcpy(h,"esquerda");
  else strcpy(h,"direita");

  if(a.y < b.y)
    strcpy(v,"baixo");
  else strcpy(v,"cima");

  printf("O primeiro ponto está à %s e em %s do segundo
ponto!\n",h,v);
}

int main()
{
  // declarar pontos p1 e p2;
  /*...*/

  // pedir ao utilizador e ler as coordenadas dos pontos p1 e p2
  /*...*/

  // chamar o subprograma escreve_ponto para mostrar os pontos lidos
  /*...*/

  // chamar o subprograma funcao aplicado aos pontos p1 e p2
  /*...*/

  return 0;
}
```

2.1. Complete o programa seguindo as indicações dos comentários:

a) Indique o que faz o subprograma *funcao*.

b) Crie um subprograma *escreve_ponto* que apenas escreve as coordenadas de um ponto na forma usual. (Exemplo: o ponto com coordenadas 1 e 2 é apresentado como (1,2).)

c) Complete o *main* como indicado nos comentários.

2.2. No programa considere uma nova estrutura *rectângulo*:

a) Defina a estrutura *rectangulo* que tem 4 campos do tipo real: *xmin*, *xmax*, *ymin* e *ymax*.

b) Crie um subprograma *area* que, dado um rectangulo devolva a sua área.

c) Crie um subprograma *esta_dentro* que, dado um ponto e um rectangulo, escreve se o ponto está dentro ou fora do rectangulo. Considere que os pontos do perímetro *estão dentro* do rectângulo.

3. Defina uma estrutura que permita representar e efectuar operações com números racionais. Esta estrutura deverá ter dois campos: um para o numerador e outro para o denominador do número. Elabore funções que desempenhem as seguintes tarefas:

a) Ler uma fracção;

b) Somar duas fracções;

c) Subtrair duas fracções;

d) Multiplicar duas fracções;

e) Dividir duas fracções;

f) Determinar a potência (com expoente inteiro) de uma fracção.

Tabelas de *structs*

É possível declarar tabelas de estruturas do mesmo modo que se declaram tabelas de tipos simples.

```
aluno turma[15];
//variável turma é uma tabela com 15 elementos do tipo aluno
```

A leitura e a escrita realizam-se como descrito anteriormente mas deve indicar-se o elemento da tabela.

```
//leitura
for(i=0;i<15;i++)
{
        printf("curso: ");
        gets(turma[i].curso);
        printf("nome: ");
        gets(turma[i].nome);
        printf("numero: ");
        scanf("%i",&turma[i].numero);
        printf("media: ");
        scanf("%f",&turma[i].media);
}

//escrita
for(i=0;i<15;i++)
{
        printf("nome: %s\n",turma[i].nome);
        printf("curso: %s\n",turma[i].curso);
        printf("numero: %i\n",turma[i].numero);
        printf("media: %f\n",turma[i].media);
}
```

4. Uma pequena empresa familiar pretende organizar os dados dos seus funcionários: número, nome, tarefa, salário.

a) Defina uma estrutura de dados que permita representar cada funcionário.

b) Declare uma tabela que permita armazenar informação pretendida.

c) Elabore subprogramas para

i. introduzir os dados de um funcionário na tabela;

ii. listar todos os funcionários e respectivos dados;

iii. listar os funcionários com salário superior a 500€;

iv. procurar e listar todos os dados de um funcionário, usando o seu nome (caso o funcionário não exista, deverá ser devolvida uma "estrutura vazia": números a zero e *strings* vazias);

v. actualizar os dados de um funcionário (usando o seu número);

vi. ordene a tabela por ordem crescente dos números dos funcionários.

vii. ordene a tabela por ordem alfabética dos nomes dos funcionários.