

**10.1** Sejam  $i$  é uma variável inteira e  $p$  e  $q$  apontadores para inteiros. Indique quais das seguintes atribuições são legais.

- |                |                |                |
|----------------|----------------|----------------|
| (a) $p = i;$   | (d) $p = \&i;$ | (g) $p = *q;$  |
| (b) $*p = i;$  | (e) $\&i = p;$ | (h) $*p = q;$  |
| (c) $p = \&q;$ | (f) $p = q;$   | (i) $*p = *q;$ |

**10.2** Sejam  $i$  é uma variável `int` e  $p$  é um apontador para  $i$ . Indique quais das seguintes instruções de leitura e escrita são corretas.

- |                                       |                                    |                                    |
|---------------------------------------|------------------------------------|------------------------------------|
| (a) <code>printf("%d", i);</code>     | (d) <code>printf("%d", p);</code>  | (g) <code>printf("%p", p);</code>  |
| (b) <code>scanf("%d", i);</code>      | (e) <code>printf("%d", *p);</code> | (h) <code>printf("%p", i);</code>  |
| (c) <code>scanf("%d", &amp;i);</code> | (f) <code>scanf("%d", p);</code>   | (i) <code>printf("%p", *p);</code> |

**10.3** Escreva uma função

```
void decompor(int total_seg, int *horas, int *mins, int *secs);
```

que decompõe um total inteiro de segundos `total_seg` em horas, minutos (0–59) e segundos (0–59); os resultados devem ser atribuídos ao conteúdo dos apontadores `horas`, `mins` e `secs`. Pode assumir que o total de segundos é maior que zero.

▷ **10.4** Escreva uma função

```
void max_min(int vec[], int size, int *pmax, int *pmin);
```

que determina o valor máximo e mínimo de um vector; os resultados devem ser atribuídos ao conteúdo dos apontadores `pmax` e `pmin`. Pode assumir que `size` é sempre maior que zero.

**10.5** Escreva uma função `void reduzir(int *pnum, int *pdenom)` que reduz uma fração de numerador e denominador `*pnum` e `*pdenom` à forma simplificada. O numerador e denominador devem ser modificados por meio dos apontadores `pnum` e `pdenom`. Pode assumir que o denominador é sempre diferente de zero.

*Sugestão:* para simplificar uma fração basta dividir o numerador e o denominador pelo seu máximo divisor comum (pode usar o algoritmo de Euclides apresentado na aula 7 para o calcular).

**10.6** Re-escreva a função seguinte para usar apontadores em vez de índices (ou seja: eliminar a variável  $i$  e os usos de indexação `vec[...]`).

```
void store_zeros(int vec[], int n) {
    int i;
    for(i = 0; i < n; i++)
        vec[i] = 0;
}
```

**10.7** Considere a função apresentada na aula teórica 13 para contar espaços de uma cadeia de caracteres. Pretende-se que escreva uma versão alternativa desta função usando apontadores em vez de índices. A função deve ter a declaração

```
int contar_espacos(char *str);
```

O resultado deve ser o número de espaços na cadeia.

**10.8** Considere a função apresentada na aula teórica 13 para inverter a ordem de caracteres uma cadeia. Pretende-se que escreva uma versão alternativa desta função usando apontadores em vez de índices. A função deve ter a declaração `void inverter(char *str);`

▷ **10.9** Considere a função apresentada na aula teórica 14 para procurar um carácter numa cadeia. Pretende-se que escreva uma versão alternativa desta função usando apontadores em vez de índices. A função deve ter a declaração

```
char *procurar(char *str, char ch);
```

O resultado deve ser um apontador para a primeira ocorrência do carácter `ch` (se este ocorrer) ou `NULL` caso contrário.

**10.10** Considere a função apresentada na aula teórica 14 para comparar igualdade de cadeias de caracteres. Pretende-se que escreva uma versão alternativa desta função usando apontadores em vez de índices. A função deve ter a declaração

```
int comparar(char *str1, char *str2);
```

O resultado deve ser 1 se as cadeias são iguais e 0 se são diferentes.