Programação em C - Exercícios de revisão sobre apontadores

Ligações úteis:

- Apontamentos de Programação Imperativa (CC1003) 2019/2020 - Apontadores
- Apontamentos de Programação Imperativa (CC1003) 2019/2020 - Programação com apontadores
- C Programming Videos (Neso Academy) - Apontadores - Video 101 e seguintes

---

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

(a) `p = i;`
(b) `*p = i;`
(c) `p = &q;`
(d) `p = &i;`
(e) `&i = p;`
(f) `p = q;`
(g) `p = *q;`
(h) `*p = q;`
(i) `*p = *q;`

2.2 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) `printf("%d",i);`
(b) `scanf("%d",i);`
(c) `scanf("%d",&i);`
(d) `printf("%d",p);`
(e) `printf("%d",*p);`
(f) `scanf("%d",p);`
(g) `printf("%p",p);`
(h) `printf("%p",i);`
(i) `printf("%p",*p);`

2.3 Tendo como referência o exemplo abaixo, escreva um programa que use apontadores para cada um dos três tipos de dados básicos (`char`; `short`, `int` ou `long`; e `float` ou `double`). Comece por declarar e iniciar uma variável para cada tipo básico juntamente com um apontador para essa variável. De seguida, utilize o operador sizeof para imprimir o tamanho de cada variável, o tamanho do seu endereço de memória, o tamanho de cada apontador e o tamanho do conteúdo apontado por cada apontador. Por fim, imprima o endereço e o conteúdo de cada uma das variáveis.

```
char c, *cptr;
c = 'a';
cptr = &c;
printf("tamanho de um char: %lu\n", sizeof(c));
printf("tamanho do endereço de um char: %lu\n", sizeof(&c));
```

```
printf("tamanho de um apontador para um char: %lu\n", sizeof(cptr));
printf("tamanho do conteúdo apontado por um apontador para um char:
       %lu\n", sizeof(*cptr));
printf("Os valores apontados pelos endereços '%p' e '%p' são '%c' e
       '%c'\n", &c, cptr, c, *cptr);
```

2.4 Considere a seguinte declaração:

```
int x[3] = {23, 41, 17};
```

Qual é o valor das expressões que se seguem:

(a) `x[0]`
(b) `x[1]`
(c) `x[2]`
(d) `x`
(e) `*x`
(f) `x+1`
(g) `*(x+1)`
(h) `x+2`
(i) `*(x+2)`
(j) `&(x[0])`
(k) `*&(x[0])`
(l) `&*(x[0])`

2.5 Considere o seguinte exemplo de código C:

```
#include <stdio.h>
#define SIZE 10

void show_vector(char *a, int n);

int main() {
  char v[SIZE] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};
  show_vector(v, SIZE);
  return 0;
}

void show_vector(char *a, int n) {
  int i;
  for (i = 0; i < n; i++) {
    /* imprima aqui os valores e endereços de cada elemento do vector */
  }
}
```

(a) Complete o código de forma a imprimir os valores e os endereços de um vector de caracteres com as 10 primeiras letras do alfabeto.

(b) Que alterações são necessárias fazer ao programa para se poder imprimir o conteúdo do vector v com a instrução `printf("%s", v);`.

2.6 Escreva uma função

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

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

2.7 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 atribuidos ao conteúdo dos apontadores `pmax` e `pmin`. Pode assumir que `size` é sempre maior que zero.

2.8 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 para o determinar)).

2.9 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;
}
```

2.10 Pretende-se que escreva uma função para para contar espaços de uma cadeia de caraters. Na implementação use 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.

2.11 Pretende-se que escreva uma função para inverter a ordem de carateres uma cadeia. Na implementação use apontadores em vez de índices. A função deve ter a declaração `void inverter(char *str);`

2.12 Pretende-se que escreva uma função para procurar um carater numa cadeia. Na implementação use 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 carater `ch` (se este ocorrer) ou `NULL` caso contrário.

2.13 Pretende-se que escreva uma função para comparar igualdade de cadeias de carateres. Na implementação use 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.