Soluções da primeira aula de exercícios de MC202

1. Dê a interface (apenas) de um Tipo Abstrato de Dados que armazena a relação entre um número inteiro e um texto (por exemplo, RA e Nome do aluno) e permite, dado um número inteiro, adicionar, alterar, remover ou encontrar a string associada a esse número. Esse TAD é chamado de um *mapa* de inteiros em strings.

Solução:

```
typedef Mapa * p_mapa;

char * pega_valor(p_mapa mapa, int n);
void define_valor(p_mapa mapa, int n, char *string);
void altera_valor(p_mapa mapa, int n, char *string);
void apaga_valor(p_mapa mapa, int n);
```

2. Prove que:

a) $2n + 42 = O(n)$

Solução: Como $2n + 42 \leq 3n$ para todo $n \geq 42$, basta escolher $n_0 = 42$ e $c = 3$.

b) $n^3 + 2n^2 - n + 10 = O(n^3)$

Solução: Como $n^3 + 2n^2 - n + 10 \leq n^3 + 2n^2 + 10 \leq 13n^3$ para todo $n \geq 10$, basta escolher $n_0 = 10$ e $c = 13$.

3. Prove ou desprove que $2^{2n} = O(2^n)$.

Solução: Suponha que $2^{2n} = O(2^n)$. Então existem constantes n_0 e c tal que $2^{2n} \leq c \cdot 2^n$ para todo $n \geq n_0$. Assim, $c \cdot 2^n = 2^{\lg c} \cdot 2^n = 2^{n+\lg c} \geq 2^{2n}$ e, portanto, $n + \lg c \geq 2n$, de onde concluímos que $n \leq \lg c$, uma contradição.

4. Crie uma função que aloca dinamicamente uma matriz tridimensional $n \times m \times p$ de números `double`, isto é, uma matriz para qual podemos acessar a posição ijk através do `matriz[i][j][k]` e que devolve tal matriz (usando um ponteiro do tipo correto).

Solução:

```
double ***aloca_tridimensional(int n, int m, int p) {
  int i,j;
  double ***matriz;
  matriz = malloc(n * sizeof(double **));
  for (i = 0; i < n; i++) {
    matriz[i] = malloc(m * sizeof(double *));
    for (j = 0; j < m; j++)
      matriz[i][j] = malloc(p * sizeof(double));
  }
  return matriz;
}
```

5. Escreva uma função que dada uma lista duplamente ligada com cabeça e dois de seus nós, troca os dois nós de lugar na lista.

Solução:

```
void troca(p_no lista, p_no n1, p_no n2) {
  p_no temp = n2->ant;
  n2->ant = n1->ant;
  n2->ant->prox = n2;
  n1->ant = temp;
  n1->ant->prox = n1;
  temp = n2->prox;
  n2->prox = n1->prox;
  n1->prox = temp;
  if (n1->prox)
    n1->prox->ant = n1;
  if (n2->prox)
    n2->prox->ant = n2;
}
```

6. Escreva uma função que devolve o número de nós em uma lista circular dado um ponteiro para um dos nós da lista.

Solução:

```
int conta(p_no lista) {
  p_no p;
  int contador = 0;
  if (lista == NULL)
    return 0;
  p = lista;
  do {
    contador++;
    p = p->prox;
  } while (p != lista)
}
```

7. Escreva uma função que devolve a concatenação de duas listas circulares dadas. Sua função pode destruir a estrutura das listas dadas.

Solução:

```
p_no concatena(p_no lista1, p_no lista2) {
  p_no ult1, ult2;
  if (lista1 == NULL)
    return lista2;
  else if (lista2 == NULL)
    return lista1;
  else {
    for(ult1 = lista1->prox; ult1->prox != lista1; ult1 = ult1->prox) ;
    for(ult2 = lista2->prox; ult2->prox != lista2; ult2 = ult2->prox) ;
    ult1->prox = lista2;
    ult2->prox = lista1;
  }
  return lista1;
}
```

8. Escreva uma função recursiva que verifica se uma lista ligada de inteiros está ordenada de maneira não-decrescente. Garanta que sua função tenha recursão de cauda.

Solução:

```
int nao_decrescente(p_no lista) {
  if (lista == NULL || lista->prox == NULL)
    return 1;
  if (lista->dado > lista->prox->dado)
    return 0;
  return nao_decrescente(lista->prox);
}
```

9. Elimine a recursão da função do exercício anterior.

Solução:

```
int nao_decrescente(p_no lista) {
  while (!(lista == NULL || lista->prox == NULL) && !(lista->dado >
   lista->prox->dado))
    lista = lista->prox;
  if (lista == NULL || lista->prox == NULL)
    return 1;
  if (lista->dado > lista->prox->dado)
    return 0;
}
```