MC202: Estrutura de Dados
Instituto de Computação - UNICAMP

Prof. Rafael C. S. Schouery

Avaliação Diagnóstica - Gabarito

1. Calcule o valor de y em função das constantes n e k, quando houver, deixando a expressão o mais simples possível.

   (a) $2^y = 42$
   $y = \log_2 42$

   (b) $2^y = 3^{10}$
   $y = 10 \log_2 3$

   (c) $y = \sum_{i=1}^{10} 3 \cdot i$
   $y = 3 \sum_{i=1}^{10} i = 3 \cdot \frac{10 \cdot 11}{2} = 165$

   (d) $y = \sum_{i=1}^{n} k \cdot i$
   $y = k \frac{(n+1)n}{2}$

   (e) $y = \sum_{i=0}^{10} 2^i$
   $y = \frac{2^{11}-1}{2-1} = 2047$

   (f) $y = \sum_{i=0}^{n} k^i$
   $y = \frac{k^{n+1}-1}{k-1}$

   (g) $y = \sum_{i=0}^{\infty} k^i \quad (0 < k < 1)$
   $y = \frac{1}{k-1}$

2. Escreva uma função em C que recebe um inteiro `n` e um inteiro `m`, aloca dinamicamente uma matriz de inteiros com `n` linhas e `m` colunas, lê `nm` inteiros do teclado armazenando esses números na matriz (preenchendo linha por linha) e devolve um ponteiro para a matriz.

```c
/* uma matriz é um vetor de vetores, onde cada vetor é uma linha da matriz -
    por isso usamos int** */
int **le_matriz(int n, int m) {
  int i, j, **matriz;
  /* alocamos um vetor de n ponteiros para inteiro (i.e, vetores) */
  matriz = malloc(n * sizeof(int *));
  for (i = 0; i < n; i++)
    /* alocamos cada uma das linhas da matriz */
    matriz[i] = malloc(m * sizeof(int *));
  /* lemos a matriz posição a posição */
  for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
      scanf("%d", &matriz[i][j]);
  return matriz;
}
```

3. Escreva uma função em C que troca os valores armazenados em duas variáveis passadas por referência (isto é, usando ponteiros). Ou seja, se a variável `a` tem valor `42` e a variável `b` tem valor `10`, após a execução da sua função teremos que `a == 10` e `b == 42`.

```c
/* recebe um ponteiro a e um ponteiro b */
void troca(int *a, int *b) {
  int aux;
  /* *a é o valor armazenado na posição apontada por a */
  /* guardamos *a */
  aux = *a;
  /* atualizamos o valor armazenado na posição apontada por a para *b */
  *a = *b;
  /* atualizamos o valor armazenado na posição apontada por b para aux */
  *b = aux;
}
```

4. Faça um algoritmo recursivo que dado dois números inteiros não-negativos `a` e `b`, calcula `a`$\times$`b` utilizando apenas a operação de soma. Lembre-se que multiplicar é o mesmo que somar várias vezes.

```c
int multiplica(int a, int b) {
  /* base da recursão: se b == 0, então a * b == 0 */
  if (b == 0)
    return 0;
  /* caso geral: a * b == a + a*(b-1) */
  return a + multiplica(a, b-1);
}
```

5. Escreva um programa completo em C que utiliza uma struct `Aluno` com campos `nome` (um vetor de char com 20 posições) e `ra` (do tipo int). Seu programa deverá ler um número `n` ≤ 10 do teclado, ler as informações `nome` e `ra` de `n` alunos e armazenar essas informações em um vetor de alunos. Posteriormente, copie esse vetor para um segundo vetor em ordem reversa e faça a impressão desse segundo vetor.

```c
#include <stdio.h>

/* define a struct */
typedef struct {
  char nome[20];
  int ra;
} Aluno;

int main() {
  /* como o número de alunos era no máximo 10, não usei alocação dinâmica, mas
     poderia ter usado...  */
  Aluno v[10], copia[10];
  int i, n;
  scanf("%d", &n);
  /* lemos os elementos do vetor campo a campo */
  for (i = 0; i < n; i++)
    scanf("%s %d", v[i].nome, &v[i].ra);
  /* podemos atribuir (=) de uma struct para outra que os valores dos campos
    são copiados automaticamente, mesmo se o conteúdo for uma string - isto é,
     não precisa usar strcpy nesse caso */
  for (i = 0; i < n; i++)
    copia[i] = v[n-i-1];
  for (i = 0; i < n; i++)
    printf("%s %d\n", copia[i].nome, copia[i].ra);
  return 0;
}
```