Instituto Federal do Espírito Santo — IFES (Campus Serra)
Coordenadoria de Informática — Cin

Aluno(a): GABARITO
Disciplina: Estrutura de dados
Valor: 20 pontos
Professor: Thiago Meireles Paixão (`thiago.paixao@ifes.edu.br`)
Data: 24 de junho de 2024

Prova 1

Responder as questões nas folhas de papel almaço.

Pontuação			
1 (5 pts.)	2 (5 pts.)	3 (4 pts.)	4 (6 pts.)

1. (4 pontos) Elabore um programa em C com uma função com assinatura `int busca(float M[MAX][MAX], int n_lin, int n_col, float v, int *lin, int *col)`. Esta função retorna por meio das variáveis `lin` e `col` a linha e coluna, respectivamente, em que se encontra o valor `v` na matriz (caso haja algum). As variáveis `n_lin` e `n_col` representam, respectivamente, o número de linhas e colunas da matriz. Se o valor foi encontrado, a função deve retornar 1 ao final (usando o comando `return`). Se o valor não for encontrado no matriz, a função deve retornar 0 ao final. Em casos onde houver valores repetidos, a função deve retornar a primeira ocorrência encontrada, seguindo o percorrimento de linhas e colunas no sentido *raster*.

   Exemplo de Execução Para o valor de busca 23 e matriz

   $$\begin{bmatrix} 1 & 2 & 1 & 4 \\ 5 & 15 & 23 & 23 \\ 9 & -1 & 4 & 10 \end{bmatrix},$$

   seu programa deve fornecer como respostas linha 1 e coluna 2 na variáveis `lin` e `col`, respectivamente, e retornar 1.

   Resposta:

```c
int busca(float M[MAX][MAX], int n_lin, int n_col, float v, int *lin, int *col)
    ↪ {
   for (int i = 0; i < n_lin; i++) {
      for (int j = 0; j < n_col; j++) {
         if (M[i][j] == v) {
            *lin = i;
            *col = j;
            return 1; // Valor encontrado
         }
      }
   }
   return 0; // Valor não encontrado
}
```

2. (5 pontos) Sobre alocação dinâmica, faça o que se pede:

   (a) (2 pontos) Explique a diferença ente alocação dinâmica e alocação estática. Mencione vantagens e desvantagens em relação simplicidade do uso e flexibilidade.

   (b) (2 pontos) Escreva uma função com assinatura `int* aloca_vetor(int n)` em linguagem C que aloque dinamicamente um vetor de inteiros com `n` elementos e retorne um ponteiro para o primeiro elemento desse vetor.

   (c) (1 ponto) Informe o tamanho – em bytes – do vetor alocado no item anterior. Forneça sua resposta em função de `n`.

   Respostas:

   (a) A alocação estática de memória em C ocorre em tempo de compilação, com o tamanho e o tempo de vida da memória definidos pelo compilador. A vantagem é a simplicidade e ausência de fragmentação de memória, enquanto a desvantagem é a falta de flexibilidade, pois o tamanho não pode ser alterado em tempo de execução.

   A alocação dinâmica de memória ocorre em tempo de execução usando funções como malloc, calloc e realloc. A vantagem é a flexibilidade para ajustar o tamanho da memória conforme necessário em tempo de execução. A desvantagem é a complexidade, exigindo que o programador gerencie manualmente a alocação e liberação da memória, o que pode causar vazamentos de memória.

   (b)

```c
int* aloca_vetor(int n) {
   // Aloca dinamicamente um vetor de inteiros com n elementos
   int *vetor = (int *)malloc(n * sizeof(int));
   // Verifica se a alocação foi bem-sucedida
   if (vetor == NULL) {
      printf("Erro ao alocar memória\n");
      return NULL; // Retorna NULL em caso de falha na alocação
   }
   return vetor; // Retorna o ponteiro para o primeiro elemento do vetor
}
```

   (c) 4 bytes $\times\ n = 4n$.

3. (4 pontos) Implemente sua própria versão da função `char* strcpy(char *dest, const char *src)`, usada para copiar uma string de origem (`src`) para uma string de destino (`dest`). Assuma que já foi alocada memória para os endereços referenciados por `src` e `dest`.

   Observação: a única função da biblioteca `string.h` que pode ser utilizada para auxiliar no exercício é a `strlen`, utilizada para calcular o tamanho de uma string.

```c
char *my_strcpy(char *dest, const char *src) {
   for (int i = 0; i <= strlen(src); i++)
      dest[i] = src[i];
   return dest; // retorna a ponteiro para a string cópia
}
```

4. (6 pontos) Essa questão aborda um programa para gerenciamento de dados de alunos, onde cada aluno é definido por um nome (string de no máximo 100 caracteres), uma idade (número inteiro) e uma nota (número real). Baseado nesse descrição, faça o que se pede.

(a) (4 pontos) Complete a implementação da estrutura e das funções nos campos indicados (1 ponto para a estrutura, 2 pontos para cada função).

(b) (1 ponto) Qual a vantagem da passagem de parâmetro por referência (ponteiro), como ocorre com a estrutura `aluno`, em vez de passagem por valor?

(c) (1 ponto) Qual a utilidade da palavra reservada `const` em presente na assinatura da função `inicializar_aluno`?

Código 1: Código base (questão 1).

```c
#include <stdio.h>
#include <string.h>

// Definição da struct aluno
typedef struct {
   // COMPLETAR
} aluno;

/*
   Inicializa os dados de um aluno com o nome, idade e nota fornecidos.
   Parâmetros:
   - a: Ponteiro para a estrutura aluno que será inicializada.
   - nome: String contendo o nome do aluno.
   - idade: Idade do aluno.
   - nota: Nota do aluno.
 */
void inicializar_aluno(aluno *a, const char *nome, int idade, float nota) {
   // COMPLETAR
}

/*
   Imprime na saída padrão os dados de um aluno (nome, idade e nota).
   Parâmetros:
   - a: Ponteiro constante para a estrutura aluno cujos dados serão impressos.
 */
void imprimir_aluno(const aluno *a) {
   // COMPLETAR
}

int main() {
   aluno a;

   // Inicializa o aluno com os dados fornecidos
   inicializar_aluno(&a, "João Silva", 20, 8.5);
   // Imprime os dados do aluno
   imprimir_aluno(&a);

   return 0;
}
```

(a)

```c
#include <stdio.h>
```

```c
#include <string.h>

// Definição da struct aluno
typedef struct {
   char nome[100];
   int idade;
   float nota;
} aluno;

/*
   Inicializa os dados de um aluno com o nome, idade e nota fornecidos.
   Parâmetros:
   - a: Ponteiro para a estrutura aluno que será inicializada.
   - nome: String contendo o nome do aluno.
   - idade: Idade do aluno.
   - nota: Nota do aluno.
 */
void inicializar_aluno(aluno *a, const char *nome, int idade, float nota) {
   strcpy(a->nome, nome);
   a->idade = idade;
   a->nota = nota;
}

/*
   Imprime na saída padrão os dados de um aluno (nome, idade e nota).
   Parâmetros:
   - a: Ponteiro constante para a estrutura aluno cujos dados serão impressos.
 */
void imprimir_aluno(const aluno *a) {
   printf("Nome: %s\n", a->nome);
   printf("Idade: %d\n", a->idade);
   printf("Nota: %.1f\n", a->nota);
}

int main() {
   aluno a;

   // Inicializa o aluno com os dados fornecidos
   inicializar_aluno(&a, "João Silva", 20, 8.5);

   // Imprime os dados do aluno
   imprimir_aluno(&a);

   return 0;
}
```

(b) Passar um ponteiro para uma estrutura como `aluno` em vez de uma cópia completa da estrutura, evita-se o custo computacional associado à duplicação de dados, especialmente útil para estruturas grandes. As funções podem modificar diretamente os dados da estrutura original, eliminando a necessidade de retornar uma nova estrutura com valores modificados.

(c) A palavra `const` serve para garantir da imutabilidade do parâmetro `nome`, ou seja, a função

não modificará o conteúdo apontado por nome.

[Extra] O uso desta palavra reservada promove a segurança e a integridade do código, evitando alterações acidentais nos dados que deveriam ser tratados como somente leitura. Além disso, documenta claramente a intenção do código.