CENTRO DE MATEMÁTICA, COMPUTAÇÃO E COGNIÇÃO – UFABC
MCTA028-15 – Programação Estruturada – 3º Quadrimestre de 2024
http://professor.ufabc.edu.br/~m.sambinelli/courses/2024Q3-PE/index.html

Laboratório 4

Atenção: de agora em diante está terminantemente **proibido** usar arrays estáticos de tamanho variado como os exibidos abaixo:

```
int n;
int vetor[n]; // nunca definir a dimensão de um array com uma variável
```

Essa proibição aplica-se a: listas, trabalho e avaliação. Qualquer programa usando o tipo de construção acima será severamente penalizado. Todo array dinâmico deverá ser construído usando alocação dinâmica de memória:

```
int* vetor = calloc(n, sizeof(int)); // ou
int* vetor2 = malloc(n * sizeof(int));
```

INSTRUÇÕES

- Em todos os programas desenvolvidos, é indispensável o gerenciamento adequado da memória, garantindo a liberação de toda a memória alocada ao término da execução. Para verificar a presença de vazamentos de memória em seu programa, utilize o comando

  ```
  valgrind --leak-check=full /caminho/para/o/seu/programa
  ```

  Para que o comando acima funcione corretamente, você deve compilar o seu programa com a *flag* de *debug* habilitada. O exemplo a seguir ilustra como fazer tal compilação com o *gcc*.

  ```
  gcc -Wall -Wextra -Wvla -g -std=c99 arquivo.c
  ```

- Em diversos exercícios, será solicitado que vocês implementem uma função de um tipo específico. Além de implementar essa função, é imprescindível desenvolver uma função `main()` que a utilize com dados fornecidos pelo usuário. Em outras palavras, sua função `main()` deverá solicitar a entrada de dados ao usuário e repassá-los como parâmetros para a função desenvolvida.

Questão 1. Escreva um programa que, dado um inteiro `n` fornecido pelo usuário, leia `n` números inteiros fornecidos pelo usuário e compute a média desses números.

Questão 2. Escreva um programa que, dado um inteiro `n` fornecido pelo usuário, leia `n` caracteres fornecidos pelo usuário e as armazene em um vetor. Na sequência, para cada uma das `n` entradas armazenadas no vetor, o seu programa deve imprimir a seguinte linha: "`entrada xxx, conteudo yyy, endereco: zzz`", onde `xxx` deve ser substituído pelo índice da entrada, `yyy` pelo conteúdo dessa entrada no array e `zzz` pelo endereço de memória dessa entrada em decimal.

Questão 3. Escreva a função `swap(a, b)`. Essa função recebe como parâmetro dois inteiros `a` e `b` e não tem nenhum retorno. O comportamento dessa função deve ser o seguinte: após a execução da mesma, os valores de `a` e `b` devem estar trocados, como ilustra o exemplo a seguir.

```c
int a = 5;
int b = 7;
swap(a, b);
printf("a: %d\n", a); // imprime: 'a: 7'
printf("b: %d\n", b); // imprime: 'b: 5'
```

Questão 4. Escreva a função `inc(x)`. Essa função recebe apenas um parâmetro do tipo inteiro e não tem retorno. Seu comportamento é o seguinte: após a execução de `inc(x)`, o valor de `x` está incrementado em uma unidade, como ilustra o exemplo a seguir.

```c
int a = 5;
inc(a);
printf("%d\n", a); // imprime 6
```

Questão 5. Escreva a função `swap(a, b)`. Essa função recebe como parâmetro dois arrays de inteiros `a` e `b` e não tem nenhum retorno. O comportamento dessa função deve ser o seguinte: após a execução da mesma, os valores de `a` e `b` devem estar trocados, como ilustra o exemplo a seguir. **Você deve realizar essa tarefa apenas usando manipulação de ponteiro, não copie os elementos de um vetor no outro!**

```c
int a[3] = {5, 6, 7};
int b[4] = {1, 2, 0, 8};
swap(a, b);
// conteúdo de a = {1, 2, 0, 8}
// conteúdo de b = {5, 6, 7}
```

Questão 6. Implemente a função `strcat(a, b)`. Essa função recebe dois parâmetros `a` e `b` do tipo "string em C", i.e., array de caracteres, e retorna uma string que é a concatenação da string a com a string b. Além disso, a sua função não deve alterar o conteúdo das variáveis `a` e `b`, que ainda podem ser úteis para o cliente da sua função.

Questão 7. Escreva a função `cartesiano(v1, n1, v2, n2)`. Essa função recebe como parâmetro dois vetores de inteiro, `v1` e `v2`, e os seus respectivos tamanhos, `n1` e `n2`. O retorno dessa função deve ser um array com os elementos do conjunto obtido pelo produto cartesiano de `v1` por `v2`.

Questão 8. Implemente a função `char* revert(char* s)`. Essa função recebe uma "string de C" s e retornar uma nova "string de C" com os caracteres de s revertidos, i.e., listados da direita para a esquerda.

Questão 9. Implemente as funções:

- `matriz_le(n, m)` essa função lê e retorna uma matriz `M` de números inteiros fornecida pelo usuário. Ela recebe dois parâmetros `n` e `m`, onde `n` é o número de linhas e `m`, o de colunas. Essa função é responsável por alocar a memória para `M` e por ler todo o conteúdo de `M`, que deverá ser fornecido pelo usuário.
- `matriz_print(M, n, m)` essa função imprime na tela do usuário a matriz `M` que tem `n` linhas e `m` colunas.

Questão 10. É muito comum que linguagens tenham um tipo de array dinâmico transparante para o usuário. Vamos adicionar tal funcionalidade à C para vetores de inteiros. Primeiro, a definição do tipo:

```
typedef struct {
  int*  data;
  int capacity;  // capacidade do 'array' data
  int nelements; // número de elementos guardados em data
}* VectorInt;
```

Para esse TAD, implemente as seguintes funções:

- `VectorInt vectorint(void)`: cria um elemento do tipo VectorInt no qual data tem capacidade 1. O ponteiro para esse objeto é o retorno dessa função.
- `void vectorint_insert(VectorInt v, int a)`: insere o inteiro `a` no vetor dinâmico `v`. Se não houver espaço em `data` para inserir o elemento `a`, então você deve alocar um novo vetor para data com o dobro da capacidade usada para o vetor `data` atual, copiar os valores do vetor `data` antigo para o novo, e atualizar o endereço do ponteiro `data` no registro `v`.
- `void vectorint_remove(VectorInt v, int a)` remove todas as ocorrências do elemento `a` do vetor `v`. Se, após a remoção, o número de elementos em `data` for menor do que a metade da capacidade do array `data`, então você deve criar um novo array `data` com a metade da capacidade do atual, copiar os valores do array `data` antigo para o novo, e atualizar o endereço do vetor `data` no registro `v` (fazemos isso para evitar desperdício de memória).
- `int vectorint_at(VectorInt v, int i)` retorna o elemento que está na i-ésima entrada do vetor.
- Crie funções adicionais para o seu TAD que sejam capazes de testar um elemento pertence à `VectorInt`, que retorne a quantidade de elementos no vetor, e que destrua (libere a memória) corretamente o vetor.

Questão 11. Crie um TAD `Turma` para armazenar o nome de todos os alunos de uma turma. Para manipular esse tipo, você deverá implementar as seguintes funções:

1. `turma_matricula(turma, nome)` essa função recebe o `nome` de um aluno a ser inserido na `turma`;
2. `turma_lista(turma)` essa função lista o nome de todos os alunos matriculados em `turma`.
3. `turma_jubila(turma, nome)` essa função remove o aluno `nome` da `turma`.

A sua função deve ser capaz de lidar com um número arbitrariamente grande de alunos, portanto, você não pode definir "um número máximo de alunos".

Questão 12. Modifique o seu TAD VectorInt para que ele seja um vetor dinâmico para qualquer tipo de dados. Para isso, precisaremos usar o tipo genérico `void`. Nossa estrutura ficará assim:

```
typedef struct {
  void**  data;
  int capacity;
  int nelements;
}* Vector;

Vector vector(void);
void vector_insert(Vector v, void* e);
void vector_remove(Vector v, void* e);
void* vector_at(Vector v, int i);
```

Para passar e utilizar valores para as funções do seu TAD, você precisará ficar convertendo os tipos de ponteiro (fazer cast), como mostram os exemplos a seguir:

```
int* a = malloc(sizeof(int));
*a = 10;

int* b = malloc(sizeof(int));
*b = 21;

Vector v = vector();

vector_insert(v, (void*) a);
vector_insert(v, (void*) b);


int *c = (int*) vector_at(v, 0);

vector_remove(v, (void*) a);
```

Outro exemplo:

```
typedef struct _ponto {
  int x;
  int y;
}* Ponto;

Ponto a = malloc(sizeof(_ponto));
a->x = 10; // a->x é equivalente a (*a).x
a->y = 11;

Vector v = vector();

vector_insert(v, (void*) a);
```

Questão 13. Seja `Aluno` a estrutura definida da seguinte forma:

```
typedef struct _aluno {
  int ra;
  char nome[1000];
  char sexo;
  int idade;

}* Aluno;
```

Implemente a função `char aluno_cmp(Aluno *a, Aluno *b)` que recebe dois ponteiros para a estrutura `Aluno` e returna 1 se o conteúdo dos alunos apontados por `a` e `b` é o mesmo e 0, caso contrário. Uma observação importante: não estou pedindo para comparar os endereços de `a` e `b`, mas sim os conteúdos desses objetos.

Questão 14. Muitas linguagens possuem uma função chamada `map`, que é muito útil. Essa função geralmente tem a seguinte assinatura `map(v, func)`, onde `v` é uma coleção de itens e `func` é uma função que será aplicada a cada um dos elementos dessa coleção. Em C, podemos implementar essa funcionalidade da seguinte maneira.

```
#include <stdio.h>

int square(int a) {
  return a * a;
```

```
}

int cube(int a) {
  return a * a * a;
}

// eu sei que pode parecer esquisito, mas isso é a definição de uma
// variável `func` que é do tipo "função que recebe um parâmetro inteiro
// e tem um retorno inteiro" -----|
//                                |
//                                v
void map(int array[], int n, int func(int)) {
  for (int i = 0; i < n; i++)
    array[i] = func(array[i]);
}

int main(int argc, char *argv[]) {
  int vet[5] = {1, 2, 3, 4, 5};

  map(vet, 5, square);

  for (int i = 0; i < 5; i++)
    printf("%d\n", vet[i]);


  int vet2[5] = {1, 2, 3, 4, 5};
  map(vet2, 5, cube);
  for (int i = 0; i < 5; i++)
    printf("%d\n", vet2[i]);

  return 0;
}
```

Crie uma função que receba outra função como parâmetro.