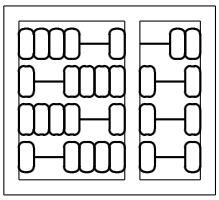


MC202 – Lista 1
Revisão do Conteúdo
Instituto de Computação — UNICAMP
Prof.: Anderson Rocha
anderson.rocha@ic.unicamp.br

Observação I

Os exercícios de 1 a 9 são de aquecimento.

Exercício 1 - Ordem crescente

Escreva uma função recursiva que mostra, em ordem crescente, os números do intervalo $[1, n]$.

Exercício 2 - Menor elemento

Faça uma função recursiva que retorna a posição do menor elemento de um vetor.

Exercício 3 - Intercalação de vetores

Implemente uma função que intercala dois vetores ordenados. Exemplo:

```
V1 = {1,3,5,7,11}
V2 = {2,4,8,12}
Resultado = {1,2,3,4,5,7,8,11,12}
```

Exercício 4 - Arquivos I

Crie manualmente um arquivo, ‘‘`dados.txt`’’, com o seguinte conteúdo:

```
 8.1  7.5  9.9
 3.5  9.1  7.5
 0.1  0.2  0.3
10.0 10.0  0.5
```

Faça um programa que abre este arquivo e mostra cada um dos números armazenados nele.

Exercício 5 - Arquivos II

Usando o arquivo ‘‘`dados.txt`’’ do exercício anterior, escreva um programa que abre o arquivo, calcula a média dos números da mesma linha (basta calcular a média 3 a 3), e grava o resultado em um arquivo ‘‘`medias.txt`’’, como no exemplo abaixo:

```
 8.1  7.5  9.9  -  8.3
 3.5  9.1  7.5  -  6.7
 0.1  0.2  0.3  -  0.2
10.0 10.0  0.5  -  6.8
```

Exercício 6 - Três somas

Implemente uma função que recebe três números `A`, `B` e `C` e retorna o seguinte: `A` = `B` + `C` `B` = `A` + `C` e `C` = `A` + `B`. Observe que a função deve necessariamente alterar os valores das variáveis (utilizar ponteiros).

Exercício 7 - Produtório

Escreva uma função que recebe um vetor de dez elementos e calcula o produto de todos os seus elementos.

Exercício 8 - Pares e ímpares

Leia dez números inteiros, armazene-os em um vetor e depois conte o número de pares e ímpares e mostre estas quantidades na tela.

Exercício 9 - Fatorial se primo

Escreva um programa que calcula o fatorial de um número inteiro n ($n > 0$), digitado pelo usuário, somente se este número é primo.

Exercício 10 - Operações com Matrizes I

Escreva um programa em C que lê uma matriz 5×5, e executa as seguintes operações:

1. Calcula e mostra a soma dos elementos da diagonal principal.
2. Mostra o menor e maior elemento da diagonal principal.
3. Calcula e mostra a transposta da matriz.

Exercício 11 - Operações com Matrizes II

Defina a estrutura

```
#define MAX 10

typedef struct matriz{
    int n_lin;
    int n_col;
    float m[MAX][MAX];
} matriz;
```

e escreva funções para ler, somar, subtrair e multiplicar matrizes de tamanho arbitrário. Observe o exemplo abaixo de como inicializar e mostrar estas matrizes:

```
matriz A = {3, 3, {{1, 2, 3},
                   {4, 5, 6},
                   {7, 8, 9}}
           };
for (i = 0; i < A.n_lin; ++i) {
    for (j = 0; j < A.n_col; ++j)
        printf("%.1f ", A.m[i][j]);
    printf("\n");
}
```

Exercício 12 - Lista ligada

Considere a seguinte declaração para os nós de uma lista ligada:

```
typedef struct refnode {
             int valor;
             struct refnode *prox;
       } tipoelemento, *tipolista, *ap_no;
```

(a) Escreva uma função que remove o k-ésimo nó de uma lista ligada.

(b) Escreva uma função que remove um elemento com valor v de uma lista ligada ordenada. Caso v não pertença à lista, uma mensagem de erro deve ser exibida.

(c) Escreva uma função que recebe duas listas ligadas ordenadas $x = (x_1, ..., x_n)$ e $y = (y_1, \ldots, y_m)$ como parâmetros e retorna uma lista formada pelos elementos de x e y intercalados.

(d) Escreva uma função que concatena duas listas encadeadas.

(e) Escreva uma função que copie uma lista encadeada para um vetor.

(f) Escreva uma função que verifica se duas listas ligadas dadas são iguais.

(g) Escreva uma função **não recursiva** que retorna um apontador para a lista p invertida. Nenhum nó adicional deve ser criado.

```
ap_no inverte(ap_no p);
```

(h) Escreva uma função que remove um elemento com valor v de uma lista ligada ordenada. Caso v não pertença à lista, uma mensagem de erro deve ser exibida.

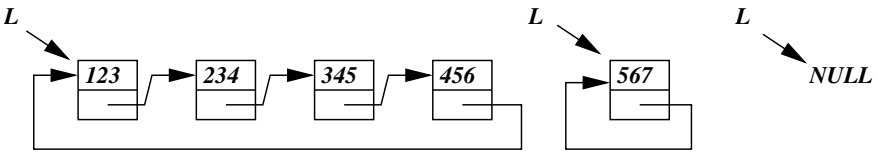
```
void remove(ap_no* p, int v);
```

Por que um apontador para `ap_no` é passado como parâmetro? Se a lista tivesse um nó cabeça, poderíamos passar apenas um ap_no?

(i) Faça uma rotina **recursiva** com dois parâmetros. No primeiro parâmetro deve ser enviado uma lista definida com o tipo acima. O segundo parâmetro receberá uma cópia desta lista contendo apenas os elementos de valor par.

Exercício 13 - Lista ligada circular

Uma lista ligada circular é parecida com uma lista ligada simples. Ela também é identificada por uma variável apontando para o primeiro elemento, mas a diferença é que o último elemento, em vez de apontar para *NULL*, aponta para o primeiro elemento. Use a definição de mesmo tipo/estrutura apresentada na questão anterior. Obs.: Se a lista está vazia, a variável aponta para *NULL*. Se a lista só tem um elemento, o campo `prox` aponta para o próprio elemento. Segue uma figura exemplificando a representação da lista circular.



(a) Faça uma rotina para inserir um novo elemento no início da lista ligada circular.

(b) Faça uma rotina para remover o último elemento da lista ligada circular.

Exercício 14 - Cadeias de caracteres I

Faça uma função, `strdel`, que elimina todas as ocorrências de uma dada cadeia, dentro de outra cadeia. A função deve receber uma cadeia auxiliar, onde armazenará o resultado, a cadeia original, e a cadeia a ser retirada. Um exemplo de uso seria:

```
int main(){
    char resultado[100];
    char original[] = "Lá está o velho Sócrates com seu velho livro.";
    char apagar[] = " velho";

    strdel(resultado, original, apagar);
    printf("%s\n", resultado);

    return 0;
}
```

Deve mostrar na tela:

```
Lá está o Sócrates com seu livro.
```

Exercício 15 - Cadeias de caracteres II

Escreva uma função que verifique se uma *string* de entrada é da forma

`str_1Cstr_2`

tal que str_1 é uma *string* composta apenas por caracteres A e B e str_2 é a *string* reversa de str_1. Por exemplo, a cadeia ABABBACABBABA é do formato especificado. A *string* de entrada deve ser percorrida uma única vez da esquerda para a direita!

```
int st1Cstr2(char* str);
```

Exercício 16 - Cadeias de caracteres II

Escreva uma função que verifique se uma *string* de entrada formada apenas por '(' e ')' está balanceada. É necessária a utilização de uma pilha para resolver este problema?

```
int balanc_parenteses(char* str);
```

Observação II

Para os dois próximos exercícios, você só pode utilizar uma pilha ou uma fila através das funções definidas na interface destes tipos abstratos (por exemplo, as funções `cria_pilha`, `pilha_vazia`, `empilha`, `desempilha`, `destroi_pilha` no caso de uma pilha; ou as funções `cria_fila`, `fila_vazia`, `enfileira`, `desenfileira`, `destroi_fila` no caso de uma fila). Não faça nenhuma suposição a respeito das implementações ou acesso direto a um campo de estrutura.

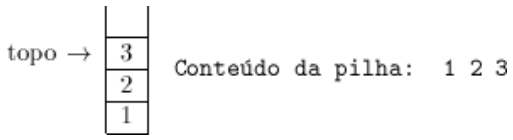
Exercício 17 - Filas

Escreva uma função que inverte os elementos de uma fila usando uma pilha.

```
void inverte_fila(Fila *fila);
```

Exercício 18 - Pilhas I

Escreva uma função que imprime os elementos de uma pilha na mesma ordem em que eles foram empilhados, como no exemplo abaixo:

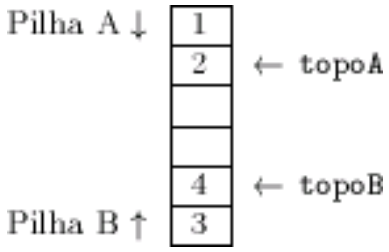


O conteúdo da pilha deve ser o mesmo antes e depois da execução da função. Pode ser utilizada uma pilha auxiliar.

```
void imprime(Pilha* pilha);
```

Exercício 19 - Pilhas II

Duas pilhas A e B podem compartilhar o mesmo vetor, como esquematizado na figura abaixo.



Faça as declarações de constantes e tipos necessárias e escreva as seguintes rotinas:

(a) cria_pilhas, que inicia os valores de topoA e topoB;

(b) vazia_A e vazia_B;

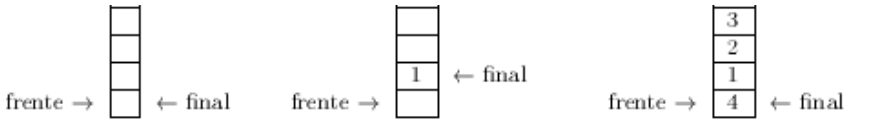
(c) empilha_A e empilha_B;

(d) desempilha_A e desempilha_B.

Obs.: Só deve ser emitida uma mensagem de pilha cheia se todas as posições do vetor estiverem ocupadas.

Exercício 20 - Pilhas III

Por que na implementação de filas em vetor este é, em geral, circular? Considere que estamos utilizando apontadores (índices) para a frente e o final da fila: frente aponta para a posição imediatamente anterior ao primeiro elemento da fila e final aponta para o último elemento inserido, se existir. Comente a dificuldade para se diferenciar fila cheia de fila vazia.

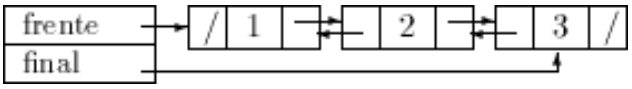


(a) Fila vazia (b) Fila com um elemento (c) Fila com quatro elementos

Implemente as funções cria_fila, fila_vazia, insere_fila e remove_fila utilizando um *flag* (variável booleana) para indicar se a fila está cheia ou vazia.

Exercício 21 - Fila simétrica

Uma fila simétrica permite a realização das operações de inserção e remoção em ambas as extremidades. Considere uma implementação em que o tipo `Fila_Simetrica` é uma estrutura que contém apontadores para o primeiro e o último elemento da fila, sendo esta implementada por meio de uma lista duplamente ligada. Implemente as funções cria_fila, enfileira, desenfileira e destroi_fila.




```
typedef struct no* ap_no;

struct no {
```

```
  int v;
  ap_no ant, prox;
};

struct fila_simetrica {
  ap_no frente;
  ap_no final;
};

typedef struct fila_simetrica Fila_Simetrica;
```

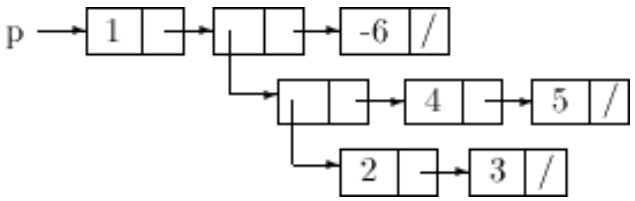
Exercício 22 - Lista generalizada

Considere a seguinte declaração para os nós de uma lista generalizada.

```
enum tipo_no {tipo_atomo, tipo_sublista};

typedef struct no* ap_no;

struct no {
   enum tipo_no tipo;
   union {
     int atomo;
     ap_no sublista;
   } info;
   ap_no prox;
};
```



(a) Escreva uma função que inicializa uma lista generalizada como uma lista vazia.

(b) Escreva uma função que determina se uma lista generalizada está vazia.

(c) Escreva uma função que retorne o tipo de um elemento de uma lista generalizada (isto é, se ele é lista ou átomo).

(d) Escreva uma função que insira um átomo no início de uma lista generalizada.

(e) Escreva uma função que retorna a lista obtida removendo-se o primeiro elemento (a cabeça) de uma lista generalizada – isto é, o ponteiro para a cauda.

(f) Escreva uma função que soma o valor dos átomos de uma lista generalizada.

(g) Escreva uma função que determina se duas listas generalizadas são iguais.

(h) Escreva uma função que exibe o conteúdo de uma lista generalizada na forma (1, ((2, 3), 4, 5), -6). Seja cuidadoso ao posicionar os parênteses e vírgulas.

(i) Escreva uma função que libera os nós de uma lista generalizada.

Exercício 23 - Estruturas

Dada a estrutura `Ponto` a seguir, defina as funções associadas deste tipo abstrato de dados:

```
typedef struct ponto{
   float x,y;
}Ponto;

float dist(Ponto, Ponto);
float soma(Ponto, Ponto);
int colinear(Ponto, Ponto, Ponto); /* Use tolerancia de 10^-4 */
```

Exercício 24 - Estruturas

Dada a estrutura `Ponto` acima, defina uma estrutura `Triangulo` as funções associadas deste tipo abstrato de dados

```
float area(Triangulo);
float perimetro(Triangulo);
```

Exercício 25 - Vetores

Implemente o algoritmo *Crivo de Eratóstenes* para verificação dos números primos de 1 a N. Faça as devidas consultas para tal exercício.

Exercício 26 - Vetores

Escreva um programa que verifique o número de inteiros diferentes entre 1 e N que aparecem em uma sequência fornecida pelo usuário.

Exercício 27 - Vetores

Escreva um programa que determina empiricamente o número de inteiros positivos pseudo-aleatórios menores que N que você espera gerar antes de receber um valor repetido.

Exercício 28 - Listas

Escreva um programa que retorna o número de nós em uma lista circular, dado um ponteiro para qualquer dos nós desta lista.

Exercício 29 - Listas

Escreva uma função que, dados dois ponteiros `x` e `t` para duas listas circulares disjuntas, insira a lista apontada por `t` na lista apontada por `x` no ponto seguinte a `x`.

Exercício 30 - Listas e Ordenação

Implemente o algoritmo de ordenação por inserção para listas ligadas. Faça as suposições que achar necessárias.

Exercício 31 - Listas

Implemente um algoritmo que reverte a ordem dos elementos de uma lista ligada em apenas uma passagem pela lista.

Exercício 32 - Listas

Escreva uma função que move o maior elemento de uma lista para a última posição desta.

Exercício 33 - Listas

Escreva uma função que coloca todos os nós pares de uma lista de inteiros após os nós ímpares.

Exercício 34 - Listas

Escreva uma função que troca dois nós `x` e `t` em uma lista duplamente encadeada. Faça as suposições que achar necessárias.

Exercício 35 - Listas

Escreva uma função que libera todos os nós de uma dada lista.

Exercício 36 - Listas

Escreva uma função que libera os nós de uma lista de acordo com uma lista de índices passada por parâmetro.

```
void libera_posicoes(Lista l, Lista posicoes);
```

Exercício 37 - Listas

Escreva uma função que receba uma lista e uma função *callback* que calcula o fatorial de um número. Retorne uma lista com os fatoriais calculados.

Exercício 38 - Pilhas

Considere que uma letra signifique a operação de *Push* e um asterisco uma operação de *Pop*. Na sequência a seguir, dê a sequência de valores retornados pela operação de *Pop*.

```
SEQU*E*NCIA***F*A*CI**L*****
```

Exercício 39 - Pilhas

Dadas duas sequências, apresente um algoritmo capaz de determinar se asteriscos podem ou não ser adicionados para fazer a primeira sequência produzir a segunda a partir de uma sequência de *pops*. Exemplo:

```
Sequencia 1: FACIL
Sequencia 2: LICAF

Solucao: FACIL*****
```

Exercício 40 - Pilhas

Escreva uma função que, ao receber uma sequência de operações (*, +) entre inteiros, identifique se esta sequência está em notação pré-fixa ou pós-fixa e faça a conversão para a outra notação. Se pós-fixo converta para pré-fixo e vice-versa.

Exercício 41 - Filas

Considere que uma letra signifique a operação de *insercao* e um asterisco uma operação de *remoção* na sequência a seguir, dê a sequência de valores retornados pela operação de *remoção*.

```
SEQU*E*NCIA***F*A*CI**L*****
```

Exercício 42 - Filas

Considere que uma letra maiúscula signifique a operação de *Inserção* no início, uma minúscula inserção no final, um sinal de adição (+) uma operação de *remoção* do início e um asterisco uma operação de *Remoção* do final. Na sequência a seguir, dê a sequência de valores retornados pelas operações de *Remoção*.

```
SEQU*E*NCIA***F*A*CI**L*****
```

Exercício 43 - Filas

Dadas duas sequências, apresente um algoritmo capaz de determinar se asteriscos e adições podem ou não ser adicionados para fazer a primeira sequência produzir a segunda a partir de uma sequência de *Remoções*. Exemplo:

```
Sequencia 1: FAciL
Sequencia 2: ciFAL

Solucao: FAciL*****
```

Exercício 44 - Filas

Defina um tipo abstrato de dados *Fila* (dados e funções associadas) que impeça a inserção de elementos repetidos.

Exercício 45 - Listas

Escreva uma função para a representação de polinômios via listas ligadas. Cada nó deve conter o coeficiente do monômio e o expoente. De acordo com esta estrutura, faça uma função que receba dois polinômios e retorne o polinômio soma. Trate possíveis monômios ausentes.

Exercício 46 - Listas

Modifique os algoritmos do exercício anterior para efetuar multiplicação de polinômios.

Exercício 47 - Recursão – Aquecimento para Árvores

Escreva uma função que calcule a função `N! % M`, onde % é a função módulo.

Exercício 48 - Recursão – Aquecimento para Árvores

Escreva um algoritmo recursivo capaz de avaliar uma expressão em notação pós-fixa.

Exercício 49 - Recursão – Aquecimento para Árvores

Escreva uma função recursiva para achar o maior elemento de um vetor.

Exercício 50 - Recursão – Aquecimento para Árvores

Escreva uma função que calcule a função `log(N!)`

Exercício 51 - Árvores

Escreva um programa que calcule o número de folhas em uma árvore que não estejam no nível mais baixo desta. Exemplo: em uma árvore binária completa, a resposta seria zero pois todos os nós folha estão no nível mais profundo desta árvore. Em uma árvore binária cujo filho esquerdo é degenerado em uma lista e o filho direito não tem filhos, a resposta seria 1.

Exercício 52 - Árvores

Escreva um programa que calcule o número de nós em uma árvore binária que contém um filho interno e um filho externo.

Exercício 53 - Árvores

Escreva um programa que calcule a altura de uma árvore ternária.

Exercício 54 - Árvores

Nas questões abaixo, considere a seguinte estrutura de dados usada para representar árvores binárias.

```
typedef struct No{
  int valor;
  struct No *dir;
  struct No *esq;
} Arvore;
```

(a) Escreva uma função em C que determina a altura de uma árvore binária.

```
int altura (Arvore *arv);
```

(b) Escreva uma função em C que libera todos os nós de uma árvore binária.

```
void liberaArvore (Arvore **arv);
```

(c) Escreva uma função em C que conta o número de nós do tipo folha.

```
int contaFolhas(Arvore *arv);
```

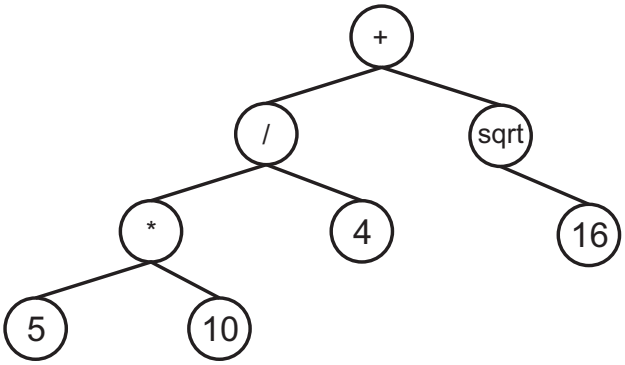
(d) Uma árvore é equilibrada se, para cada nó, o número de nós da sub-árvore esquerda é igual ao número de nós da sub-árvore direita. Escreva uma função em C que determina se uma árvore binária é equilibrada.

(e) Considere o uso de árvores binárias para representar expressões matemáticas (veja a estrutura e a figura a seguir). Escreva uma função que avalia uma árvore.

```
typedef enum {tipo_char, tipo_float} tipo_no;

typedef struct No {
  union {
    float valor;
    char oper;
  }   info;
  tipo_no tipo;
  struct No *dir;
  struc No *esq;
} ArvoreMat;

float avalia(ArvoreMat *arv);
```



(f) Considere a estrutura definida na questão anterior. Escreva uma função que recebe uma expressão matemática em notação pós-fixa e constrói uma árvore binária.

```
ArvoreMat *constroiArvoreMat(char *posfixa);
```

(g) Escreva uma função em C que constrói uma árvore binária a partir de percursos *In*-ordem e Pré-ordem armazenados em vetores de tamanho n.

```
Arvore *constroiArvore(int *inordem, int *preordem, int n);
```

(h) Escreva uma função em C que faça um percurso em profundidade do tipo Pós-ordem sem usar recursão.

```
void posordem(Arvore *arv);
```

Observação III

Além dos exercícios propostos nesta lista, seria interessante a resolução dos exercícios do livro abaixo:

- *Introdução a Estruturas de Dados*. Waldemar Celes, Renato Cerqueira e José Lucas Rangel. Editora Campus, 2004.

  1. Exercícios 1 a 6, páginas 114–119.
  2. Exercícios 1 a 5, páginas 214–220.