Mecanismos construtores de tipos

1) Faça um programa que recebe os detalhes da venda de produtos e determina o valor da venda. Cada produto tem um código numérico do tipo inteiro, uma descrição de até 20 caracteres, a unidade de medida (que pode ser UN, KG, LT ou MT), a quantidade disponível em estoque (é um inteiro se a unidade de medida for UN e é um real nos outros casos) e o preço unitário correspondente. Criar um *array* que permita armazenar os dados de 25 produtos e carregar esse array com dados. Em seguida, receber os detalhes de uma venda (código e quantidade de cada produto vendido naquela transação e imprimir o valor total da venda). Usar uma `struct` para descrever o produto, uma `enum` para as unidades de medida e uma `union` para a quantidade do produto.

2) É dada a estrutura de dados a seguir, que permite alocar um dado de 4 bytes e manipulá-lo tanto byte a byte como também em pares de bytes ou em um conjunto único de 4 bytes. Faça um programa que cria uma variável desse tipo e permite:

- **a)** Atribuir um valor a um byte específico da variável.
- **b)** Atribuir um valor a uma determinada sequência de dois bytes da variável.
- **c)** Atribuir, de uma só vez, um valor ao conjunto de 4 bytes.
- **d)** Consultar o conteúdo de um determinado byte da variável, imprimindo esse conteúdo tanto em formado decimal como hexadecimal e binário.
- **e)** Consultar o conteúdo de uma determinada sequência de dois bytes da variável, imprimindo esse conteúdo tanto em formado decimal como hexadecimal e binário.
- **f)** Consultar o conteúdo da variável como um todo, imprimindo esse conteúdo tanto em formado decimal como hexadecimal e binário.

```
#define DWORD unsigned int
#define WORD  unsigned short
#define BYTE  unsigned char
typedef union _DWORD_PART_ {DWORD dwWord;
                            struct {WORD dwMSB;
                                    WORD dwLSB;
                                   } hw;
                            struct {BYTE byMSB;
                                    BYTE byMSBL;
                                    BYTE byLSBH;
                                    BYTE byLSB;
                                   } b;
                           } DWORD_PART;
```

3) Fazer um programa que define um tipo de dados para cliente, considerando as especificações a seguir, e:

- **a)** Declara uma variável desse tipo.
- **b)** Recebe os dados de um cliente informado pelo usuário
- **c)** Exibe os dados informados na tela.

O item de dado para indicar o tipo do cliente ('P' para pessoa, 'E' para empresa) é uma enumeração. Não é possível um cliente ser simultaneamente “pessoa” e “empresa”.

Se cliente é do tipo “pessoa”, ele tem:

- nome, string de 40 caracteres úteis
- cpf, string de 14 caracteres úteis
- rg, string de 14 caracteres úteis
- sexo, char
- data de nascimento, composto por dia, mês e ano

Se cliente é do tipo “empresa”, ele tem:

- razão social, string de 50 caracteres úteis
- cnpj, string de 18 caracteres úteis
- inscrição estadual, string de 14 caracteres úteis

Todo cliente, seja ele “pessoa” ou “empresa”, tem um endereço, composto por:

- logradouro (nome da rua, avenida, rodovia, etc e respectivo número)
- complemento (indicativo do apartamento, casa ou equivalente)
- bairro (nome do bairro)
- cidade (nome da cidade)
- cep (código e 8 dígitos numéricos)
- estado (sigla com 2 caracteres alfabéticos maiúsculos)

Todo cliente tem um limite de crédito, que é um número real de precisão simples.

Todo cliente tem uma data de cadastramento, composta por dia, mês e ano.

TDA – Tipos Abstratos de Dados

4) Um ponto geométrico é um objeto descrito por duas coordenadas no plano cartesiano: sua ordenada (posição do ponto no eixo *y*) e sua abscissa (posição do ponto no eixo *x*). Neste exercício você deverá considerar que ordenadas e abscissas de um ponto são números reais. Faça uma implementação de TDA para suportar a representação de pontos geométricos e as seguintes operações:

a) Operações fundamentais de qualquer TDA: inicialização, inclusão e exclusão de elementos daquele tipo.

b) Quadrante de um ponto: para um ponto recebido como parâmetro a rotina retorna o inteiro 0 se o ponto corresponde à origem ($y = 0$ e $x = 0$); 1 se estiver no primeiro quadrante; 2 se estiver no segundo quadrante; etc. Consulte a figura 1 caso tenha dúvidas.

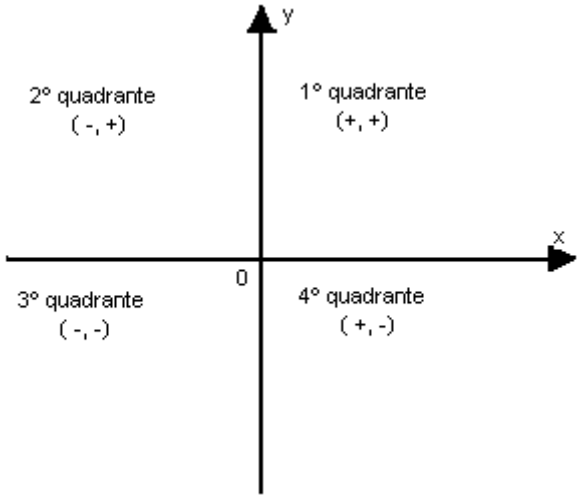


Figura 1: Quadrantes no plano cartesiano

c) Distância entre dois pontos: a rotina recebe dois pontos como parâmetro e determina a distância entre eles, conforme a figura 2.

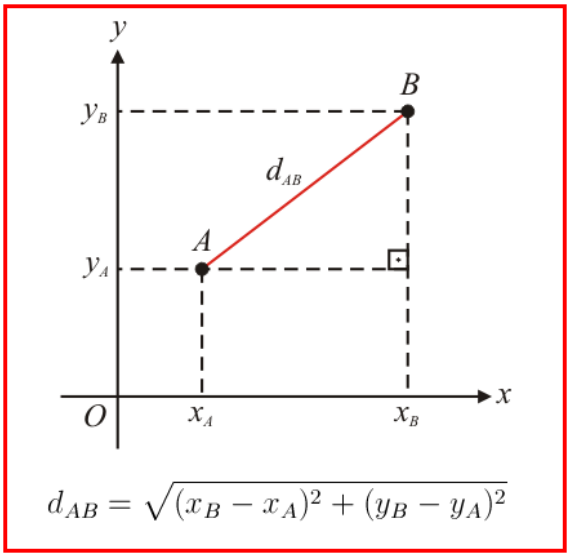


Figura 2: Distância entre dois pontos no plano

Faça depois um programa cliente que utiliza os recursos do TDA para determinar a distância entre dois pontos geométricos quaisquer informados pelo usuário e os respectivos quadrantes

de cada um. Depois adapte o programa para receber três pontos e determinar se com eles pode ser construído um triângulo.

5) Defina um TDA para representar datas no formato convencional gregoriano e suas operações básicas.

O TDA deverá permitir armazenar o conteúdo referente a dia, mês e ano de datas válidas no período de 3000 a.C. até pelo menos 3000 d.C. e oferecer operações que suportem:

a) Validação de datas: para uma data válida recebida, determinar se ela é válida ou não.

b) Verificação se um determinado ano é bissexto.

c) Aritmética de datas

a.1. Quantos dias existem entre duas datas informadas?

a.2. Quantos meses existem entre duas datas informadas?

a.3. Quantos anos existem entre duas datas informadas?

a.4. Se somarmos uma quantidade de dias a uma data, que nova data teremos?

d) Dia da semana correspondente a uma data válida informada

Listas lineares

6) Para se determinar o número de lâmpadas necessárias para cada cômodo de uma residência, existem normas que dão o mínimo de potência de iluminação exigida por metro quadrado (m^2) conforme a utilização desse cômodo. Seja a seguinte tabela tomada como exemplo:

Utilização	***Classe***	***Potência / m^2***
Quarto	1	15
Sala de TV	1	15
Salas	2	18
Cozinha	2	18
Varandas	2	18
Escritório	3	20
Banheiro	3	20

Faça um programa que recebe e armazena em uma lista encadeada o nome do cômodo, a largura, o comprimento, a classe e a potência por m^2 requerida. Exiba o conteúdo da lista. Depois, em um *looping*, receba do usuário qual a potência das lâmpadas a serem usadas e determine:

a) Para cada cômodo:

a.1. O cômodo;
a.5. A área do cômodo;
a.6. Número de lâmpadas necessárias.

b) Para toda a residência:

b.1. Total de lâmpadas;

b.2. Total de potência requerida.

Encerrar o programa quando o usuário informar uma potência de lâmpada ≤ 0.

7) Com base no algoritmo a seguir, faça o que se pede nos itens a), b) e c)

```
tipo noh :: reg( dado: int; prox: ref noh);
var prim : ref noh;   vr : int ;
início
    prim ← NULL ; vr ← 0 ;
    faça enquanto vr != -1
    início
        Receba( vr ) ;
        Se valor != -1 então CriaNo( vr ) ;
    fim
    Escreva( 'Informe um número:' );  Receba( vr ) ;
    Escreva( 'Existem na lista ', ContaNo( vr ), ' números maiores que o valor informado' );
fim
```

a) Crie a rotina ContaNo, que recebe como parâmetro um número e descobre quantos nós possuem dado maior que esse valor.

b) Escreva uma rotina chamada SOMA para retornar a soma dos valores positivos da lista.

c) Crie a rotina ImprimeLista que exibe na tela os valores de todos os elementos da lista.

8) Faça um programa que implementa uma lista linear por encadeamento simples que armazene palavras de até 15 caracteres. Utilize descritor estruturado e sub-rotinas para as operações

básicas de manutenção da lista. Cuide para que a inclusão de elementos na lista seja feita de maneira a mantê-la sempre em ordem alfabética. Após a inclusão de cada palavra, imprima a lista inteira.

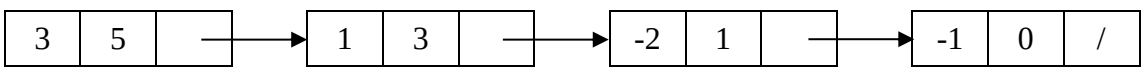
9) Crie uma versão do exercício anterior que implemente a lista por contiguidade.

10) Faça um programa que administra uma lista duplamente encadeada contendo, em cada elemento, o nome, a população e a área de uma cidade. A tela de entrada do programa deverá oferecer as opções 'Esvaziar lista', 'Incluir elemento pela cabeça', 'Incluir elemento pela cauda', 'Imprimir lista', 'Imprimir resumo', 'Excluir cidade' e 'Modificar cidade' e 'Encerrar'. Separar a implementação do cliente, utilizando um arquivo header para expor os recursos disponíveis na implementação. A seguir, uma rápida apresentação de cada opção:

- **'Esvaziar lista':** Excluir, por meio da função `free()` todos os elementos da lista e colocar nulo nos ponteiros de início e final da lista. Antes de excluir os elementos, solicitar confirmação do usuário. Após o processamento desta opção, retornar ao menu inicial.
- **'Incluir elemento pela cabeça':** Receber os valores referentes a nome, população e área digitados pelo usuário e colocá-los como primeiro elemento da lista. Para encerrar as entradas, o usuário deverá digitar 'XXX'. Após o processamento desta opção, retornar ao menu inicial.
- **'Incluir elemento pela cauda':** Receber os valores referentes a nome, população e área digitados pelo usuário e colocá-los como último elemento da lista. Para encerrar as entradas, o usuário deverá digitar 'XXX' para o nome da cidade. Após o processamento desta opção, retornar ao menu inicial.
- **'Imprimir lista':** Percorrer a lista a partir do início, imprimindo para cada cidade o seu nome, população e área. Após a impressão, aguardar o pressionamento de uma tecla, limpar a tela e retornar ao menu inicial.
- **'Imprimir resumo':** Calcular e imprimir os seguintes totais: quantidade de elementos da lista, maior população, menor população, média da população das cidades constantes na lista. Após o processamento desta opção, retornar ao menu inicial.
- **'Excluir cidade':** Solicitar do usuário o nome da cidade a excluir. Pesquisá-la na lista e, caso seja encontrada, eliminá-la com `free()`, cuidando de refazer o encadeamento. Se a cidade informada não existir na lista, emitir uma mensagem de erro. Após o processamento desta opção, retornar ao menu inicial.
- **'Modificar cidade':** Solicitar do usuário o nome da cidade a modificar. Pesquisá-la na lista e, caso seja encontrada, exibir seus dados (nome, população e área) na tela e permitir que o usuário os modifique, regravando-os na lista. Se a cidade informada não existir na lista, emitir uma mensagem de erro. Após o processamento desta opção, retornar ao menu inicial.

11) Crie uma versão para o exercício anterior com implementação da lista por contiguidade.

12) Polinômios podem ser representados por meio de listas, cujos nós são registros com 3 membros: coeficiente, expoente e referência ao seguinte. Por exemplo, o polinômio $3x^5 + x^3 - 2x - 1$ seria representado por



Faça um programa que recebe um polinômio de uma única variável (no caso será a variável *x*) e o armazena em uma lista encadeada, conforme ilustrado no exemplo acima. Em seguida, o programa deve receber uma sequência de valores para *x* e, para cada um desses valores, calcula e exibe na tela o resultado correspondente à substituição de *x* no polinômio dado. Encerrar quando for informado para *x* o valor -999.

Exemplos:

Valores informados	*Resultado a ser exibido*
$3x^5 + x^3 - 2x - 1$	
0	-1
1	1
2	99
3	749

Valores informados	*Resultado a ser exibido*
$5x^3 - 3x^2 + x - 25$	
-1	-34
3	86
8	2351
25	76250

13) Uma aplicação controla a atividade de uma sala de bate-papo *online*, utilizando uma lista encadeada para registrar os indivíduos participantes. Cada participante é descrito pela estrutura de dados apresentada abaixo, que contém o endereço ip utilizado pelo indivíduo, o apelido (*nickname*) usado pela pessoa no *chat*, o momento de entrada no bate papo (momento de *login*) e o momento da saída (momento de *logout*). Quando alguém entra na sala de bate-papo um registro é colocado no final da lista, com todos os campos preenchidos exceto aquele referente ao momento de saída. Quando a pessoa sai da sala de bate papo o campo referente ao momento de saída é atualizado. Um momento é representado por um número real que indica tanto a data quanto a hora, e são obtidos da data e hora do sistema e convertidos por meio de uma fórmula própria cujos detalhes estão fora do escopo deste problema. A lista é controlada por meio de um descritor, cuja estrutura também é apresentada a seguir. Escreva as rotinas para:

a) Determinar qual a pessoa que ficou a maior quantidade de tempo no *chat*, entre um *login* e um *logout*. A rotina deverá receber o descritor da lista como parâmetro e retornar o *nickname* correspondente. Se mais de uma pessoa ficou a maior quantidade de tempo, retornar o *nickname* daquela que entrou primeiro no chat.

b) Registrar a desativação do bate papo. A rotina deverá receber como parâmetro o descritor da lista e o número real correspondente ao momento de fechamento da sala. Para cada participante ativo (que fez a entrada na sala, mas não fez a saída) atualizar seu campo referente ao momento de saída com o valor recebido como parâmetro.

c) Contar quantas pessoas estavam ativas na sala em um intervalo de tempo informado como parâmetro. A rotina deverá receber o descritor da lista, o número real referente ao início do período e o número real referente ao final do período, e retornar um inteiro representando quantas pessoas estiveram conectadas naquele intervalo.

d) Verificar se duas pessoas estiveram juntas simultaneamente na sala. Considere que elas estiveram juntas simultaneamente se houve pelo menos um instante em que ambas tinham entrado na sala, mas ainda não tinham saído. Por exemplo se uma pessoa A entrou no momento 40202.33 e saiu no momento 40202.55 e outra pessoa B entrou no momento 40202.50 e saiu no momento 40203.01, elas estiveram simultaneamente no *chat*. A rotina deverá receber como parâmetros o descritor da lista e os dois *nicknames* das pessoas a serem verificadas. Considere que cada pessoa possui apenas um registro na lista. Retornar 1 se as pessoas estiveram simultaneamente no *chat* e zero em caso contrário.

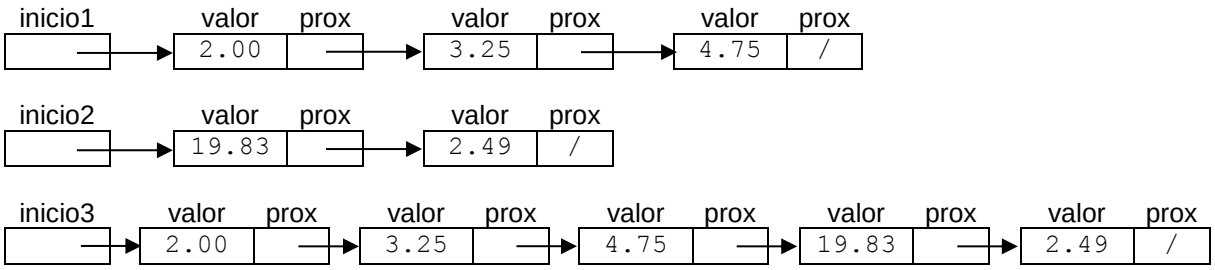
```
struct particip {char ip[16];
                 char nickname[21];
                 float login;
                 float logout;
```

```
                    struct particip* prox;};
typedef struct particip TParticip;
typedef struct {TParticip *inicio; TParticip *final;} TDescr;

TDescr descritor;
```

14) Um programa escrito em C constrói duas listas encadeadas simples utilizando os valores digitados pelo usuário. A primeira lista é referenciada pelo ponteiro `inicio1` e a segunda lista é referenciada pelo ponteiro `inicio2`. Escreva uma rotina que constrói uma terceira lista, referenciada pelo ponteiro `inicio3`, cujo conteúdo é a concatenação das duas primeiras listas, ou seja, uma lista cujos elementos iniciais provêm da primeira lista, e após estes, os elementos da segunda lista, conforme o exemplo abaixo. Escreva também uma rotina para imprimir a nova lista na tela. Considere as definições de tipos de dados e variáveis abaixo, que foram feitas fora da rotina `main()` do programa.

```
struct elemento {float valor ; struct elemento* prox ;};
typedef struct elemento TElemento;
TElemento *inicio1 = NULL, *inicio2 = NULL, *inicio3 = NULL;
float vrdig = 0;
```



15) Escreva um programa em C para determinar se uma *string* de até 30 caracteres informada pelo usuário possui a forma `xCy`, onde `x` é uma *string* consistindo na combinação de apenas das letras 'A' e 'B', e `y` é o inverso de `x` (isto é, se `x` for a cadeia "ABABBA", `y` deve equivaler a "ABBABA"). Em cada ponto, você só poderá ler o próximo caracter da *string*. (Adaptado do ex. 2.1.3 de Tenenbaum e outros. Estruturas de Dados Usando C. São Paulo: Makron Books, 1995, pag. 97).

Dica: Utilize uma pilha para armazenar os valores da primeira parte da palavra (que antecedem o 'C'). Depois, para cada caracter da segunda parte da *string* (depois do 'C'), retire um caracter da pilha e o compare com o obtido da *string*: se forem iguais, prossiga, senão emita mensagem informando que a *string* digitada pelo usuário não é da forma `xCy`. Se, após processar toda a *string* a pilha estiver vazia, a cadeia está no formato `xCy`.

16) O Estacionamento de Scratchemup contém uma única alameda que guarda até dez carros. Os carros entram pela extremidade sul do estacionamento e saem pela extremidade norte. Se chegar um cliente para retirar um carro que não esteja estacionado na posição do extremo-norte, todos os carros ao norte do carro desejado serão deslocados para fora, o carro sairá do estacionamento e os outros carros voltarão à mesma ordem em que se encontravam inicialmente. Sempre que um carro deixa o estacionamento, todos os carros ao sul são deslocados para a frente, de modo que o tempo inteiro todos os espaços vazios estejam na parte sul do estacionamento.

Escreva um programa que leia um grupo de linhas de entrada. Cada linha contém um 'C', de chegada ou um 'P', de partida, além de um número de placa de licenciamento. Presume-se que

os carros chegarão e partirão na ordem especificada pela entrada. O programa deve imprimir uma mensagem cada vez que um carro chegar ou partir. Quando um carro chegar, a mensagem deverá especificar se existe ou não vaga para o carro dentro do estacionamento. Se não existir vaga, o carro esperará pela vaga ou até que uma linha de partida seja lida para o carro. Quando houver espaço disponível, outra mensagem deverá ser impressa. Quando um carro partir, a mensagem deverá incluir o número de vezes que o carro foi deslocado dentro do estacionamento, incluindo a própria partida mas não a chegada. Esse número será zero se o carro for embora a partir da fila de espera. *(Extraído de: Tenenbaum, Aaron M. & outros. Estruturas de dados usando C. São Paulo: Makron Books, 1995. pag. 222)*

Árvores

17) A rotina recursiva descrita a seguir realiza a busca de um valor em uma árvore binária ordenada do mesmo tipo que a construída em aula. Escreva a versão não recursiva correspondente.

```
struct regNo *Busca(struct regNo *r, int v)
{  if (r == NULL || r->valor == v)
        return r;
   if (r->valor > v)
        return Busca(r->esq, v);
   else
        return Busca(r->dir, v);
}
```

18) Escreva uma rotina para determinar a quantidade total de espaço em bytes ocupada por uma árvore. A rotina deverá receber como parâmetro o ponteiro para a raiz da árvore e retornar um inteiro correspondente à quantidade total de bytes que a árvore ocupa no momento. Considere que os nós da árvore possuem o mesmo tipo utilizado no programa criado durante as aulas.

19) Dadas duas árvores binárias A e B, diz-se que A *eq* B (lê-se A é equivalente a B) se:

a) Ambas são vazias, ou

b) info(raiz(A)) = info(raiz(B)) e esq(A) *eq* esq(B) e dir(A) *eq* dir(B)

Faça um programa que permite montar duas árvores binárias ordenadas e determinar se elas são ou não equivalentes.

20) Segundo as páginas 187 e 188 da Introdução a Estruturas de Dados, de Waldemar Celes e outros, "Para descrever árvores binárias, podemos usar a seguinte notação textual: a árvore vazia é representada por `< >`, e árvores não-vazias, por `<raiz sae sad>`. Com essa notação, a árvore da Figura 13.4 é representada por:

```
<a<b< ><d< >< >>><c<e< >< >><f< >< >>>>
```

Pela definição, uma subárvore de uma árvore binária é sempre especificada como sendo a *sae* ou a *sad* de uma árvore maior, e qualquer das duas subárvores pode ser vazia. Assim, as duas subárvores da Figura 13.5 são distintas.

Isso também pode ser visto pelas representações textuais das duas árvores que, em pre-order, são, respectivamente: `<a<b< >< >>< >> e <a< ><b< >< >>>`."

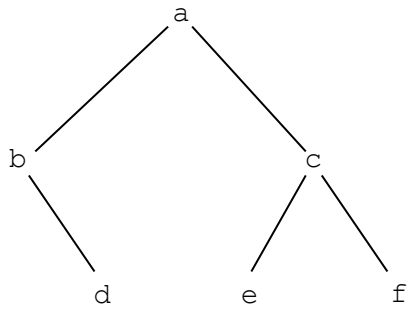


Fig. 13.4 Exemplo de árvore binária.

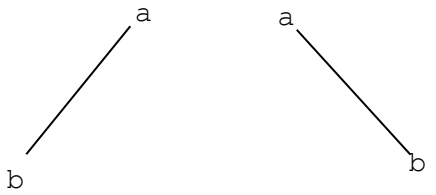


Fig. 13.5 Duas árvores binárias distintas

Escreva programas que:

a) Recebe uma sequência de inteiros e monta a árvore binária ordenada correspondente. Em seguida exibir a descrição da árvore na notação textual indicada anteriormente.

b) Recebe uma *string* contendo a descrição textual de uma árvore em pre-order e constrói a árvore correspondente, imprimindo-a e exibindo a sua altura e quantidade de elementos.

21) Com base na implementação da rotina de impressão não-recursiva Pré-Order apresentada a seguir, crie uma versão em que a abordagem adotada seja In-Order e, quando um nó tiver nulo em algum de seus ponteiros, seja impresso um sinal '/'.

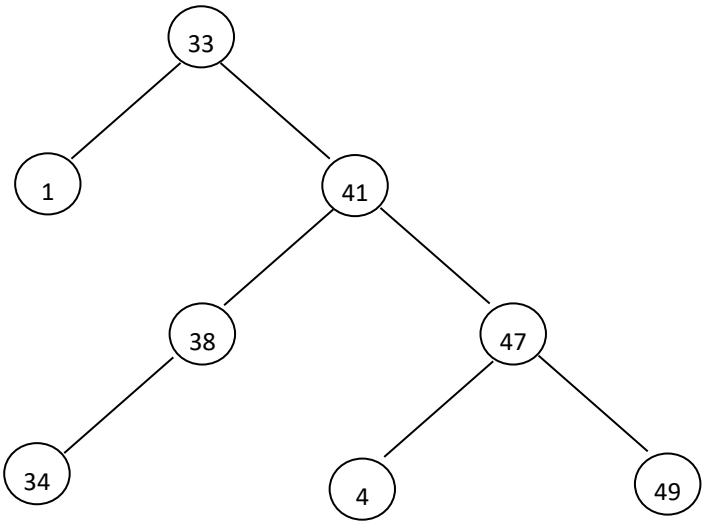
```
void ImprimeArvore(TNo*r, int n)
{     typedef struct{ TNo *ender; int nivel;} TPilha;

      TPilha pilha[1000];
      TNo *noh;
      int d, topo = 0;

      noh = r;
      while (noh != NULL || topo > 0)
      {     if (noh != NULL)
            {     for (d = 0; d < n; d++) printf("    ");
                  printf("%d\n", noh->valor);

                  pilha[topo].ender = noh;
                  pilha[topo].nivel = n;
                  topo++;
                  noh = noh->esq;
                  n++;
            }
            else
            {     topo--;
                  noh = pilha[topo].ender;
                  n   = pilha[topo].nivel;
                  noh = noh->dir;
                  n++;
            }
      }
}
```

22) [Questão 39 (discursiva) do ENADE 2005, para Bacharel em Sistemas de Informação]



Tendo como base a árvore acima, faça o que se pede nos itens a seguir.

a) Descreva uma ordem de visita dos nós para uma busca em profundidade a partir do nó de valor 41.

b) Considerando que o nó de valor 33 seja a raiz da árvore, descreva a ordem de visita para uma varredura em pré-ordem (r-e-d, ou pré-fixado à esquerda) na árvore.

c) Considerando que a árvore cuja raiz é o nó de valor 33 represente uma árvore de busca binária, desenhe a nova árvore que será obtida após a realização das seguintes operações: inserir um nó de valor 21; remover o nó de valor 47; inserir um nó de valor 48.