Para todas as questões desta prova que envolvam implementação de código utilizando árvores binárias, considere a seguinte declaração:

```
typedef struct no {
    int v;
    struct no *esq, *dir;
} No;
```

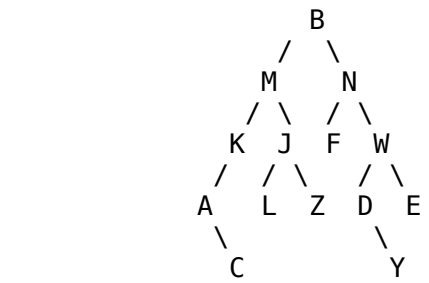
Para todas as questões desta prova que envolvam implementação de código utilizando heaps, considere a declaração e as funções abaixo:

```
typedef struct heap {
  int *h;     /* Vetor de inteiros a ser alocado
                 no momento da inicialização do heap */
  int n;      /* Número de elementos no heap */
  int max_n;  /* Número máximo de elementos */
} Heap;

int pai(int i) { return (i-1)/2; }
int esq(int i) { return 2*i + 1; }
int dir(int i) { return 2*i + 2; }

int existe_pai(int i) { return i > 0; }
int existe_esq(int i, int n) { return esq(i) < n; }
int existe_dir(int i, int n) { return dir(i) < n; }
```

Questão 1

Desenhe a árvore binária que apresenta os percursos descritos abaixo.

```
Pré-ordem: B M K A C J L Z N F W D Y E

In-ordem: A C K M L J Z B F N D Y W E
```



Questão 2

Escreva uma função recursiva que conta o número de nós em uma árvore binária que possuem pelo menos um filho.

```
int conta_pais(No* t) {
  if (t == NULL ||
      (t->esq == NULL && t->dir == NULL)
    return 0;
  return 1 + conta_pais(t->esq) + conta_pais(t->dir);
}
```

Questão 3

Seja dh(t) = |h_e - h_d| o módulo da diferença entre as alturas das sub-árvores esquerda h_e e direita h_d de uma árvore binária t, sendo dh(t) = 0 quando t == NULL. A função dh_max(t) deve calcular o valor máximo de dh para a árvore t. Implemente o código da função recursiva auxiliar aux_dh_max(t), que, em um mesmo percurso, calcula a altura da árvore t (armazenando o resultado em *h) e retorna o dh máximo encontrado. Você pode utilizar as funções max(a,b) (retorna o máximo entre a e b) e abs(a) (retorna o valor absoluto de a).

```
int aux_dh_max(No *t, int* h);

int dh_max(No *t) {
  int h;
  return aux_dh_max(t, &h);
}

int aux_dh_max(No *t, int* h) {
  if (t == NULL) {
    *h = 0;
    return 0;
  }
  int he, hd, dhe, dhd;
  dhe = aux_dh_max(t->esq, *he);
  dhd = aux_dh_max(t->dir, *hd);
  *h = 1 + max (he, hd);
  return max(max(dhe, dhd), abs(he- hd));

}
```

Questão 4

Escreva um função que retorna 1 caso o heap passado como argumento respeite as restrições de heap de máximo (para todo elemento que tenha pelo menos um filho, seu valor associado deve ser maior do que o do(s) seu(s) filho(s)) e 0 caso contrário.

```
int verifica(Heap *heap) {
  int i;
  for (i = heap->n-1; existe_pai(i); i--)
    if (heap->h[i] > heap->h[pai(i)])
      return 0;
  return 1;
}
```

Questão 5

Escreva um função que retorna o valor mínimo em um heap de máximo. Apenas as posições do vetor que podem armazenar o elemento mínimo devem ser visitadas (evite comparações desnecessárias). A função deve retornar -1 caso o heap esteja vazio.

```
int valor_minimo(Heap heap) {
  if (heap->n == 0)
```

```
    return -1;
  int min = heap->h[heap->n-1];
  int i = heap->n-2;
  while (!existe_esq(i)) {
    if (min > heap->h[i])
      min = heap->h[i];
      i--;
  }
  return min;
}
```