CP2 Exercise 6 **27/11/2006**

1. Using the framework below, write a program which allows a binary tree (*with stored elements ordered ascending from left to right*) to be generated and printed out. To do this you will have to complete the functions ***insert*** and ***depth*** as well as the three tree printing functions ***printInOrder***, ***printPreOrder*** and ***printPostOrder***.

Note: *the function* **depth** *should return '0' (zero) if the tree is empty. The function* **insert** *should not insert elements that are already present within the tree.*

2. Modify the ***printInOrder*** function in the above exercise to print the values in the tree indented (*depending on the depth of their position in the tree*), so that if the tree would contain the 3 elements 7, 10 and 13 with 10 being the root node the output would look something like:

```
      13
   10
      7
```

Exercise framework:

```
/* exercise framework – binary trees, Eike Anderson, 2006 */

#include<stdlib.h>
#include<stdio.h>


/* type declarations */
typedef struct _node
{
   struct _node *left;
   struct _node *right;
   int data;
}  node;
typedef node*  nodePtr;


/* function prototypes */

int insert(nodePtr*,int);
/* tree node insertion function (ascending left->right) */

int  depth(nodePtr);
/* tree depth query (depth 0 -> tree empty) function  */

void printInOrder(nodePtr);
/* tree content output function - inOrder traversal */

void printPreOrder(nodePtr);
/* tree content output function - preOrder traversal */

void printPostOrder(nodePtr);
/* tree content output function - postOrder traversal */

void clear(nodePtr*);
/* tree content deletion function - postOrder traversal */
/* main function */
int main(void)
{
  int input;
  char c;
  nodePtr base=NULL;

  do /* a simple menu */
  {
      printf("\nmake your choice:\n");
      printf("i) add element to tree\n");
      printf("d) print depth of tree\n");
      printf("p) print tree in-order\n");
      printf("r) print tree pre-order\n");
      printf("o) print tree post-order\n");
      printf("c) clear tree structure\n");
      printf("q) quit program\n - ");
      c=getchar();  fflush(stdin);
      switch(c)
      {
        case 'c': printf("\ndeleting all tree-nodes\n");
                  clear(&base);
                  break;
        case 'd': printf("\ntree depth is %d.\n",depth(base));
                  break;
        case 'o': printPostOrder(base);
                  break;
        case 'p': printInOrder(base);
                  break;
        case 'r': printPreOrder(base);
                  break;
        case 'i': printf("\nPlease enter a number>0 - ");
                  scanf("%d",&input);
                  fflush(stdin);
                  /* flush stdin after using scanf */
                  insert(&base,input);
                  break;
        default: printf("\n");
      }
    } while(c!='q'&&c!='Q');
  printf("\n");
  printInOrder(base);
  clear(&base);
  return 0;
}

int  insert(nodePtr* root,int value)
{
  /* insert function body here */
}

int depth(nodePtr root)
{
   /* insert function body here */
}

void printInOrder(nodePtr root)
{
  /* insert function body here */
}
```

```
void printPreOrder(nodePtr root)
{
  /* insert function body here */
}

void printPostOrder(nodePtr root)
{
  /* insert function body here */
}

void clear(nodePtr *root)
{
  nodePtr temp;
  /* note - this is a post-order operation */
  if(*root==NULL)
    return;
  temp=*root;
  clear(&(temp->left));
  clear(&(temp->right));
  free(temp);
  *root=NULL;
}
```

3. Expand the above exercise by a function ***int AVLtest(nodePtr root)***. This function should recursively check if the tree is conforming with the AVL-tree condition for perfectly height-balanced trees (*“the difference in the depth at each branch-node of the tree is 1 or less”*). The function should return 1 if the tree is an AVL-tree and 0 if it isn't.

Note: *you might need to call the* **depth** *function from within* ***AVLtest.***