Stuparu Elena Natalia

```
    1.  #include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct bintree{

  /* Cuvantul pe care il va contine arborele biner */
  char* info;
  /* Copilul stang si copilul drept */
  struct bintree* left;
  struct bintree* right;

  /* Alt arbore binar unde vom stoca orasele si mai apoi locatiile */
  struct bintree *cities;
};



struct bintree* newtree(char* x){
   /* Creem un arbore*/
   struct bintree* t = (struct bintree*) malloc (sizeof(struct bintree));

   if(t != NULL){

      /* Alocam memorie pentru Cuvantul din arbore */
      t->info = (char *)malloc(100 * sizeof(char));
      /* Copiem valoarea in informatia arborelui*/
      strncpy(t->info, x, strlen(x));
      t->left = NULL;
      t->right = NULL;


      return t;
   }


   return NULL;


}




/* Functia de insert
 * Vom insera dupa valoarea in cod ascii a cuvantului pe care vrem sa-l bagam
 * Astfel vom mentine ordinea alfabetica
 * Intai vom compara valorile, iar in functie de valoarea cuvantului pe care
 * vrem sa-l introducem, ne ducem pe ramura stanga/dreapta
 */
struct bintree* insert(struct bintree* b, char *word){


   int x = atoi(word);
   if(b == NULL){


      b = newtree(word);


   }else if(x > atoi(b->info)){


      b->right = insert(b->right, word);


   }else if(x <= atoi(b->info)){


              b->left = insert(b->left, word);
```

```
  }

  return b;
}



/* Functie de printat arborele
 * Recursiv, vom printa arborele cu nume
 * iar mai apoi, la fiecare nod printam arborele de localitati
 * iar mai apoi, la fiecare nod din arborele de localitati printam
 * arborele de locatii(in mod recursiv pana cand nodul este NULL)
 */
void printTree(struct bintree* b){

  if(b == NULL){
        return;
  }

  printf("%s\n",b->info);
  printTree(b->cities);

  printTree(b->left);
  printTree(b->right);

}



struct bintree* search(struct bintree* root, char* word)
{
   if(root == NULL || strcmp(root->info, word) == 0) /* Daca valoarea din root este egala cu word atunci am gasit elementul*/
```

```
     return root;
   else if(atoi(word) > atoi(root->info)) /* daca valorea este mai mare cautam in dreapta*/
     return search(root->right, word);
   else /* daca valorea este mai mica cautam in stanga*/
     return search(root->left, word);
}

int main(){

  struct bintree* b = NULL;

  /* Inseram datele */
  b = insert(b, (char*)"Ionel");
  b = insert(b, (char*)"Petre");


  /* Penttru Ionel ii adaugam locatia bucuresti cu locurile de vizitat aferente*/
  struct bintree *searchedNode = NULL;
  searchedNode = search(b, (char*)"Ionel");
  searchedNode->cities = insert(searchedNode->cities, (char*)"Bucuresti");

  searchedNode = search(searchedNode->cities, (char*)"Bucuresti");
  searchedNode->cities = insert(searchedNode->cities, (char*)"Arcul de Triumf");
  searchedNode->cities = insert(searchedNode->cities, (char*)"Centrul Vechi");


  searchedNode = search(b, (char*)"Ionel");
  searchedNode->cities = insert(searchedNode->cities, (char*)"Targul-Jiu");


   /* Penttru Ionel ii adaugam locatia targul jiu cu locurile de vizitat aferente*/
  searchedNode = search(searchedNode->cities, (char*)"Targul-Jiu");
  searchedNode->cities = insert(searchedNode->cities, (char*)"Coloana Infinitului");
  searchedNode->cities = insert(searchedNode->cities, (char*)"Masa Tacerii");



  /* Iar pentru petre la fel ca mai sus*/
  searchedNode = NULL;
  searchedNode = search(b, (char*)"Petre");
  searchedNode->cities = insert(searchedNode->cities, (char*)"Alba Iulia");
  searchedNode->cities->cities = insert(searchedNode->cities->cities, (char*)"Coloana Unirii");

  printTree(b);

  return 0;
}
```

Output-ul generat de prima problema este:

```
Ionel
Bucuresti
Arcul de Triumf
Centrul Vechi
Targul-Jiu
Coloana Infinitului
Masa Tacerii
Petre
Alba Iulia
Coloana Unirii
```

```
2. #include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
```

```
/*  Nodul din arbore */
struct Node
{
   int info;
   struct Node *left;
   struct Node *right;
};

/* Functie cu care ne alocam un nou nod */
struct Node *newNode(int k)
{
   struct Node node = (struct Node)malloc(sizeof(struct Node));
   node->info = k;
   node->right = NULL;
   node->left = NULL;
   return node;
}

/* functia aceasta numara cate noduri avem in arborele binar in mod recursiv
 * Cauta pe arborele stang pana cand ajunge la frunze si vede ca, copii frunzei
 * sunt NULL, iar mai apoi, identic, pe subarborele drept.
 */
int countNodes(struct Node* root)
{
   if (root == NULL)
      return 0;
   return (1 + countNodes(root->left) + countNodes(root->right));
}

/* Functia aceasta verifica daca un arbore este complet sau nu */
int isComplete (struct Node* root, int startPosition,
            int nr_nodes)
{
   /* UN ARBORE NULL ESTE COMPLET*/
   if (root == NULL)
      return (true);

   /* Daca pozitia de start, care acum a ajuns la un nod oarecare din arbore
    * ajunge sa fie mai mare decat numarul de noduri din arbore, atunci cu siguranta
    * arborele nu este complet.
    */
   if (startPosition >= nr_nodes)
      return 0;

   /* Apelam recursiv functia pe subarborele stang si mai apoi pe cel drept
    * De mentionat ca, pe subarborele stang indexul va fii intodeauna un numar
    * impar, iar pe subarobrele drept va fi par, teoretic, acest numar este un index
    * care spune al catelea nod este in arbore de la stanga la dreapta(impar la par).
    */
   return (isComplete(root->left, 2 * startPosition + 1, nr_nodes) &&
         isComplete(root->right, 2 * startPosition + 2, nr_nodes));
}

int main()
{
   /* Creem arborele care arata asa

      1
     2   3
    4 5    6

    Acest arboore nu este complet
```

```
*/
struct Node* root = NULL;
root = newNode(1);
root->left = newNode(2);
root->right = newNode(3);
root->left->left = newNode(4);
root->left->right = newNode(5);
root->right->right = newNode(6);
int index = 0;

if (isComplete(root, index, countNodes(root)))
    printf("The Binary Tree is complete\n");
else
    printf("The Binary Tree is not complete\n");

/* Acum, mai punem un nod*/
root->right->left = newNode(7);
/* Creem arborele care arata asa

    1
  2   3
 4 5 7 6

 Acest arboore este complet
*/
index = 0;
if (isComplete(root, index, countNodes(root)))
    printf("The Binary Tree is complete\n");
else
    printf("The Binary Tree is not complete\n");
return 0;
```

```
}
```

Output-ul generat de aceasta problema este:

```
The Binary Tree is not complete

The Binary Tree is complete
```