Task 1

Final Submission Date - 25.11.2020

Required Data Structures of the Job

In order to carry out this work, it is necessary to implement the data structures of the list, the fifo queue, the lifo or stack queue and the binary search tree described below.

These structures store integers of type int (or long int) or any other data type you think is useful for your case. Each of these structures is implemented by two files, a file with the extension .h and a file with the extension .c. The files with the .h ending contain one or more structs that describe the data structure and the prototypes of the functions that implement it, and the .c file only the implementations of the corresponding functions described in the .h file.

The files describing each of the following structures should be in the directory of files you submit, because they are required when compiling from the associated Makefile. If a structure you think you will not need in the main programs you will write (list1.c, tree1.c, tree2.c, tree3.c), you can leave the corresponding files empty.

A recommended good practice for all .h files (header files) to avoid multiple inclusions of the .h file in the compiled code is to write your code inside a declaration of the form below

```
# ifndef    __ HEADER_FILE _H_
# define    __ HEADER_FILE _H_

/* insert all its code
  * header file here
  * /

# endif
```

replacing the section HEADER_FILE with the file name ending in .h. In the example below, you can see how the content of the header file of the doubly linked list can be declared.

```
# ifndef    __DLIST_H_
# define    __DLIST_H_

typedef struct dlist_node {
    struct     dlist_node*      next?
    struct     dlist_node*      previous?
    int data;
}   dnode_t;

typedef struct dlist {
    struct dlist_node *head, *tail; int size;

}   dlist_t;

/* here you can put the prototypes of the list functions. */


# endif
```

The doubly linked list

In the files**dlist.c, dlist.h** implement a doubly linked list of your choice.

The FIFO queue

In the files**fifo.c, fifo.h** implement a queuefirst-in-first-out of your choice.

The STACK stack

In the files**stack.c, stack.h** implement a queuelast-in-first-out of your choice.

The binary search tree

In the files**tree.c, tree.h** implement a binary search tree of your choice.

Lists 1

Write the programlist1.c which reads a sequence of integers from the standard input, for which we do not know its length in advance, and stores it in a list of your choice. The numbers in the sequence are separated by one or more spaces, and the sequence ends by reading the number zero (zero is not stored in the list). The program finds the maximum subsequence within the list whose sum is zero (0) and prints its characteristics to the standard output (stdout). If there are more sub-sequences with the same length it chooses the first one encountered, as we traverse the sequence of elements from beginning to end.

When printing, it prints the following:

- the string “start: START_POS, stop: END_POS, size: SIZE” followed by newline character , where START_POS the position of the first element of the subsequence within the original sequence (assuming the first element of the original sequence is numbered zero), END_POS the position of the last element of the sub-array and SIZE the size of subsequence.

- the elements of the sub-sequence. Each element is followed by a blank character and the last blank character is followed by a newline character.

For example, for the sequence of integers below

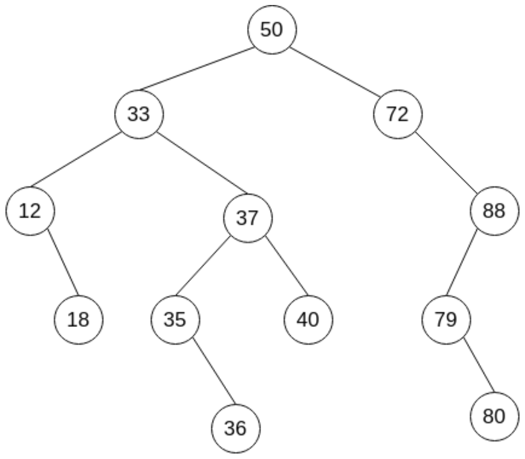
1 -3 3 3 -4 4 -4 2 2 -3 5

the maximal zero-sum subsequence is the one starting at position 1 and ending at position 9, as follows:**-3 3 3 -4 4 -4 2 2 -3**

Binary Trees 1

Write the programtree1.c , which reads a sequence of positive integers from standard input (*stdin*) , which constitutes the *mail order*traversal of a binary search tree. The numbers in the sequence are separated by one or more spaces and the sequence ends when a negative number is read (the negative number is not recorded). The program constructs the binary search tree corresponding to the given penetration and prints it in *stdout*her level-order penetration of it. A blank character is printed after each number is printed, and a newline character is printed after the last blank.

For example, for the following binary tree the post-order and pre-order traversals are as follows:



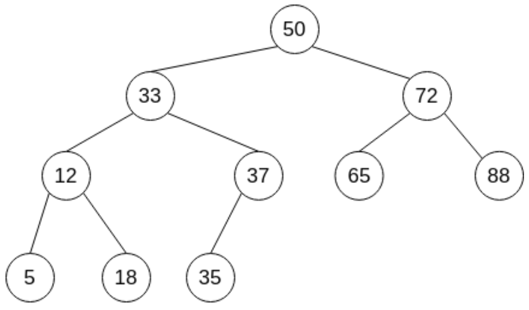
Post Order:
18, 12, 36, 35, 40, 37, 33, 80, 79, 88, 72, 50

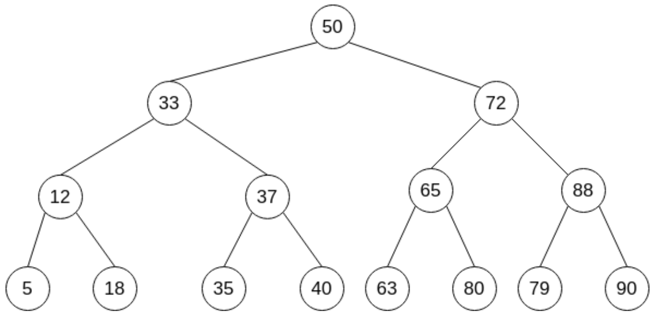
Level Order:
50, 33, 72, 12, 37, 88, 18, 35, 40, 79, 36, 80

Binary Trees 2

Write the programtree2.c , which reads a sequence of positive integers from standard input (*stdin*) , which constitutes thelevel-order penetration of one full binary tree . The numbers in the sequence are separated by one or more spaces, and the sequence ends by reading a negative number. After constructing the binary tree, the program checks whether this tree is a binary search tree or a simple binary tree and prints the message accordingly
“Binary Search Tree” the"Binary Tree" , followed by a newline character.

We remind you that complete is the binary tree that*a).*all its internal nodes have two children (full tree), *b).*leaves are found only in the last two levels and *c).*the leaves of the latter level is as far to the left as possible. Examples of complete trees are given below:





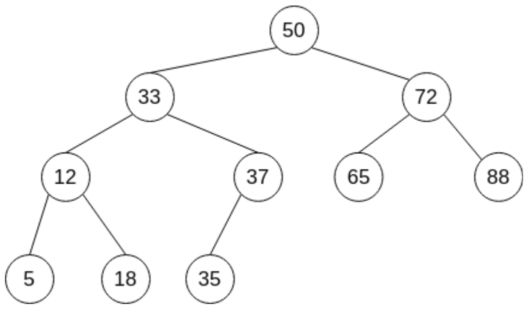
Observation: the left of the two trees is a binary search tree unlike the right one which is not.

Binary Trees 3

Write the programtree3.c , which reads a sequence of positive integers from standard input (*stdin*) , which constitutes thepre-order traversal of a binary search tree. The numbers in the sequence are separated by one or more spaces, and the sequence ends by reading a negative number. The program constructs the binary search tree corresponding to the given penetration.

The program then prints the message "Enter 2 integers: " and reads two integers numbers separated from each other by a space character. Every integer he reads, he looks for him in the tree. If one of the two integers does not exist in the tree, the message is printed "X not found!” followed by a newline character and the program terminates, where X is the first of the two numbers not found. Otherwise it prints a newline character and t~~he message~~ "Path is: ". Then the minimum path from the node corresponding to the value that is printed read first, to the node corresponding to the value read second. A blank character is printed after each number is printed, while a newline character is printed after the last number ~~and after the last blank.~~

For example, in the adjacent search tree given by the pre-order penetration50, 33, 12, 5, 18, 37,35, 72, 65, 88 the shortest path from node 35 to node 88 is 35, 37, 33, 50, 72, 88.



Shipping Method

The work will be sent through the platform autolab (no VPN connection required). Follow these steps:

1. Create a folder named **hw1submit**and copy the files in there**dlist.c, dlist.h, stack.c, stack.h, fifo.c, fifo.h, tree.c, tree.h, list1.c, tree1.c, tree2.c, tree3.c** .
2. Compress as tar.gz. On Linux/KDE go to the folder, right click and select **Compress -> Here (as tar.gz).**The file is created**hw1submit.tar.gz** .

3. You connect to the address https://autolab.e-ce.uth.gr and choose the course **ECE215-F20 (f20)** and from this work**HW1** .
4. To submit your work, click on the option **"I affirm that I have compiled with this course academic integrity policy..."**and press**submit** . Then select the file **hw1submit.tar.g** z that you created in step 2.