HACETTEPE UNIVERSITY COMPUTER SCİENCE ENGINEERING DEPARTMENT

BIL-341 LABORATORY

2. EXPERIMENT REPORT

Name :Serdar
Surname :GÜL
Number :20421689

Subject :C and Assembly Programming
Language :C and Assembly
System :Linux or Windows(dev.cs.hacettepe.edu.tr)
Platform :MinGW Developer Studio and Nasm in Linux

Advisors :Dr. Ayça Tarhan, R.A. Yiğitcan AKSARI

Due Date :30.10.2008

SOFTWARE USAGE

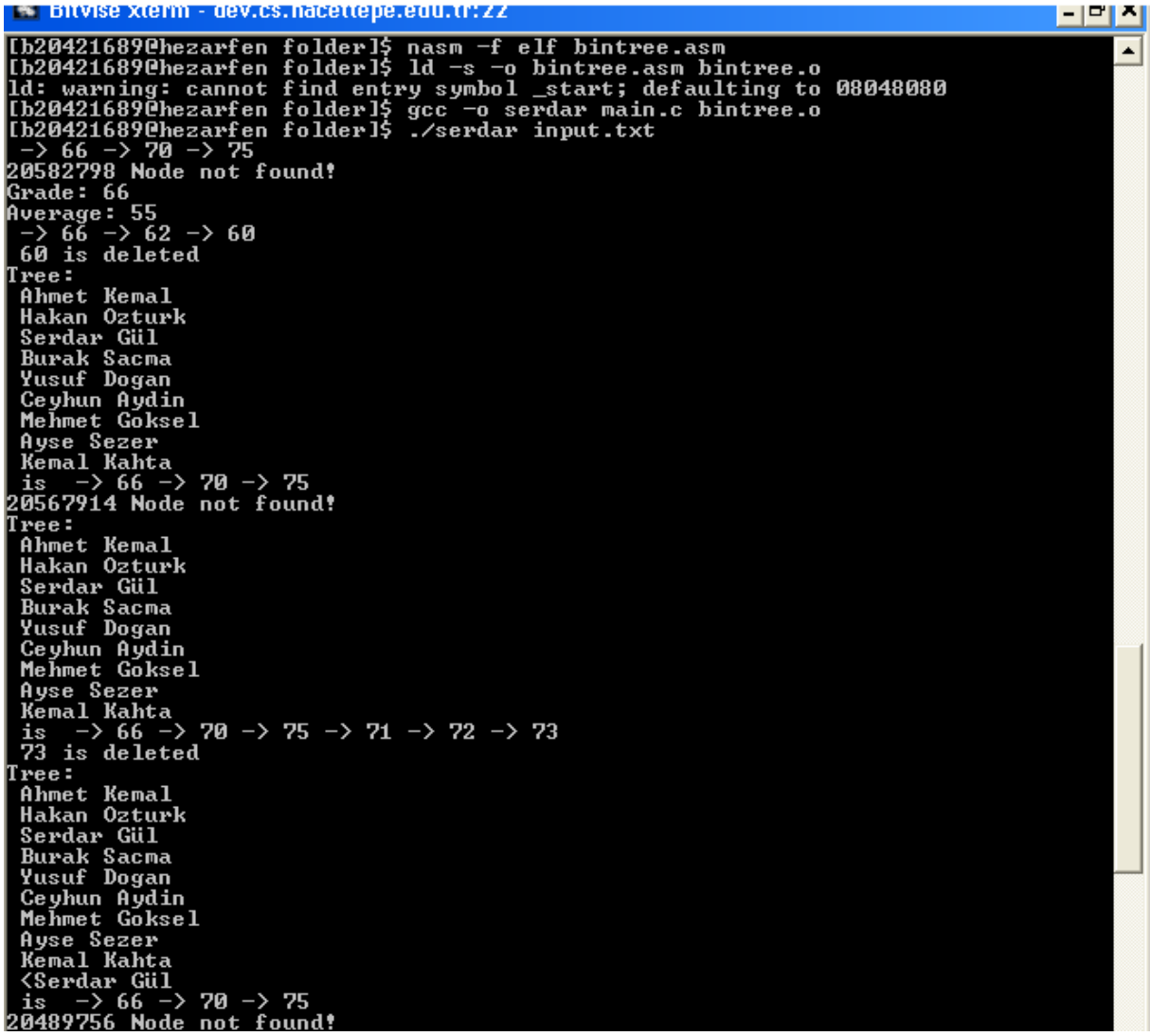
It is used so easily.With the given makefile you can run the program in the Linux kernal.

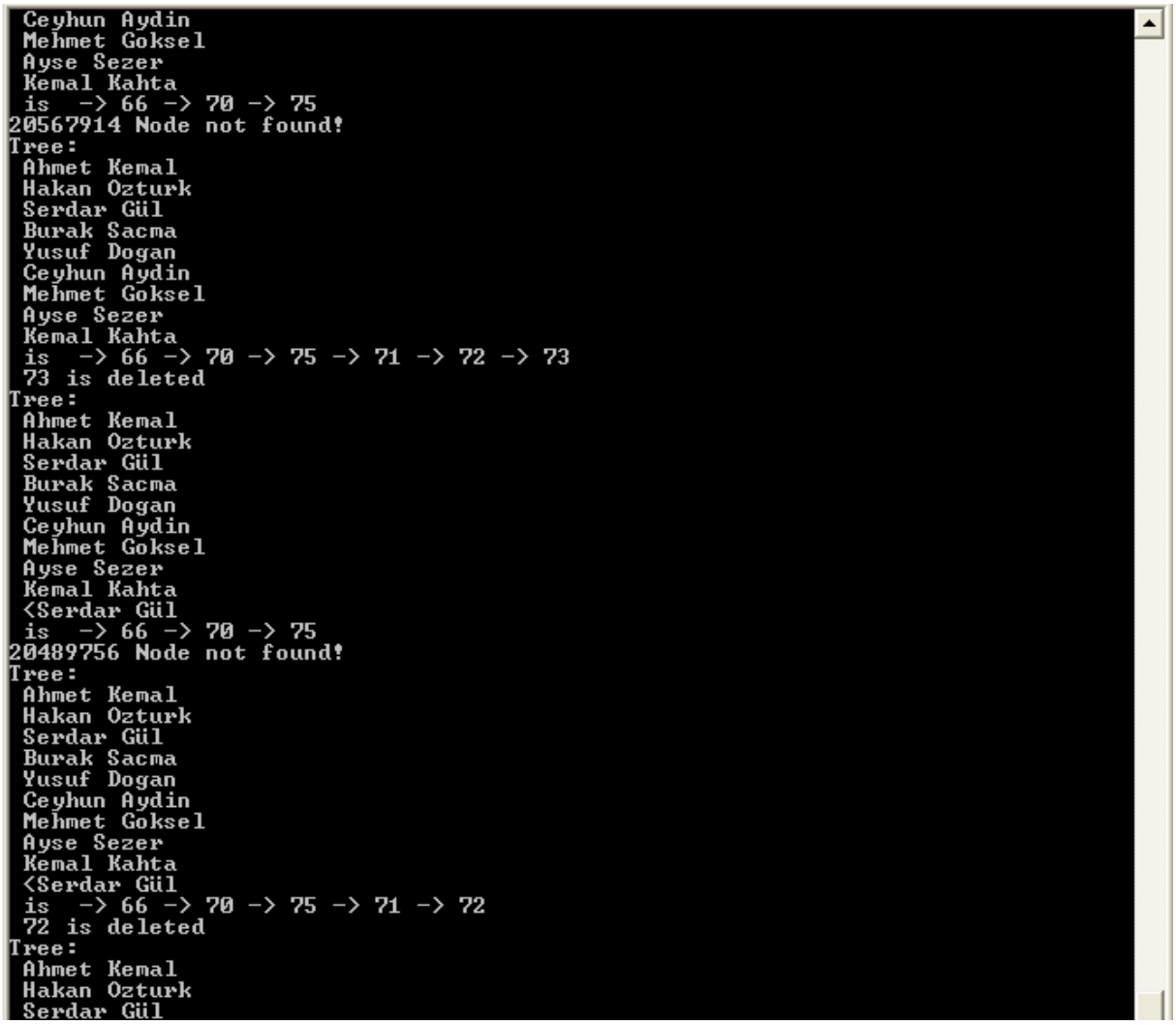
So we can say that is has a basic usage…

I use a random input in this execution but it Works in the given input file..

It takes 2 parameters while execution that are input and output files.

We will print the results to the console and to the output file…


Bitvise xterm - dev.cs.hacettepe.edu.tr:22
[b20421689@hezarfen folder]$ nasm -f elf bintree.asm
[b20421689@hezarfen folder]$ ld -s -o bintree.asm bintree.o
ld: warning: cannot find entry symbol _start; defaulting to 08048080
[b20421689@hezarfen folder]$ gcc -o serdar main.c bintree.o
[b20421689@hezarfen folder]$ ./serdar input.txt
-> 66 -> 70 -> 75
20582798 Node not found!
Grade: 66
Average: 55
-> 66 -> 62 -> 60
60 is deleted
Tree:
Ahmet Kemal
Hakan Ozturk
Serdar Gül
Burak Sacma
Yusuf Dogan
Ceyhun Aydin
Mehmet Goksel
Ayse Sezer
Kemal Kahta
is -> 66 -> 70 -> 75
20567914 Node not found!
Tree:
Ahmet Kemal
Hakan Ozturk
Serdar Gül
Burak Sacma
Yusuf Dogan
Ceyhun Aydin
Mehmet Goksel
Ayse Sezer
Kemal Kahta
is -> 66 -> 70 -> 75 -> 71 -> 72 -> 73
73 is deleted
Tree:
Ahmet Kemal
Hakan Ozturk
Serdar Gül
Burak Sacma
Yusuf Dogan
Ceyhun Aydin
Mehmet Goksel
Ayse Sezer
Kemal Kahta
<Serdar Gül
is -> 66 -> 70 -> 75
20489756 Node not found!



ERROR MESSAGES

"The parameters are wrong" : If user enters less than 3 arguments to the command line program will give this error message to the user.

"Input File can not be opened" : If the user gives a wrong name of an input file or program fails about opening the input file , program will give this error message to the user..

"Output File can not be opened.." :If the user can not open the output file for printing the results

to the output file or program fails about opening the output file , program will give this error message to the user..

"The parameters are too much" : If the user enters too much parameters there are not necessary , program will give this error message to the user…

SOFTWARE DESIGN NOTES

PROBLEM

In this experiment our main problem is combining the C source code and Assembly code.
Link both of them
And execute completely true..

But we have a lot of problems in this case..These are like this..

-User can give wrong parameters
-User can give less parameters to the command line while executing the program.(This program will run in 2 arguments for the command line)
-User can give more parameters.(User can give unnecessary parameters to the command line..)

In this experiment we will have student data structures that has name,surname,no and grade values

We will make a binary search tree that take the parameter of the student number.

And we will call some functions which are on the tree like add,delete,findgrade etc.…

And we should put the results to the output..

SOLUTION

At first we will take the parameters from the command line.

We will read the command and
İf the command equals to add
    We will read a line and take the elements and
    We will add a node
Else if the command is average
    We will find the average of the student’s grades.
Else if the command is print
    We will print the tree element with the depth search algorithm
Else if the command is find grade
    We will read a line
    Take the parameters of the name and surname
    And we will find the grade of the student with the given paramaters
    İf does not found in the tree
    We will say that that student is not in the tree
Else if the command is delete
We will read a line and take the necessary parameters and we will delete the node
If we don’t find the node in the tree we will give an error message
Else we will delete the node , write the path to reach the node to command line

And we will maket his operations until the input file will be finished…

The functions we called in the C source code

SYSTEM CHARTS

INPUT	PROGRAM	OUTPUT
input.txt	bintree.asm main.c makefile	command prompt output.txt

MAIN DATA STRUCTURES

I will have a student data structure that is used for the binary search tree and my struct is like this.

```
typedef struct stu *stu_ptr;
```

```
typedef struct stu{
  stu_ptr leftChild;
  int no;
  char name[20];
  char surname[20];
  int grade;
  stu_ptr rightChild;
}student;
```

and also that I have pointers to store the informations of them and I use local variables like integer etc…

In the assembly source code of the experiment I use stack and registers..

These are like

Esi
Edi
Eax
Ebx
Edx

ALGORITHM

With the makefile run the program
Take the second argument as an input file
Open the input file
    Read a line(as command)

        İf the command is ADD

Read the next line
Take the parameters no , name, surname , grade and make a student object.
Call the assembly function in the C code
Add the element to the binary search tree
If the tree is empty , make the given element as a parameter root

Else if the command is PRINT
Call the assembly function in the C code
With starting at node print the student’s name and surname informations to the command line
If the tree is empty print an error message to the command line and to the output file

Else if the command is FINDGRADE
Read the next line
Take the parameters of teh name and surname
Call the assembly function of findGrade() with the given parameters
In assembly search the tree with the given parameters
If you find the student print the student’s grade to the command line and output file
If the student does not exist in the tree print an error message to the command line and output file.
If the tree is empty print an error message to the command line and to the output file

Else if the command is AVERAGE
Call the assembly function of calculateAverage ()
Wander all of the tree nodes
Take the grade of the students

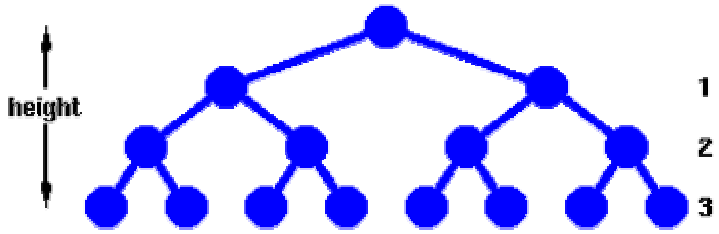
```
                Divide it to the number of the nodes
                And print the final average value of
the tree to the command line and output file..
             If the tree is empty print an error
message to the command line and to the output file


             Else if the command is DELETE
                Take the number parameter
                Call the assembly function of
deleteNode () with the given parameters
                Wander in the tree
                İf you find the node with given
number delete it and print the path to Access the
node to the command line and output file.

                İf the tree is empty print an error
message to the command line and output file

                İf you don't find the node with the
given parameter give an error message to the output
file and command line that says that node not found..
```

ALGORITHMS OF BINARY SEARCH TREES

Binary Search Tree

Binary Search tree is a binary tree in which each internal node x stores an element such that the element stored in the left subtree of x are less than or equal to x and elements stored in the right subtree of x are greater than or equal to x. This is called binary-search-tree property.

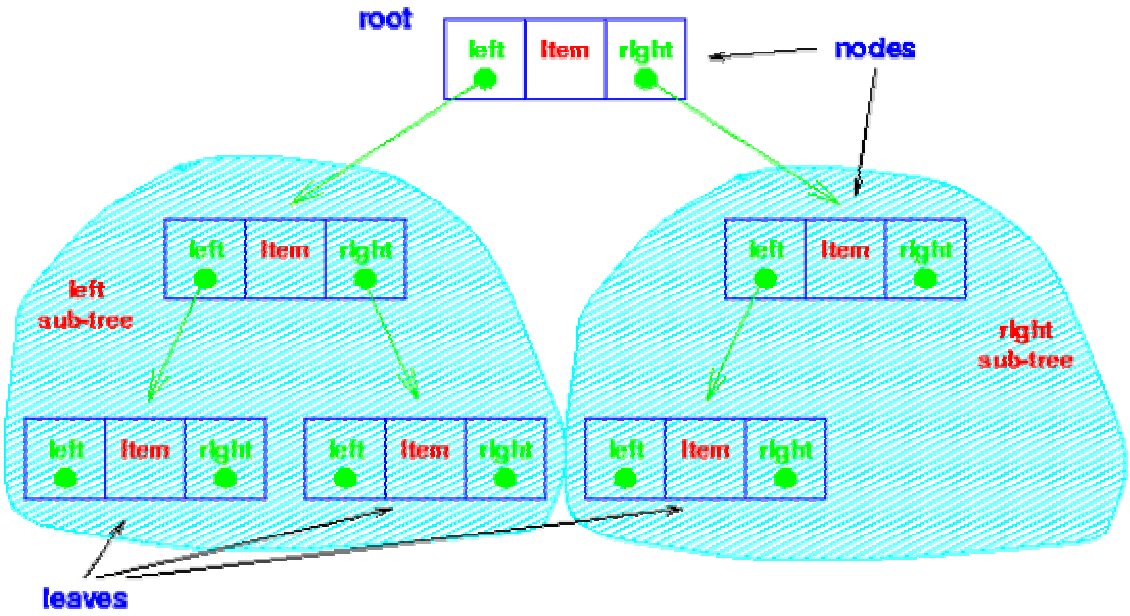
The basic operations on a binary search tree take time proportional to the height of the tree. For a complete binary tree with node n, such operations runs in $\Theta(\lg n)$ worst-case time. If the tree is a linear chain of n nodes, however, the same operations takes (n) worst-case time.



The height of the Binary Search Tree equals the number of links from the root node to the deepest node.

Implementation of Binary Search Tree

Binary Search Tree can be implemented as a linked data structure in which each node is an object with three pointer fields. The three pointer fields left, right and p point to the nodes corresponding to the left child, right child and the parent respectively NIL in any pointer field signifies that there exists no corresponding child or parent. The root node is the only node in the BTS structure with NIL in its p field.



Inorder Tree Walk

During this type of walk, we visit the root of a subtree between the left subtree visit and right subtree visit.

INORDER-TREE-WALK (*x*)

If $x \neq$NIL then
    INORDER-TREE-WALK (left[x])
    print key[x]
    INORDER-TREE-WALK (right[x])

It takes $\Theta(n)$ time to walk a tree of n nodes. Note that the Binary Search Tree property allows us to print out all the elements in the Binary Search Tree in sorted order.

Preorder Tree Walk

In which we visit the root node before the nodes in either subtree.

PREORDER-TREE-WALK (*x*)

```
If x not equal NIL then
    PRINT key[x]
    PREORDER-TREE-WALK (left[x])
    PREORDER-TREE-WALK (right[x])
```

Postorder Tree Walk

In which we visit the root node after the nodes in its subtrees.

POSTORDER-TREE-WALk (*x*)

```
If x not equal NIL then
    POSTORDER-TREE-WALK (left[x])
    PREORDER-TREE-WALK (right[x])
    PRINT key [x]
```

It takes $O(n)$ time to walk (inorder, preorder and pastorder) a tree of *n* nodes.

Binary-Search-Tree property Vs Heap Property

In a heap, a nodes key is greater than equal to both of its children's keys. In binary search tree, a node's key is greater than or equal to its child's key but less than or equal to right child's key. Furthermore, this applies to entire subtree in the binary search tree case. It is very important to note that the heap property does not help print the nodes in sorted order because this property does not tell us in which subtree the next item is. If the heap property could used to print the keys (as we have shown above) in sorted order in $O(n)$ time, this would contradict our known lower bound on comparison sorting.

The last statement implies that since sorting n elements takes $\Omega(n \lg n)$ time in the worst case in the comparison model, any comparison-based algorithm for

constructing a Binary Search Tree from arbitrary list n elements takes $\Omega(n \lg n)$ time in the worst case.

We can show the validity of this argument (in case you are thinking of beating $\Omega(n \lg n)$ bound) as follows: let $c(n)$ be the worst-case running time for constructing a binary tree of a set of n elements. Given an n-node BST, the inorder walk in the tree outputs the keys in sorted order (shown above). Since the worst-case running time of any computation based sorting algorithm is $\Omega(n \lg n)$, we have

$$c(n) + O(n) = \Omega(n \lg n)$$

Therefore, $$c(n) = \Omega(n \lg n).$$

Querying a Binary Search Tree

The most common operations performed on a BST is searching for a key stored in the tree. Other operations are MINIMUM, MAXIMUM, SUCCESSOR and PREDESESSOR. These operations run in $O(h)$ time where h is the height of the tree i.e., h is the number of links root node to the deepest node.

The TREE-SEARCH (x, k) algorithm searches the tree root at x for a node whose key value equals k. It returns a pointer to the node if it exists otherwise NIL

TREE-SEARCH (x, k)

```
if x = NIL      .OR.     k = key[x]
   then return x
if k < key[x]
   then return TREE-SEARCH (left[x], k)
   else return TREE-SEARCH (right[x], k)
```

Clearly, this algorithm runs in $O(h)$ time where h is the height of the tree.

The iterative version of above algorithm is very easy to implement.

ITERATIVE-TREE-SEARCH (*x*, *k*)

```
1. while x not equal NIL     .AND.     key ≠
   key[x] do
2.         if k < key [x]
3.                 then x ← left[x]
4.                 else x ← right [x]
5. return x
```

The TREE-MINIMUN (x) algorithm returns a point to the node of the tree at x whose key value is the minimum of all keys in the tree. Due to BST property, an minimum element can always be found by following left child pointers from the root until NIL is uncountered.

TREE-MINIMUM (x)

```
while left[x] ≠ NIL do
   x ← left [x]
return x
```

Clearly, it runs in $O(h)$ time where h is the height of the tree. Again thanks to BST property, an element in a binary search tree whose key is a maximum can always be found by following right child pointers from root until a NIL is encountered.

TREE-MAXIMUM (x)

```
while right[x] ≠ NIL do
   x ← right [x]
return x
```

Clearly, it runs in $O(h)$ time where h is the height of the tree.

The TREE-SUCCESSOR (x) algorithm returns a pointer to the node in the tree whose key value is next higher than key [x].

TREE-SUCCESSOR (x)

```
if right [x] ≠ NIL
   then return TREE-MINIMUM (right[x])
   else y ← p[x]
      while y ≠ NIL     .AND.    x = right[y] do
            x ← y
            y ← p[y]
      return y
```

Note that algorithm TREE-MINIMUM, TRE-MAXIMUM, TREE-SUCCESSOR, and TREE-PREDESSOR never look at the keys.

An inorder tree walk of an n-node BST can be implemented in $\Theta(n)$-time by finding the minimum element in the tree with TREE-MINIMUM (x) algorithm and then making n-1 calls to TREE-SUCCESSOR (x).

Another way of Implementing Inorder walk on Binary Search Tree

Algorithm

- find the minimum element in the tree with TREE-MINIMUM
- Make n-1 calls to TREE-SUCCESSOR

Let us show that this algorithm runs in $\Theta(n)$ time. For a tree T, let m_T be the number of edges that are traversed by the above algorithm. The running time of the algorithm for T is $\Theta(m_T)$. We make following claim:

m_T is zero if T has at most one node and $2e - r$ otherwise, where e is the number of edges in the tree and r is the length of the path from root to the node holding the maximum key.

Note that $e = n - 1$ for any tree with at least one node. This allows us to prove the claim by induction on e (and therefore, on n).

Base case Suppose that $e = 0$. Then, either the tree is empty or consists only of a single node. So, $e = r = 0$. Therefore, the claim holds.

Inductive step Suppose $e > 0$ and assume that the claim holds for all $e' < e$. Let T be a binary search tree with e edges. Let x be the root, and T1 and T2 respectively be the left and right subtree of x. Since T has at least one edge, either T1 or T2 respectively is nonempty. For each $i = 1, 2$, let e_i be the number of edges in Ti, p_i the node holding the maximum key in Ti, and r_i the distance from p_i to the root of Ti. Similarly, let e, p and r be the correspounding values for T. First assume that both T1 and T2 are nonempty. Then $e = e_1 + e_2 + 2$, $p = p_2$, and $r = r_2 + 1$. The action of the enumeration is as follows:

- Upon being called, the minimum-tree(x) traverses the left branch of x and enters T1.
- Once the root of T1 is visited, the edges of T1 are traversed as if T1 is the input tree. This situation will last until p_1 is visisted.
- When the Tree-Successor is called form p_1. The upward path from p_1 and x is traversed and x is discovered to hold the successor.
- When the tree-Successor called from x, the right branch of x is taken.
- Once the root of T2 is visited, the edges of T2 are traversed as if T2 is the input tree. This situation will last until p_2 is reached, whereby the algorithm halts.

By the above analysis, the number of edges that are traversed by the above algorithm, m_T, is

$$\begin{aligned} m_T &= 1 + (2e_1 - r_1) + (r_1 + 1) + 1 + (2e_2 - r_2) \\ &= 2(e_1 + e_2 + 2) - (r_2 + 1) \\ &= 2e - r \end{aligned}$$

Therefore, the claim clearly holds for this case.

Next suppose that T2 is emply. Since $e > 0$, T1 is nonempty. Then $e = e_1 + 1$. Since x does not have a right child, x holds the maximum. Therefore, $p = x$ and $r = 0$. The action of the enumeration algorithm is the first two steps. Therefore, the number of edges that are traversed by the algorithm in question is

$$\begin{aligned} m_T &= 1 + (2e_1 - r_1) + (r_1 + 1) \\ &= 2(e_1 + 1) - 0 \\ &= 2e - r \end{aligned}$$

Therefore, the claim holds for this case.

Finally, assume that T1 is empty. Then T2 is nonempty. It holds that $e = e_2 + 1$, $p = p2$, and $r = r2 + 1$. This time x holds the minimum key and the action of the enumeration algorithm is the last two steps. Therefore, the number of edges that are traversed by the algorithm is

$$\begin{aligned} m_T &= 1 + (2e_2 - r_2) \\ &= 2(e_2+1) - (r_2 + 1) \\ &= 2e - r \end{aligned}$$

Therefore, the claim holds for this case.

The claim is proven since $e = n - 1$, $m_T \leq 2n$. On the other hand, at least one edge has to be traversed when going from on node to another, so $m_T \geq n - 1$. Therefore, the running time of the above algorithm is $\Theta(n)$.

Consider any binary search tree T and let y be the parent of a leaf z. Our goal is to show that key[y] is

either the smallest key in T larger than key[x]

or the largest key in the T smaller than key[x].

Proof Suppose that x is a left child of y. Since key[y] ≥key[x], only we have to show that there is no node z with key[y] > key[z] > key[x]. Assume, to the contrary, that there is such a z. Choose z so that it holds the smallest key among such nodes. Note for every node $u \neq z$, x, key[z] ≤ dey[u] if and only if key[x] ≤ key[u]. If we search key[z], then the search path is identical to that of key[x] until the path rearches z or x. Since x is a leaf (meaning it has no children), the search path never reaches x. Therefore, z is an ancestor of x. Since y is the parent of x (it is given, in case you've forgotton!) and is not z, z has to be an ancestor of y. So, key[y] > dey[z] >dey[x]. However, we are assuming key[y] > key[z] > key[x], so this is clearly impossible. Therefore, there is no such z.

The case when x is a right child of y is easy. Hint: symmetric.

INSERTION

To insert a node into a BST

1. find a leaf st the appropriate place and
2. connect the node to the parent of the leaf.

TREE-INSERT (T, z)

```
y ← NIL
x ← root [T]
while x ≠ NIL do
    y ← x
    if key [z] <  key[x]
        then x ← left[x]
        else x ← right[x]
p[z] ← y
if y = NIL
    then root [T] ← z
    else if key [z] < key [y]
```

then left [y] ← z
else right [y] ← z

Like other primitive operations on search trees, this algorithm begins at the root of the tree and traces a path downward. Clearly, it runs in $O(h)$ time on a tree of height h.

Sorting

We can sort a given set of n numbers by first building a binary search tree containing these number by using TREE-INSERT (x) procedure repeatedly to insert the numbers one by one and then printing the numbers by an inorder tree walk.

Analysis

Best-case running time

Printing takes $O(n)$ time and n insertion cost $O(\lg n)$ each (tree is balanced, half the insertions are at depth $\lg(n) - 1$). This gives the best-case running time $O(n \lg n)$.

Worst-case running time

Printing still takes $O(n)$ time and n insertion costing $O(n)$ each (tree is a single chain of nodes) is $O(n^2)$. The n insertion cost 1, 2, 3, . . . n, which is arithmetic sequence so it is $n^2/2$.

Deletion

Removing a node from a BST is a bit more complex, since we do not want to create any "holes" in the tree. If the node has one child then the child is spliced to the parent of the node. If the node has two children then its successor has no left child; copy the successor into the node and delete the successor instead TREE-DELETE

(T, z) removes the node pointed to by z from the tree T. IT returns a pointer to the node removed so that the node can be put on a free-node list, etc.

TREE-DELETE (T, z)

```
1.  if left [z] = NIL    .OR.    right[z] = NIL
2.     then y ← z
3.     else y ← TREE-SUCCESSOR (z)
4.  if left [y] ≠ NIL
5.     then x ← left[y]
6.     else x ← right [y]
7.  if x ≠ NIL
8.     then p[x] ← p[y]
9.  if p[y] = NIL
10.    then root [T] ← x
11.    else if y = left [p[y]]
12.       then left [p[y]] ← x
13.       else right [p[y]] ← x
14. if y ≠ z
15.    then key [z] ← key [y]
16.       if y has other field, copy them, too
17. return y
```

The procedure runs in $O(h)$ time on a tree of height h.

EXECUTION FLOW BETWEEN SUBPROGRAMS

There are some functions that are used combi nely.These are;

void Operations(FILE * , FILE *); :it is used to open inout and output file , read the input file and call the assembly functions

stu_ptr deleteNode(int ,stu_ptr); : it is used tot delete a node with the given parameters in the binary search treee

void findGrade(char [], char [],stu_ptr,FILE *,int*); : it is used to find grade of the student with the given parameters in the binary search tree

extern int addNode(stu_ptr, int); : this assembly funcitons add an element to the binary search tree with the given parameter

extern void printTree(stu_ptr); : this assembly function is used top rint the whole tree and the name and surname of the students..

extern int calculateAverage(stu_ptr); : this assembly function is used to find the average grades of the students in the binary search tree

```
int depth(stu_ptr);
int MAX(int,int);
stu_ptr wander(stu_ptr,int);
void wanderAndWrite(stu_ptr,int,FILE *);
stu_ptr findMax(stu_ptr);
```

: these functions are used in delete and find grade functions they are as a sub program that makes the necessary operations on nodes and tree..

I will give you some example codes

```
int main (int argc , char *argv[])
```

```
{
    if (argc < 2)
    {
        printf("The parameters are wrong.\n");
        exit(-1);
    }
    else if(argc > 2)
    {
        printf("There are too much parameters..\n");
        exit(1);
    }
    else
    {
        FILE *input,*output;
        input = fopen(argv[1],"r");
        if((input = fopen(argv[1],"r")) == NULL)
        {
            printf("Input File can not be
opened.\n");
        }
        output = fopen("output.txt","w");
        if((output = fopen("output.txt","w")) ==
NULL)
        {
            printf("Output File can not be
opened.\n");
        }
        Operations(input,output);
    }
    return 0;
}// end of the main function
```

This is my operations function

```
void Operations(FILE * inputFile , FILE * outputFile)
{

    char name[20],surname[20];
    char command[10];
```

```
    int no,grade,result;
    int i = 0 ,a = 0, b = 0;
    stu_ptr root = (stu_ptr) malloc
(sizeof(student));
    root->leftChild = NULL ;
    root->rightChild = NULL ;
    //stu_ptr temp = (stu_ptr) malloc
(sizeof(student));
    while( !feof(inputFile) )
    {
        fscanf(inputFile, "%s", command);

        if(strcmp(command,"ADD") == 0)
        {
            stu_ptr tempstudent = (stu_ptr) malloc
(sizeof(student));
            tempstudent->leftChild = NULL ;
            tempstudent->rightChild = NULL ;
            fscanf(inputFile,"%d %s %s
%d",&no,name,surname,&grade);
            strcpy(tempstudent->name,name);
            strcpy(tempstudent->surname,surname);
            tempstudent->no = no;
            tempstudent->grade = grade ;
            if(b == 0)
            {
                strcpy(root->name,name);
                strcpy(root->surname,surname);
                root->no = no;
                root->grade = grade ;
                //printf("%s %s %d %d\n",root-
>name,root->surname,root->no,root->grade);
                i = addNode(root,no);
            }
            else
            {
                i = addNode(tempstudent,no);
                //fprintf(outputFile,"%d\n",i);
            }
            b++;
        }//if the line equals to "ADD"

        else if(strcmp(command,"DELETE") == 0)
        {
```

```
            stu_ptr temp ;
            fscanf(inputFile,"%d",&no);
            if(root == NULL)
            {
                fprintf(outputFile,"Tree is
empty\n");
            }
            else
            {
                fprintf(outputFile,"Route:");
                wanderAndWrite(root,no,outputFile);
                temp = wander(root,no);
                if(temp == NULL)
                {
                    printf("\n%d Node not
found!\n");
                    fprintf(outputFile,"\n%d Node
not found!\n",no);
                }
                else
                {
                    printf("\n %d is
deleted\n",temp->no);
                    deleteNode(no , temp);

                }
            }
        }//if the line equals to "DELETE"

        else if(strcmp(command,"AVERAGE") == 0)
        {
            int sum = 0 ,acounter = 0;
            acounter = calculateAverage(root);
            if(acounter == 0)
            {
                printf("Tree is empty.\n");
                fprintf(outputFile,"Tree is
empty.\n");
            }
            else
            {
                printf("Average: %d\n",acounter);
                fprintf(outputFile,"Average:
%d\n",acounter);
```

```
            }

        }//if the line equals to "AVERAGE"

        if(strcmp(command,"PRINT") == 0)
        {
            int e = 0 ;
            if(root)
            {
                printTree(root);
            }
            else
            {
                fprintf(outputFile,"Tree is
empty.\n");
            }

        }//if the line equals to "PRINT"


        else if(strcmp(command,"FINDGRADE") == 0)
        {
            int d ;
            d = 0;
            fscanf(inputFile,"%s %s",name,surname);

    findGrade(name,surname,root,outputFile,&d);
            if(d == 0)
            {
                printf("Node not found\n");
                fprintf(outputFile,"Node not
found\n");
            }

        }//if the line equals to "PRINT"
        else
        {
            //printf("This context is not suitable
for the suitable input.\n");
        }

    }

}//end of the Operation function
```

SOFTWARE TESTING NOTES

SOFTWARE RELIABILITY

It is a good program that has too little bugs.
Except this , this program runs correctly and it has lots of controls that user can give wrong expression.
It controls these events.
-User can give less parameters to the command line while executing the program
-User can give more parameters.(User can give unnecessary parameters to the command line..)
-User can give a wrong input or output file..

And I was not sure that we can put the results to the output file or command line , so I put the results to the both of this

But my program runs in 2 parameters.
Program name and input file..

SOFTWARE EXTENDIBILITY AND UPRADIBILITY

This program can be extended with some small changes some processes in the C programming language

And with this changes we will make an excellent processor and we can make a task manager from this source code..

We can combine the assembly and c source codes and we can make huge projects with them…

And we can change the subject of the program and we can make a lot of programs with a huge variety..

PERFORMANCE CONSİDERATIONS

This program runs rapidly and with some small changes we can make it faster then all.
And it gives the true results which are expected from us in a very little time.

COMMENTS

I think that this experiment will be so useful for us now and after these days.
For example I learn C and Assembly language again and I don’t forget it after this experiment.
And it was so useful to us to finding great ideas for the programming and it causes us to study much more than.
Briefly it was a so useful and good experiment.

And I take the Bil-235 course,because of this , program gives false results in sometimes rarely .So I apologise from you.And while you were giving the not for the experiment if you think about this I will be so appreciated to you..

Regards..

RESOURCES

How To Learn C/C++ by DEITEL
The C Programming Language 3rd Edition by BJORNE STROUSTRAP
Data Structures and Algorithms in C
Pc Book of the Assembly
The Art of the Assembly Language
İleri C Programlama by MURAT TAŞBAŞI
C Programlama Dili

These are the e-books I used for dealing with this experiment.

Web Sites

www.programlama.com
en.wikipedia.org/wiki/**Assembly**_language
thestarman.pcministry.com/asm/index.html