Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603 110
(An Autonomous Institution, Affiliated to Anna University, Chennai)
Computer Science and Engineering
CAT 2 – Assignment - Regulations – R2021

Degree & Branch	B.E & CSE				**Semester**	III
Subject Code & Name	**UCS 2302 – Data Structures**					
Academic Year	2022-2023 ODD/EVEN	**Batch**	2021-2025	**Date**	**05.01.2023**	**FN / AN**
Time: 11.00 am – 12.30 pm	**Answer All Questions**				**Maximum: 50 Marks**	

(i) Find an optimal method to solve the problem of Distribution of Smiley Ball Bags to children.
[CO6,K5]
[2.2.4, 3.2.1, 14.5.2]

Let A be the array of *n* integers where each value represents the number of smiley balls in a bag. Each bag can have a variable number of smiley balls. There are *m* children. The task is to choose m bags out of n bags and create it as a subset S such that S belongs to A and Maximum (S) - Minimum (S) should be the smallest. Write a program in C to find one subset with minimum difference. [CO5, K3]
[2.1.2, 2.3.1, 12.3.1, 13.3.1, 13.3.2, 1.4.1, 4.1.2]

Input : arr[] = {3, 4, 1, 8, 56, 7, 9, 12} , m = 3
Subset = {8, 7, 9}
Output: Minimum Difference is 2

Input : arr[] = {12, 4, 7, 9, 2, 23, 25, 41, 30,} , m = 5
Subset = {4, 7, 9, 2, 12}
Output: Minimum Difference is 10

Prepare a report containing the following: [10.1.1, 10.1.2]

i. Exploration of Design Alternatives [3]
ii. Rationale between choices [3]
iii. Identify modules with their prototypes in the program [2]
iv. Write appropriate ADT procedures to implement modules [12]
v. Discuss the time and space complexity of the solution [3]
vi. Program with Output snapshot [2]

(ii) Choose an efficient data structure to construct English Dictionary by the following operations with minimum time complexity. [CO5, K3]
[2.1.2, 2.3.1, 12.3.1, 13.3.1, 13.3.2, 1.4.1, 4.1.2]

- Insertion of word and meaning into the dictionary
- Display all the words along with meanings in the dictionary
- Find the meaning given a word from the dictionary

Prepare a report containing the following: [10.1.1, 10.1.2]

vii. Exploration of Design Alternatives [3]
viii. Rationale between choices [3]
ix. Identify modules with their prototypes in the program [2]

x.	Write appropriate ADT procedures to implement modules	[12]
xi.	Discuss the time and space complexity of the solution	[3]
xii.	Program with Output snapshot	[2]

Name:Sai Rahul

CSE B

3122215001090

(i) Find an optimal method to solve the problem of Distribution of Smiley Ball Bags to children.
[CO6,K5]
[2.2.4, 3.2.1, 14.5.2]

Let A be the array of *n* integers where each value represents the number of smiley balls in a bag. Each bag can have a variable number of smiley balls. There are *m* children. The task is to choose m bags out of n bags and create it as a subset S such that S belongs to A and Maximum (S) - Minimum (S) should be the smallest. Write a program in C to find one subset with minimum difference. [CO5, K3]
[2.1.2, 2.3.1, 12.3.1, 13.3.1, 13.3.2, 1.4.1, 4.1.2]

Input : arr[] = {3, 4, 1, 8, 56, 7, 9, 12} , m = 3
Subset = {8, 7, 9}
Output: Minimum Difference is 2

Input : arr[] = {12, 4, 7, 9, 2, 23, 25, 41, 30,} , m = 5
Subset = {4, 7, 9, 2, 12}
Output: Minimum Difference is 10

Prepare a report containing the following: [10.1.1, 10.1.2]

i.	Exploration of Design Alternatives	[3]
ii.	Rationale between choices	[3]
iii.	Identify modules with their prototypes in the program	[2]
iv.	Write appropriate ADT procedures to implement modules	[12]
v.	Discuss the time and space complexity of the solution	[3]
vi.	Program with Output snapshot	[2]

i)we can use ***queues*** as an alternative

We use **stack or queue** instead of arrays/lists when we want the elements in a specific order i.e. in the order we put them (queue) or in the reverse order (stack)

Stack is **a linear data structure that follows a particular order in which the operations are performed**. The order may be LIFO(Last In First Out) or FILO(First In Last Out). Stack Data Structure. There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen.

A Queue is defined as **a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order**. We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end.

ii)
Array is better than queue/stack in this way:
Insertion and deletion in Queues takes place only from rear and front respectively. **Insertion and deletion in array can be done at any index in the array**. Insertion and deletion in stacks take place only from one end of the list called the top.

iii) Identify modules with their prototypes in the program

- struct array get_submaxarr(int arr[],int n, int y)
  type:struct

  arguments: int arr[],int n, int y

  input:no input taken from the user.
  But uses the arguments as inputs

  Output:submaxarray is returned

- void getMinDifference(int Arr[], int N,int Y)
  - type: void
  - arguments: int Arr[], int N,int Y
  - input:no input taken from the user.
    But uses the arguments as inputs
  - Output:Mindifference

iv) Write appropriate ADT procedures to implement modules
the fns are:

- struct array get_submaxarr(int arr[],int n, int y)
  type:struct

  arguments: int arr[],int n, int y

  input:no input taken from the user.
  But uses the arguments as inputs

Output:submaxarray is returned

Algorithm:

Initialize struct array get_submaxarr(int arr[],int n, int y)

initialize truct array maxarray,stack1, submax
int j,p,i = 0

set maxarray.len=0
set stack1.len=-1
set submax.len=0
set stack1.data[++stack1.len]=0

for (i = 1; i < n; i++) =>
while (stack1.len >=0 && arr[i] > arr[stack1.data[stack1.len]])

maxarray.data[stack1.data[stack1.len]] = i
stack1.len--

stack1.data[++stack1.len]=i

while (stack1.len>=0)

maxarray.data[stack1.data[stack1.len]] = n
stack1.len--

j=0
for (i = 0; i <= n - y; i++) =>
while (maxarray.data[j] < i + y - 1 || j < i) =>
j++

submax.data[submax.len++]=arr[j]

return submax

- void getMinDifference(int Arr[], int N,int Y)

  type: void

  arguments: int Arr[], int N,int Y

  input:no input taken from the user.
  But uses the arguments as inputs

Output:
Mindifference

Algorithm:

```
void getMinDifference(int Arr[], int N,int Y)
defining the fn and the arguements
        struct array submin     = get_subminarr(Arr, N, Y)
int i,j,p=0 initialized
        struct array submax= get_submaxarr(Arr, N, Y)
        int minn = submax.data[0] - submin.data[0]
        int dif
        int b = submax.len

        for ( i = 1; i < b; i++) in this case

                dif = submax.data[i] - submin.data[i]
                if (minn>dif)=>

                p=i
                minn=dif



   print("\nSubset ...\n")
        for(i=0;i<N;i++)

                if(Arr[i]=submin.data[p])

                        for(j=i;j>=i-Y+1;j--)
                        print(Arr[j])
                        break



                print("\nMinium differnce %d \n",minn)
```

v)
the time complexity is o(n^2)

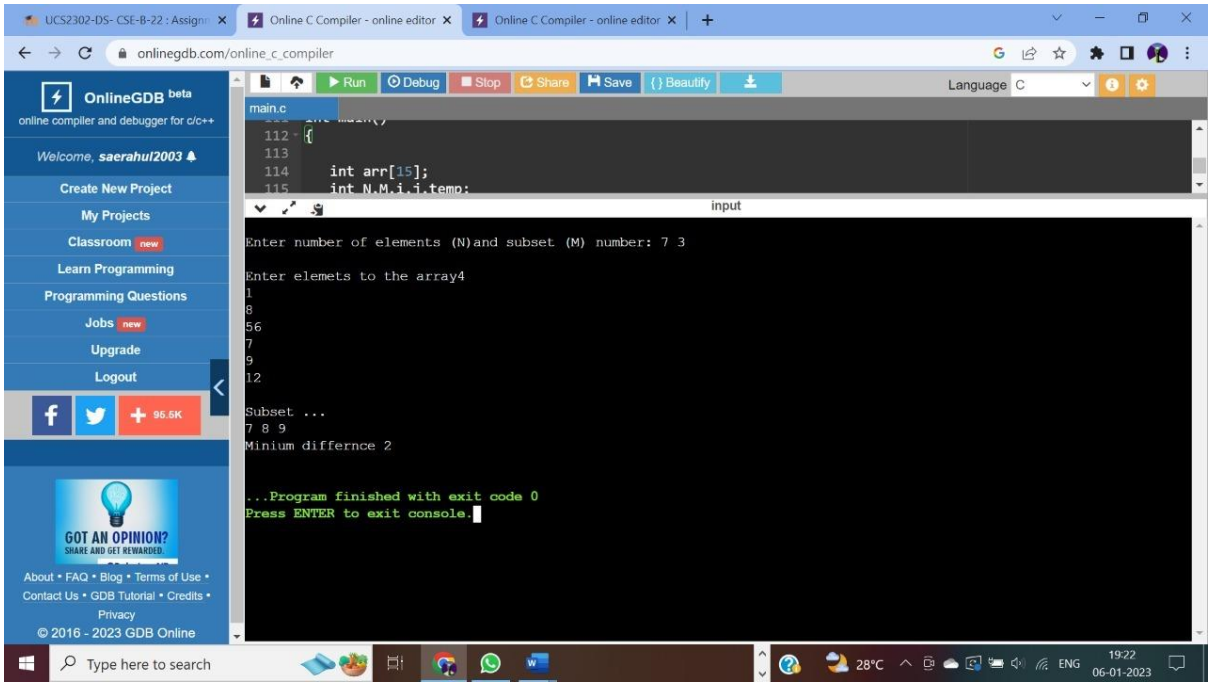
Space complexity refers to **the total amount of memory space used by an algorithm/program, including the space of input values for execution**. Calculate the space occupied by variables in an algorithm/program to determine space complexity.

vi)
Code:

```
#include <stdio.h>


struct array
{int len;
 int data[10];
};
struct array get_subminarr(int arr[],int n, int y)
{
        struct array minarray,stack, submin;
                int j=0,p,i = 0;

        minarray.len=0;
        stack.len=-1;
        submin.len=0;
        stack.data[++stack.len]=0;

        for (i = 1; i < n; i++) {
                while (stack.len >=0 && arr[i] < arr[stack.data[stack.len]]) {

                        minarray.data[stack.data[stack.len]] = i;
                        stack.len--;
                }
                stack.data[++stack.len]=i;
        }
        while (stack.len>=0) {

                minarray.data[stack.data[stack.len]] = n;
                stack.len--;
        }

        for (i = 0; i <= n - y; i++) {
                while ( minarray.data[j] <= i + y - 1 || j < i) {
                        j++;
                }
      submin.data[submin.len++]=arr[j];
        }
        return submin;
}

struct array get_submaxarr(int arr[],int n, int y)
{
        struct array maxarray,stack1, submax;
```

```
        int j,p,i = 0;

    maxarray.len=0;
    stack1.len=-1;
    submax.len=0;
    stack1.data[++stack1.len]=0;

    for (i = 1; i < n; i++) {
        while (stack1.len >=0 && arr[i] > arr[stack1.data[stack1.len]]) {

            maxarray.data[stack1.data[stack1.len]] = i;
            stack1.len--;
        }
        stack1.data[++stack1.len]=i;
    }
    while (stack1.len>=0) {

        maxarray.data[stack1.data[stack1.len]] = n;
        stack1.len--;
    }
  j=0;
    for (i = 0; i <= n - y; i++) {
        while ( maxarray.data[j] < i + y - 1 || j < i) {
            j++;
        }
     submax.data[submax.len++]=arr[j];
    }
    return submax;
}

void getMinDifference(int Arr[], int N,int Y)
{
    struct array submin     = get_subminarr(Arr, N, Y);
int i,j,p=0;
    struct array submax= get_submaxarr(Arr, N, Y);
    int minn = submax.data[0] - submin.data[0];
    int dif;
    int b = submax.len;

    for ( i = 1; i < b; i++) {

        dif = submax.data[i] - submin.data[i];
        if (minn>dif)
        {
        p=i;
        minn=dif;
    }

    }
  printf("\nSubset ...\n");
```

```
        for(i=0;i<N;i++)
        {
                if(Arr[i]==submin.data[p])
                {
                        for(j=i;j>=i-Y+1;j--)
                        printf("%d ",Arr[j]);
                        break;
                }
        }

                printf("\nMinium differnce %d \n",minn);



}

int main()
{

  int arr[15];
  int N,M,i,j,temp;
  printf("\nEnter number of elements (N)and subset (M) number: ");
  scanf("%d%d",&N,&M);
  printf("\nEnter elemets to the array");

  for(i=0;i<N;i++)
  scanf("%d",&arr[i]);
for(i=0;i<N;i++)
  for(j=0;j<N;j++)
  if(arr[i]>arr[j])
  {
        temp=arr[i];
        arr[i]=arr[j];
        arr[j]=temp;
  }
        getMinDifference(arr, N, M);
        return 0;
}
```
Output:



2)

(ii) Choose an efficient data structure to construct English Dictionary by the following operations with minimum time complexity. [CO5, K3]

[2.1.2, 2.3.1, 12.3.1, 13.3.1, 13.3.2, 1.4.1, 4.1.2]

- Insertion of word and meaning into the dictionary
- Display all the words along with meanings in the dictionary
- Find the meaning given a word from the dictionary

Prepare a report containing the following: [10.1.1, 10.1.2]
vii)Exploration of Design Alternatives [3]
viii)Rationale between choices [3]
ix)Identify modules with their prototypes in the program [2]
x)Write appropriate ADT procedures to implement modules [12]
x1)Discuss the time and space complexity of the solution [3]
xii)Program with Output snapshot [2]

ANSWERS:

vii)AVL is a data alternative

The AVL tree is always height-balanced, and its height is always O(log N), where N is the total number of nodes in the tree

AVL trees have a faster retrieval. The first balanced binary tree in computing history was the AVL tree which introduced the wide area of balanced binary search trees. An AVL tree is a binary search tree which has an additional balanced condition.

Hash is a data alternative

Hashing in Java is the technique that **enables us to store the data in the form of key-value pairs**, by modifying the original key using the hash function so that we can use these modified keys as the index of an array and store the associated data at that index location in the Hash table for each key.

viii) Rationale between choices

BST Has been used as

A BST **supports operations like search, insert, delete, floor, ceil, greater, smaller, etc in O(h) time where h is height of the BST**. To keep height less, self balancing BSTs (like AVL and Red Black Trees) are used in practice. These Self-Balancing BSTs maintain the height as O(Log n).

BSTs are **easy to implement** compared to hashing, we can easily implement our own customized BST. To implement Hashing, we generally rely on libraries provided by programming languages. With Self-Balancing BSTs, all operations are guaranteed to work in O(Logn) time.

ix) Identify modules with their prototypes in the program

void insert(char *word, char *meaning)

- input:the arguments
- output:the word and the meaning is inserted
- type:void
- arguments: char *word, char *meaning

void find(char *str)

input:arguments
output:returns if element is found or not
arguments: char *str
type:void

void inorder(struct BST *node)

type:void
arguments: struct BST *node
input:arguments
output: node->word,node->meanin

void trimTrailing(char * str)

input:argument (str)
o/p:sets index
type:void
arguments: char * str

x)
Write appropriate ADT procedures to implement modules

- void insert(char *word, char *meaning)
    - input:the arguments
    - output:the word and the meaning is inserted
    - type:void
    - arguments: char *word, char *meaning

Algorithm:

```
Initialize void insert(char *word, char *meaning)
    struct BST*parent = NULL, *ptr = NULL, *newnode = NULL
    int res = 0
    if (!root)
        root = create(word, meaning)
        return

    for (ptr = root; ptr !=NULL;ptr = (res > 0) ? ptr->right : ptr->left)=

        res = strcasecmp(word, ptr->word);
        if (res == 0)
            printf("\nDuplicate entry!!\n")
            return

        parent = ptr

    newnode = create(word, meaning)
    res > 0 ? (parent->right = newnode)  (parent->left = newnode)
    return
```

- void find(char *str)
  input:arguments
  output:returns if element is found or not
  arguments: char *str
  type:void

Algorithm:

```
Initialize  void find(char *str)
```

```
struct BST *temp = NULL
int flag = 0, res = 0
if (root = NULL)
        return

temp = root
while (temp) =
        if ((res = strcasecmp(temp->word, str)) == 0)
                print("\nWord   : %s", str)
                print("\nMeaning: %s", temp->meaning)
                flag = 1
                break

        temp = (res > 0) ? temp->left : temp->right

if (!flag)
        print("\nSearch Element not found in Binary Search Tree\n")
return
```

- void inorder(struct BST *node)
  type:void
  arguments: struct BST *node
  input:arguments
  output: node->word,node->meanin

  Algorithm:

Initialize void inorder(struct BST *node)

```
        if (node)
                inorder(node->left)
                printf("%s\t%s\n", node->word,node->meaning)
                //printf("\n")
                inorder(node->right)

        return
```

void trimTrailing(char * str)
input:argument (str)
o/p:sets index
type:void
arguments: char * str

initialize void trimTrailing(char * str)

```
    int index, i
            index = -1
```

```
i = 0
while(str[i] != '\0')=>

    if(str[i] != ' ' && str[i] != '\t' && str[i] != '\n')

        index= i


    i++


/* Mark next character to last non-white space character as NULL */
str[index + 1] = '\0'
```

xi)
time complexity:
O(1) average :O(log n) and worst O(n)(skewed trees)

Space Complexity:
Space complexity refers to **the total amount of memory space used by an algorithm/program, including the space of input values for execution**. Calculate the space occupied by variables in an algorithm/program to determine space complexity. However, people frequently confuse Space-complexity with auxiliary space.

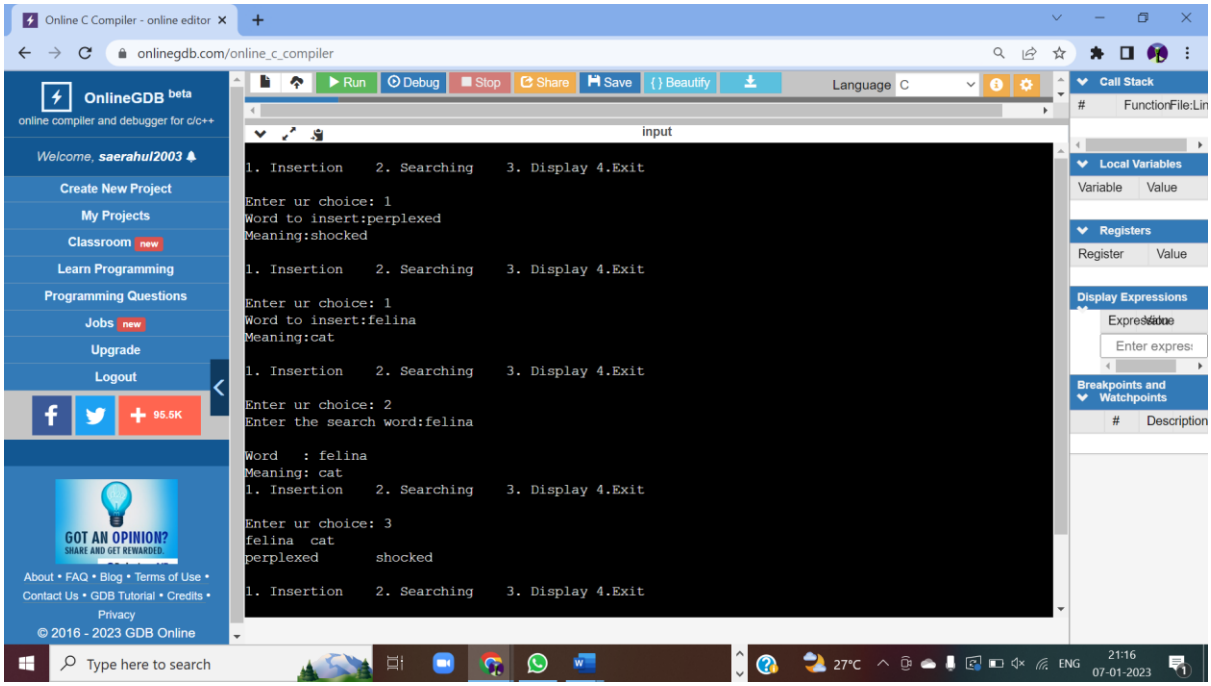
xii)

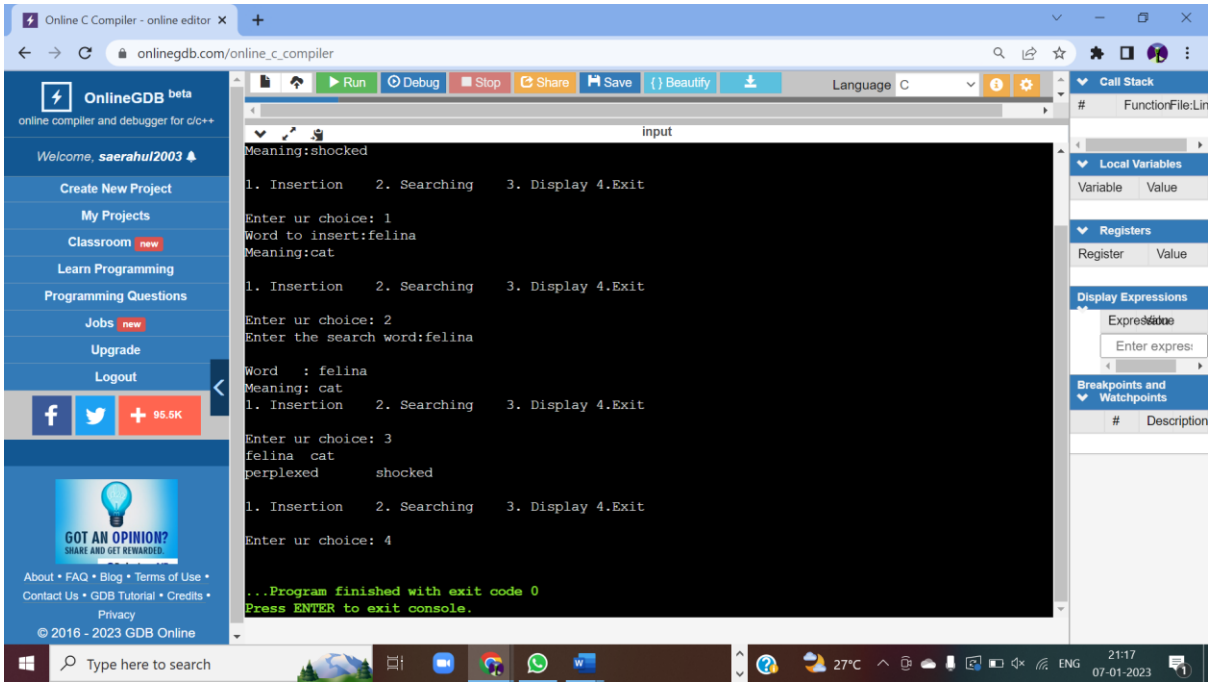
```
#include <stdio.h>
 #include <stdlib.h>
 #include <string.h>

 struct BST {
     char word[128], meaning[256];
     struct BST *left, *right;
 };

 struct BST *root = NULL;

 struct BST * create(char *word, char *meaning) {
     struct BST *newnode;
     newnode = (struct BST *)malloc(sizeof(struct BST));
     strcpy(newnode->word, word);
     strcpy(newnode->meaning, meaning);
     newnode->left = newnode->right = NULL;
```

```
    return newnode;
}

void insert(char *word, char *meaning) {
    struct BST*parent = NULL, *ptr = NULL, *newnode = NULL;
    int res = 0;
    if (!root) {
        root = create(word, meaning);
        return;
    }
    for (ptr = root; ptr !=NULL;ptr = (res > 0) ? ptr->right : ptr->left)
            {
        res = strcasecmp(word, ptr->word);
        if (res == 0) {
            printf("\nDuplicate entry!!\n");
            return;
        }
        parent = ptr;
    }
    newnode = create(word, meaning);
    res > 0 ? (parent->right = newnode) : (parent->left = newnode);
    return;
}



void find(char *str) {
    struct BST *temp = NULL;
    int flag = 0, res = 0;
    if (root == NULL) {
        return;
    }
    temp = root;
    while (temp) {
        if ((res = strcasecmp(temp->word, str)) == 0) {
            printf("\nWord   : %s", str);
            printf("\nMeaning: %s", temp->meaning);
            flag = 1;
            break;
        }
        temp = (res > 0) ? temp->left : temp->right;
    }
    if (!flag)
        printf("\nSearch Element not found in Binary Search Tree\n");
    return;
}

void inorder(struct BST *node) {
    if (node) {
        inorder(node->left);
```

```
            printf("%s\t%s\n", node->word,node->meaning);
            //printf("\n");
            inorder(node->right);
        }
        return;
  }
void trimTrailing(char * str)
{
   int index, i;
          index = -1;
   i = 0;
   while(str[i] != '\0')
   {
      if(str[i] != ' ' && str[i] != '\t' && str[i] != '\n')
      {
         index= i;
      }

      i++;
   }

   /* Mark next character to last non-white space character as NULL */
   str[index + 1] = '\0';
}

  int main() {
        int ch;
        char str[20], meaning[25];
        while (1) {
              printf("\n1. Insertion\t2. Searching\t3. Display 4.Exit\n");
              printf("\nEnter ur choice: ");
              scanf("%d", &ch);
              getchar();
              switch (ch) {
                    case 1:
                          printf("Word to insert:");
                          fgets(str, 100, stdin);
                          trimTrailing(str);
                          printf("Meaning:");

                          fgets(meaning, 100, stdin);
                          trimTrailing(meaning);
                          insert(str, meaning);
                          break;
                    case 2:
                          printf("Enter the search word:");
                          fgets(str, 100, stdin);
                          trimTrailing(str);
                          find(str);
                          break;
```

```
                case 3:
                        inorder(root);
                        break;
                case 4:
                        exit(0);
                default:
                        printf("You have entered wrong option\n");
                        break;
            }
        }
        return 0;
}
```




Online C Compiler - online editor
onlinegdb.com/online_c_compiler
OnlineGDB beta
online compiler and debugger for c/c++
Welcome, saerahul2003
Create New Project
My Projects
Classroom new
Learn Programming
Programming Questions
Jobs new
Upgrade
Logout
95.5K
GOT AN OPINION?
SHARE AND GET REWARDED.
About • FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy
© 2016 - 2023 GDB Online
Run
Debug
Stop
Share
Save
{ } Beautify
Language C
input
Meaning:shocked
1. Insertion 2. Searching 3. Display 4.Exit
Enter ur choice: 1
Word to insert:felina
Meaning:cat
1. Insertion 2. Searching 3. Display 4.Exit
Enter ur choice: 2
Enter the search word:felina
Word : felina
Meaning: cat
1. Insertion 2. Searching 3. Display 4.Exit
Enter ur choice: 3
felina cat
perplexed shocked
1. Insertion 2. Searching 3. Display 4.Exit
Enter ur choice: 4
...Program finished with exit code 0
Press ENTER to exit console.
Call Stack
Local Variables
Variable Value
Registers
Register Value
Display Expressions
Breakpoints and Watchpoints
Type here to search
27°C
ENG
21:17
07-01-2023