**Programming Languages Dictionary**

**Table of Contents**

1. Complexity Table
2. The Main Question in Computer Science
3. Chapter 1:
4. Declarations, Queries and Logic
5. Programming Challenges
6. Two Examples of Link List
7. AI Presentation and Challenges
8. Binary Search in CPP and Python
9. Lisp Functions

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Complexity Table**

0(1) – Constant number of steps

Log2n – Binary search

N – Min, max, linear search

n Log2n – Merge sort, quick sort, leap sort

n2 – Bubble sort, standard, selection sort

n3 – Sorting matrix

2n – Subset sum

n! – Combinational problem

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**The Main Open Question in Computer Science**

Slogan – “Hard to solve, easy to verify”

? P = N P

P – Class of problems with polynomial algorithms

NP – Class of problems with non-deterministically polynomial algorithms

Reference: <https://www.claymath.org/millennium-problems/p-vs-np-problem>

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Chapter 1:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Declarations, Queries and Logic**

**FAMILY TREE IN PROLOG**

male(bob).

male(opa).

male(scott).

male(jay).

male(spencer).

male(colby).

male(jarod).

male(bryce).

female(carol).

female(oma).

female(kelly).

female(melanie).

female(brittany).

female(lauren).

parent(opa,scott).

parent(opa,melanie).

parent(oma,scott).

parent(oma,melanie).

parent(bob, kelly).

parent(bob, jay).

parent(carol, kelly).

parent(carol, jay).

parent(scott, brittany).

parent(scott, spencer).

parent(scott, colby).

parent(kelly, brittany).

parent(kelly, spencer).

parent(kelly, colby).

parent(melanie, lauren).

parent(melanie, bryce).

parent(jarod, lauren).

parent(jarod, bryce).

father(X,Y):-male(X),

parent(X,Y).

mother(X,Y):-female(X),

parent(X,Y).

grandfather(X,Y):-male(X),

parent(X,P), parent(P,Y), X \= Y.

grandmother(X,Y):-female(X),

parent(X,P), parent(P,Y), X \= Y.

brother(X,Y):-male(X),

mother(M,Y), mother(M,X), X \= Y.

sister(X,Y):-female(X),

mother(M,Y), mother(M,X), X \= Y.

aunt(X,Y):- female(X),

parent(Z,Y), sister(X,Z).

uncle(X,Y):- male(X),

parent(Z,Y), brother(Z,X).

cousin(X,Y) :- parent(Z, X),

parent(W,Y),

sibling(Z,W),

X \= Y.

ancestor(X,Y):- parent(X,Y).

ancestor(X,Y):- parent(X,Z),

ancestor(Z,Y).

descendent(X,Y) :-

ancestor(Y,X).

sibling(X, Y) :- parent(Z, X),

parent(Z, Y), X \= Y.

relative(X,Y) :- ancestor(Z,X),

ancestor(Z,Y),

X \= Y.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Programming Challenges**

**Swap without Using a New Variable**

void swapNoNewVar(int x, int y) {

cout << "Without Using a New Variable: x becomes x+y -> y becomes x-y -> x becomes x-y" << endl;

cout << "Original: " << x << " and " << y << endl;

x = x + y;

y = x - y;

x = x - y;

cout << "Output: " << x << " and " << y << endl << endl;

}

**Swap Using a New Variable**

void swapNewVar(int x, int y) {

cout << "Using a New Variable: z will hold y -> y then holds x -> x then holds z" << endl;

cout << "Original: " << x << " and " << y << endl;

int z;

z = y;

y = x;

x = z;

cout << "Output: " << x << " and " << y << endl << endl;

}

**Template Swap**

//Array no extra space

case 2:

cout << " Pseudocode for Reversing an Array without Allocating Extra Space" << endl;

cout << R"(

for(int i = 0; i < arrlen / 2; i++){

swap(a[i],a[arrlen - 1 - i]);

}

for(int i = 0, int j = size - 1; i < j; i++, j--){

swap(a[i], a[j]);

}

)" << endl;

break;

**Complexity n and n2**

void array\_accommodation(int\* a, int size);

void print\_array(int\* a, int size);

void linear\_search(int\* a, int target, int size);

void bubble\_sort(int\* a, int size);

int main() {

int \*a;

int size;

printf("How many numbers do you want to enter? \n");

printf("\n");

scanf\_s("%d", &size);

a = (int\*)calloc(size, sizeof(int));

array\_accommodation(a, size);

printf("Enter target number to find in array: \n");

scanf\_s("%d", &target);

bubble\_sort(a, size);

cout << endl << "Output from Complexity n^2" << endl;

print\_array(a, size);

cout << endl << "Output from Complexity n" << endl;

linear\_search(a, target, size);

break;

bubble\_sort(a, size);

}

void array\_accommodation (int\* a, int size) {

int i;

printf("Please enter %d numbers in a row: ", size);

for (i = 0; i < size; i++)

scanf\_s("%d", &a[i]);

}

void linear\_search(int\* a, int target, int size)

{

int found = 0;

int i;

for (i = 0; i < size; i++) {

if (a[i] == target) {

found = 1;

break;

}

}

if (found == 1)

printf("The target number is in the array at position %d. \n", i + 1);

if (found == 0)

printf("The target number is not in the array. \n");

}

void bubble\_sort(int\* a, int size){

int i, j;

int temp;

for (j = 0; j < size - 1; j++)

for (i = 0; i < size - 1; i++)

if (a[i] > a[i + 1]) {

temp = a[i];

a[i] = a[i + 1];

a[i + 1] = temp;

}

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Two Examples of Link List**

English Example First then Pseudocode

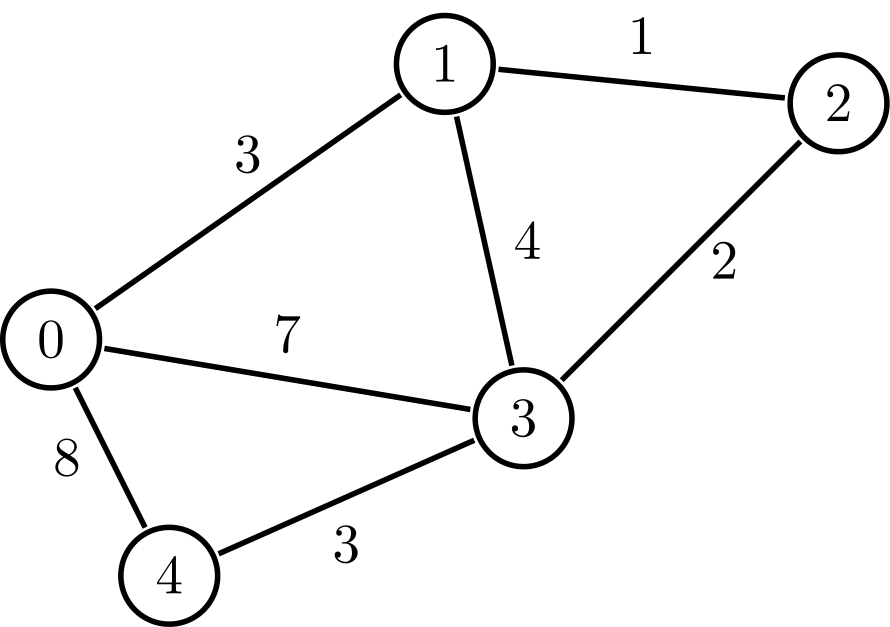
for (temp = head; temp! = NULL)

cout << temp->data;

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Dynamic Programming: Constructing Solutions Step by Step**

**Djikstra Graphing Nodes**

0 3 0 7 11

3 0 1 0 0

0 1 0 2 0

7 0 2 0 3

11 0 0 3 0

Numbers as input.txt file

#include "pch.h"

#include<iostream>

#include<fstream>

using namespace std;

#define NUMBER\_OF\_NODES 5

#define INF 99999

struct Node

{

int arrow[NUMBER\_OF\_NODES];

};

void initialize(int start, int\* distance, int\* predecessor, int\* allowed, struct Node \*node);

void one\_round(int next, int\* distance, int\* predecessor, int\* allowed, struct Node \*node);

void graph\_read(ifstream &instream, struct Node \*node);

void travel(int next, int j, int\* distance, int\* predecessor, int\* allowed, struct Node \*node);

int find\_next(int\* distance, int\* allowed);

void print\_arrays(int\* distance, int\* predecessor, int\* allowed);

//void shortest\_path(int start, int destination, int\* predecessor);

int main()

{

ifstream instream;

Node node[NUMBER\_OF\_NODES];

int i, j;

int distance[NUMBER\_OF\_NODES];

int allowed[NUMBER\_OF\_NODES];

int predecessor[NUMBER\_OF\_NODES];

int start, destination;

int next;

int NUMBER\_OF\_ALLOWED\_NODES = 1;

cout << "Please enter the start node" << endl;

cin >> start;

cout << "First Node is " << start << endl;

cout << "Please enter the destination node" << endl;

cin >> destination;

initialize(start, distance, predecessor, allowed, node);

cout << "Initialize" << endl;

print\_arrays(distance, predecessor, allowed);

instream.open("input.txt");

graph\_read(instream, node);

next = start;

while (NUMBER\_OF\_ALLOWED\_NODES < NUMBER\_OF\_NODES)

{

one\_round(next, distance, predecessor, allowed, node);

next = find\_next(distance, allowed);

NUMBER\_OF\_ALLOWED\_NODES++;

}

cout << "Round 1" << endl;

print\_arrays(distance, predecessor, allowed);

}

void initialize(int start, int\* distance, int\* predecessor, int\* allowed, struct Node \*node)

{

int i;

for (i = 0; i < NUMBER\_OF\_NODES; i++)

{

if (i == start)

{

distance[i] = 0 ;

predecessor[i] = i ;

allowed[i] = 1 ;

}

else

{

distance[i] = INF;

predecessor[i] = -1 ;

allowed[i] = 0 ;

}

}

}

void graph\_read(ifstream &instream, struct Node \*node)

{

int i, j;

for (j = 0; j < NUMBER\_OF\_NODES; j++)

for (i = 0; i < NUMBER\_OF\_NODES; i++)

instream >> node[j].arrow[i];

}

void print\_arrays(int\* distance, int\* predecessor, int\* allowed)

{

int i;

cout << "Printing distance, allowed and predecessor arrays" << endl;

cout << "The distance array is" << endl;

for (i = 0; i < NUMBER\_OF\_NODES; i++)

cout << distance[i] << " ";

cout << endl;

cout << "The predecessor array is" << endl;

for (i = 0; i < NUMBER\_OF\_NODES; i++)

cout << predecessor[i] << " ";

cout << endl;

cout << "The allowed array is" << endl;

for (i = 0; i < NUMBER\_OF\_NODES; i++)

cout << allowed[i] << " ";

cout << endl;

}

void travel(int next, int j, int\* distance, int\* predecessor, int\* allowed, struct Node \*node)

{

int new\_distance;

new\_distance = distance[next] + j;

if (new\_distance < distance[j])

{

distance[j] = new\_distance ;

predecessor[j] = next;

}

}

void one\_round(int next, int\* distance, int\* predecessor, int\* allowed, struct Node \*node)

{

int j;

//allowed[next] = 1;

for (j = 0; j < NUMBER\_OF\_NODES; j++)

if (allowed[j] == 0 && node[next].arrow[j] != -1)

travel(next, j, distance, predecessor, allowed, node);

}

int find\_next(int\* distance, int\* allowed) {

int min\_distance, i;

int next = 1;

min\_distance = INF;

for (i = 0; i < NUMBER\_OF\_NODES; i++) {

if (allowed[i] == 0 && distance[i] <= min\_distance)

{

min\_distance = distance[i] ;

next = i;

}

allowed[next] = 1;

return next;

}

}

**Djikstra Reading Nodes**

/\*

20

10 | 30 |

7 | 15 | 26| 35|

3| 8| 12| |19 x| x| x| x|

\*/

#include "pch.h"

#include <iostream>

using namespace std;

struct Node {

int data;

struct Node \*left, \*right;

};

int find\_min(Node \*root);

int main()

{

Node \*root, \*temp;

//first node with 20

temp = new Node;

temp->data = 20;

temp->left = NULL;

temp->right = NULL;

root = temp;

//node with 30

temp = new Node;

temp->data = 30;

temp->left = NULL;

temp->right = NULL;

root->right = temp;

//node with 26

temp = new Node;

temp->data = 26;

temp->left = NULL;

temp->right = NULL;

root->right->left = temp;

//node with 10

temp = new Node;

temp->data = 10;

temp->left = NULL;

temp->right = NULL;

root->left = temp;

//node with 15

temp = new Node;

temp->data = 15;

temp->left = NULL;

temp->right = NULL;

root->left->right = temp;

//node with 7

temp = new Node;

temp->data = 7;

temp->left = NULL;

temp->right = NULL;

root->left->left = temp;

//node with 3

temp = new Node;

temp->data = 3;

temp->left = NULL;

temp->right = NULL;

root->left->left->left = temp;

//node with 8

temp = new Node;

temp->data = 8;

temp->left = NULL;

temp->right = NULL;

root->left->left->right = temp;

//node with 12

temp = new Node;

temp->data = 12;

temp->left = NULL;

temp->right = NULL;

root->left->right->left = temp;

//node with 19

temp = new Node;

temp->data = 19;

temp->left = NULL;

temp->right = NULL;

root->left->right->right = temp;

//node with 35

temp = new Node;

temp->data = 35;

temp->left = NULL;

temp->right = NULL;

root->right->right = temp;

find\_min(root);

//Used to test data values

//cout << root->left->left->data << endl;

}

int find\_min(Node \*root) {

int min = 0;

Node \*temp;

temp = root;

while (temp->left != root) {

temp = temp->left;

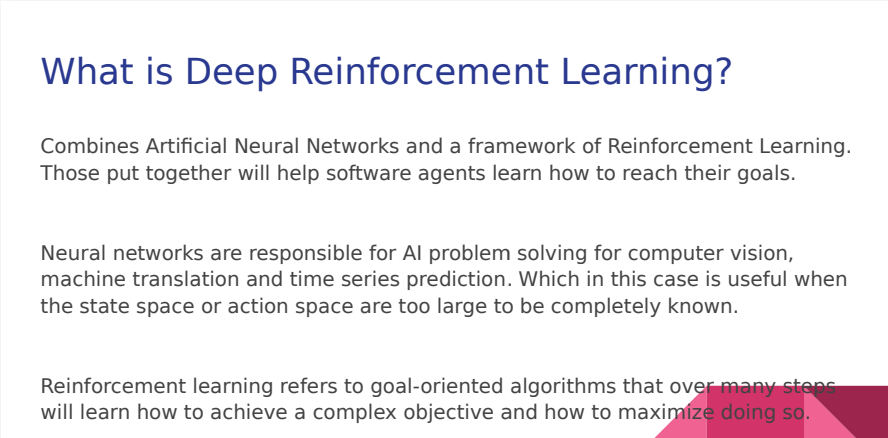
};

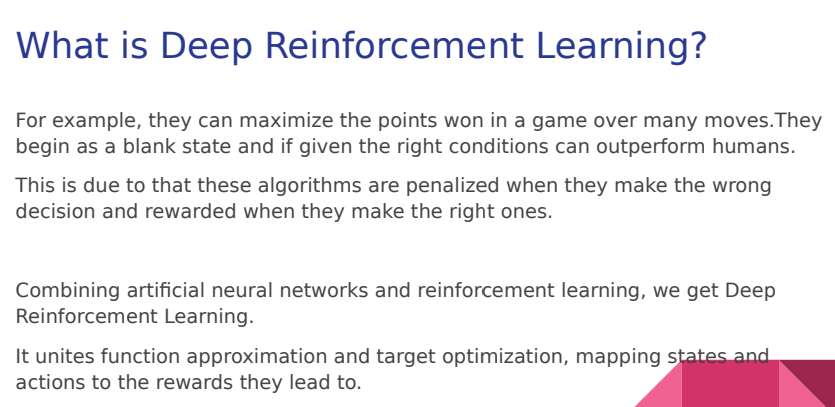
return min;

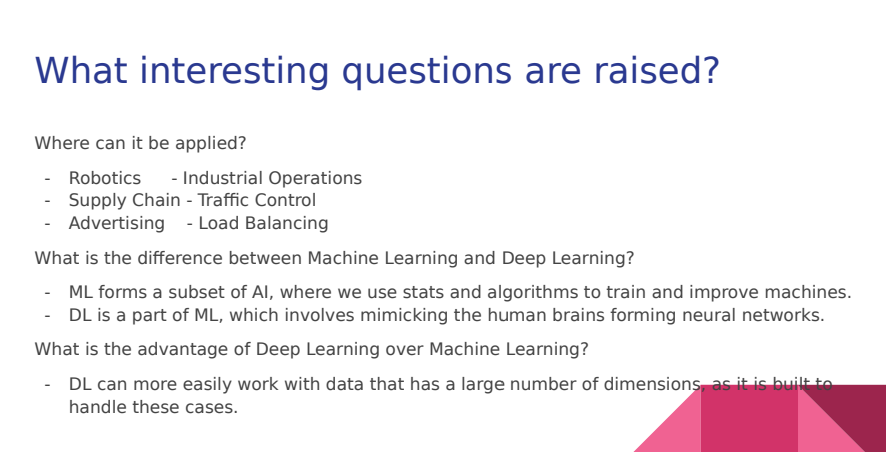
}

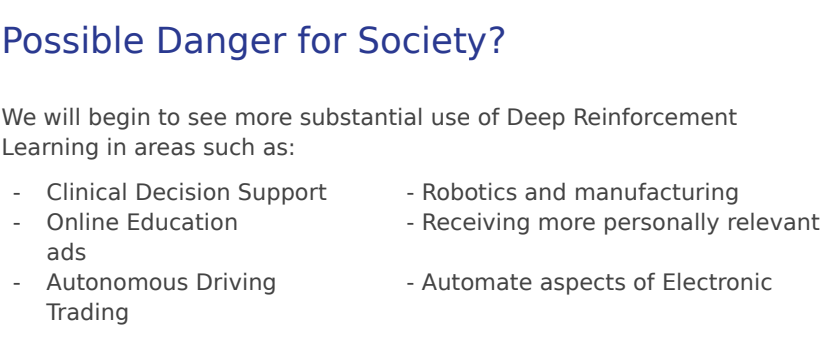
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AI Presentations and Challenges**

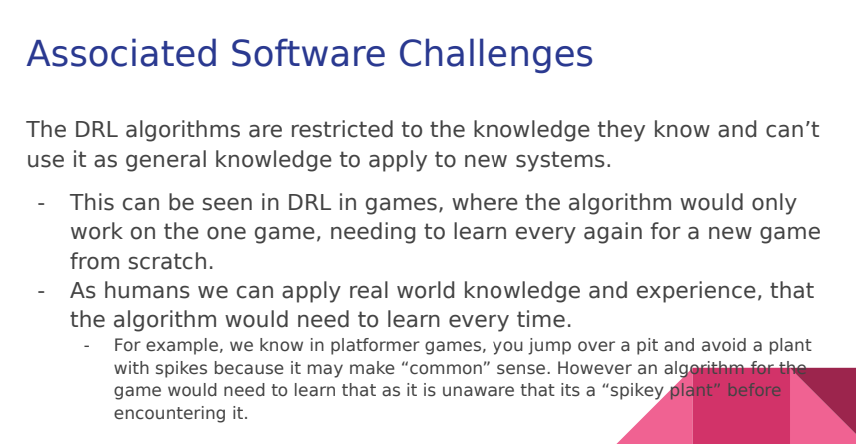












**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Binary Search Tree (Python then CPP)**

#include <iostream>

using namespace std;

struct node {

int key;

struct node \*left, \*right;

};

struct node\* newNode(int item) {

struct node\* temp

= (struct node\*)malloc(sizeof(struct node));

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

void inorder(struct node\* root) {

if (root != NULL) {

inorder(root->left);

cout << root->key;

inorder(root->right);

}

}

struct node\* insert(struct node\* node, int key) {

if (node == NULL)

return newNode(key);

if (key < node->key)

node->left = insert(node->left, key);

else

node->right = insert(node->right, key);

return node;

}

struct node\* minValueNode(struct node\* node) {

struct node\* current = node;

while (current && current->left != NULL)

current = current->left;

return current;

}

struct node\* deleteNode(struct node\* root, int key) {

// base case

if (root == NULL)

return root;

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else {

// node has no child

if (root->left==NULL and root->right==NULL)

return NULL;

// node with only one child or no child

else if (root->left == NULL) {

struct node\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL) {

struct node\* temp = root->left;

free(root);

return temp;

}

// node with two children: Get the inorder successor

// (smallest in the right subtree)

struct node\* temp = minValueNode(root->right);

// Copy the inorder successor's content to this node

root->key = temp->key;

// Delete the inorder successor

root->right = deleteNode(root->right, temp->key);

}

return root;

}

int main() {

// Let us create following BST

50

/ \

30 70

/ \ / \

20 40 60 80 \*/

struct node\* root = NULL;

root = insert(root, 50);

root = insert(root, 30);

root = insert(root, 20);

root = insert(root, 40);

root = insert(root, 70);

root = insert(root, 60);

root = insert(root, 80);

cout << "Inorder traversal of the given tree \n";

inorder(root);

cout << "\nDelete 20\n";

root = deleteNode(root, 20);

cout << "Inorder traversal of the modified tree \n";

inorder(root);

cout << "\nDelete 30\n";

root = deleteNode(root, 30);

cout << "Inorder traversal of the modified tree \n";

inorder(root);

cout << "\nDelete 50\n";

root = deleteNode(root, 50);

cout << "Inorder traversal of the modified tree \n";

inorder(root);

return 0;

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Lisp Functions**

(write(/ (\* 6 2) 2))

(defun cubeMylist(lst)

(mapcar #'(lambda(x) (\* x x x)) lst)

)

(write (cubeMylist '(2 3 4 5 6 7 8 9)))

(write ((lambda (a b c x)

(+ (\* a (\* x x)) (\* b x) c))

4 2 9 3)

)

(defun addMylist(lst)

(mapcar #'(lambda(x) (+ x x)) lst)

)

(write (addMylist '(2 3 4 5 6 7 8 9)))

(write(\* (/ (\* 2 3) 8) 7))