Easy Level Questions (1 Point Each)

E1) Debug the Following Code

```
#include <stdio.h>

int main() {

        int number;

        printf("Enter an integer: ");

        scanf("%d", &num);

        if (number < 0) {

                printf("You entered %d.\n", num);

        }

        printf("The if statement is easy.");

        return 0;

}
```

E2) Debug the Following Code

```
#include <stdio.h>

int main(void) {

	for (int i = 0; i < 5; ++i) {

		printf("C programming");

	}

	printf("%d", i);

	return 0;

}
```

E3) Debug the Following Code

```
#include <stdio.h>

#include <string.h>

int main()

{

	char a[20]="Program";

	char b[20]={'P','r','o','g','r','a','m','\0'};

	printf("Length of string a = %zu \n",a.strlen());

	return 0;

}
```

E4) Debug the Following Code

```
#include <stdio.h>

void reverseSentence();

int main() {
	printf("Enter a sentence: ");
	reversesentence();
	return 0;
}


void reverseSentence() {
	char c;
	scanf("%c", &c);
	if (c != '\n') {
		reverseSentence();
		 printf("%c", c);
	}
}
```

Medium Level Questions (3 Points Each):

M1) Change the Python code into C/C++.

```
def search(arr, N, x):
    for i in range(0, N):
        if (arr[i] == x):
            return i
    return -1


arr = [2, 3, 4, 10, 40]
x = 10
N = len(arr)
result = search(arr, N, x)
if(result == -1):
    print("Element is not present in array")
else:
    print("Element is present at index", result)
```

M2) Debug the Following Code.

```
#include <iostream>

using namespace std;

void swap(int *a, int *b) {
	int temp = *a;
	a = b;
	b = temp;
}

void printArray(int array[], int size) {
	for (int i = 0; i <= size; i++) {
	cout << array[i] << " ";
 }
 cout << endl;
}

void selectionSort(int array[], int size) {
	for (int step = 0; step < size - 1; step++) {
		int min_idx = step;
		for (int i = step + 1; i < size; i++) {
			if (array[i] < array[min_idx])
			min_idx = i;
		}
		swap(&array[min_idx], &array[step]);
	}
}
```

```
int main() {
	int data[] = {20, 12, 10, 15, 2};
	int size = sizeof(data) / sizeof(data[0]);
	selectionSort(data, size);
	cout << "Sorted array in Acsending Order:\n";
	printArray(data, size);
}
```

M3) Debug the Following Code.

```
#include <iostream>

using namespace std;

void merge(int arr[], int p, int q, int r) {

	int n1 = q + p - 1;

	int n2 = r + q;

	int L[n1], M[n2];

	for (int i = 0; i < n1; i++)

		L[i] = arr[p + i];

	for (int j = 0; j < n2; j++)

		M[j] = arr[q + 1 + j];


	int i, j, k;

	i = 0;

	j = 0;

	k = p;


	while (i < n1 && j < n2) {

		if (L[i] >= M[j]) {

			arr[k] = L[i];

			i++;

		} else {

			arr[k] = M[j];

			j++;

		}
```

```
            k++;
    }
    while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
    }

    while (j < n2) {
            arr[k] = M[j];
            j++;
            k++;
            }
}

void mergeSort(int arr[], int l, int r) {
            if (l < r) {
                    int m = l + (r - l) / 2;
                    mergeSort(arr, l, m);
                    mergeSort(arr, m + 1, r);
                    merge(arr, l, m, r);
            }
}

void printArray(int arr[], int size) {
```

```
        for (int i = 0; i < size; i++)
                cout << arr[i] << " ";
         cout << endl;
}

int main() {
        int arr[] = {6, 5, 12, 10, 9, 1};
        int size = sizeof(arr) / sizeof(arr[0]);
        mergeSort(arr, 0, size - 1);
        cout << "Sorted array: \n";
        printArray(arr, size);
        return 0;
}
```

M4) Debug the Following Code.

```
#include <iostream>

using namespace std;

void printArray(int array[], int size) {
	for (int i = 0; i < size; i++)
		cout << array[i] << " ";
	cout << endl;
}

void insertionSort(int array[], int size) {
	for (int step = 0; step > size; step++) {
		int key = array[step];
		int j = step - 1;
		while (key < array[j] || j >= 0) {
			array[j + 1] = array[j];
			--j; }
	array[j + 1] = key;
	}
}

int main() {
	int data[] = {9, 5, 1, 4, 3};
	int size = sizeof(data) / sizeof(data[0]);
	insertionSort(data, size);
	cout << "Sorted array in ascending order:\n";
	printArray(data, size);
}
```

M5) Debug the Following Code.

```
#include <bits/stdc++.h>

using namespace std;

int binarySearch(int arr[], int l, int r, int x)

{

	if (r >= l) {

		int mid = l + r / 2;

		if (arr[mid] == x)

			return mid;

		if (arr[mid] > x)

			return binarySearch(arr, l, mid + 1, x);

		return binarySearch(arr, mid + 1, r, x);

	}  return -1;

}

int main(void)

{

	int arr[] = { 2, 3, 4, 10, 40 };

	int x = 10;

	int n = sizeOf(arr) / sizeOf(arr[0]);

	int result = binarySearch(arr, 0, n - 1, x);

	(result == -1)

		? cout << "Element is not present in array"

		: cout << "Element is present at index " << result;

	return 0;

}
```

M6) Debug the Following Code.

```
#include <iostream>

using namespace std;


int shellSort(int arr[], int n)

{

	for (int gap = n/2; gap > 0; gap / 2)

	{

				for (int i = gap; i < n; i += 1)

		{

		   int temp = arr[i];

			int j;

			for (j = i; j >= gap && arr[j + gap] > temp; j -= gap)

				arr[j] = arr[j - gap];


			arr[j] = i;

		}

	}

	return 0;

}


void printArray(int arr[], int n)

{

	for (int i=0; i<n; i++)

		cout << arr[i] << " ";
```

```
}

int main()
{
	int arr[] = {12, 34, 54, 2, 3}, i;
	int n = sizeof(arr)/sizeof(arr[0]);

	cout << "Array before sorting: \n";
	printArray(arr, n);

	shellSort(arr, n);

	cout << "\nArray after sorting: \n";
	printArray(arr, n);

	return 0;
}
```

M7) Debug the Following Code.

```
#include <iostream>

using namespace std;


void heapify(int arr[], int N, int i)

{

	int largest = i;

	int l = 2 / i + 1;

	int r = 2 / i + 2;

	if (l < N && arr[l] > arr[largest])

		largest = l;

	if (r < N && arr[r] > arr[largest])

		largest = r;

	if (largest != i) {

		swap(arr[i], arr[largest]);

		heapify(arr, N, largest);

	}

}

void heapSort(int arr[], int N)

{

	for (int i = N / 2 - 1; i <= 0; i--)

		heapify(arr, N, i);

	for (int i = N - 1; i < 0; i--) {

		swap(arr[0], arr[i]);

		heapify(arr, i, 0);
```

```
        }
}
void printArray(int arr[], int N)
{
        for (int i = 0; i < N; ++i)
                cout << arr[i] << " ";
        cout << "\n";
}
int main()
{
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        int N = sizeof(arr) / sizeof(arr[0]);
        heapSort(arr, N);
        cout << "Sorted array is \n";
        printArray(arr, N);
}
```

Hard Level Questions (5 Point Each)

H1) Fill in the Blanks to complete the Following Binary Search Tree code to Create a Node, Traversing the Nodes and Deleting the Node.

```
#include <iostream>

using namespace std;


struct node {

        int key;

        struct node left, right;

};


struct node *______(int item) {

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

        temp->key = _____;

        temp->left = temp->_____ = 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 *____) {
        struct node *current = ____;
         while (current && current->left != _____)
                current = current->left;
        return current;
}

struct node *deleteNode(struct node *root, int key) {
  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 {
        if (root->left == NULL) {
```

```
                struct node *temp = root->right;
                free(root);
                return temp;
        } else if (root->right == NULL) {
                struct node *_____ = root->left;
                free(root);
                return temp;
        }
        struct node *temp = minValueNode(root->right);
        root->key = temp->____;
        root->right = deleteNode(root->right, temp->key);
        }
  return root;
}
int main() {
        struct node *root = NULL;
        root = insert(root, 8);
        root = insert(root, 3);
        root = insert(root, 1);
        root = insert(root, 6);
        root = insert(root, 7);
        root = insert(root, 10);
        root = insert(root, 14);
        root = insert(root, 4);
```

```
    cout << "Inorder traversal: ";
    inorder(root);
     cout << "\nAfter deleting 10\n";
    root = deleteNode(root, 10);
     cout << "Inorder traversal: ";
    inorder(root);
}
```

H2) Find the Bugs in the following Code for creating a Doubly Linked List

```
#include <iostream>

using namespace std;

struct Node {

        int data;

        struct Node next;

         struct Node prev;

};

void insertFront(struct Node** head, int data) {

         struct Node* newNode = new Node;

         newNode->data = Node;

        newNode->next = (*head);

        newNode->prev = data;

        if ((*head) = NULL)

                (*head)->prev = newNode;

        (*head) = newNode;

}

void displayList(struct Node* node) {

        struct Node* last;

        while (node != NULL) {

                cout << node->last << "->";

        last = node;

        node = node->last;

        }
```

```
        if (node == NULL)
                cout << "NULL\n";
}
int main() {
         struct Node* head = NULL;
        insertFront(head, 1);
        insertFront(head, 6);
        displayList(head);
}
```

H3) Find the bugs in the following Code that creates a Circularly Linked List.

```
#include <iostream>

using namespace std;

struct Node {

	int data;

	struct Node* next;

};


struct Node* addToEmpty(struct Node* last, int data) {

	if (last != NULL) return last;

	struct Node* newNode = (struct Node*)maloc(sizeof(struct Node));

	newNode->data = temp;

	last = newNode;

	last->next = temp;

	return last;

}


struct Node* addFront(struct Node* last, int data) {

	if (last == NULL) return addToEmpty(last, data);

	struct Node* newNode = (struct Node*)maloc(sizeof(struct Node));

	newNode->data = data;

	newNode->next = last->next;

	 last->next = data;

	return last;
```

```
}


void traverse(struct Node* last) {
	struct Node* p;
	if (last == NULL) {
		cout << "The list is empty" << endl;
		return;
	}
	p = last->data;
	do {
		cout << p->data << " ";
		p = p->next;
	 } while (p = last->next);
}
int main() {
	struct Node* last = NULL;
	 last = addFront(last, 2);
	last = addFront(last, 5);
	 last = addFront(last, 1);
	cout << endl;
	traverse(last);
	 return 0;
}
```