1. Library Borrowing Records Management in Python

```
class Library:
    def __init__(self):
        self.records = {}

    def borrow_book(self, name, book):
        self.records[name] = book
        print(f"{name} borrowed {book}")

    def return_book(self, name):
        if name in self.records:
            print(f"{name} returned {self.records[name]}")
            del self.records[name]
        else:
            print("No record found")

library = Library()
library.borrow_book("Alice", "Python Programming")
library.return_book("Alice")
```

2. Linear and Binary Search for Customer Account ID Lookup (Python)

```
def linear_search(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i
    return -1

def binary_search(arr, x):
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1

arr = [101, 102, 103, 104, 105]
x = 104
print("Linear Search Result:", linear_search(arr, x))
print("Binary Search Result:", binary_search(arr, x))
```

3. Sorting Employee Salaries using Selection Sort and Bubble Sort (Python)

```
def selection_sort(arr):
    for i in range(len(arr)):
        min_idx = i
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr

def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

salaries = [45000, 23000, 67000, 54000, 32000]
print("Selection Sort:", selection_sort(salaries.copy()))
print("Bubble Sort:", bubble_sort(salaries.copy()))
```

4. Real-Time Undo/Redo System Using Stack (C++)

```
#include <iostream>
#include <stack>
using namespace std;

int main() {
    stack<string> undo, redo;

    undo.push("Typed 'Hello'");
    undo.push("Deleted 'o'");

    redo.push(undo.top());
    undo.pop();

    cout << "Undo Stack: ";
    while(!undo.empty()) {
        cout << undo.top() << " ";
        undo.pop();
    }
    cout << "\nRedo Stack: ";
    while(!redo.empty()) {
        cout << redo.top() << " ";
        redo.pop();
    }
    return 0;
}
```

5. Simulation of Call Queue System Using Queue (C++)

```
#include <iostream>
#include <queue>
using namespace std;

int main() {
    queue<string> calls;
    calls.push("Customer 1");
    calls.push("Customer 2");

    cout << "Answering: " << calls.front() << endl;
    calls.pop();
    cout << "Answering: " << calls.front() << endl;
    calls.pop();
    return 0;
}
```

6. Student Record Management System Using Linked List (C++)

```
#include <iostream>
using namespace std;

struct Node {
    string name;
    int roll;
    Node* next;
};

class LinkedList {
public:
    Node* head = NULL;

    void insert(string name, int roll) {
        Node* newNode = new Node();
        newNode->name = name;
        newNode->roll = roll;
        newNode->next = head;
        head = newNode;
    }

    void display() {
        Node* temp = head;
```

```
        while (temp != NULL) {
            cout << "Name: " << temp->name << ", Roll: " << temp->roll << endl;
            temp = temp->next;
        }
    }
};

int main() {
    LinkedList list;
    list.insert("Alice", 1);
    list.insert("Bob", 2);
    list.display();
    return 0;
}
```

7. Implementation of Hash Table Using Division Method with Linear Probing (C++)

```
#include <iostream>
using namespace std;

class HashTable {
    int size;
    int* table;
public:
    HashTable(int s) {
        size = s;
        table = new int[size];
        for (int i = 0; i < size; i++) table[i] = -1;
    }

    int hashFunc(int key) { return key % size; }

    void insert(int key) {
        int index = hashFunc(key);
        while (table[index] != -1) {
            index = (index + 1) % size;
        }
        table[index] = key;
    }

    void display() {
        for (int i = 0; i < size; i++)
            cout << i << " --> " << table[i] << endl;
    }
};

int main() {
    HashTable ht(5);
    ht.insert(1);
    ht.insert(6);
    ht.display();
    return 0;
}
```

8. Optimal Pizza Delivery Routing Using Prim’s Algorithm (C++)

```
#include <iostream>
using namespace std;

#define INF 9999999
#define V 5

int main() {
    int G[V][V] = {
        {0, 9, 75, 0, 0},
        {9, 0, 95, 19, 42},
        {75, 95, 0, 51, 66},
        {0, 19, 51, 0, 31},
        {0, 42, 66, 31, 0}
    };
```

```
    int selected[V] = {0};
    selected[0] = 1;
    int no_edge = 0;

    cout << "Edge : Weight" << endl;

    while (no_edge < V - 1) {
        int min = INF, x = 0, y = 0;
        for (int i = 0; i < V; i++) {
            if (selected[i]) {
                for (int j = 0; j < V; j++) {
                    if (!selected[j] && G[i][j]) {
                        if (min > G[i][j]) {
                            min = G[i][j];
                            x = i;
                            y = j;
                        }
                    }
                }
            }
        }
        cout << x << " - " << y << " : " << G[x][y] << endl;
        selected[y] = 1;
        no_edge++;
    }
    return 0;
}
```

9. Implementation of Various Operations on a Binary Search Tree (BST) (C++)

```
#include <iostream>
using namespace std;

struct Node {
    int key;
    Node* left;
    Node* right;
};

Node* createNode(int key) {
    Node* newNode = new Node();
    newNode->key = key;
    newNode->left = newNode->right = NULL;
    return newNode;
}

Node* insert(Node* root, int key) {
    if (root == NULL) return createNode(key);
    if (key < root->key)
        root->left = insert(root->left, key);
    else
        root->right = insert(root->right, key);
    return root;
}

void inorder(Node* root) {
    if (root != NULL) {
        inorder(root->left);
        cout << root->key << " ";
        inorder(root->right);
    }
}

int main() {
    Node* root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 70);
    root = insert(root, 20);
    root = insert(root, 40);
    inorder(root);
    return 0;
}
```