DS October Assignments

Natya Vidhan Biswas

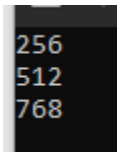
B.Sc. (H) C.S. 25771

1. Write a C++ program to add two long integers using stacks.

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

class Node {
public:
    int val;
    Node* next;

    Node(int v = 0) : val(v), next(0) {};
    friend class Stack;
};

class Stack {
    Node *head;
    int cap;
public:
    Stack() {
        head = 0;
        cap = 0;
    };
    void push(int val) {
        Node *p;
        p = new Node(val);
        if (!head) {
            head = p;
        } else {
            Node *q;
            q = head;
            head = p;
            head->next = q;
        };
        cap++;
        return;
    };

    void pop() {
        if (head) {
            Node *q;
            q = head;
            head = head->next;
            delete(q);
            cap--;
        }
        return;
    }

    int peek() {
        return head->val;
    }

    bool isEmpty() {
        return head == 0;
    }

    int size() {
        return cap;
```

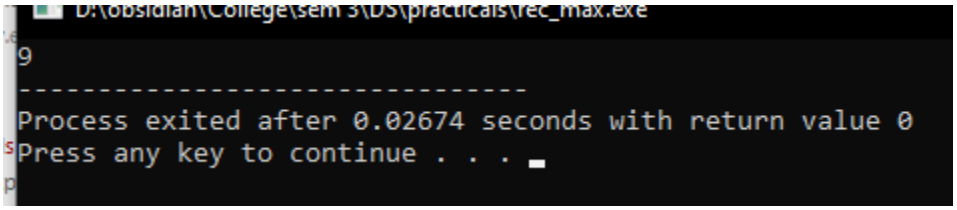
```
    }

    void clear() {
        // free all nodes and reset size
        while (head) {
            Node* t = head;
            head = head->next;
            delete t;
        }
        cap = 0;
        return;
    }

    int search(int v) {
        Node *q = head;
        int n = 0;
        while (q) {
            if (q->val == v) return n;
            q = q->next;
            n++;
        }
        return -1;
    }

    void print() {
        Node *q = head;
        while (q) {
            cout << q->val;
            q = q->next;
        }
        cout << endl;
        return;
    }
};

int main() {
    string a, b;
    cin >> a >> b;

    Stack *s1 = new Stack;
    Stack *s2 = new Stack;
    Stack *sr = new Stack;

    for (size_t i = 0; i < a.size(); ++i) {
        if (a[i] >= '0' && a[i] <= '9') s1->push(a[i] - '0');
    }
    for (size_t i = 0; i < b.size(); ++i) {
        if (b[i] >= '0' && b[i] <= '9') s2->push(b[i] - '0');
    }

    int carry = 0;
    while (!s1->isEmpty() || !s2->isEmpty() || carry) {
        int x = 0, y = 0;
        if (!s1->isEmpty()) { x = s1->peek(); s1->pop(); }
        if (!s2->isEmpty()) { y = s2->peek(); s2->pop(); }
        int sum = x + y + carry;
        sr->push(sum % 10);
        carry = sum / 10;
    }
```

```
    sr->print();
    return 0;
}
```



2. Describe a recursive algorithm for finding the maximum element in an array A of n elements. What is your running time and space usage?

```cpp
#include <iostream>

using namespace std;

int rec_max(const int arr[], int n) {
    if (n == 1) return arr[0];
    int next = rec_max(arr + 1, n - 1);
    return arr[0] > next ? arr[0] : next;
}

int main() {
    int nums[5] = {5, 6, 7, 8, 9};
    cout << rec_max(nums, 5);
}
```



Time Complexity: O(n)

- The function makes exactly n-1 recursive calls
- Each call performs a constant-time comparison operation
- Total time: O(n) where n is the array size

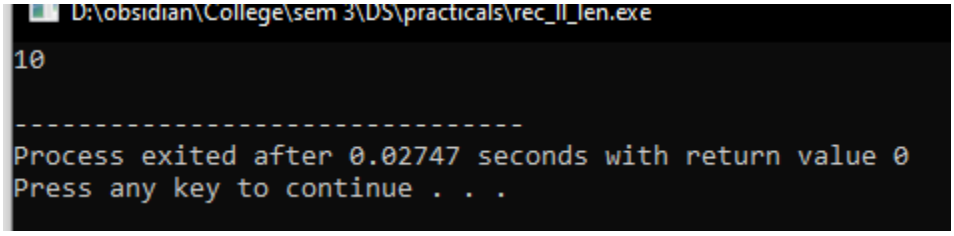
Space Complexity: O(n)

- Call Stack Space: O(n) due to recursive calls
- Maximum recursion depth is n
- Each call frame stores local variables (pointer, int)

3. Write a recursive C++ function to count the number of nodes in a singly linked list.

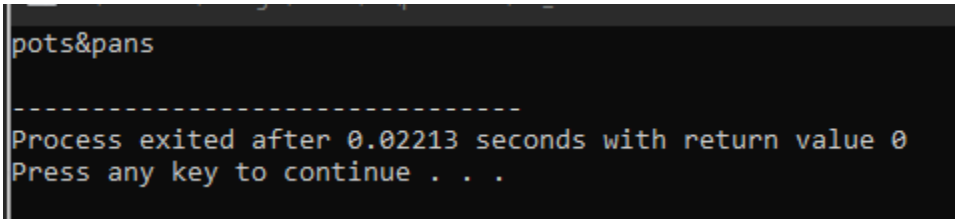
```
#include <iostream>

using namespace std;

class Node{
    int val;
    Node *next;
public:
    Node(int v) : val(v), next(0) {}
    friend class LinkedList;
};

class LinkedList {
    Node *head;
public:
    LinkedList() : head(0) {}
    void insert_end(int v) {
        Node *p, *q;
        p = new Node(v);
        if (!head) {
            head = p;
            return;
        }
        q = head;
        while (q->next)
        {
            q = q->next;
        }
        q->next = p;
        return;
    }

    void delete_end() {
        Node *p, *q;
        q = head;
        while (q->next->next)
        {
            q = q->next;
        }
        p = q->next;
        q->next = 0;
        delete(q);
        return;
    }


    int length(Node *n = 0) {
        if (!n) n = head;
        if (!n) return 0;
        if (!n->next) {
            return 1;
```

```
        }
        return 1 + length(n->next);
    }
};

int main() {
    LinkedList LL;
    int nums[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
    for (int i=0; i<10; i++) {
        LL.insert_end(nums[i]);
    }
    cout << LL.length() << endl;
    return 0;
}
```



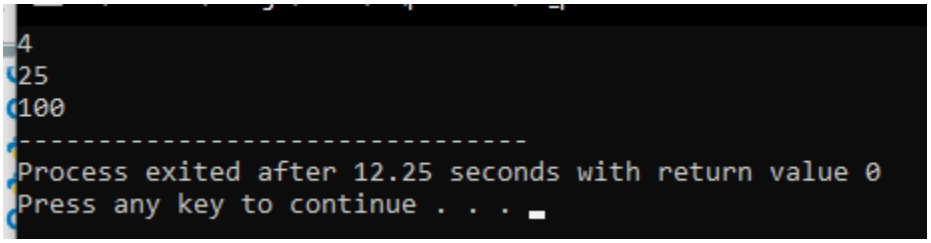
4. Write a short recursive C++ function that takes a character string s and outputs its reverse. For example, the reverse of "pots&pans" would be "snap&stop".

```cpp
#include <iostream>
#include <string>

using namespace std;

string reverse(string s) {
    if (s.length() == 0) {
        return "";
    }
    return reverse(s.substr(1)) + s[0];
}

int main() {
    string str = "snap&stop";
    string rev_str = reverse(str);
    cout << rev_str << endl;
    return 0;
}
```



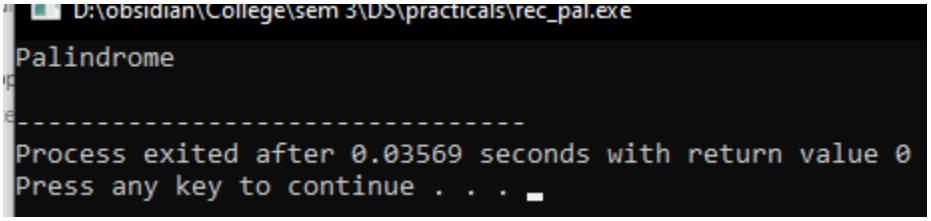
5. Give a recursive algorithm to compute the product of two positive integers, m and n, using only addition and subtraction.

```cpp
#include <iostream>

using namespace std;

int product(int m, int n) {
    if (n == 0) {
        return 0;
    }
    return m + product(m, n - 1);
}

int main() {
    int a, b;
    cin >> a;
    cin >> b;
    cout << product(a, b);
    return 0;
}
```



6. Write a short recursive C++ function that determines if a string s is a palindrome, that is, it is equal to its reverse. For example, "racecar" and "gohangasalamiimalasagnahog" are palindromes.

```cpp
#include <iostream>
#include <string>

using namespace std;

bool palindrome(string s) {
    int len = s.length();
    if (len <= 1) return true;
    if (s[0] != s[len-1]) return false;
    return palindrome(s.substr(1, len-2));
}

int main() {
    bool isit = palindrome("gohangasalamiimalasagnahog");
    if (isit) {
        cout << "Palindrome" << endl;
    } else {
        cout << "Not Palindrome" << endl;
    }
    return 0;
}
```



7. Write a program to implement Binary Search Tree as an ADT which supports the following operations:

- Insert an element x
- Delete an element x
- Search for an element x in the BST
- Display elements of the BST using breadth First Traversal
- Display the elements of the BST using preorder, inorder, and postorder traversal
- Find the height of the tree
- Count the number of nodes in the tree
- Count the number of internal and terminal nodes (leaves) of the tree

```
#include <iostream>

using namespace std;

template<typename T>
class QueueNode {
    T val;
    QueueNode *next;
public:
    QueueNode(T v): val(v), next(0) {}
    template <typename u>
    friend class Queue;
};

template<typename T>
class Queue {
    QueueNode<T> *head;
public:
    Queue(): head(0) {}
    bool empty() const {return head == 0;}
    void enqueue(T v) {
        QueueNode<T> *p;
        p = new QueueNode<T>(v);
        if (!head) {
            head = p;
            return;
        }
        QueueNode<T> *q = head;
        while (q->next) {
            q = q->next;
        }
        q->next = p;
        return;
    }
```

```
    T dequeue() {
        if (!head) return T();
        QueueNode<T> *p = head;
        head = head->next;
        T val = p->val;
        delete(p);
        return val;
    }
};

template <typename T>
class Node {
    T val;
    Node *left;
    Node *right;
public:
    Node(T v): val(v), left(0), right(0) {}

    template <typename U>
    friend class BST;
};

template <typename T>
class BST {
    Node<T> *head;
public:
    BST(): head(0) {};
    void insert(T v) {
        Node<T> *p;
        p = new Node<T>(v);
        if (!head) {
            head = p;
            return;
        }
        Node<T> *q = head;
        while (true) {
            if (v < q->val) {
                if (!q->left) {
                    q->left = p;
                    return;
                } else {
                    q = q->left;
                }
            } else {
                if (!q->right) {
                    q->right = p;
                    return;
                } else {
                    q = q->right;
                }
            }
        }
    }
```

```
void remove(T v) {
    if (!head) return;
    Node<T> *q = head;
    Node<T> *prev = 0;
    while (true) {
        if (q->val == v) {
            break;
        }
        prev = q;
        if (v < q->val && q->left) {
            q = q->left;
        } else if (v > q->val && q->right) {
            q = q->right;
        } else {
            cout << "Not Found" << endl;
            return;
        }
    }
    if (!q->left && !q->right) {
        if (!prev) head = 0;
        else if (prev->left == q) prev->left = 0;
        else prev->right = 0;
    } else if (!(q->right && q->left)) {
        if (q->right) {
            if (!prev) {
                head = q->right;
            } else {
                if (v < prev->val) {
                    prev->left = q->right;
                } else {
                    prev->right = q->right;
                }
            }
        } else if (q->left) {
            if (!prev) {
                head = q->left;
            } else {
                if (v < prev->val) {
                    prev->left = q->left;
                } else {
                    prev->right = q->left;
                }
            }
        }
    } else {
        Node<T> *right = q->right;
        Node<T> *leftmost = right;
        while (leftmost->left) {
            leftmost = leftmost->left;
        }
        leftmost->left = q->left;
        if (!prev) {
            head = right;
```

```
        } else {
            if (v < prev->val) {
                prev->left = right;
            } else {
                prev->right = right;
            }
        }
    }
    delete(q);
    return;
}

void search(T v) {
    search(v, head);
}

void search(T v, Node<T> *p) {
    if (p == 0) return;
    if (p->val == v) {
        cout << "Found" << endl;
        return;
    }
    if (v < p->val) {
        search(v, p->left);
    } else {
        search(v, p->right);
    }
    return;
}

void bft() {
    if (!head) return;
    Queue<Node<T>*> node_queue;
    node_queue.enqueue(head);
    while (!node_queue.empty()) {
        Node<T> *q = node_queue.dequeue();
        cout << q->val << " ";
        if (q->left) node_queue.enqueue(q->left);
        if (q->right) node_queue.enqueue(q->right);
    }
    cout << endl;
    return;
}

void preorder() {
    preorder(head);
    cout << endl;
}

void preorder(Node<T> *p) {
    if (p==0) return;
    cout << p->val << " ";
    preorder(p->left);
    preorder(p->right);
```

```
    return;
}

void inorder() {
    inorder(head);
    cout << endl;
}

void inorder(Node<T> *p) {
    if (p==0) return;
    inorder(p->left);
    cout << p->val << " ";
    inorder(p->right);
    return;
}

void postorder() {
    postorder(head);
    cout << endl;
}

void postorder(Node<T> *p) {
    if (p==0) return;
    postorder(p->left);
    postorder(p->right);
    cout << p->val << " ";
    return;
}

int height() {
    return height(head);
}

int height(Node<T> *p) {
    if (p==0) return 0;
    if (p->left == 0 && p->right == 0) return 1;
    return (1 + max(height(p->left), height(p->right)));
}

void count(int& i, int& t) {
    count(head, i, t);
}

void count(Node<T> *p, int& i, int& t) {
    if (p==0) return;
    if (!p->left && !p->right) {
        t = t + 1;
    } else {
        i = i + 1;
        count(p->left, i, t);
        count(p->right, i, t);
    }
    return;
}
```

```
};

int main() {
    BST<int> tree;
    int choice, value;

    int nums[13] = {50, 48, 70, 47, 49, 60, 90, 55, 65, 80, 100, 75, 85};
    for (int i = 0; i < 13; i++) {
        tree.insert(nums[i]);
    }

    while (true) {
        cout << "\n=== BST Menu ===" << endl;
        cout << "1. Insert" << endl;
        cout << "2. Remove" << endl;
        cout << "3. Search" << endl;
        cout << "4. Breadth First Traversal" << endl;
        cout << "5. Preorder Traversal" << endl;
        cout << "6. Inorder Traversal" << endl;
        cout << "7. Postorder Traversal" << endl;
        cout << "8. Height of Tree" << endl;
        cout << "9. Count Internal and Terminal Nodes" << endl;
        cout << "0. Exit" << endl;
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to insert: ";
                cin >> value;
                tree.insert(value);
                cout << "Inserted " << value << endl;
                break;

            case 2:
                cout << "Enter value to remove: ";
                cin >> value;
                tree.remove(value);
                break;

            case 3:
                cout << "Enter value to search: ";
                cin >> value;
                tree.search(value);
                break;

            case 4:
                cout << "BFT: ";
                tree.bft();
                break;

            case 5:
                cout << "Preorder: ";
                tree.preorder();
```

```
            break;

        case 6:
            cout << "Inorder: ";
            tree.inorder();
            break;

        case 7:
            cout << "Postorder: ";
            tree.postorder();
            break;

        case 8:
            cout << "Height: " << tree.height() << endl;
            break;

        case 9: {
            int internal = 0, terminal = 0;
            tree.count(internal, terminal);
            cout << "Internal nodes: " << internal << endl;
            cout << "Terminal nodes: " << terminal << endl;
            cout << "Total nodes: " << (internal + terminal) << endl;
            break;
        }

        case 0:
            cout << "Exiting..." << endl;
            return 0;

        default:
            cout << "Invalid choice!" << endl;
    }
  }

  return 0;
}
```

```
D:\obsidian\College\sem 3\DS\practicals\bst.exe

=== BST Menu ===
1. Insert
2. Remove
3. Search
4. Breadth First Traversal
5. Preorder Traversal
6. Inorder Traversal
7. Postorder Traversal
8. Height of Tree
9. Count Internal and Terminal Nodes
0. Exit
Enter choice: 1
Enter value to insert: 25
Inserted 25

=== BST Menu ===
1. Insert
2. Remove
3. Search
4. Breadth First Traversal
5. Preorder Traversal
6. Inorder Traversal
7. Postorder Traversal
8. Height of Tree
9. Count Internal and Terminal Nodes
0. Exit
Enter choice: 2
Enter value to remove: 25

=== BST Menu ===
1. Insert
2. Remove
3. Search
4. Breadth First Traversal
5. Preorder Traversal
6. Inorder Traversal
7. Postorder Traversal
8. Height of Tree
9. Count Internal and Terminal Nodes
0. Exit
Enter choice: 3
Enter value to search: 75
Found

=== BST Menu ===
1. Insert
2. Remove
3. Search
4. Breadth First Traversal
5. Preorder Traversal
6. Inorder Traversal
7. Postorder Traversal
8. Height of Tree
9. Count Internal and Terminal Nodes
0. Exit
Enter choice: 4
BFT: 50 48 70 47 49 60 90 55 65 80 100 75 85
```

```
D:\obsidian\College\sem 3\DS\practicals\bst.exe
=== BST Menu ===
1. Insert
2. Remove
3. Search
4. Breadth First Traversal
5. Preorder Traversal
6. Inorder Traversal
7. Postorder Traversal
8. Height of Tree
9. Count Internal and Terminal Nodes
0. Exit
Enter choice: 5
Preorder: 50 48 47 49 70 60 55 65 90 80 75 85 100

=== BST Menu ===
1. Insert
2. Remove
3. Search
4. Breadth First Traversal
5. Preorder Traversal
6. Inorder Traversal
7. Postorder Traversal
8. Height of Tree
9. Count Internal and Terminal Nodes
0. Exit
Enter choice: 6
Inorder: 47 48 49 50 55 60 65 70 75 80 85 90 100

=== BST Menu ===
1. Insert
2. Remove
3. Search
4. Breadth First Traversal
5. Preorder Traversal
6. Inorder Traversal
7. Postorder Traversal
8. Height of Tree
9. Count Internal and Terminal Nodes
0. Exit
Enter choice: 7
Postorder: 47 49 48 55 65 60 75 85 80 100 90 70 50

=== BST Menu ===
1. Insert
2. Remove
3. Search
4. Breadth First Traversal
5. Preorder Traversal
6. Inorder Traversal
7. Postorder Traversal
8. Height of Tree
9. Count Internal and Terminal Nodes
0. Exit
Enter choice: 8
Height: 5
```

```
=== BST Menu ===
1. Insert
2. Remove
3. Search
4. Breadth First Traversal
5. Preorder Traversal
6. Inorder Traversal
7. Postorder Traversal
8. Height of Tree
9. Count Internal and Terminal Nodes
0. Exit
Enter choice: 9
Internal nodes: 6
Terminal nodes: 7
Total nodes: 13

=== BST Menu ===
1. Insert
2. Remove
3. Search
4. Breadth First Traversal
5. Preorder Traversal
6. Inorder Traversal
7. Postorder Traversal
8. Height of Tree
9. Count Internal and Terminal Nodes
0. Exit
Enter choice: 0
Exiting...
```