Name : Arin Rai

UID : 22BCS11773

Section : KPIT-901/B

Winter Winning Camp Day-3 Questions and Solutions

Questions:

Q1: Write a recursive function to print the reverse of a given string.

Q2: Given the head of a singly linked list, return true if it is a palindrome or false otherwise.

Q3: You are given a positive integer p. Consider an array nums (1-indexed) that consists of the integers in the inclusive range

[1, 2p - 1] in their binary representations.

Q4: Given a string s representing a valid expression, implement a basic calculator to evaluate it, and return the result of the evaluation.

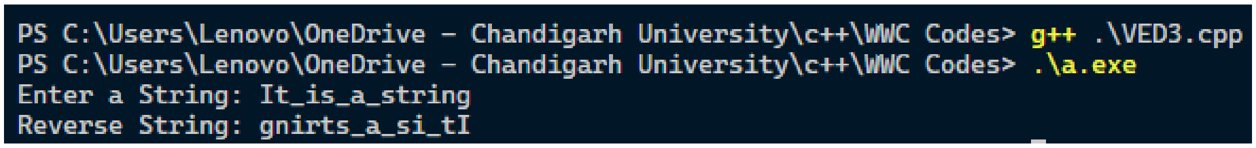
Q5: You are given a string expression representing a Lisp-like expression to return the integer value.

Solutions & Outputs

S1:

```
#include<iostream>

using namespace std;
```

```
void ReverseString(string &str, int start, int end){
    if (start >= end){
        return;
    }
    swap (str[start], str[end]);
    ReverseString(str, start+1, end-1);
}
int main () {
    string str;
    cout<<"Enter a String: ";
    cin>>str;

    ReverseString(str, 0, str.length()-1);
    cout<<"Reverse String: "<<str<<endl;

    return 0;
}
```



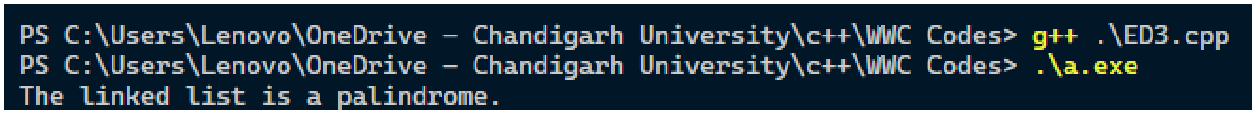
```
PS C:\Users\Lenovo\OneDrive - Chandigarh University\c++\WWC Codes> g++ .\VED3.cpp
PS C:\Users\Lenovo\OneDrive - Chandigarh University\c++\WWC Codes> .\a.exe
Enter a String: It_is_a_string
Reverse String: gnirts_a_si_tI
```

S2:

```
#include <iostream>
#include <vector> // Include vector header
using namespace std;

// Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode* next;
```

```cpp
    ListNode(int x) : val(x), next(nullptr) {}
};

// Function to reverse a linked list
ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* curr = head;
    while (curr != nullptr) {
        ListNode* nextTemp = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextTemp;
    }
    return prev;
}

// Function to check if the linked list is a palindrome
bool isPalindrome(ListNode* head) {
    if (head == nullptr || head->next == nullptr) return true;

    // Step 1: Find the middle of the linked list
    ListNode* slow = head;
    ListNode* fast = head;
    while (fast != nullptr && fast->next != nullptr) {
        slow = slow->next;
        fast = fast->next->next;
    }

    // Step 2: Reverse the second half of the list
    ListNode* secondHalf = reverseList(slow);
    // Step 3: Compare the two halves
    ListNode* firstHalf = head;
    ListNode* temp = secondHalf; // To restore the list later
    bool isPalin = true;

    while (temp != nullptr) {
        if (firstHalf->val != temp->val) {
            isPalin = false;
            break;
        }
        firstHalf = firstHalf->next;
        temp = temp->next;
    }

    // Step 4: Restore the list to its original state
    reverseList(secondHalf);

    return isPalin;
}

// Helper function to create a linked list from an array
ListNode* createLinkedList(const vector<int>& values) {
    if (values.empty()) return nullptr;
    ListNode* head = new ListNode(values[0]);
    ListNode* current = head;
    for (size_t i = 1; i < values.size(); i++) {
        current->next = new ListNode(values[i]);
        current = current->next;
    }
```

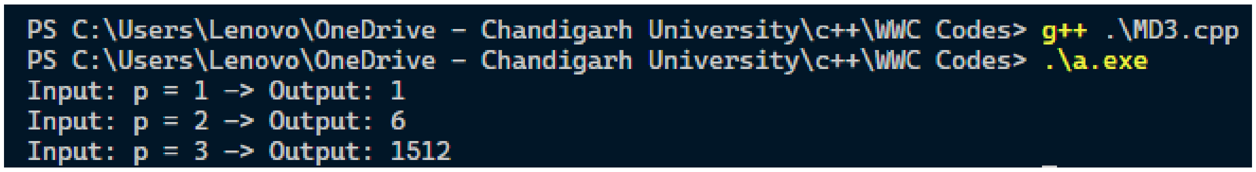
```cpp
    return head;
}

// Helper function to free the linked list
void freeLinkedList(ListNode* head) {
    while (head != nullptr) {
        ListNode* temp = head;
        head = head->next;
        delete temp;
    }
}

// Main function to test the implementation
int main() {
    vector<int> values = {1, 2, 2, 1}; // Example 1
    // vector<int> values = {1, 2}; // Example 2

    ListNode* head = createLinkedList(values);
    if (isPalindrome(head)) {
        cout << "The linked list is a palindrome." << endl;
    } else {
        cout << "The linked list is not a palindrome." << endl;
    }
    freeLinkedList(head);
    return 0;
}
```



```
PS C:\Users\Lenovo\OneDrive - Chandigarh University\c++\WWC Codes> g++ .\ED3.cpp
PS C:\Users\Lenovo\OneDrive - Chandigarh University\c++\WWC Codes> .\a.exe
The linked list is a palindrome.
```

S3:

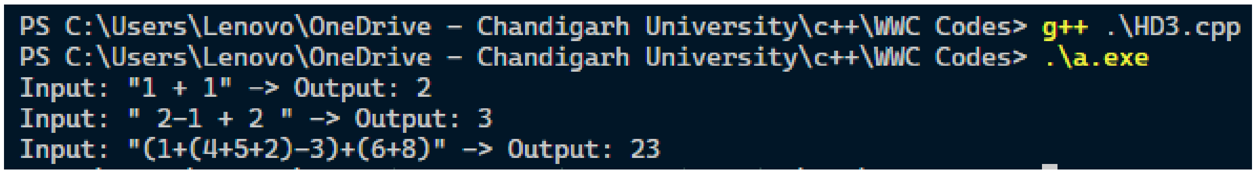
```
#include <iostream>

#include <cmath>

using namespace std;


const long long MOD = 1e9 + 7;


// Function for modular exponentiation

long long modExp(long long base, long long exp, long long mod) {

    long long result = 1;

    while (exp > 0) {

        if (exp % 2 == 1) { // If exp is odd

            result = (result * base) % mod;

        }

        base = (base * base) % mod; // Square the base

        exp /= 2;

    }

    return result;

}


// Function to compute the minimum non-zero product

long long minNonZeroProduct(int p) {

    if (p == 1) return 1;


    long long maxVal = (1LL << p) - 1;          // 2^p - 1

    long long secondMax = maxVal - 1;          // 2^p - 2

    long long numPairs = (1LL << (p - 1)) - 1; // 2^(p-1) - 1
    // Result = maxVal * (secondMax ^ numPairs) % MOD
    long long result = maxVal % MOD;
    result = (result * modExp(secondMax, numPairs, MOD)) % MOD;

    return result;
}

int main() {
    // Example Inputs
    int p1 = 1, p2 = 2, p3 = 3;

    cout << "Input: p = " << p1 << " -> Output: " <<
minNonZeroProduct(p1) << endl;
    cout << "Input: p = " << p2 << " -> Output: " <<
minNonZeroProduct(p2) << endl;
    cout << "Input: p = " << p3 << " -> Output: " <<
minNonZeroProduct(p3) << endl;

    return 0;
}
```



```
PS C:\Users\Lenovo\OneDrive - Chandigarh University\c++\WWC Codes> g++ .\MD3.cpp
PS C:\Users\Lenovo\OneDrive - Chandigarh University\c++\WWC Codes> .\a.exe
Input: p = 1 -> Output: 1
Input: p = 2 -> Output: 6
Input: p = 3 -> Output: 1512
```

S4:

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

class BasicCalculator {
public:
    int calculate(string s) {
        stack<int> values;     // Stack to hold values
        stack<int> operators;  // Stack to hold signs (1 for +, -1 for -
)
        int result = 0, num = 0, sign = 1;

        for (int i = 0; i < s.length(); i++) {
            char c = s[i];

            if (isdigit(c)) {
                // Form the number
                num = num * 10 + (c - '0');
            } else if (c == '+' || c == '-') {
                // Add the previous number to the result
                result += sign * num;
                num = 0;

                // Update the sign for the next number
                sign = (c == '+') ? 1 : -1;
            } else if (c == '(') {
                // Push the result and sign onto their stacks
                values.push(result);
                operators.push(sign);

                // Reset result and sign for the new sub-expression
                result = 0;
                sign = 1;
            } else if (c == ')') {
                // Add the current number to the result
                result += sign * num;
                num = 0;

                // Multiply by the sign from the stack and add to the
previous result
                result = values.top() + operators.top() * result;
                values.pop();
                operators.pop();
            }
        }

        // Add any remaining number to the result
        result += sign * num;
        return result;
    }
};

int main() {
    BasicCalculator calculator;

    // Test cases
    string s1 = "1 + 1";
    string s2 = " 2-1 + 2 ";
    string s3 = "(1+(4+5+2)-3)+(6+8)";


    cout << "Input: \"" << s1 << "\" -> Output: " <<
calculator.calculate(s1) << endl;

    cout << "Input: \"" << s2 << "\" -> Output: " <<
calculator.calculate(s2) << endl;

    cout << "Input: \"" << s3 << "\" -> Output: " <<
calculator.calculate(s3) << endl;


    return 0;

}
```

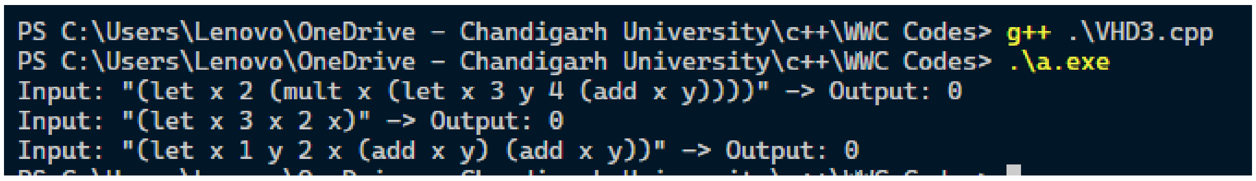


```
PS C:\Users\Lenovo\OneDrive - Chandigarh University\c++\WWC Codes> g++ .\HD3.cpp
PS C:\Users\Lenovo\OneDrive - Chandigarh University\c++\WWC Codes> .\a.exe
Input: "1 + 1" -> Output: 2
Input: " 2-1 + 2 " -> Output: 3
Input: "(1+(4+5+2)-3)+(6+8)" -> Output: 23
```

S5:

```
#include <iostream>

#include <string>

#include <unordered_map>

#include <sstream>

#include <vector>

using namespace std;


class LispParser {

public:

    int evaluate(string expression) {

        unordered_map<string, vector<int>> scope;

        return evaluate(expression, scope);

    }
```

```
private:
    int evaluate(const string& expr, unordered_map<string, vector<int>>&
scope) {
        if (isdigit(expr[0]) || expr[0] == '-') { // Direct integer
            return stoi(expr);
        }
        if (isalpha(expr[0]) && scope.count(expr)) { // Variable lookup
            return scope[expr].back();
        }

        stringstream ss(expr.substr(1, expr.size() - 2)); // Remove
outer parentheses
        string command;
        ss >> command;

        if (command == "let") {
            return evaluateLet(ss, scope);
        } else if (command == "add") {
            return evaluateAdd(ss, scope);
        } else if (command == "mult") {
            return evaluateMult(ss, scope);
        }
        return 0;
    }

    int evaluateLet(stringstream& ss, unordered_map<string,
vector<int>>& scope) {
        string token;
        vector<string> variables;
        while (ss >> token) {
            if (isExpression(token)) { // The final expression
                int result = evaluate(token, scope);
                for (const string& var : variables) {
                    scope[var].pop_back();
                    if (scope[var].empty()) {
                        scope.erase(var);
                    }
                }
                return result;
            }

            string valueExpr;
            ss >> valueExpr;
            int value = evaluate(valueExpr, scope);

            scope[token].push_back(value);
            variables.push_back(token);
        }
        return 0; // Should never reach here
    }

    int evaluateAdd(stringstream& ss, unordered_map<string,
vector<int>>& scope) {
        string expr1, expr2;
        ss >> expr1 >> expr2;
        return evaluate(expr1, scope) + evaluate(expr2, scope);
    }
    int evaluateMult(stringstream& ss, unordered_map<string,
vector<int>>& scope) {

        string expr1, expr2;

        ss >> expr1 >> expr2;

        return evaluate(expr1, scope) * evaluate(expr2, scope);

    }


    bool isExpression(const string& token) {

        return token[0] == '(' || isdigit(token[0]) || token[0] == '-'
|| isalpha(token[0]);

    }

};


int main() {

    LispParser parser;

    string expr1 = "(let x 2 (mult x (let x 3 y 4 (add x y))))";

    string expr2 = "(let x 3 x 2 x)";

    string expr3 = "(let x 1 y 2 x (add x y) (add x y))";

    cout << "Input: \"" << expr1 << "\" -> Output: " <<
parser.evaluate(expr1) << endl;

    cout << "Input: \"" << expr2 << "\" -> Output: " <<
parser.evaluate(expr2) << endl;

    cout << "Input: \"" << expr3 << "\" -> Output: " <<
parser.evaluate(expr3) << endl;

    return 0;

}
```



```
PS C:\Users\Lenovo\OneDrive - Chandigarh University\c++\WWC Codes> g++ .\VHD3.cpp
PS C:\Users\Lenovo\OneDrive - Chandigarh University\c++\WWC Codes> .\a.exe
Input: "(let x 2 (mult x (let x 3 y 4 (add x y))))" -> Output: 0
Input: "(let x 3 x 2 x)" -> Output: 0
Input: "(let x 1 y 2 x (add x y) (add x y))" -> Output: 0
```