

**15B17CI371 – Data Structures Lab**  
**ODD 2024**  
**Week 5-LAB B**

- 1. Write a program to find whether the number is Palindrome or not. A number is a Palindrome if it remains the same when its digits are reversed. Assumption: N is a positive integer.**

```
#include <iostream>

using namespace std;

int reversenum(int num, int temp) {

    if (num == 0)

        return temp;

    temp = (temp * 10) + (num % 10);

    return reversenum(num / 10, temp);

}

bool ispalindrome(int num) {

    int reversednum = reversenum(num, 0);

    return (num == reversednum);

}

int main() {

    int num;

    cout << "Enter a number: ";
```

```

cin >> num;

if (ispalindrome(num))

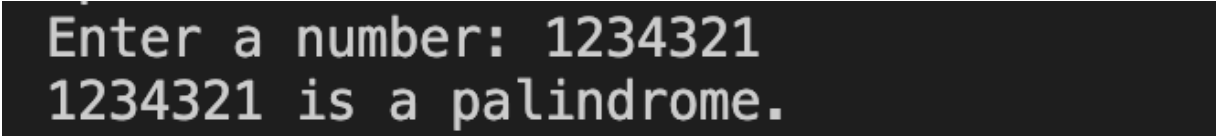
    cout << num << " is a palindrome." << endl;

else

    cout << num << " is not a palindrome." << endl;

return 0;
}

```



```

Enter a number: 1234321
1234321 is a palindrome.

```

**2. Write a program to implement a recursive function to calculate the sum of digits of a given number.**

```

#include <iostream>

using namespace std;

int sum(int num,int temp)

{

    if(num==0)

    {

        return temp;

    }

    else{

        temp=temp+(num%10);

        return sum(num/10,temp);

    }

}

```

```

    }

}

int main() {

    int num;

    cout << "Enter a number: ";

    cin >> num;

    cout<<"sum of the digits of "<<num<<" is "<<sum(num,0);

    return 0;

}

```

```

Enter a number: 1234
sum of the digits of 1234 is 10%

```

**3. Write a program to implement a recursive function to find the maximum and minimum elements in a given array.**

```

#include <iostream>

using namespace std;

int fmin(int arr[],int n)

{

    if (n==1)

        return arr[0];

        return min(arr[n-1],fmin(arr,n-1));

}

int main() {

    int num;

```

```
cout << "Enter size: ";

cin >> num;

int arr[num];

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

{

    cin>>arr[i];

}

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

{

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

}

cout<<"min is " <<fmin(arr,num);

return 0;

}
```

```
Enter size: 4
12
13
7
11
12 13 7 11 min is 7%
```

**4. Write a program to reverse a string using recursion.**

```
#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 = "Hello, World!";

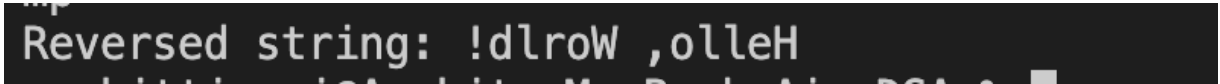
    reverseString(str, 0, str.length() - 1);

    cout << "Reversed string: " << str << endl;

    return 0;

}
```

```
}
```



```
Reversed string: !dlrow ,olleH
```

```
#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";
```

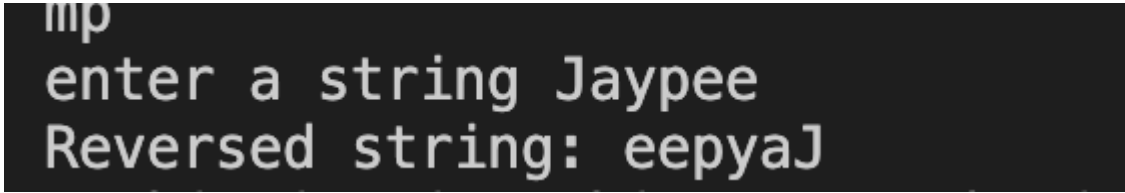
```
    cin>>str;
```

```
    reversestring(str, 0, str.length() - 1);
```

```
    cout << "Reversed string: " << str << endl;
```

```
    return 0;
```

```
}
```



```
enter a string Jaypee  
Reversed string: eepyaJ
```

**5. Write a program to implement a recursive function to reverse a linked list.\**

```
#include <iostream>

using namespace std;

struct Node {

    int data;

    Node* next;

};

Node* reverseList(Node* head) {

    if (head == nullptr || head->next == nullptr)

        return head;

    Node* revHead = reverseList(head->next);

    head->next->next = head;

    head->next = nullptr;

    return revHead;

}

void printList(Node* head) {

    Node* curr = head;

    while (curr != nullptr) {

        cout << curr->data << " ";

        curr = curr->next;

    }

    cout << endl;

}
```

```
Node* createNode(int data) {  
  
    Node* newNode = new Node;  
  
    newNode->data = data;  
  
    newNode->next = nullptr;  
  
    return newNode;  
  
}  
  
int main() {  
  
    Node* head = nullptr;  
  
    Node* tail = nullptr;  
  
    int n, data;  
  
    cout << "Enter the number of nodes: ";  
  
    cin >> n;  
  
  
    for (int i = 0; i < n; i++) {  
  
        cout << "Enter node " << i + 1 << " data: ";  
  
        cin >> data;  
  
        Node* newNode = createNode(data);  
  
  
        if (head == nullptr) {  
  
            head = newNode;  
  
            tail = head;  
  
        } else {  
  
            tail->next = newNode;  
  
            tail = newNode;  
  
        }  
    }  
}
```



```

    }
}

cout << "Original linked list: ";

printList(head);

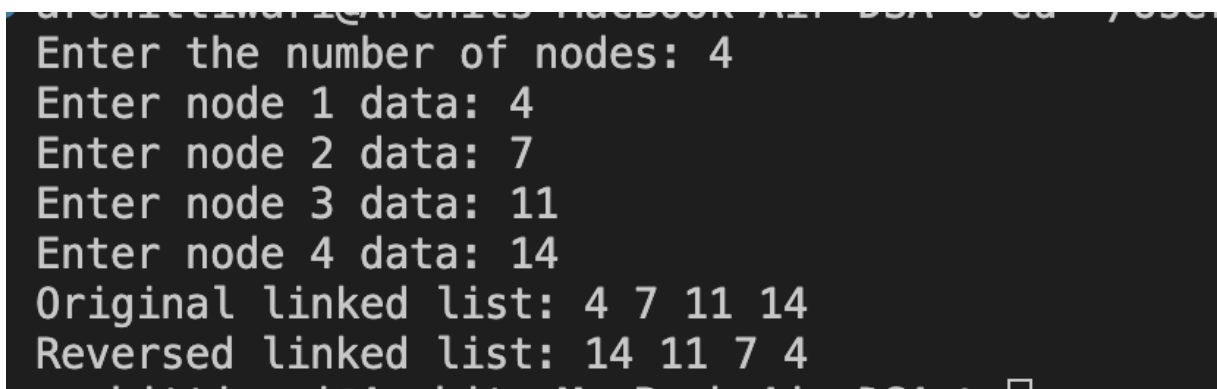
head = reverseList(head);

cout << "Reversed linked list: ";

printList(head);

return 0;
}

```



```

Enter the number of nodes: 4
Enter node 1 data: 4
Enter node 2 data: 7
Enter node 3 data: 11
Enter node 4 data: 14
Original linked list: 4 7 11 14
Reversed linked list: 14 11 7 4

```

**6. Write a program to implement a recursive function to find the greatest common divisor and Least Common Multiple.**

```

#include <iostream>
using namespace std;
int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}
int lcm(int a, int b) {
    return (a * b) / gcd(a, b);
}
int main() {

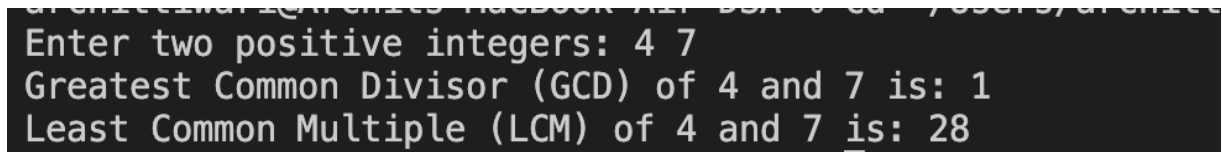
```

```

int num1, num2;

cout << "Enter two positive integers: ";
cin >> num1 >> num2;
int gcdValue = gcd(num1, num2);
int lcmValue = lcm(num1, num2);
cout << "Greatest Common Divisor (GCD) of " << num1 << " and " << num2 << " is: " <<
gcdValue << endl;
cout << "Least Common Multiple (LCM) of " << num1 << " and " << num2 << " is: " <<
lcmValue << endl;
return 0;
}

```



```

Enter two positive integers: 4 7
Greatest Common Divisor (GCD) of 4 and 7 is: 1
Least Common Multiple (LCM) of 4 and 7 is: 28

```

**7. Write a program to implement a recursive function to generate all permutations of a given set of numbers.**

```

#include <iostream>

using namespace std;

void printPermutation(int* nums, int n) {

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

        cout << nums[i] << " ";

    }

    cout << endl;
}

void generatePermutations(int* nums, int start, int n) {

    if (start >= n) {

        printPermutation(nums, n);
    }
}

```

```

        return;
    }

    for (int i = start; i < n; i++) {

        swap(nums[start], nums[i]);

        generatePermutations(nums, start + 1, n);

        swap(nums[start], nums[i]);
    }
}

int main() {

    int n;

    cout << "Enter the number of elements: ";

    cin >> n;

    int* nums = new int[n];

    cout << "Enter the elements: ";

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

        cin >> nums[i];
    }

    cout << "All permutations are:\n";

    generatePermutations(nums, 0, n);

    delete[] nums;

    return 0;
}

```

}

```
Enter the number of elements: 3 4 7 25
Enter the elements: All permutations are:
4 7 25
4 25 7
7 4 25
7 25 4
25 7 4
25 4 7
archittiwari@Archits-MacBook-Air DSA %
```

8. Assume that you are given a string. You can now form continuous substrings from the given string. Write a program to count the number of **UNIQUE** continuous sets of substrings that have the same starting and ending characters. (You can use a mix of recursive and non-recursive functions).

Eg:

Input string: "andisan"

Possible substrings:

"a", "an", "and", "andi", "andis", "andisa", "andisan",  
"n", "nd", "ndi", "ndis", "ndisa", "ndisan",  
"d", "di", "dis", "disa", "disan",  
"i", "is", "isa", "isan",  
"s", "sa", "san",  
~~"a", "an", "n",~~ (already considered earlier)

Output: 7 ("a", "andisa", "n", "ndisan", "d", "i", "s")

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

```
bool isUnique(string* uniqueSubstrings, int count, const string& substring) {
    for (int i = 0; i < count; i++) {
        if (uniqueSubstrings[i] == substring) {
            return false;
        }
    }
    return true;
}
```

```
int findUniqueSubstrings(const string& str, string* uniqueSubstrings) {
    int n = str.length();
    int count = 0;
```

```

    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            string substring = str.substr(i, j - i + 1);
            if (substring.front() == substring.back() &&
isUnique(uniqueSubstrings, count, substring)) {
                uniqueSubstrings[count] = substring;
                count++;
            }
        }
    }
    return count;
}

int main() {
    string input;
    cout << "Enter the input string: ";
    cin >> input;

    const int maxSubstrings = 100;
    string uniqueSubstrings[maxSubstrings];

    int uniqueCount = findUniqueSubstrings(input, uniqueSubstrings);

    cout << "Count of unique substrings with the same starting and ending
characters: "
        << uniqueCount << endl;

    cout << "Unique substrings are: " << endl;
    for (int i = 0; i < uniqueCount; i++) {
        cout << uniqueSubstrings[i] << endl;
    }

    return 0;
}

```

```

Enter the input string: School
Count of unique substrings with the same starting and ending characters: 6
Unique substrings are:
S
c
h
o
oo
l

```