C++ Notes

Bhanu Prasanna Koppolu

2023-10-25

Table of contents

Day - 1

Time Complexity

Important Points

- ***Definition*** - Time Complexity is the relationship about how the time will grow as the input grows.
- Time Complexity != Total Time Taken by a Machine to execute.
- There are mainly 4 points for Time Complexity:

    1. Always look for the **Worst Case Time Complexity**.
    2. Always look at the **Complexity for Large Data**.
    3. Always **Ignore Constants**.
    4. Always **Ignore Less Dominant Terms**. *From Point 2*

Akra-Bazzi Time Complexity Method

If the recurrence become more complex then finding the time complexity will become harder. So, Akra-Bazzi method is the easy and simple method to find the Time Complexity.

The method goes as follows:

Let p be the unique real number for which

$$\sum_{i=1}^{k} a_i b_i^p = 1$$

Then,

$$T(x) = \Theta\left(x^p + x^p \int_1^x \frac{g(u)}{u^{p+1}} du\right)$$

Asymptotic Notation

The usage of Asymptotic Notation is for specifying and for the identification of the upper and lower bounds.

$$\lim_{x \to \infty} \frac{f(n)}{g(n)}$$

Below are the Asymptotic Notations:

Big - O Notation ($\leq$ Upper Bound)

This is the Upper Bound and the complexity cannot go beyond this Upper Bound.

$$\lim_{x \to \infty} \frac{f(n)}{g(n)} < \infty$$

Big - Ω Notation ($\geq$ Lower Bound)

This is the opposite of Big - O Notation. This is the Lower Bound and the function complexity will atleast require Lower Bound complexity.

$$\lim_{x \to \infty} \frac{f(n)}{g(n)} > 0$$

Big - Θ Notation (Average)

The Big - Θ Notation was created when a function lies in both Upper Bound (Big - O) and Lower Bound (Big - Ω).

$$0 < \lim_{x \to \infty} \frac{f(n)}{g(n)} < \infty$$

Little - o Notation ($<$ Upper Bound)

The Little - o is same as Big - O but it is slightly loose and strictly lesser than.

$$\lim_{x \to \infty} \frac{f(n)}{g(n)} = 0$$

Little - ω Notation ($>$ Lower Bound)

The Little - ω is same as Big - Ω but it is slightly loose and strictly greater than.

$$\lim_{x \to \infty} \frac{f(n)}{g(n)} = \infty$$

Space Complexity

- Space Complexity is measured by combining the Input Space and the Auxilary Space. The same Asymptotic Notations discussed above are the ones used for Space Complexity.

Linear Recurrence (Homogenous and Non-Homogenous)

- Resources Used:
  - Click Here!

This is the end of ***Day - 1***.

Day - 2

Recursion

Important Points

- Recursion helps in solving bigger/complex problems into smaller problems which can be solved in a simple way.
- You can convert a Recursion solution into Iterative and Vice Versa.
- Space Complexity is not constant due to Recursive Calls.

Types of Recurrence Relations

1. Linear Recurrence Relation
2. Divide & Conquer Recurrence Relation

How to understand & approach a problem?

- Identify if you can break down problem into smaller problems.
- Write the Recurrence Relations if needed.
- Draw the Recursive Tree.
- About the Tree:
  - See the flow of functions, how they are getting into the stack.
  - Identify and Focus on Left Tree Calls and Right Tree Calls.
  - Draw the tree and pointers using Pen & Paper for better understanding. *
  - Use a debugger to see the flow.
- See how the values and what type of values are returned at each step. See where the function will come out. In the end you will come out of the main function.
- Three Variables to concentrate on:
  - Arguments
  - Return Type
  - Body of Function
- Resources used:

  - Click Here!

Math for DSA

- Resources used:
  - Math for DSA 1 - Click Here!
  - Math for DSA 2 - Click Here!

This is the end of ***Day - 2***.

Day - 3

C++ Start

- I have solved 18 Hackerrank Problems in the C++ section. - Click Here!.

Day - 4

- I have solved total of 28 Hackerrank Problems in the C++ section. ***I feel confident in C++.*** - Click Here!.

C++ from Book

- C++ developed by Bjarne Stroustroup at Bell Labs (1979).

Learning Language Tradition - Hello, World! Program

```cpp
// Preprocessor Directive
#include <iostream>

// Start of the Program
int main() {
  // Tell the compiler what namespace to search in
  using namespace std;

  // Write to the screen using std::cout
  cout << "Hello, World!" << endl;

  // Return a value to the OS
  return 0;
}
```

Day - 5

- I have solved 5 LeetCode Questions using C++ in the Arrays Section. - Click Here!.

LeetCode Questions & Answers

Two Sum

```cpp
//  Two_Sum.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna

class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int l = nums.size();

        for (int i = 0; i < l; i++) {
            for (int j = 0; j < l && i != j; j ++) {
                if (nums[i] + nums[j] == target) {
                    return {i, j};
                }
            }
        }

        return {};
    }
};
```

Remove Duplicates from Sorted Array

```cpp
//  Remove_Duplicates_from_Sorted_Array.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna

class Solution {
public:
    int removeDuplicates(vector<int>& v) {
        int j = 1;

        for (auto i = v.begin() + 1; i != v.end(); i++) {
            if (*i != *(i - 1)) {
                v[j] = *i;
                j++;
            }
        }

        return j;
    }
};
```

Remove Element

```cpp
//  Remove_Element.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna

class Solution {
public:
    int removeElement(vector<int>& v, int val) {
        int j = 0;

        for (auto i = v.begin(); i != v.end(); i++) {
            if (*i != val) {
                v[j] = *i;
                j++;
            }
        }
```

```
        return j;
    }
};
```

Search Insert Position

```
//  Search_Insert_Position.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna

class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        auto x = std::lower_bound(nums.begin(), nums.end(), target);

        cout << (x - nums.begin()) << endl;

        return x - nums.begin();
    }
};
```

Plus One

```
//  Plus_One.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna
//  Not the Best Solution - Must Learn How to Optimize a Solution.


class Solution {
public:
    vector<int> plusOne(vector<int>& v) {
        int c = 0, l = v.size();

        for (int i = l-1; i >= 0; i--) {
            if (i == l-1) {
                if (v[i] == 9) {
                    v[i] = 0;
                    c = 1;
```

```
                } else {
                    v[i] += 1;
                    return v;
                }
            } else {
                if (v[i] == 9) {
                    v[i] = 0;
                    c = 1;
                } else {
                    v[i] += c;
                    return v;
                }
            }
        }

        if (*v.begin() == 0) {
            v.insert(v.begin(), 1);
        }

        return v;
    }
};
```

Day - 6

- I have solved 5 LeetCode Questions using C++ in the Arrays Section. - Click Here!.

LeetCode Questions & Answers

Merge Sorted Array

```cpp
//  Merge_Sorted_Array.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna

class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int i = m - 1;
        int j = n - 1;
        int k = m + n - 1;

        while (j >= 0) {
            if (i >= 0 && nums1[i] > nums2[j]) {
                nums1[k] = nums1[i];
                i--;
            } else {
                nums1[k] = nums2[j];
                j--;
            }
            k--;
        }
    }
};
```

Pascal's Triangle

```cpp
//  Pascal's_Triangle.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna

class Solution {
public:
    vector<vector<int>> generate(int numRows) {
        vector<vector<int>> v;

        for (int i = 0; i < numRows; i++) {
            vector<int> x(i+1, 1);
            for (int j = 1; j < i; j++) {
                x[j] = v[i-1][j] + v[i-1][j-1];
            }
            v.push_back(x);
        }
        return v;
    }
};
```

Pascal's Triangle 2

```cpp
//  Pascal's_Triange_2.cpp
//  LeetCode - C++
//  Created by Bhanu Prasanna

class Solution {
public:
    vector<int> getRow(int rowIndex) {
        vector<vector<int>> v;
        vector<int> xv;

        for (int i = 0; i <= rowIndex; i++) {
            vector<int> x(i+1, 1);

            for (int j = 1; j < i; j++) {
                x[j] = v[i-1][j] + v[i-1][j-1];
            }
```

```
            if (i == rowIndex) {
                return x;
            }

            v.push_back(x);
        }

        return xv;
    }
};
```

Best Time to Buy and Sell Stock

```
// Best_Time_to_Buy_and_Sell_Stock.cpp
// LeetCode - C++
// Created by Bhanu Prasanna

class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int lsf = INT_MAX;
        int op = 0;
        int pist = 0;

        for (int i = 0; i < prices.size(); i++) {
            if (prices[i] < lsf) {
                lsf = prices[i];
            }
            pist = prices[i] - lsf;
            if (pist > op) {
                op = pist;
            }
        }
        return op;
    }
};
```

Single Number

```cpp
// Single_Number.cpp
// LeetCode - C++
// Created by Bhanu Prasanna

class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int ans = 0;

        for (auto x:nums) {
            ans ^= x;
        }

        return ans;
    }
};
```

Day - 7

- I am going to implement Data Structures in C++.

Single Linked List

```cpp
//  Single_Linked_List.cpp
//  DS - C++
//  Created by Bhanu Prasanna

#include <iostream>
using namespace std;

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

    Node () {
        data = 0;
        next = nullptr;
    }

    Node (int d) {
        data = d;
        next = nullptr;
    }
};

class SLL : public Node {
    Node* head;

public:
    SLL () {
```

```
        head = nullptr;
    }

    // Insert Data at the Front of SLL
    void insert_front(int);

    // Insert Data at the Back of SLL
    void insert_back(int);

    // Insert Data in the Middle of SLL
    void insert(int, int);

    // Delete Data from the Front of SLL
    void delete_front();

    // Delete Data from the Back of SLL
    void delete_back();

    // Delete Data from the Middle of SLL
    void del(int);

    // Length of the SLL
    int length();

    // Print SLL
    void printll();

};

/* Delete Functions for Single Linked List */

void SLL::delete_front () {
    Node *t = head;

    if (t == nullptr) {
        cout << "There are no element in the SLL to delete!" << endl;
        return;
    }

    head = head->next;
}
```

```
void SLL::delete_back () {
    Node *t = head;

    if (t == nullptr) {
        cout << "There are no element in the SLL to delete!" << endl;
        return;
    }

    while (t->next->next != nullptr) {
        t = t->next;
    }
    t->next = nullptr;
}

void SLL::del(int pos) {
    Node *x = head;
    Node *y = head->next;

    if (x == nullptr) {
        cout << "There are no element in the SLL to delete!" << endl;
        return;
    }

    for (int i = 0; i < pos - 2; i++) {
        x = x->next;
        y = y->next;
    }
    x->next = y->next;
}

/* Insertion Functions for Single Linked List */

void SLL::insert_front (int d) {
    Node *t = new Node(d);

    if (head == nullptr) {
        head = t;
        return;
    }

    t->next = head;
```

```
    head = t;

}

void SLL::insert_back (int d) {
    Node *t = new Node(d);

    if (head == nullptr) {
        head = t;
        return;
    }

    Node *x = head;

    while (x->next != nullptr) {
        x = x->next;
    }
    x->next = t;
}

void SLL::insert (int d, int pos) {

    if (pos > SLL::length()) {
        SLL::insert_back(d);
        return;
    }

    if (pos == 1) {
        SLL::insert_front(d);
        return;
    }

    Node *t = new Node(d);
    Node *x = head;
    Node *y = head->next;

    for (int i = 0; i < pos - 2; i++) {
        x = x->next;
        y = y->next;
    }
```

```
    t->next = y;
    x->next = t;
}

/* Length of Single Linked List */

int SLL::length () {
    Node *t = head;
    int c = 0;

    while (t != nullptr) {
        t = t->next;
        c++;
    }

    return c;
}

/* Print Single Linked List */

void SLL::printll () {
    Node *t = head;

    if (t == nullptr) {
        cout << "Linked List is Empty" << endl;
        return;
    }

    while (t != nullptr) {
        if (t->next == nullptr) {
            cout << t->data << endl;
        } else {
            cout << t->data << " -> ";
        }
        t = t->next;
    }
}

int main () {
    SLL l;
```

```
    l.insert_front(10);
    l.insert_back(20);
    l.insert_back(40);
    l.insert_back(50);
    l.insert_back(60);

    l.insert(30, 3);

    l.insert(30,3);

    l.del(3);

    l.delete_back();
    l.delete_front();

    cout << "Elements of Linked List are:" << endl;

    l.printll();

    cout << endl << "Count: " << l.length() << endl;

    return 0;
}
```

Double Linked List

```
//  Double_Linked_List.cpp
//  DS - C++
//  Created by Bhanu Prasanna

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node* prev;
```

```
    Node () {
        data = 0;
        next = nullptr;
        prev = nullptr;
    }

    Node (int d) {
        data = d;
        next = nullptr;
        prev = nullptr;
    }
};

class DLL : public Node {
    Node* head;

public:
    DLL () {
        head = nullptr;
    }

    // Insert Data at the Front of DLL
    void insert_front(int);

    // Insert Data at the Back of DLL
    void insert_back(int);

    // Insert Data in the Middle of DLL
    void insert(int, int);

    // Delete Data from the Front of DLL
    void delete_front();

    // Delete Data from the Back of DLL
    void delete_back();

    // Delete Data from the Middle of DLL
    void del(int);

    // Length of the DLL
    int length();
```

```
    // Print DLL
    void printll();

};

/* Delete Functions for Double Linked List */

void DLL::delete_front () {
    Node *t = head;

    if (t == nullptr) {
        cout << "There are no element in the DLL to delete!" << endl;
        return;
    }

    head = head->next;
    head->prev = nullptr;
}

void DLL::delete_back () {
    Node *t = head;

    if (t == nullptr) {
        cout << "There are no element in the DLL to delete!" << endl;
        return;
    }

    while (t->next->next != nullptr) {
        t = t->next;
    }
    t->next = nullptr;
}

void DLL::del(int pos) {
    Node *x = head;
    Node *y = head->next;

    if (x == nullptr) {
        cout << "There are no element in the DLL to delete!" << endl;
        return;
    }
```

```
    for (int i = 0; i < pos - 2; i++) {
        x = x->next;
        y = y->next;
    }
    y->next->prev = x;
    x->next = y->next;
}

/* Insertion Functions for Double Linked List */

void DLL::insert_front (int d) {
    Node *t = new Node(d);

    if (head == nullptr) {
        head = t;
        return;
    }

    t->next = head;
    head->prev = t;

    head = t;

}

void DLL::insert_back (int d) {
    Node *t = new Node(d);

    if (head == nullptr) {
        head = t;
        return;
    }

    Node *x = head;

    while (x->next != nullptr) {
        x = x->next;
    }
    x->next = t;
    t->prev = x;
}
```

```
void DLL::insert (int d, int pos) {

    if (pos > DLL::length()) {
        DLL::insert_back(d);
        return;
    }

    if (pos == 1) {
        DLL::insert_front(d);
        return;
    }

    Node *t = new Node(d);
    Node *x = head;
    Node *y = head->next;

    for (int i = 0; i < pos - 2; i++) {
        x = x->next;
        y = y->next;
    }

    t->next = y;
    y->prev = t;
    t->prev = x;
    x->next = t;
}

/* Length of Double Linked List */

int DLL::length () {
    Node *t = head;
    int c = 0;

    while (t != nullptr) {
        t = t->next;
        c++;
    }

    return c;
}
```

```
/* Print Double Linked List */

void DLL::printll () {
    Node *t = head;

    if (t == nullptr) {
        cout << "Linked List is Empty" << endl;
        return;
    }

    while (t != nullptr) {
        if (t->next == nullptr) {
            cout << t->data << endl;
        } else {
            cout << t->data << " -> ";
        }
        t = t->next;
    }
}

int main () {
    DLL l;

    l.insert_front(10);
    l.insert_back(20);
    l.insert_back(40);
    l.insert_back(50);
    l.insert_back(60);

    l.insert(30, 3);

    l.insert(30,3);

    l.del(3);

    l.delete_back();
    l.delete_front();

    cout << "Elements of Linked List are:" << endl;

    l.printll();
```

```
    cout << endl << "Count: " << l.length() << endl;

    return 0;
}
```

Circular Linked List

```
//  Circular_Linked_List.cpp
//  DS - C++
//  Created by Bhanu Prasanna

#include <iostream>
using namespace std;

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

    Node () {
        data = 0;
        next = nullptr;
    }

    Node (int d) {
        data = d;
        next = nullptr;
    }
};

class CLL : public Node {
    Node* head;
    Node* last;

public:
    CLL () {
        head = nullptr;
        last = nullptr;
    }
```

```
    // Insert Data at the Front of CLL
    void insert_front(int);

    // Insert Data at the Back of CLL
    void insert_back(int);

    // Insert Data in the Middle of CLL
    void insert(int, int);

    // Delete Data from the Front of CLL
    void delete_front();

    // Delete Data from the Back of CLL
    void delete_back();

    // Delete Data from the Middle of CLL
    void del(int);

    // Length of the CLL
    int length();

    // Print CLL
    void printll();
};


/* Delete Functions for Circular Linked List */

void CLL::delete_front() {
    Node *t = head;

    if (t == nullptr) {
        cout << "There are no element in the CLL to delete!" << endl;
    }

    head = head->next;
    last->next = head;
}

void CLL::delete_back() {
    Node *t = head;
```

```
    if (t == nullptr) {
        cout << "There are no element in the CLL to delete!" << endl;
    }

    if (t->next == head) {
        head = nullptr;
        last = nullptr;
    }

    while (t->next->next != head) {
        t = t->next;
    }

    last = t;

    last->next = head;
}

void CLL::del(int pos) {
    Node *x = head;
    Node *y = head->next;

    if (x == nullptr) {
        cout << "There are no element in the CLL to delete!" << endl;
    }

    if (pos == CLL::length()) {
        CLL::delete_back();
    }

    for (int i = 0; i < pos - 2; i++) {
        x = x->next;
        y = y->next;
    }

    x->next = y->next;
}

/* Insertion Functions for Circular Linked List */

void CLL::insert_front(int d) {
```

```
    Node *t = new Node(d);

    if (head == nullptr) {
        head = t;
        last = t;
        last->next = head;
        return;
    }

    t->next = head;
    head = t;
    last->next = head;
}

void CLL::insert_back (int d) {
    Node *t = new Node(d);

    if (head == nullptr) {
        insert_front(d);
        return;
    }

    t->next = head;
    last->next = t;
    last = t;
}

void CLL::insert (int d, int pos) {
    if (pos > CLL::length()) {
        CLL::insert_back(d);
        return;
    }

    if (pos == 1) {
        CLL::insert_front(d);
        return;
    }

    Node *t = new Node(d);
    Node *x = head;
    Node *y = head->next;
```

```
    for (int i = 0; i < pos - 2; i++) {
        x = x->next;
        y = y->next;
    }

    t->next = y;
    x->next = t;
}

/* Length of Double Linked List */

int CLL::length () {
    Node *t = head;
    int c = 0;

    if (t == nullptr) {
        return 0;
    }

    c++;
    t = t->next;

    while (t != head) {
        t = t->next;
        c++;
    }

    return c;
}

/* Print Double Linked List */

void CLL::printll () {
    Node *t = head;

    if (t == nullptr) {
        cout << "Linked List is Empty" << endl;
        return;
    }

    cout << t->data << "-> ";
```

```
    t = t->next;

    while (t != head) {
        if (t->next == head) {
            cout << t->data << endl;
        } else {
            cout << t->data << " -> ";
        }
        t = t->next;
    }
}


int main () {
    CLL l;

    l.insert_front(10);
    l.insert_back(20);
    l.insert_back(40);
    l.insert_back(50);
    l.insert_back(60);

    l.insert(30, 3);

    l.insert(30,3);

    l.del(3);

    l.delete_back();
    l.delete_front();

    cout << "Elements of Linked List are:" << endl;

    l.printll();

    cout << endl << "Count: " << l.length() << endl;

    return 0;
}
```

Stack

```cpp
//  Stack.cpp
//  DS - C++
//  Created by Bhanu Prasanna

#include <iostream>
using namespace std;

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

    Node () {
        data = 0;
        next = nullptr;
    }

    Node (int d) {
        data = d;
        next = nullptr;
    }
};

class Stack : public Node {
    Node* head;
    int MAX;

public:

    Stack (int m) {
        head = nullptr;
        MAX = m;
    }

    // Push an element to the Stack
    void push (int d);

    // Pop an element from the Stack
    int pop ();
```

```
    // Check if Stack is Empty
    bool isEmpty ();

    // Check if Stack is Full
    bool isFull ();

    // Get the Top element from the Stack
    int peek ();

    // Print the Stack
    void print ();
};

/* Push an element to the Stack */
void Stack::push (int d) {
    if (Stack::isFull()) {
        cout << "The Stack is Full. " << d << " element cannot be Inserted." << endl;
        return;
    }

    Node *t = new Node(d);

    if (head == nullptr) {
        t->next = nullptr;
        head = t;
        return;
    }

    t->next = head;
    head = t;
}

/* Pop an element from the Stack */
int Stack::pop () {
    if (Stack::isEmpty()) {
        cout << "The Stack is Empty. No Element is Present." << endl;
        return -1;
    }

    Node *t = head;
```

```
    head = head->next;

    return t->data;
}

/* Check if the Stack is Empty */
bool Stack::isEmpty () {
    Node *t = head;

    if (head == nullptr) {
        return true;
    }

    return false;
}

/* Check if the Stack is Full */
bool Stack::isFull () {
    Node *t = head;
    int c = 0;

    while (t != nullptr) {
        c++;
        t = t->next;
    }

    if (c == MAX) {
        return true;
    }

    return false;
}

/* Get the Top element from the Stack */
int Stack::peek () {
    Node *t = head;

    return t->data;
}

/* Print the Stack */
```

```
void Stack::print () {
    Node *t = head;

    while (t != nullptr) {
        if (t->next == nullptr) {
            cout << t->data << endl;
        }
        else {
            cout << t->data << " -> ";
        }
        t = t->next;
    }
}

int main () {
    Stack s(5);

    s.push(1);
    s.push(2);
    s.push(3);
    s.push(4);
    s.push(5);
    s.push(6);

    cout << s.isEmpty() << endl;
    cout << s.isFull() << endl;

    cout << "Element Pop is : " << s.pop() << endl;

    cout << s.peek() << endl;

    s.print();

    return 0;
}
```

Day - 8

- I am going to further implement Data Structures.

Simple Queue

```
//  Queue.cpp
//  DS - C++
//  Created by Bhanu Prasanna

#include <iostream>
using namespace std;

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

    Node () {
        data = 0;
        next = nullptr;
    }

    Node (int d) {
        data = d;
        next = nullptr;
    }
};

class Simple_Queue : public Node {
    Node* front;
    Node* rear;
    int MAX;

public:
```

```
    Simple_Queue (int m) {
        front = nullptr;
        rear = nullptr;
        MAX = m;
    }

    /* Enqueue an Element */
    void Enqueue (int);

    /* Dequeue an Element */
    int Dequeue ();

    /* Check if Queue is Empty */
    bool isEmpty ();

    /* Check if Queue is Full */
    bool isFull ();

    /* Peek Queue */
    int peek ();

    /* Print Queue */
    void print ();

    /* Number of Elements in Simple Queue */
    int size ();
};

/* Enqueue an Element */
void Simple_Queue::Enqueue (int d) {
    if (isFull()) {
        cout << "The Simple Queue is Full. " << d << " cannot be inserted." << endl;
        return;
    }

    Node* t = new Node(d);
    t->next = nullptr;

    if (front == nullptr && rear == nullptr) {
        front = t;
        rear = t;
```

```
        return;
    }

    if (front->next == nullptr && rear->next == nullptr) {
        front->next = t;
        rear = t;
        return;
    }

    rear->next = t;
    rear = rear->next;
}

/* Dequeue an Element */
int Simple_Queue::Dequeue () {
    if (isEmpty()) {
        cout << "There are no Elements in the Simple Queue." << endl;
        return -1;
    }

    Node *t = front;

    cout << t->data << " is Dequeued from the Simple Queue." << endl;

    if (front == rear) {
        front = nullptr;
        rear = nullptr;
    }
    else {
        front = front->next;
    }

    return t->data;
}

/* Check if Queue is Empty */
bool Simple_Queue::isEmpty () {
    if (front == nullptr && rear == nullptr) {
        return true;
    }
```

```
    return false;
}

/* Check if Queue is Full */
bool Simple_Queue::isFull () {
    Node *t = front;
    int c = 0;

    while (t != nullptr) {
        t = t->next;
        c++;
    }

    if (c == MAX) {
        return true;
    }
    return false;
}

/* Peek Queue */
int Simple_Queue::peek () {
    return front->data;
}

/* Print Queue */
void Simple_Queue::print () {
    Node *t = front;

    while (t != nullptr) {
        if (t->next == nullptr) {
            cout << t->data << endl;
        }
        else {
            cout << t->data << " -> ";
        }
        t = t->next;
    }
}

/* Number of Elements in Simple Queue */
int Simple_Queue::size () {
```

```
    Node *t = front;
    int c = 0;

    while (t != nullptr) {
        c++;
        t = t->next;
    }

    return c;
}

int main (void) {
    Simple_Queue sq(5);

    sq.Enqueue(1);
    sq.Enqueue(1);
    sq.Enqueue(1);
    sq.Enqueue(1);
    sq.Enqueue(1);
    sq.Enqueue(1);
    sq.Enqueue(1);

    sq.print();

    sq.Dequeue();
    sq.Dequeue();

    sq.print();

    cout << sq.size() << endl;

    return 0;
}
```

Reverse a Single Linked List using Iteration

```
/* Reverse a SLL using Iteration */
void SLL::rev_iter () {
    Node *cur = head, *prev = nullptr;

    while (cur != nullptr) {
        Node *t = cur->next;
        cur->next = prev;
        prev = cur;
        cur = t;
    }
    head = prev;
}
```