Inclass Lab-5

* Array implemented stack is faster than LinkedList implemented stack for push operations.

- In LinkedList implemented stack, a new node creation (memory allocation) is required for 'push' operation. But in array based stack it can easily access the index number and add the element. ∴ LinkedList based stack is slower than array based stack.

* Array based stack is faster than Linked List based stack for 'pop' operation.

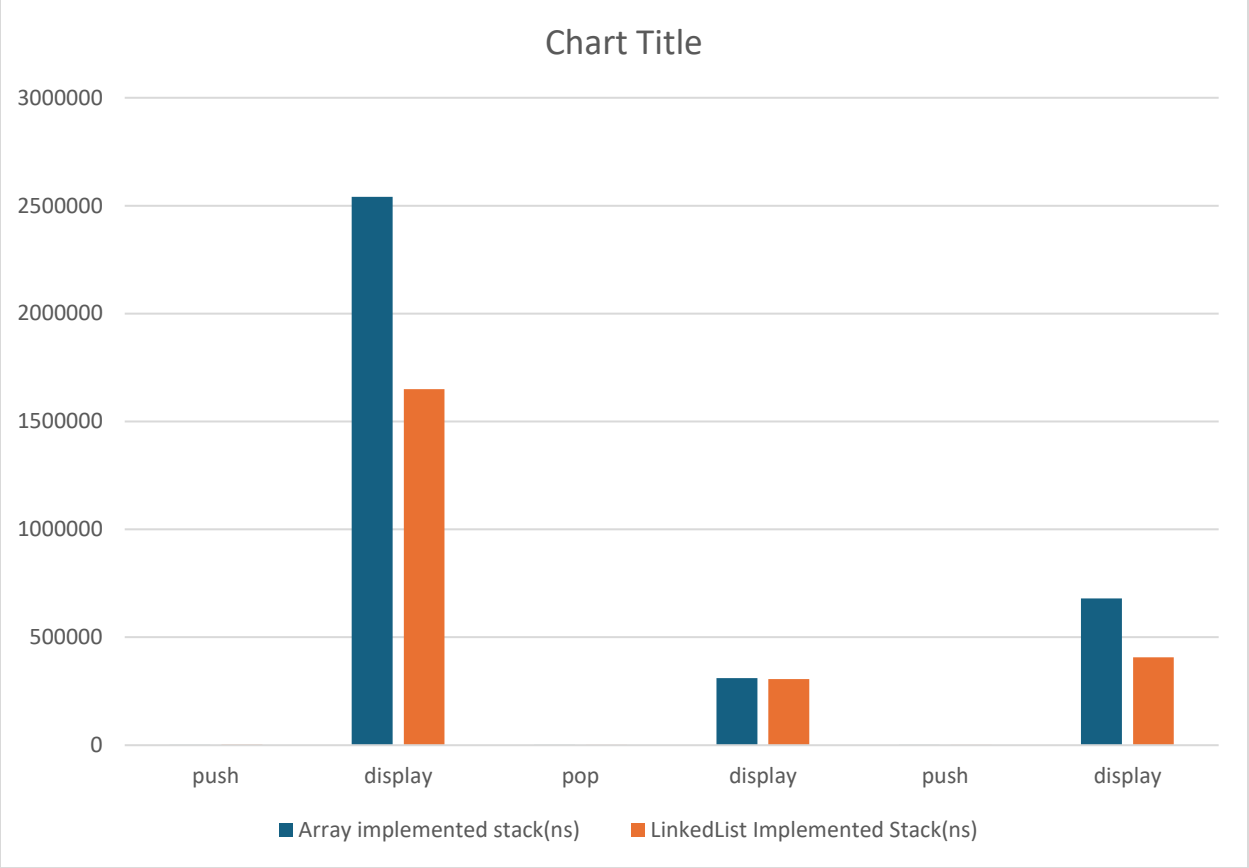
- Array based stack only needs to update the pointer to the topmost element when popping elements. But linklist based one has to create a temporary node and set the top pointer to the next node and delete temporary node. Hence it takes more time for removal.

* Linklist based stacks show a slightly faster performance for display operation. But unlike push, pop operations, the ratio between two runtimes is very low.

- Array has to iterate through the elements while linked list has to iterate through nodes. Both takes O(n) time complexity. LinkedList's node traversal can be cache-friendly compared to array's reverse order traversal. Because of this link list based stack shows a better performance.

Gunasekara S.L. 220191F

	Array implemented stack(ns)	LinkedList Implemented Stack(ns)
push	1000	2800
display	2541100	1649100
pop	100	1400
display	311200	306600
push	100	800
display	680100	407700
Total Time	3533600	2368400



Github link for time measuring codes

https://github.com/sahan974/Inclass-Lab5-Stacks.git

Gunasekara S.L. 220191F

Array based Stack

```cpp
#include <iostream>
#include <chrono>

class Stack {
private:
    int* array;
    int top;
    int size;

public:
    Stack(int maxSize) {…

    //To check whether the stack is empty or not
    bool isEmpty() {…

    //To check whether the stack is full or not
    bool isFull() {…

    // To insert data into the stack
    void push(int x) {…

    //To remove/delete data from the stack
    void pop() {…

    // To find what is at the top of the stack
    int stackTop() {…

    //Print the elements from the top of the stack to the bottom, separated by spaces
    void display() {…

    // Function to measure time and execute the given function
    template<typename Func>
    long long measureTime(Func func) {
        auto start = std::chrono::high_resolution_clock::now();
        func();
        auto end = std::chrono::high_resolution_clock::now();
        return std::chrono::duration_cast<std::chrono::nanoseconds>(end - start).count();
```

timeOne.cpp > main()

```cpp
class Stack {
    // Function to measure time and execute the given function
    template<typename Func>
    long long measureTime(Func func) {
        auto start = std::chrono::high_resolution_clock::now();
        func();
        auto end = std::chrono::high_resolution_clock::now();
        return std::chrono::duration_cast<std::chrono::nanoseconds>(end - start).count();
    }
};

int main() {
    Stack myStack(10);

    // Push operations
    long long pushTime = myStack.measureTime([&]() {
        myStack.push(8);
        myStack.push(10);
        myStack.push(5);
        myStack.push(11);
        myStack.push(15);
        myStack.push(23);
        myStack.push(6);
        myStack.push(18);
        myStack.push(20);
        myStack.push(17);
    });
    std::cout << "Time taken for push operations: " << pushTime << " nanoseconds\n";

    // Display operation
    long long displayTime = myStack.measureTime([&]() {
        myStack.display();
    });
    std::cout << "Time taken for display operation: " << displayTime << " nanoseconds\n";

    // Pop operations
    long long popTime = myStack.measureTime([&]() {
```

Gunasekara S.L. 220191F

```cpp
int main() {
    long long displayTime = myStack.measureTime([&]() {
        myStack.display();
    });
    std::cout << "Time taken for display operation: " << displayTime << " nanoseconds\n";

    // Pop operations
    long long popTime = myStack.measureTime([&]() {
        for (int i = 0; i < 5; ++i) {
            myStack.pop();
        }
    });
    std::cout << "Time taken for pop operations: " << popTime << " nanoseconds\n";

    // Display operation after pops
    long long displayAfterPopTime = myStack.measureTime([&]() {
        myStack.display();
    });
    std::cout << "Time taken for display operation after pops: " << displayAfterPopTime << " nanoseconds\n";

    // Push operations
    long long pushTime2 = myStack.measureTime([&]() {
        myStack.push(4);
        myStack.push(30);
        myStack.push(3);
        myStack.push(1);
    });
    std::cout << "Time taken for second set of push operations: " << pushTime2 << " nanoseconds\n";

    // Display operation
    long long displayTime2 = myStack.measureTime([&]() {
        myStack.display();
    });
    std::cout << "Time taken for display operation after second set of pushes: " << displayTime2 << " nanoseconds\n";

    return 0;
}
```

LinkedList based Stack

```cpp
#include <iostream>
#include <chrono>

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

    Node(int value) : data(value), next(nullptr) {}
};

class Stack {
private:
    Node* top;

public:
    Stack() : top(nullptr) {}

    // To insert data into the stack
    void push(int data) {
        Node* newNode = new Node(data);
        newNode->next = top;
        top = newNode;
    }

    //To remove/delete data from the stack
    void pop() {
        if (!isEmpty()) {
            Node* temp = top;
            top = top->next;
            delete temp;
        } else {
            std::cerr << "Stack underflow\n";
        }
    }

    //To check whethet the stack is empty or not
```

Gunasekara S.L. 220191F

```
12  class Stack {
27      void pop() {
35      }
36
37      //To check whethet the stack is empty or not
38      bool isEmpty() {
39          return top == nullptr;
40      }
41
42      // To find what is at the top of the stack
43      int stackTop() {
44          if (!isEmpty()) {
45              return top->data;
46          } else {
47              std::cerr << "Stack is empty\n";
48              return -1;
49          }
50      }
51      //Print the elements from the top of the stack to the bottom, separated by spaces
52      void display() {
53          Node* current = top;
54          while (current != nullptr) {
55              std::cout << current->data << " ";
56              current = current->next;
57          }
58          std::cout << "\n";
59      }
60
61      // Function to measure time and execute the given function
62      template<typename Func>
63      long long measureTime(Func func) {
64          auto start = std::chrono::high_resolution_clock::now();
65          func();
66          auto end = std::chrono::high_resolution_clock::now();
67          return std::chrono::duration_cast<std::chrono::nanoseconds>(end - start).count();
68      }
```

Ln 115, Col 8 Spaces: 4 UTF-8 CRLF {} C++ Go Live Win32 Prettier

timeTwo.cpp > main()

```
69  };
70
71  int main() {
72      Stack myStack;
73
74      // Push operations
75      long long pushTime = myStack.measureTime([&]() {
76          myStack.push(8);
77          myStack.push(10);
78          myStack.push(5);
79          myStack.push(11);
80          myStack.push(15);
81          myStack.push(23);
82          myStack.push(6);
83          myStack.push(18);
84          myStack.push(20);
85          myStack.push(17);
86      });
87      std::cout << "Time taken for push operations: " << pushTime << " nanoseconds\n";
88
89      // Display operation
90      long long displayTime = myStack.measureTime([&]() {
91          myStack.display();
92      });
93      std::cout << "Time taken for display operation: " << displayTime << " nanoseconds\n";
94
95      // Pop operations
96      long long popTime = myStack.measureTime([&]() {
97          for (int i = 0; i < 5; ++i) {
98              myStack.pop();
99          }
100     });
101     std::cout << "Time taken for pop operations: " << popTime << " nanoseconds\n";
102
103     // Display operation after pops
104     long long displayAfterPopTime = myStack.measureTime([&]() {
105         myStack.display();
```

Gunasekara S.L. 220191F

```
    });
    std::cout << "Time taken for display operation: " << displayTime << " nanoseconds\n";

    // Pop operations
    long long popTime = myStack.measureTime([&]() {
        for (int i = 0; i < 5; ++i) {
            myStack.pop();
        }
    });
    std::cout << "Time taken for pop operations: " << popTime << " nanoseconds\n";

    // Display operation after pops
    long long displayAfterPopTime = myStack.measureTime([&]() {
        myStack.display();
    });
    std::cout << "Time taken for display operation after pops: " << displayAfterPopTime << " nanoseconds\n";

    // Push operations
    long long pushTime2 = myStack.measureTime([&]() {
        myStack.push(4);
        myStack.push(30);
        myStack.push(3);
        myStack.push(1);
    });
    std::cout << "Time taken for second set of push operations: " << pushTime2 << " nanoseconds\n";

    // Display operation
    long long displayTime2 = myStack.measureTime([&]() {
        myStack.display();
    });
    std::cout << "Time taken for display operation after second set of pushes: " << displayTime2 << " nanoseconds\n";

    return 0;
}
```