Lab Manual 02

DATA STRUCTURE

Submitted by: **Tahira Inam**

Roll No: 105

Instructor: Azka Mir

3rd Semester (Blue)

Department of SoftwareEngineering

University of Sialkot

Table of Contents

Introduction

This lab manual covers essential operations on data structures, specifically focusing on doubly linked lists and stack. It demonstrates insertion, deletion, searching, updating, and splitting operations, reinforcing understanding of linked list data structures.

Objectives

- Implement basic operations on a doubly linked list and stack.
- Develop problem-solving skills by manipulating linked list structures.
- Understand how to dynamically manage memory using pointers.

Exercises

Exercise 1.1: Manage collection of Books using doubly Linked List Operations

Problem Statement:
Consider a scenario where a library wants to manage its collection of books. Each book has a unique ISBN, title, author, and publication year. The data is maintained in a doubly linked list. Create the following functions for the book collection:

1. Insert: Insert a new book record into the collection.

2. Search: Search for a book record using ISBN or title.

3. Modify: Update the details of an existing book record.

4. Display: Display all book records and the total number of books in the collection.

Exercise 1.2: Manage Playlist of Songs using Doubly Linked List

Problem Statement:
Consider a scenario where a music app wants to manage its playlist of songs. Each song has a unique ID, title, artist name, and duration. The data is maintained in a doubly linked list, allowing for efficient traversal in both forward and backward directions. This is particularly useful for features like navigating through songs in a playlist and enabling users to easily move to the previous or next track. Create the following functions for the music playlist:

1. Insert: Add a new song record to the playlist.

2. Search: Find a song record using the title or artist name.

3. Modify: Update the details of an existing song record.

4. Display: Display all songs in the playlist, the total number of songs and total duration.

5. Play Next: Move to the next song in the playlist.

6. Play Previous: Move to the previous song in the playlist

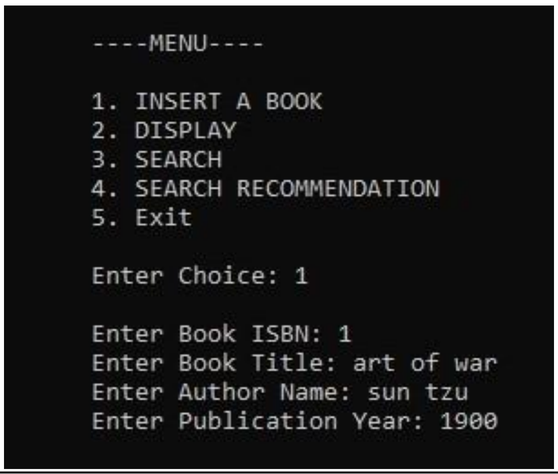
Source Code and Outputs

Exercise 1.1 Code

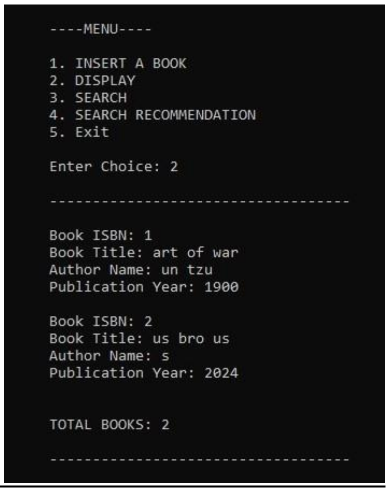
```
#include <iostream>

#include <string>

using namespace std;


struct Book {

	string author, title;

	int publicationYear, isbn;

	Book *next;

	Book *pre;

};


class Library {

	private:

		Book *head;

		Book *tail;

		int count;

	public:

		// CONSTRUCTOR

		Library() {

			head = NULL;
			tail = NULL;
			count = 0;
		}
		//INSERTION
		void insertBook() {
			Book *temp = new Book;

			cout<<"\n\tEnter Book ISBN: ";
			cin>>temp->isbn;
			cout<<"\tEnter Book Title: ";
			cin.ignore();
getline(cin, temp->title);
			cout<<"\tEnter Author Name: ";
			cin.ignore();
getline(cin, temp->author);
			cout<<"\tEnter Publication Year: ";
			cin>>temp->publicationYear;

			temp->next = NULL;
			temp->pre = NULL;

			if (head == NULL){
				head = temp;
				tail = temp;
			}
			else {
```

```
                        tail->next = temp;

                        temp->pre = tail;

                        tail = temp;

                }

                // TOTAL BOOKS/NODES

                count++;

        }

        //DISPLAYING

        void Display(){

                Book *temp = head;


                if (head == NULL) {

    cout << "\n\tNo books available.\n";

    return;

   }


                cout<<"\n\t-----------------------------------\n";

                while (temp != NULL){

                        cout<<"\n\tBook ISBN: "<<temp->isbn;

                        cout<<"\n\tBook Title: "<<temp->title;

                        cout<<"\n\tAuthor Name: "<<temp->author;

                        cout<<"\n\tPublication Year: "<<temp-
>publicationYear<<endl;;

                        temp = temp->next;

                }

                cout<<"\n\n\tTOTAL BOOKS: "<<count<<endl;
        cout<<"\n\t-----------------------------------\n";


    }
    //SEARCHING FOR RECOMMENDATION
    void searchRecommendation(){
            int num, f = 0;
            cout<<"\n\tEnter Book ISBN to search: ";
            cin>>num;
            Book *temp = head;
            while (temp != NULL){
                    if (temp->isbn == num ) {
                            f = 1;
                            temp = temp->next;
                            // To search the next book for recommendation
if (temp == NULL) {
  cout << "\n\tNo More BOOK Recommendations\n";
} else {
// New Book Recommendation
   cout << "\n\tNew Book Recommendation: \n";
   cout << "\n\tBook ISBN: " << temp->isbn;
   cout << "\n\tBook Title: " << temp->title;
   cout << "\n\tAuthor Name: " << temp->author;
   cout << "\n\tPublication Year: " << temp->publicationYear;
}
break;
                    }
                    temp = temp->next;
                }
                if (f == 0) {
  cout << "\n\tBook Not Found\n";
}
        }
        //SIMPLE SEARCH
        //SEARCHING
        void search(){
                int num, f = 0;
                cout<<"\n\tEnter Book ISBN to search: ";
                cin>>num;
                Book *temp = head;
                while (temp != NULL){
                        if (temp->isbn == num ) {
                                f = 1;

                                break;
                        }
                        temp = temp->next;
                }
                if (f == 1){
                        cout<<"\n\tBook Found\n";
                        cout<<"\n\n\tNew Book Recommendation: \n";
                        cout<<"\n\tBook ISBN: "<<temp->isbn;
                        cout<<"\n\tBook Title: "<<temp->title;
                cout<<"\n\tAuthor Name: "<<temp->author;
                cout<<"\n\tPublication Year: "<<temp->publicationYear;
            }
            else {
                cout<<"\n\tBook Not Found";
            }
        }
};
int main(){
    Library l;
    while (true) {
        cout<<"\n\n\t----MENU----\n";
        cout<<"\n\t1. INSERT A BOOK";
        cout<<"\n\t2. DISPLAY";
        cout<<"\n\t3. SEARCH";
        cout<<"\n\t4. SEARCH RECOMMENDATION";
        cout<<"\n\t5. Exit";
        int choice;
        cout<<"\n\n\tEnter Choice: ";
        cin>>choice;
        switch (choice)
        {
            case 1:
                l.insertBook();
                break;
            case 2:
                l.Display();
                break;
            case 3:
                l.search();
                break;
            case 4:
                l.searchRecommendation();
                break;
            case 5:
                return 0;
                break;
            default:
                cout<<"\n\tINVALID INPUT\n";
        }
    }
    return 0;
}
```

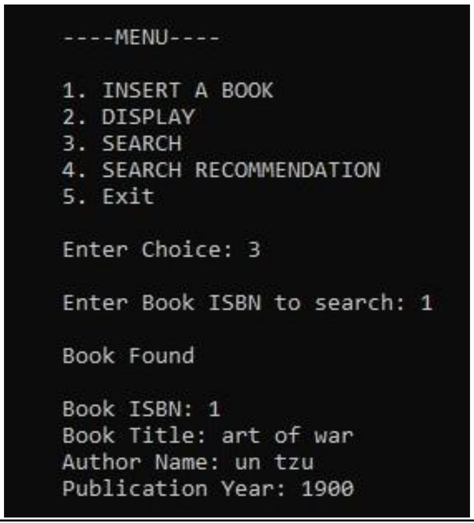
Output:

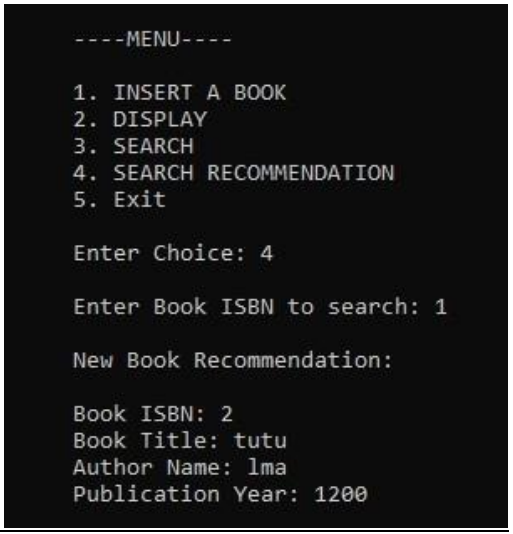


```
----MENU----

1. INSERT A BOOK
2. DISPLAY
3. SEARCH
4. SEARCH RECOMMENDATION
5. Exit

Enter Choice: 1

Enter Book ISBN: 1
Enter Book Title: art of war
Enter Author Name: sun tzu
Enter Publication Year: 1900
```



```
----MENU----

1. INSERT A BOOK
2. DISPLAY
3. SEARCH
4. SEARCH RECOMMENDATION
5. Exit

Enter Choice: 2

-----------------------------------

Book ISBN: 1
Book Title: art of war
Author Name: un tzu
Publication Year: 1900

Book ISBN: 2
Book Title: us bro us
Author Name: s
Publication Year: 2024


TOTAL BOOKS: 2

-----------------------------------
```


```
----MENU----

1. INSERT A BOOK
2. DISPLAY
3. SEARCH
4. SEARCH RECOMMENDATION
5. Exit

Enter Choice: 3

Enter Book ISBN to search: 1

Book Found

Book ISBN: 1
Book Title: art of war
Author Name: un tzu
Publication Year: 1900
```


```
----MENU----

1. INSERT A BOOK
2. DISPLAY
3. SEARCH
4. SEARCH RECOMMENDATION
5. Exit

Enter Choice: 4

Enter Book ISBN to search: 1

New Book Recommendation:

Book ISBN: 2
Book Title: tutu
Author Name: lma
Publication Year: 1200
```

Exercise 1.2 Code

```
#include <iostream>

#include <string>

using namespace std;


struct Song {

        string artist_name, title;

        int duration, id;

        Song *next;

        Song *pre;

};


class Playlist {

        private:

                Song *head;

                Song *tail;

                int count, Tduration;
```

```
public:

        // CONSTRUCTOR

        Playlist() {

                head = NULL;

                tail = NULL;

                count = 0;

                Tduration = 0;

        }

        //INSERTION

        void insertSong() {

                Song *temp = new Song;

                cout<<"\n\tEnter Artist Name: ";

                cin.ignore();

                getline(cin,temp->artist_name);

                cout<<"\n\tEnter Song Title: ";

                cin.ignore();

                getline(cin,temp->title);

                cout<<"\tEnter Duration: ";

                cin>>temp->duration;

                cout<<"\tEnter Song ID: ";

                cin>>temp->id;

                // TOTAL DURATION

                Tduration = Tduration + temp->duration;

                temp->next = NULL;

                temp->pre = NULL;

                if (head == NULL){
```

```
                head = temp;

                tail = temp;

        }

        else {

                tail->next = temp;

                temp->pre = tail;

                tail = temp;

        }

        // TOTAL SONGS/NODES

        count++;

}

//DISPLAYING

void Display(){

        Song *temp = head;

        cout<<"\n\t-----------------------------------\n";

        while (temp != NULL){

                cout<<"\n\tSong ID: "<<temp->id;

                cout<<"\n\tSong Title: "<<temp->title;

                cout<<"\n\tArtist Name: "<<temp->artist_name;

                cout<<"\n\tSong Duration: "<<temp->duration<<endl;;


                temp = temp->next;

        }

        cout<<"\n\n\tTOTAL SONGS: "<<count<<endl;

        cout<<"\n\tTOTAL DURATION: "<<Tduration<<endl;

  cout<<"\n\t----------------------------------\n";
```

```
}

//SEARCHING

void search(){

	int num, f = 0;

	cout<<"\n\tEnter Song ID to search: ";

	cin>>num;

	Song *temp = head;

	while (temp != NULL){

		if (temp->id == num ) {

			f = 1;

			break;

		}

		temp = temp->next;

	}

	if (f == 1){

		cout<<"\n\tSong Found\n";

		cout<<"\n\tSong ID: "<<temp->id;

		cout<<"\n\tSong Title: "<<temp->title;

		cout<<"\n\tArtist Name: "<<temp->artist_name;

		cout<<"\n\tSong Duration: "<<temp->duration;

	}

	else {

		cout<<"\n\tSong Not Found";

	}

}
```

```
//UPDATION
void update(){
    int num, f = 0;
    cout<<"\n\tEnter Song ID to search: ";
    cin>>num;
    Song *temp = head;
    Song *temp2 = tail;
    while (temp != NULL){
        if (temp->id == num || temp2->id == num) {
            f = 1;
            break;
        }
        temp = temp->next;
        temp2 = temp2->next;
    }
    if (f==1) {
    int choice;
    do {
        cout<<"\n\t----UPDATE----";
        cout<<"\n\t0. Exit";
        cout<<"\n\t1. Update Artist Name";
        cout<<"\n\t2. Update Song Title";
        cout<<"\n\t3. Update Song Duration";
        cout<<"\n\t-------------\n";
        cout<<"\n\tEnter Choice: ";
        cin>>choice;
```

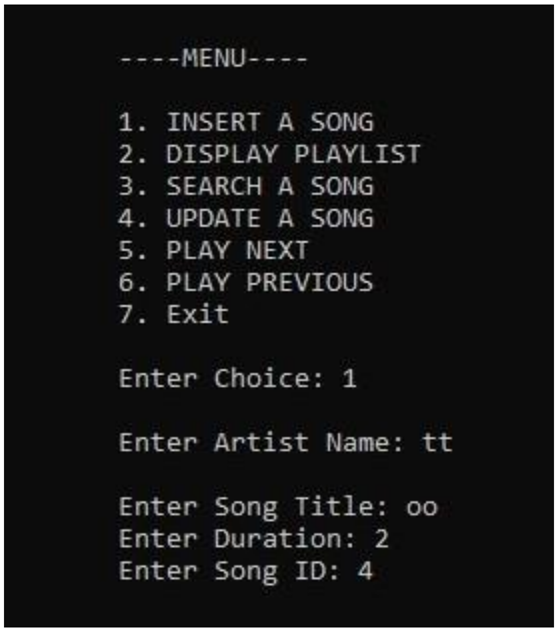
```
                switch (choice){
                        case 0:
                                cout<<"\n\tExiting from Updation...\n";
                                break;
                        case 1:
                                cout<<"\n\tUpdate Artist Name: ";
              cin>>temp->artist_name;
              cout<<"\n\tArtist Name Updated successfuly\n";
              break;
                        case 2:
                                cout<<"\n\tUpdate Song Title: ";
              cin.ignore();
        getline(cin, temp->title);
              cout<<"\n\tSong Title Updated successfuly\n";
              break;
                        case 3:
                                cout<<"\n\tUpdate Duration: ";
              cin>>temp->duration;
              cout<<"\n\tSong Duration Updated successfuly\n";
              break;
            default:
                cout<<"\n\tInvalid Choice. Try Again";
                }
        } while (choice != 0);
}
else {
```

```
                cout<<"\n\tRECORD NOT FOUND\n";
        }
        }
        //PLAY NEXT
        void next() {
                int num, f = 0;
                cout<<"\n\tEnter Song ID : ";
                cin>>num;
                Song *temp = head;
                while (temp != NULL){
                        if (temp->id == num ) {
                                f = 1;
                                temp = temp->next;
                                // To play next song
    if (temp == NULL) {
      cout << "\n\tNo More Songs\n";
    } else {
cout<<"\n\tSONG THAT WILL PLAY NEXT\n";
                            cout<<"\n\tSong ID: "<<temp->id;
                            cout<<"\n\tSong Title: "<<temp->title;
                            cout<<"\n\tArtist Name: "<<temp->artist_name;
                            cout<<"\n\tSong Duration: "<<temp->duration;
                                }
                                break;
                        }
                        temp = temp->next;
```

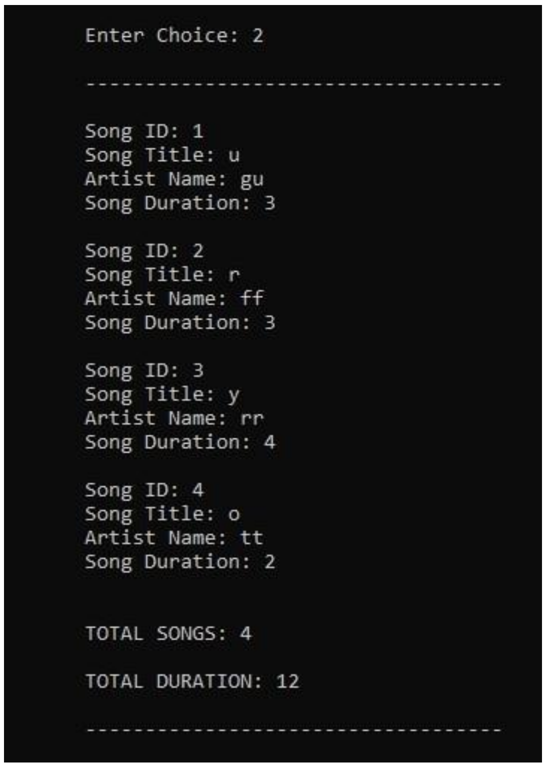
```
                    }
                    if (f == 0) {
  cout << "\n\tBook Not Found\n";
}
               }
               //PLAY PREVIOUS
               void previous() {
                    int num, f = 0;
                    cout<<"\n\tEnter Song ID : ";
                    cin>>num;
                    Song *temp = head;
                    while (temp != NULL){
                         if (temp->id == num ) {
                              f = 1;
                              temp = temp->pre;
                              // To play next song
    if (temp == NULL) {
      cout << "\n\tNo More Songs\n";
    } else {
  cout<<"\n\tPREVIOUS SONG\n";
                            cout<<"\n\tSong ID: "<<temp->id;
                            cout<<"\n\tSong Title: "<<temp->title;
                            cout<<"\n\tArtist Name: "<<temp->artist_name;
                            cout<<"\n\tSong Duration: "<<temp->duration;
                              }
                              break;
```

```
                                        }
                                        temp = temp->next;
                              }
                              if (f == 0) {
          cout << "\n\tBook Not Found\n";
        }
                    }
};

int main(){
          Playlist p;
          while (true) {
                    cout<<"\n\n\t----MENU----\n";
                    cout<<"\n\t1. INSERT A SONG";
                    cout<<"\n\t2. DISPLAY PLAYLIST";
                    cout<<"\n\t3. SEARCH A SONG";
                    cout<<"\n\t4. UPDATE A SONG";
                    cout<<"\n\t5. PLAY NEXT";
                    cout<<"\n\t6. PLAY PREVIOUS";
                    cout<<"\n\t7. Exit";
                    int choice;
                    cout<<"\n\n\tEnter Choice: ";
                    cin>>choice;
                    switch (choice)
                    {
                              case 1:
}

                p.insertSong();
                break;
            case 2:
                p.Display();
                break;
            case 3:
                p.search();
                break;
            case 4:
                p.update();
                break;
            case 5:
                p.next();
                break;
            case 6:
                p.previous();
                break;
            case 7:
                return 0;
                break;
            default:
                cout<<"\n\tINVALID INPUT\n";
        }
    }
    return 0;
}
```

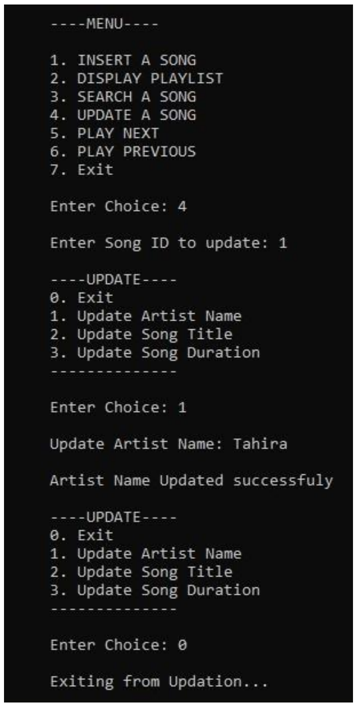
Output:



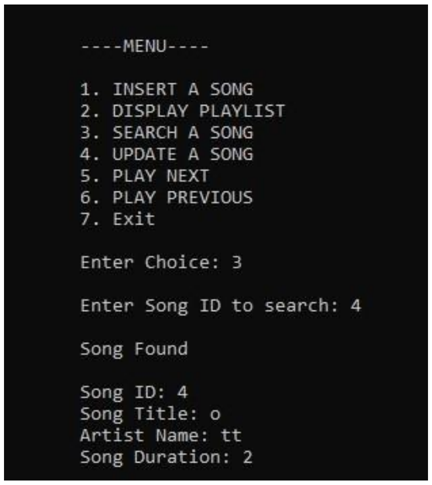
```
----MENU----

1. INSERT A SONG
2. DISPLAY PLAYLIST
3. SEARCH A SONG
4. UPDATE A SONG
5. PLAY NEXT
6. PLAY PREVIOUS
7. Exit

Enter Choice: 1

Enter Artist Name: tt

Enter Song Title: oo
Enter Duration: 2
Enter Song ID: 4
```



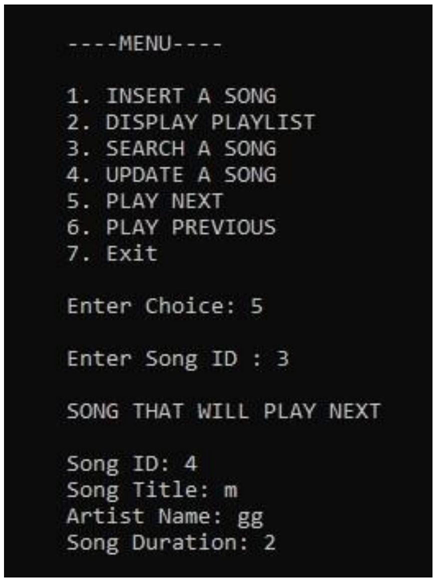
```
Enter Choice: 2

-----------------------------------

Song ID: 1
Song Title: u
Artist Name: gu
Song Duration: 3

Song ID: 2
Song Title: r
Artist Name: ff
Song Duration: 3

Song ID: 3
Song Title: y
Artist Name: rr
Song Duration: 4

Song ID: 4
Song Title: o
Artist Name: tt
Song Duration: 2


TOTAL SONGS: 4

TOTAL DURATION: 12

-----------------------------------
```



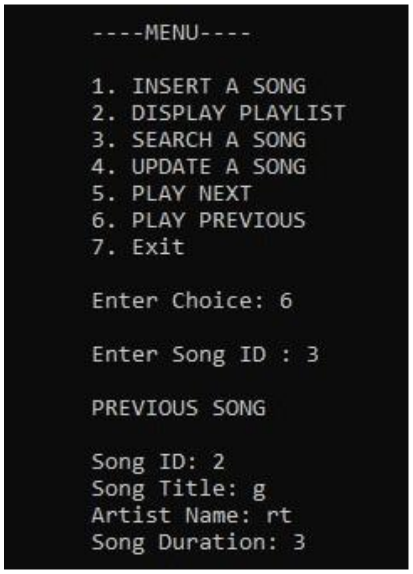
```
----MENU----

1. INSERT A SONG
2. DISPLAY PLAYLIST
3. SEARCH A SONG
4. UPDATE A SONG
5. PLAY NEXT
6. PLAY PREVIOUS
7. Exit

Enter Choice: 4

Enter Song ID to update: 1

----UPDATE----
0. Exit
1. Update Artist Name
2. Update Song Title
3. Update Song Duration
--------------

Enter Choice: 1

Update Artist Name: Tahira

Artist Name Updated successfuly

----UPDATE----
0. Exit
1. Update Artist Name
2. Update Song Title
3. Update Song Duration
--------------

Enter Choice: 0

Exiting from Updation...
```



```
----MENU----

1. INSERT A SONG
2. DISPLAY PLAYLIST
3. SEARCH A SONG
4. UPDATE A SONG
5. PLAY NEXT
6. PLAY PREVIOUS
7. Exit

Enter Choice: 3

Enter Song ID to search: 4

Song Found

Song ID: 4
Song Title: o
Artist Name: tt
Song Duration: 2
```



```
----MENU----

1. INSERT A SONG
2. DISPLAY PLAYLIST
3. SEARCH A SONG
4. UPDATE A SONG
5. PLAY NEXT
6. PLAY PREVIOUS
7. Exit

Enter Choice: 5

Enter Song ID : 3

SONG THAT WILL PLAY NEXT

Song ID: 4
Song Title: m
Artist Name: gg
Song Duration: 2
```



```
----MENU----

1. INSERT A SONG
2. DISPLAY PLAYLIST
3. SEARCH A SONG
4. UPDATE A SONG
5. PLAY NEXT
6. PLAY PREVIOUS
7. Exit

Enter Choice: 6

Enter Song ID : 3

PREVIOUS SONG

Song ID: 2
Song Title: g
Artist Name: rt
Song Duration: 3
```

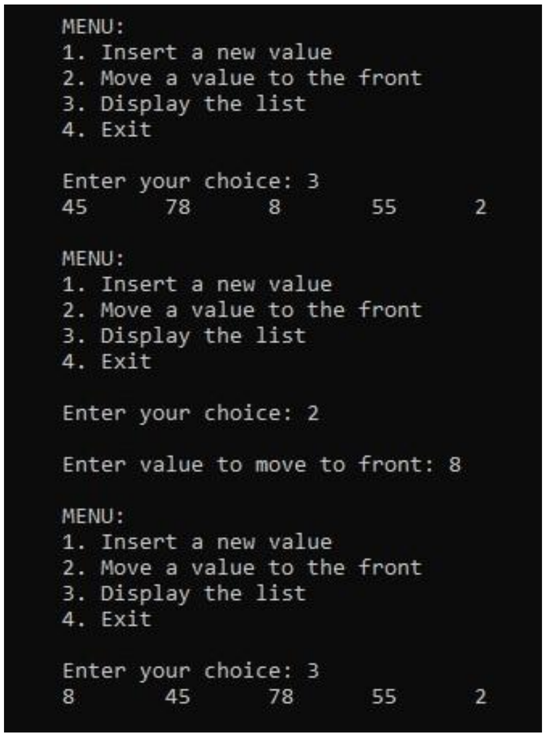
Exercise 1.3 Code

```
#include <iostream>

using namespace std;


struct Node {

  int Data;

  Node* pre;
```

```
    Node* Next;
};

class DoublyLinkedList {
private:
    Node* head;

public:
    DoublyLinkedList() {
        head = NULL;
    }

    void insert() {
        int value;
        cout << "\n\tEnter value to insert: ";
        cin >> value;

        Node* newNode = new Node();
        newNode->Data = value;
        newNode->pre = NULL;
        newNode->Next = NULL;

        if (head == NULL) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->Next != NULL) {
                temp = temp->Next;
            }
            temp->Next = newNode;
            newNode->pre = temp;
        }
    }

    void MoveToFront() {
        int value;
        cout << "\n\tEnter value to move to front: ";
        cin >> value;

        if (head == NULL)
            return;

        Node* current = head;
        Node* previous = NULL;

        while (current != NULL && current->Data != value) {
            previous = current;
            current = current->Next;
        }

        if (current == NULL || current == head)
            return;
        if (previous != NULL) {
            previous->Next = current->Next;
        }
        if (current->Next != NULL) {
            current->Next->pre = previous;
        }

        current->Next = head;
        head->pre = current;
        current->pre = NULL;
        head = current;
    }

    void display() {
        if (head == NULL) {
            cout << "\n\tList is empty." << endl;
            return;
        }

        Node* current = head;
        while (current != NULL) {
            cout << current->Data << " ";
            current = current->Next;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList dll;
    int choice;

    while (true) {
        cout << "\n\tMENU:\n";
        cout << "\t1. Insert a new value\n";
        cout << "\t2. Move a value to the front\n";
        cout << "\t3. Display the list\n";
        cout << "\t4. Exit\n";
        cout << "\n\tEnter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                dll.insert();
                break;

            case 2:
                dll.MoveToFront();
                break;

            case 3:
```

```
            dll.display();

            break;


          case 4:

            cout << "Exiting program.\n";

            return 0;


          default:

            cout << "Invalid choice, please try again.\n";

        }

    }


    return 0;

}
```

OUTPUT:



```
MENU:
1. Insert a new value
2. Move a value to the front
3. Display the list
4. Exit

Enter your choice: 3
45      78      8       55      2

MENU:
1. Insert a new value
2. Move a value to the front
3. Display the list
4. Exit

Enter your choice: 2

Enter value to move to front: 8

MENU:
1. Insert a new value
2. Move a value to the front
3. Display the list
4. Exit

Enter your choice: 3
8       45      78      55      2
```

Exercise 1.4 Code

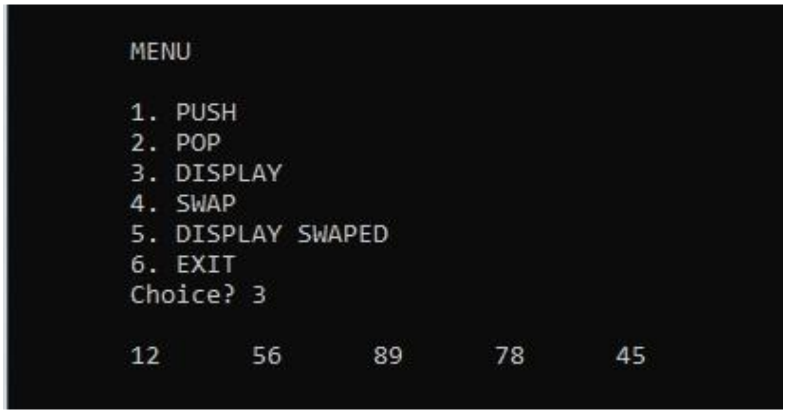
```
#include <iostream>

using namespace std;


//class

class Sstack {

        private:

                int Sarr[10];

                int Sarr2[10];

                int size;

                int top;

                int top2;

                int index;

        public:

                //constructor

                Sstack() {

                        top = -1;

                        top2 = -1;

                        size = 10;

                }

                //insertion

                void push() {

                        if (top >= size) {

                                cout<<"\n\tOVERFLOWED";

                        } else {
```

```
                top++;
                cout<<"\n\tEnter data: ";
                cin>>Sarr[top];
        }
}
//SWAP
void swap() {
        Sstack s2;
        if (top == -1) {
                cout<<"\n\tUNDERFLOWED";
        } else {
                while (top != -1) {
                        top2++;
                        Sarr2[top2] = Sarr[top];
                        top--;
                }
        }
}
//DELETION
void pop() {
        if (top == -1) {
                cout<<"\n\tUNDERFLOWED";
        } else {
                int num = Sarr[top];
                Sarr[top] = 0;
                top--;
                }
            }
            //DISPLAY
            void display() {
                cout<<"\n";
                for (int i=top; i>=0; i--) {
                    cout<<"\t"<<Sarr[i];
                }
                cout<<"\n";
            }
            //DISPLAY SWAP
            void display2() {
                cout<<"\n";
                for (int i=top2; i>=0; i--) {
                    cout<<"\t"<<Sarr2[i];
                }
                cout<<"\n";
            }
};

//main function
int main()
{
    Sstack ss;
    while (true){
        cout<<"\n\n\tMENU\n";
```

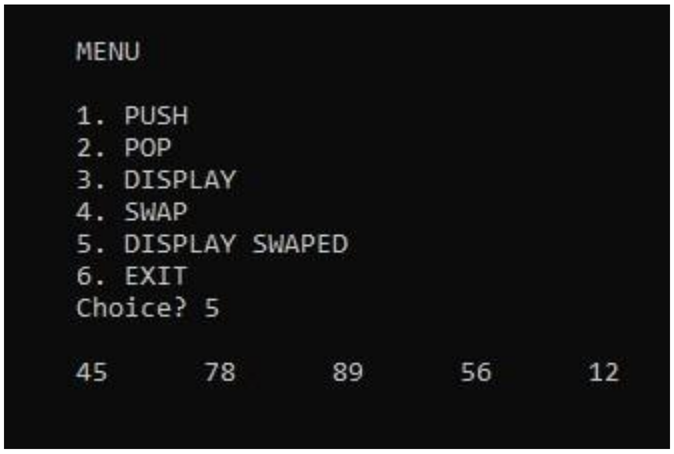
```
cout<<"\n\t1. PUSH\n";

cout<<"\t2. POP\n";

cout<<"\t3. DISPLAY\n";

cout<<"\t4. SWAP\n";

cout<<"\t5. DISPLAY SWAPED\n";

cout<<"\t6. EXIT";

int choice;

cout<<"\n\tChoice? ";

cin>>choice;

switch(choice){

        case 1:

                ss.push();

                break;

        case 2:

                ss.pop();

                break;

        case 3:

                ss.display();

                break;

        case 4:

                ss.swap();

                break;

        case 5:

                ss.display2();

                break;

        case 6:
```

```
                                return 0;

                                break;

                        default:

                                cout<<"\n\tInvalid Input";

                }

        }


        return 0;

}
```

OUTPUT:



```
MENU

1. PUSH
2. POP
3. DISPLAY
4. SWAP
5. DISPLAY SWAPED
6. EXIT
Choice? 3

12      56      89      78      45
```



```
MENU

1. PUSH
2. POP
3. DISPLAY
4. SWAP
5. DISPLAY SWAPED
6. EXIT
Choice? 5

45      78      89      56      12
```

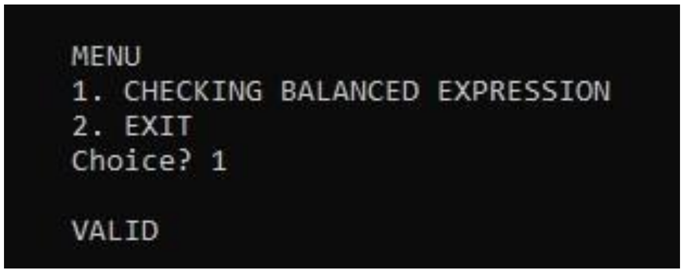
Exercise 1.5 Code

```
#include <iostream>
```

```
using namespace std;

struct Node {
  char data;
  Node *ptr;
};

class Dstack {
private:
  Node *top;
public:
  Dstack() {
    top = NULL;
  }

  // INSERTION
  void push() {
    Node *temp = new Node;
    temp->data = '(';
    temp->ptr = NULL;
    if (top == NULL) {
      top = temp;
    } else {
      temp->ptr = top;
      top = temp;
    }
}

// DELETION
void pop() {
    if (top == NULL) {
        cout << "\n\tUnderflowed\n";
    } else {
        char num = top->data;
        Node *temp = top;
        top = top->ptr;
        delete temp;
    }
}

// DISPLAY
void display() {
    Node *temp = top;
    cout << "\n";
    while (temp != NULL) {
        cout << "\t" << temp->data;
        temp = temp->ptr;
    }
    cout << "\n";
}

// CHECKING BALANCED EXPRESSION
```

```
    void balancedExpression(string expression) {
        Dstack d;
        char i = 0;
        char c;
        while (i < expression.length()) {
            c = expression[i];
            if (c == '(') {
                d.push();
            } else if (c == ')') {
                if (d.top == NULL) {
                    cout << "\n\tINVALID\n";
                    return;
                }
                d.pop();
            }
            i++;
        }
        if (d.top == NULL) {
            cout << "\n\tVALID\n";
        } else {
            cout << "\n\tINVALID\n";
        }
    }
};

int main() {
  Dstack ds;

  while (true) {

    cout << "\n\n\tMENU\n";

    cout << "\t1. CHECKING BALANCED EXPRESSION\n";

    cout << "\t2. EXIT";

    int choice;

    cout << "\n\tChoice? ";

    cin >> choice;

    switch (choice) {

      case 1:

        ds.balancedExpression("(98*0)");

        break;

      case 2:

        return 0;

        break;

      default:

        cout << "\n\tInvalid Input";

    }

  }


  return 0;

}
```

OUTPUT:



```
MENU
1. CHECKING BALANCED EXPRESSION
2. EXIT
Choice? 1

VALID
```

Conclusion

This lab manual provided hands-on experience with doubly linked lists and Stack. By implementing various operations, we reinforced our understanding of linked list manipulation, memory management, and the application of pointers in dynamic data structures.