## ***DATA STRUCTURE ASSIGMENT 1***

### Task 1:

### Answers:

### 1. O(n^2)

### 2. O(nlogn)

### 3. O(n^3)

### 4. O(nlogn)

### 5. O(n^2)

### Task 2:

### Code:

#include <iostream>

#include <cmath>

using namespace std;

class Term

{

public:

int coefficient;

int exponent;

Term \*next;

Term(int coefficient, int exponent)

{

this->coefficient = coefficient;

this->exponent = exponent;

this->next = NULL;

}

void displayTerm();

};

class Polynomial

{

private:

void insertAtTail(Term \*newTerm);

void insertAthead(Term \*newTerm);

Term \*exponentAlreadyExist(int exponent);

void deletePloynomial();

public:

Term \*firstTerm = NULL;

Term \*lastTerm = NULL;

int size = 0;

void Input();

void displayPolynomial();

Polynomial &operator+(Polynomial &obj);

int operator()(int val);

Polynomial \*operator\*(Polynomial &obj);

~Polynomial();

};

void Polynomial::insertAtTail(Term \*newTerm)

{

if (lastTerm == NULL)

{

firstTerm = newTerm;

lastTerm = newTerm;

return;

}

lastTerm->next = newTerm;

lastTerm = newTerm;

}

void Polynomial::insertAthead(Term \*newTerm)

{

if (firstTerm == NULL)

{

firstTerm = newTerm;

lastTerm = newTerm;

return;

}

newTerm->next = firstTerm;

firstTerm = newTerm;

}

Term \*Polynomial::exponentAlreadyExist(int exponent)

{

Term \*current = firstTerm;

while (current)

{

if (current->exponent == exponent)

{

return current;

}

current = current->next;

}

return NULL;

}

void Polynomial::Input()

{

deletePloynomial();

cout << "Enter Number of terms : ";

cin >> size;

int c, e;

int count = 0;

int max = 0;

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

{

cout << "Enter coffient of " << i + 1 << " Term  : ";

cin >> c;

cout << "Enter exponent of " << i + 1 << " Term  : ";

cin >> e;

if (c == 0)

{

count++;

}

if (i == 0)

{

Term \*newTerm = new Term(c, e);

max = e;

insertAthead(newTerm);

}

else

{

Term \*already = exponentAlreadyExist(e);

if (already)

{

already->coefficient = already->coefficient + c;

count++;

}

else

{

Term \*newTerm = new Term(c, e);

if (max > e)

{

insertAtTail(newTerm);

max = e;

}

else

{

insertAthead(newTerm);

}

}

}

}

size -= count;

}

void Polynomial::displayPolynomial()

{

Term \*current = firstTerm;

while (current)

{

// current->displayTerm();

if (current == firstTerm)

{

if (current->coefficient != 1)

cout << current->coefficient;

if (current->exponent != 0)

{

cout << "n^" << current->exponent << " ";

}

}

else

{

if (current->coefficient > 0)

{

cout << " + ";

}

if (current->coefficient != 1)

cout << current->coefficient;

if (current->exponent != 0)

{

if (current->exponent != 1)

cout << "n^" << current->exponent << " ";

else

cout << "n";

}

}

current = current->next;

}

}

void Polynomial::deletePloynomial()

{

while (firstTerm)

{

Term \*temp = firstTerm;

firstTerm = firstTerm->next;

delete temp;

size--;

}

}

Polynomial::~Polynomial()

{

deletePloynomial();

}

Polynomial &Polynomial::operator+(Polynomial &obj)

{

Term \*current1 = firstTerm;

Term \*current2 = obj.firstTerm;

Term \*pre1 = NULL;

Term \*pre2 = NULL;

while (current1 && current2)

{

if (current1->exponent == current2->exponent)

{

current1->coefficient += current2->coefficient;

pre1 = current1;

current1 = current1->next;

pre2 = current2;

current2 = current2->next;

}

else if (current1->exponent > current2->exponent)

{

current1 = current1->next;

}

else

{

Term \*newNode = new Term(current2->coefficient, current2->exponent);

if (current1 == firstTerm)

{

insertAthead(newNode);

}

else

{

newNode->next = pre1->next;

pre1->next = newNode;

}

current2 = current2->next;

}

}

while (current2)

{

Term \*newNode = new Term(current2->coefficient, current2->exponent);

insertAtTail(newNode);

current2 = current2->next;

}

return \*this;

}

int Polynomial::operator()(int val)

{

int result = 0;

Term \*current = firstTerm;

while (current)

{

result += current->coefficient \* pow(val, current->exponent);

current = current->next;

}

return result;

}

Polynomial \*Polynomial::operator\*(Polynomial &obj)

{

Polynomial \*newObj = new Polynomial;

if (size > obj.size)

{

Term \*current2 = obj.firstTerm;

Term \*pre1 = NULL;

Term \*pre2 = NULL;

while (current2)

{

Term \*current1 = firstTerm;

while (current1)

{

int c = current1->coefficient \* current2->coefficient;

int e = current1->exponent + current2->exponent;

Term \*newTerm = newObj->exponentAlreadyExist(e);

if (newTerm)

{

newTerm->coefficient += c;

}

else

{

newTerm = new Term(c, e);

newObj->insertAtTail(newTerm);

}

current1 = current1->next;

}

current2 = current2->next;

}

}

else

{

Term \*current2 = firstTerm;

Term \*pre1 = NULL;

Term \*pre2 = NULL;

while (current2)

{

Term \*current1 = obj.firstTerm;

while (current1)

{

int c = current1->coefficient \* current2->coefficient;

int e = current1->exponent + current2->exponent;

Term \*newTerm = newObj->exponentAlreadyExist(e);

if (newTerm)

{

newTerm->coefficient += c;

}

else

{

newTerm = new Term(c, e);

newObj->insertAtTail(newTerm);

}

current1 = current1->next;

}

current2 = current2->next;

}

}

return newObj;

}

void menu()

{

cout << "---------------------------------------------------------" << endl;

puts("| Enter 1 : Insert Polynomial 1          :              |");

puts("| Enter 2 : Insert Polynomial 2          :              |");

puts("| Enter 3 : print Polynomial 1           :              |");

puts("| Enter 4 : print Polynomial 2           :              |");

puts("| Enter 5 : To Add polynomial 1 & 2      :              |");

puts("| Enter 6 : To Multiply Polynomial 1 & 2 :              |");

puts("| Enter 7 : To Evaluate Polynomials      :              |");

puts("| Enter 8 : To Exit programe             :              |");

cout << "---------------------------------------------------------" << endl;

}

char inputHandler(string input)

{

if (input.size() > 1 || input.size() < 1)

{

return '9';

}

return input[0];

}

int main()

{

string str;

Polynomial first, second;

bool programeExit = false;

bool poly1 = false;

bool poly2 = false;

do

{

menu();

cout << "Enter your choice : ";

cin >> str;

char choice = inputHandler(str);

switch (choice)

{

case '1':

first.Input();

poly1 = true;

break;

case '2':

second.Input();

poly2 = true;

break;

case '3':

if (!poly1)

{

cout << "Please Enter Polynomial 1 first " << endl;

}

else

{

cout << "Polynomila 1 : ";

first.displayPolynomial();

cout << endl;

}

break;

case '4':

if (!poly2)

{

cout << "Please Enter Polynomial 2 first " << endl;

}

else

{

cout << "Polynomila 2 : ";

second.displayPolynomial();

cout << endl;

}

break;

case '5':

if (!poly1)

{

cout << "Please Enter Polynomial 1 first " << endl;

}

else if (!poly2)

{

cout << "Please Enter Polynomial 2 first " << endl;

}

else

{

cout << "( ";

first.displayPolynomial();

cout << " ) + ( ";

second.displayPolynomial();

cout << " )" << " =  ";

first + second;

first.displayPolynomial();

cout << endl;

}

break;

case '6':

if (!poly1)

{

cout << "Please Enter Polynomial 1 first " << endl;

}

else if (!poly2)

{

cout << "Please Enter Polynomial 2 first " << endl;

}

else

{

cout << "( ";

first.displayPolynomial();

cout << " ) \* ( ";

second.displayPolynomial();

cout << " )" << " =  ";

Polynomial \*product = first \* second;

product->displayPolynomial();

cout << endl;

delete product;

product = NULL;

}

break;

case '7':

if (!poly1)

{

cout << "Please Enter Polynomial 1 first " << endl;

}

else if (!poly2)

{

cout << "Please Enter Polynomial 2 first " << endl;

}

else

{

int n;

cout << "Enter value for n : ";

cin >> n;

cout << endl;

first.displayPolynomial();

cout << " = " << first(n) << endl;

second.displayPolynomial();

cout << " = " << second(n) << endl;

}

break;

case '8':

programeExit = true;

cout << "Programe is going to exit " << endl;

break;

default:

cout << "Invalid Input . Please Enter a valid Input " << endl;

break;

}

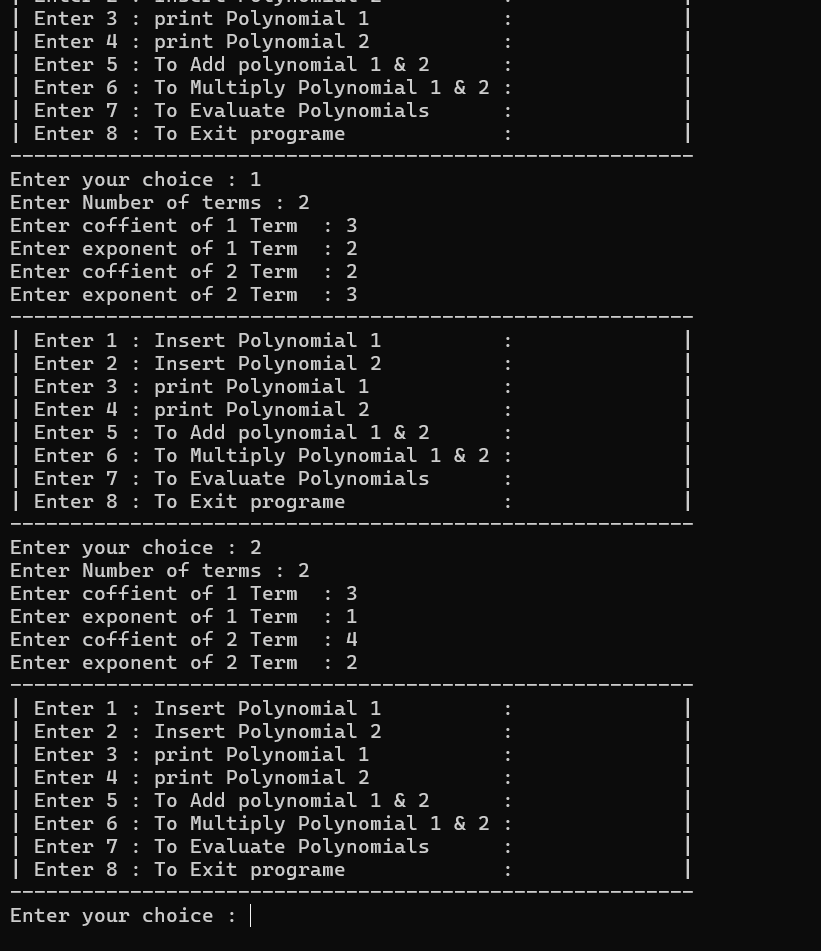
} while (!programeExit);

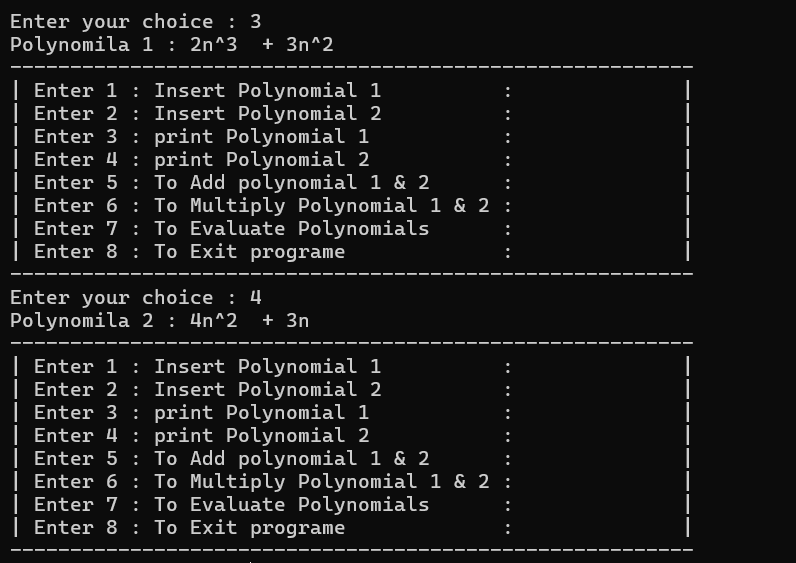
system("pause");

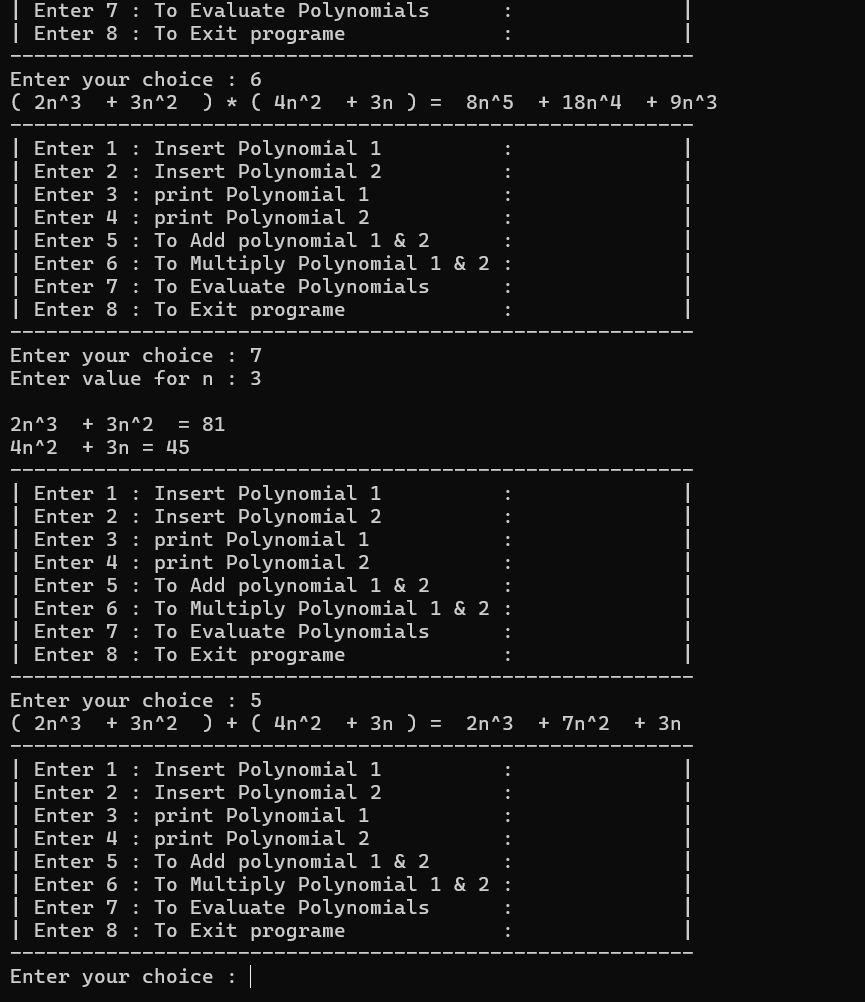
return 0;

}

### Output:







### Task 3:

### Code:

#include <iostream>

#include <string>

using namespace std;

class Node

{

public:

string action;

Node \*next;

Node \*pre;

Node(string);

};

Node::Node(string action)

{

this->action = action;

next = NULL;

pre = NULL;

}

class Actions

{

private:

Node \*orgHead;

Node \*hisHead;

Node \*orgTail;

Node \*hisTail;

int orgSize;

int hisSize;

void addActionHis(Node \*node);

public:

Actions();

void displayHis();

void displayOrg();

void addAction(string, Node \*);

void UndoAction();

void redoAction();

void deleteAction();

void deleteNode(Node \*);

void mostRecentAction();

Node \*findAction(int, Node \*);

~Actions();

};

Actions::Actions()

{

orgHead = NULL;

hisHead = NULL;

orgSize = 0;

hisSize = 0;

}

void Actions::displayHis()

{

if (!hisHead)

{

cout << "List is Empty ." << endl;

return;

}

Node \*current = hisHead;

int count = 1;

while (current)

{

cout << count << ". " << current->action << endl;

current = current->next;

count++;

}

cout << endl;

}

void Actions::displayOrg()

{

if (!orgHead)

{

cout << "List is Empty ." << endl;

return;

}

Node \*current = orgHead;

int count = 1;

while (current)

{

cout << count << ". " << current->action << endl;

current = current->next;

count++;

}

cout << endl;

}

void Actions::addActionHis(Node \*node)

{

if (!hisHead)

{

hisHead = node;

hisTail = node;

node->next = NULL;

node->pre = NULL;

return;

}

hisTail->next = node;

node->pre = hisTail;

hisTail = node;

hisTail->next = NULL;

}

void removeAction(Node \*node)

{

if (node->next)

{

node->next->pre = node->pre;

}

if (node->pre)

{

node->pre->next = node->next;

}

}

void Actions::addAction(string action, Node \*node = NULL)

{

Node \*newNode;

orgSize++;

if (!node)

{

newNode = new Node(action);

}

else

{

newNode = node;

}

if (!orgHead)

{

orgHead = newNode;

orgTail = orgHead;

newNode->next = NULL;

newNode->pre = NULL;

return;

}

orgTail->next = newNode;

newNode->pre = orgTail;

orgTail = newNode;

orgTail->next = NULL;

}

void Actions::UndoAction()

{

if (orgHead == NULL)

{

cout << "There is No Action to Undo.Please add an Action First " << endl;

return;

}

displayOrg();

int n;

cout << "Enter number of Action : ";

cin >> n;

if (n < 1 || n > orgSize)

{

cout << "Out Range : " << endl;

return;

}

Node \*action = findAction(n, orgHead);

if (action == orgTail)

{

orgTail = orgTail->pre;

orgTail->next = NULL;

}

if (action != orgHead)

{

removeAction(action);

}

else

{

orgHead = orgHead->next;

}

addActionHis(action);

orgSize--;

hisSize++;

}

Node \*Actions::findAction(int n, Node \*head)

{

int count = 1;

while (head)

{

if (count == n)

{

return head;

}

count++;

head = head->next;

}

return NULL;

}

void Actions::deleteNode(Node \*node)

{

if (node == orgHead)

{

if (orgSize == 1)

{

delete orgHead;

orgHead = NULL;

}

else

{

orgHead = orgHead->next;

delete orgHead->pre;

orgHead->pre = NULL;

}

orgSize--;

return;

}

if (node->next == NULL)

{

node = node->pre;

delete node->next;

node->next == NULL;

orgSize--;

return;

}

node->pre->next = node->next;

node->next->pre = node->pre;

delete node;

orgSize--;

return;

}

void Actions::mostRecentAction()

{

if (orgHead == NULL)

{

cout << "There is No Recent  Action. " << endl;

return;

}

cout << "Most Recent Action : " << orgTail->action << endl;

}

void Actions::redoAction()

{

if (hisHead == NULL)

{

cout << "There is No Action to redo.Please add an Action First " << endl;

return;

}

displayHis();

int n;

cout << "Enter number of Action : ";

cin >> n;

if (n < 1 || n > hisSize)

{

cout << "Out Range : " << endl;

return;

}

Node \*action = findAction(n, hisHead);

if (action != hisHead)

{

removeAction(action);

}

else

{

hisHead = hisHead->next;

}

addAction("", action);

hisSize--;

}

void Actions::deleteAction()

{

if (orgHead == NULL)

{

cout << "There is  No Action to delete ." << endl;

return;

}

displayOrg();

int n;

cout << "Enter number of Action : ";

cin >> n;

if (n < 1 || n > orgSize)

{

cout << "Out Range : " << endl;

return;

}

Node \*node = findAction(n, orgHead);

deleteNode(node);

orgSize--;

return;

}

Actions::~Actions()

{

while (orgHead)

{

deleteNode(orgHead);

}

while (hisHead && hisHead->next)

{

hisHead = hisHead->next;

delete hisHead->pre;

hisHead->pre = NULL;

}

delete hisHead;

}

void menu()

{

cout << "----------------------------------------------" << endl;

puts("| Enter 1 To add a new Action              : |");

puts("| Enter 2 To display ACtion History        : |");

puts("| Enter 3 To Undo An Action                : |");

puts("| Enter 4 To display Undo Actions          : |");

puts("| Enter 5 To redo An Action                : |");

puts("| Enter 6 To delete An Action              : |");

puts("| Enter 7 Most Recent Action               : |");

puts("| Enter 8 To Exit Programe                 : |");

cout << "----------------------------------------------" << endl;

}

char inputHandler(string input)

{

if (input.size() > 1 || input.size() < 1)

{

return '9';

}

return input[0];

}

int main()

{

Actions act;

bool programeExit = false;

do

{

cout << endl;

string input;

string action;

menu();

cout << "Enter your choice : ";

cin >> input;

char choice = inputHandler(input);

switch (choice)

{

case '1':

cin.ignore();

cout << "Enter Action : ";

getline(cin, action);

act.addAction(action);

break;

case '2':

cout << "Action History : " << endl;

act.displayOrg();

break;

case '3':

cout << "Actions List : " << endl;

act.UndoAction();

break;

case '4':

cout << "Undo Actions List : " << endl;

act.displayHis();

break;

case '5':

cout << "Undo Actions List : " << endl;

act.redoAction();

break;

case '6':

act.deleteAction();

break;

case '7':

act.mostRecentAction();

break;

case '8':

cout << "Programe is going to Exit " << endl;

programeExit = true;

break;

default:

cout << "Enter Invalid choice . Please try again Later " << endl;

break;

}

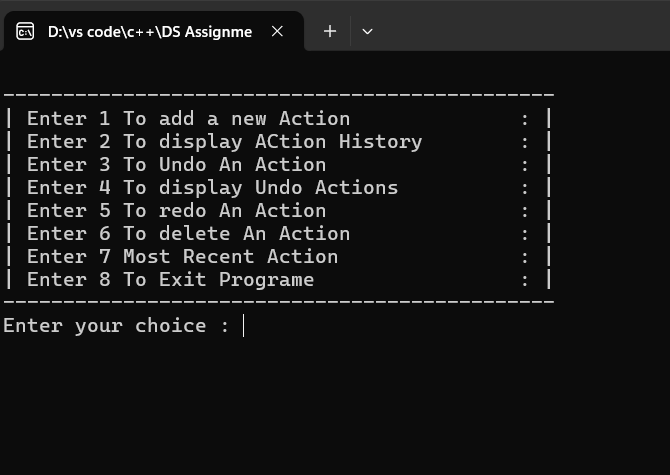
} while (!programeExit);

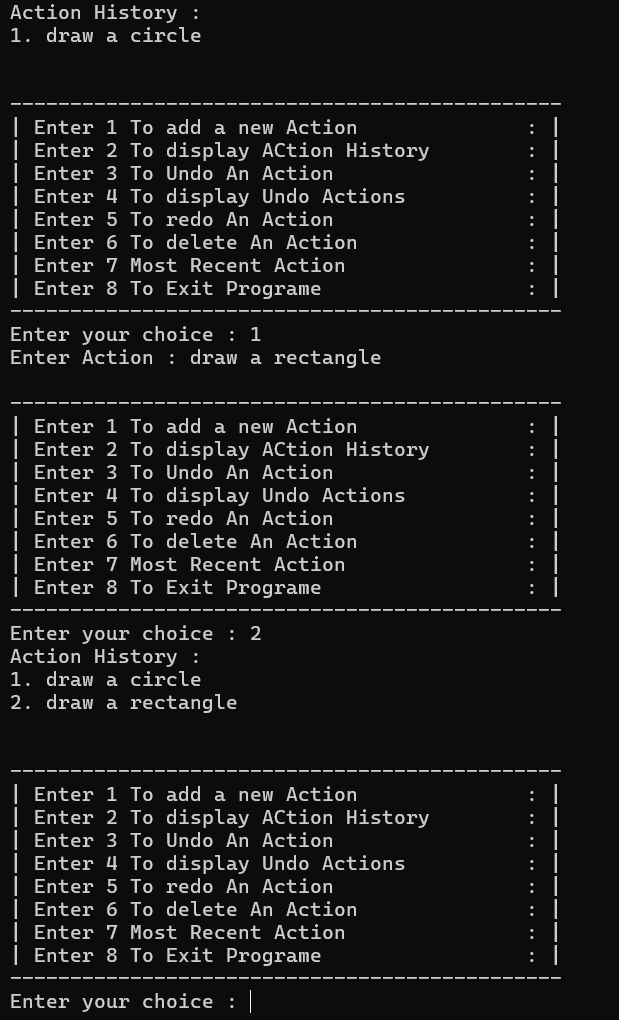
system("pause");

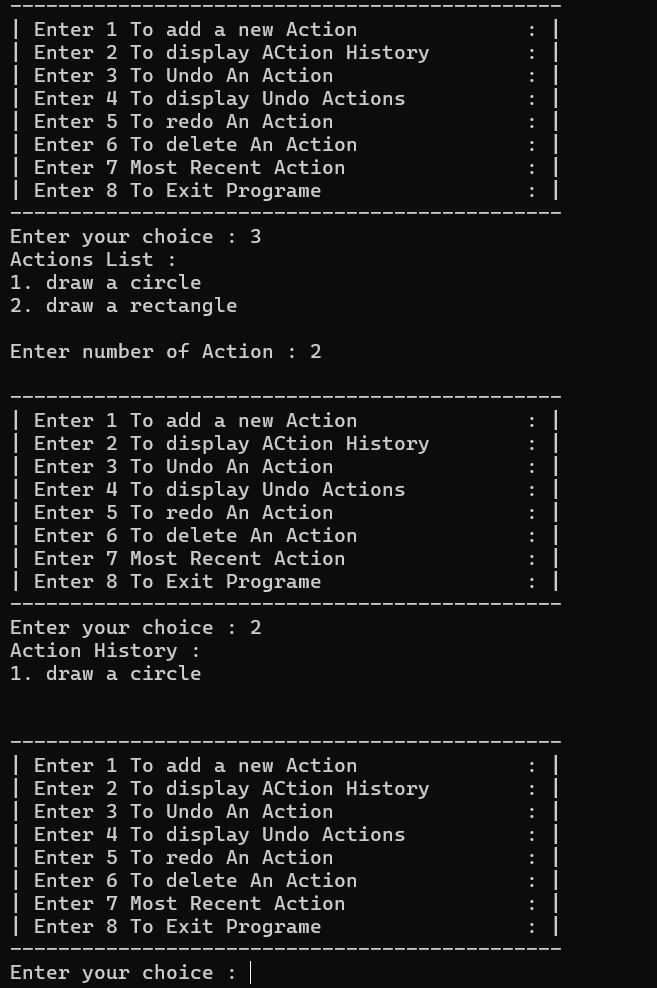
return 0;

}

### Output:







### Task 4:

### Code:

#include <iostream>

#include <string>

using namespace std;

class Node

{

public:

    int id;

    Node \*next;

    Node \*previous;

    Node \*nestedNode;

    Node(int idValue);

};

class BookList

{

public:

    Node \*head;

    BookList();

    ~BookList();

    void clearList();

    void insertBook(int id, bool hasNestedVolumes);

    void addNestedVolumes(Node \*parent);

    void flattenList();

    void printList() const;

    void inputBook();

    void displayMenu();

};

Node::Node(int idValue) : id(idValue), next(nullptr), previous(nullptr), nestedNode(nullptr) {}

BookList::BookList() : head(nullptr) {}

BookList::~BookList()

{

    clearList();

}

void BookList::clearList()

{

    Node \*current = head;

    while (current)

    {

        Node \*nextNode = current->next;

        delete current;

        current = nextNode;

    }

}

void BookList::insertBook(int id, bool hasNestedVolumes)

{

    Node \*newNode = new Node(id);

    if (!head)

    {

        head = newNode;

    }

    else

    {

        Node \*temp = head;

        while (temp->next)

        {

            temp = temp->next;

        }

        temp->next = newNode;

        newNode->previous = temp;

    }

    if (hasNestedVolumes)

    {

        addNestedVolumes(newNode);

    }

}

void BookList::addNestedVolumes(Node \*parent)

{

    int count;

    cout << "Enter number of volumes/series for book " << parent->id << ": ";

    cin >> count;

    for (int i = 0; i < count; ++i)

    {

        int nestedId;

        bool isVolume;

        cout << "Enter ID of volume/series: ";

        cin >> nestedId;

        cout << "Is it a volume? (1 for yes, 0 for no): ";

        cin >> isVolume;

        Node \*nestedNode = new Node(nestedId);

        if (!parent->nestedNode)

        {

            parent->nestedNode = nestedNode;

        }

        else

        {

            Node \*temp = parent->nestedNode;

            while (temp->next)

            {

                temp = temp->next;

            }

            temp->next = nestedNode;

            nestedNode->previous = temp;

        }

        if (isVolume)

        {

            addNestedVolumes(nestedNode);

        }

    }

}

void BookList::flattenList()

{

    if (!head)

    {

        cout << "\nNo list to flatten.\n";

        return;

    }

    Node \*current = head;

    while (current)

    {

        if (current->nestedNode)

        {

            Node \*nestedCurrent = current->nestedNode;

            while (nestedCurrent)

            {

                Node \*nextNode = nestedCurrent->next;

                nestedCurrent->next = current->next;

                if (current->next)

                {

                    current->next->previous = nestedCurrent;

                }

                current->next = nestedCurrent;

                nestedCurrent->previous = current;

                nestedCurrent = nextNode;

            }

            current->nestedNode = nullptr;

        }

        current = current->next;

    }

}

void BookList::printList() const

{

    if (!head)

    {

        cout << "\nNo list to print.\n";

        return;

    }

    Node \*current = head;

    while (current)

    {

        cout << current->id << " ";

        Node \*nestedCurrent = current->nestedNode;

        while (nestedCurrent)

        {

            cout << "\n  Volume/Series of book " << current->id << ": " << nestedCurrent->id << " ";

            nestedCurrent = nestedCurrent->next;

        }

        current = current->next;

    }

    cout << endl;

}

void BookList::inputBook()

{

    int id;

    bool hasNested;

    cout << "Enter Book ID: ";

    cin >> id;

    cout << "Does it have volumes/series? (1 for yes, 0 for no): ";

    cin >> hasNested;

    insertBook(id, hasNested);

}

void Menu()

{

    puts("-------------------------------------");

    puts("| 1. Add a book                    | ");

    puts("| 2. Print books                   | ");

    puts("| 3. Flatten list                  | ");

    puts("| 4. Exit                          | ");

    puts("-------------------------------------");

}

char inputHandler(string input)

{

    if (input.size() > 1 || input.size() < 1)

    {

        return '9';

    }

    return input[0];

}

int main()

{

    BookList books;

    bool programExit = false;

    do

    {

        Menu();

        cout << "Enter your choice: ";

        string input;

        cin >> input;

        char choice = inputHandler(input);

        switch (choice)

        {

        case '1':

            books.inputBook();

            break;

        case '2':

            books.printList();

            break;

        case '3':

            books.flattenList();

            books.printList();

            break;

        case '4':

            cout << "Program is going to Exit " << endl;

            programExit = true;

            break;

        default:

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

            break;

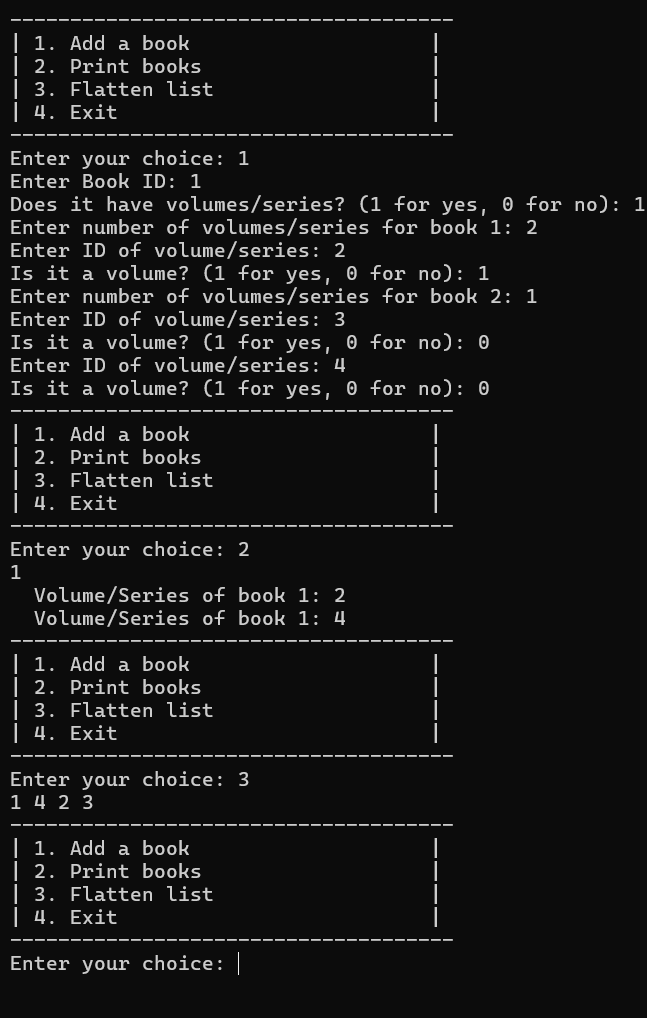
        }

    } while (!programExit);

    return 0;

}

### Output:



### Task :

### Code:

#include <iostream>

using namespace std;

class Node

{

public:

    int id;

    Node \*next;

    Node(int id)

    {

        this->id = id;

        this->next = nullptr;

    }

};

class CircularLinkedList

{

public:

    Node \*head;

    CircularLinkedList()

    {

        head = nullptr;

    }

    void insert(int id)

    {

        Node \*newNode = new Node(id);

        if (head == nullptr)

        {

            head = newNode;

            head->next = head;

        }

        else

        {

            Node \*temp = head;

            while (temp->next != head)

            {

                temp = temp->next;

            }

            temp->next = newNode;

            newNode->next = head;

        }

    }

    void moveNodeToFront(int id)

    {

        if (head == nullptr)

        {

            puts("No node to access.");

            return;

        }

        Node \*temp = head;

        Node \*tail = nullptr;

        do

        {

            tail = temp;

            temp = temp->next;

        } while (temp != head);

        temp = head;

        Node \*prev = nullptr;

        do

        {

            if (temp->id == id)

            {

                if (prev)

                {

                    prev->next = temp->next;

                    tail->next = temp;

                    temp->next = head;

                    head = temp;

                }

                puts("After accessing file id.");

                return;

            }

            prev = temp;

            temp = temp->next;

        } while (temp != head);

        puts("Desired id not found in list.");

    }

    void print()

    {

        if (head == nullptr)

        {

            puts("No list to print.");

            return;

        }

        Node \*temp = head;

        do

        {

            cout << temp->id << " ";

            temp = temp->next;

        } while (temp != head);

        cout << endl;

    }

};

void Menu()

{

    cout << "--------------------------------------" << endl;

    puts("| Enter 1. Insert Node              : |");

    puts("| Enter 2. Move Node to Front       : |");

    puts("| Enter 3. Print List               : |");

    puts("| Enter 4 To Exit Programe          : |");

    cout << "---------------------------------------" << endl;

}

char inputHandler(string input)

{

    if (input.size() > 1 || input.size() < 1)

    {

        return '9';

    }

    return input[0];

}

int main()

{

    CircularLinkedList cache;

    bool progameExit = false;

    do

    {

        Menu();

        char choice;

        int id;

        string input;

        cout << "Enter your choice: ";

        cin >> input;

        choice = inputHandler(input);

        switch (choice)

        {

        case '1':

            cout << "Enter id to insert: ";

            cin >> id;

            cache.insert(id);

            puts("Node inserted.");

            break;

        case '2':

            cout << "Enter id to move to front: ";

            cin >> id;

            cache.moveNodeToFront(id);

            break;

        case '3':

            puts("Current state of the list:");

            cache.print();

            break;

        case '4':

            puts("Exiting the program...");

            progameExit = true;

            return 0;

        default:

            puts("Invalid choice. Please try again.");

        }

    } while (!progameExit);

    system("pause");

    return 0;

}

### Output:



