

Applied Programming

Assignment



October 5, 2022

Muhammad talha Arif

22F-3769

# **Question#1**

Take inputs in a user defined array. After that if the input is even, place it at an even index and if the input is odd, place it an odd index. If the user puts in an even integer and all even indexes are occupied, ask user to enter an odd integer and vice versa. If all the even and odd entries are filled, then notify the user that the program has ended.

## **Code**

#include <iostream>

#include <string>

using namespace std;

int main()

{

// User Defined array pointer

int\* Array\_List;

// Taking Array size as input from user

int Array\_Size;

cout << "Enter Size of Array: ";

cin >> Array\_Size;

// Allocating User Defined array pointer with N-Size

Array\_List = new int[Array\_Size];

for (int i = 0; i < Array\_Size; i++)

{

Array\_List[i] = -1;

}

Array\_Size -= 1;

// Counter for even & odd index

int Even\_Index = 0;

int Odd\_Index = 1;

// Variable to take input from user

int User\_Input;

while (true)

{

if (Odd\_Index > Array\_Size && Even\_Index > Array\_Size)

{

cout << "There is no space left in array..." << endl;

cout << "Terminating Program..." << endl;

break;

}

cout << "Enter a Number: ";

cin >> User\_Input;

if (!(User\_Input % 2)) // If even number

{

if (Even\_Index > Array\_Size)

{

cout << "\nEven Indexes are full" << endl;

continue;

}

Array\_List[Even\_Index] = User\_Input;

Even\_Index += 2;

}

else // If odd number

{

if (Odd\_Index > Array\_Size)

{

cout << "\nOdd Indexes are full" << endl;

continue;

}

Array\_List[Odd\_Index] = User\_Input;

Odd\_Index += 2;

}

}

cout << "[ ";

Array\_Size += 1;

for (int i = 0; i < Array\_Size; i++)

{

cout << Array\_List[i] << " ";

}

cout << "]" << endl;

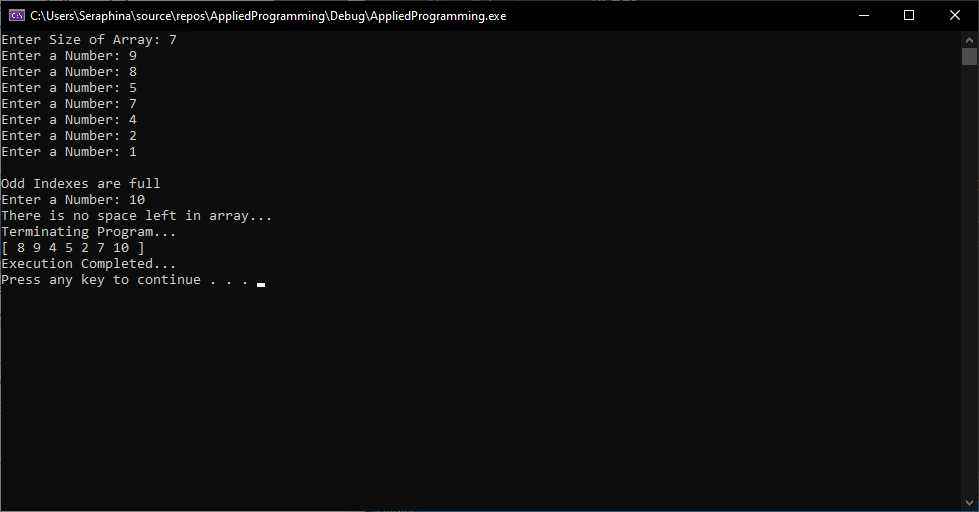
cout << "Execution Completed..." << endl;

system("PAUSE"); // Prevent console from exiting.

return 0;

}

## **Screenshot**



# **Question#2**

Create a user defined array and then take inputs. Print all the Divisors of every number user gave as input.

## **Code**

#include <iostream>

#include <string>

using namespace std;

int main()

{

// User Defined array pointer

int\* Array\_List;

// Taking Array size as input from user

int Array\_Size;

cout << "Enter Size of Array: ";

cin >> Array\_Size;

// Allocating User Defined array pointer with N-Size

Array\_List = new int[Array\_Size];

for (int i = 0; i < Array\_Size; i++)

{

cout << "Enter value for ID[" << i << "] : ";

cin >> Array\_List[i];

}

for (int i = 0; i < Array\_Size; i++)

{

cout << Array\_List[i] << ": ";

for (int j = 1; j < Array\_List[i]; j++)

{

if (Array\_List[i] % j == 0)

cout << j << " ";

}

cout << endl;

}

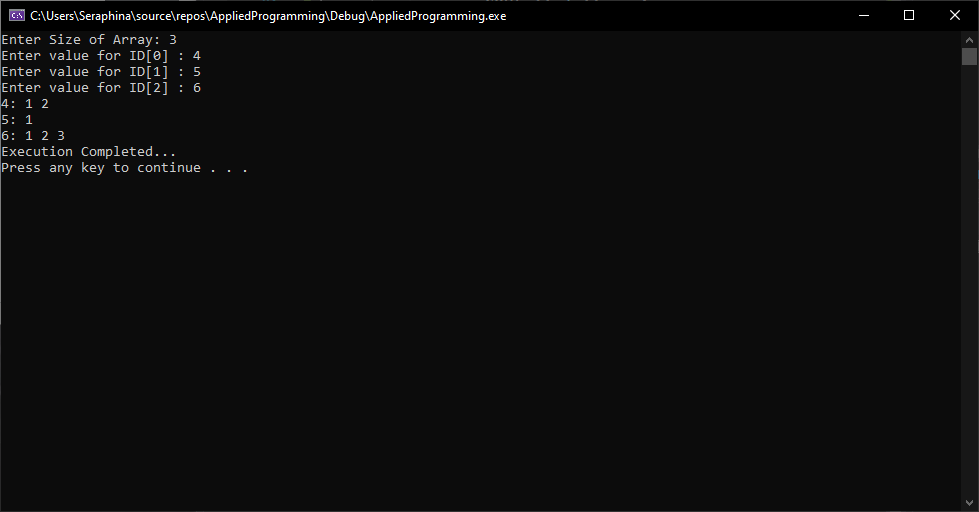
cout << "Execution Completed..." << endl;

system("PAUSE"); // Prevent console from exiting.

return 0;

}

## **Screenshot**



# **Question#3**

Your task is to input two sorted singly linked lists in descending order. Change the next pointers to obtain a single, merged linked list which also has data in descending order. Note that either head pointer given may be null meaning that the corresponding list is empty. You must use the following function prototype,

node\* merge\_lists(node \*, node \*);

## **Code**

#include <iostream>

#include <string>

using namespace std;

struct Node

{

int data;

Node\* next;

};

class ADT\_LinkedList

{

private:

Node\* head = NULL;

public:

void Insert(int value);

void SetHead(Node\* head);

Node\* GetHead();

Node\* Merge\_Lists(Node\* list1, Node\* list2);

void DisplayList();

};

void ADT\_LinkedList::Insert(int value)

{

if (value < 0)

{

cout << "Value cannot be less than 0" << endl;

return;

}

cout << "Inserting Node: " << value << endl;

Node\* temp = new Node();

temp->data = value;

if (head == NULL || head->data < value)

{

temp->next = head;

head = temp;

}

else

{

bool flag = false;

Node\* next = head;

Node\* prev = NULL;

while (next->next != NULL)

{

if (next->data == value)

{

cout << "Node Already exists..." << endl;

return;

}

prev = next;

next = next->next;

if (prev != NULL && next->data < value)

{

prev->next = temp;

temp->next = next;

flag = true;

break;

}

}

if (!flag)

{

next->next = temp;

temp->next = NULL;

}

}

}

void ADT\_LinkedList::SetHead(Node\* head)

{

this->head = head;

}

Node\* ADT\_LinkedList::GetHead()

{

return this->head;

}

Node\* ADT\_LinkedList::Merge\_Lists(Node\* list1, Node\* list2)

{

Node\* ptr = new Node();

Node\* prev = ptr;

while (list1 != NULL && list2 != NULL)

{

if (list1->data <= list2->data)

{

prev->next = list1;

list1 = list1->next;

}

else {

prev->next = list2;

list2 = list2->next;

}

prev = prev->next;

}

if (list1 == NULL)

prev->next = list2;

if (list2 == NULL)

prev->next = list1;

return ptr->next;

}

void ADT\_LinkedList::DisplayList()

{

Node\* tmp = head;

cout << "[ ";

while (tmp != NULL)

{

cout << tmp->data << " ";

tmp = tmp->next;

}

cout << "]" << endl;

}

int main()

{

ADT\_LinkedList obj1;

obj1.Insert(10);

obj1.Insert(1);

obj1.Insert(5);

obj1.Insert(20);

obj1.Insert(15);

cout << "List Number-1" << endl;

obj1.DisplayList();

cout << endl;

ADT\_LinkedList obj2;

obj2.Insert(25);

obj2.Insert(16);

obj2.Insert(12);

obj2.Insert(40);

obj2.Insert(75);

cout << "List Number-2" << endl;

obj2.DisplayList();

cout << endl;

ADT\_LinkedList obj3;

obj3.SetHead( obj3.Merge\_Lists(obj1.GetHead(), obj2.GetHead()));

cout << "Merge List" << endl;

obj3.DisplayList();

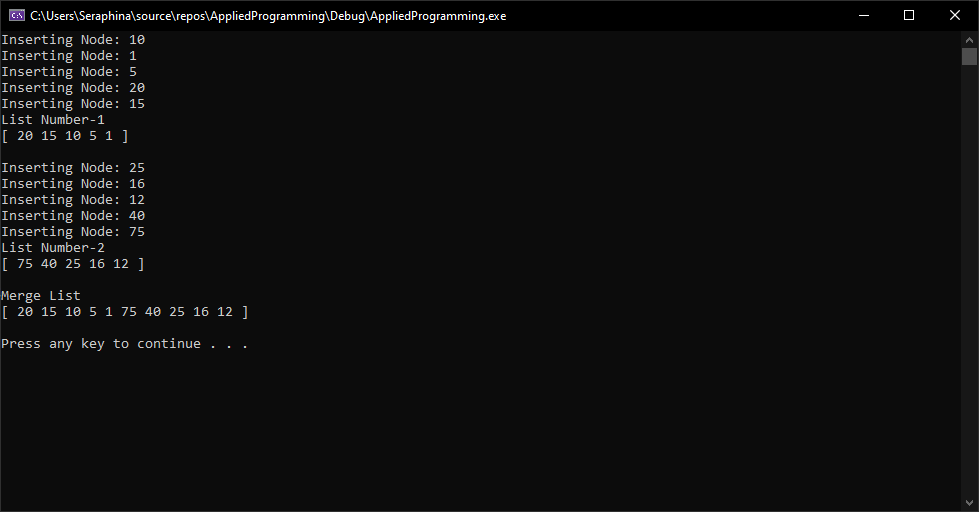
cout << endl;

system("PAUSE");

return 0;

}

## **Screenshot**



# **Question#4**

Your task is to take two singly linked lists let suppose A and B. The last index of A is connected to 3rd index of the B. Find the common elements in both linked lists. For more idea see the figure below.

## **Code**

#include <iostream>

#include <string>

using namespace std;

struct Node

{

int data;

Node\* next;

};

class ADT\_LinkedList

{

private:

Node\* head = NULL;

public:

void Insert(int value);

void SetHead(Node\* head);

Node\* GetHead();

void ConnectLists(Node\* otherList);

void FindCommonItems(Node\* otherList);

void DisplayList();

};

void ADT\_LinkedList::Insert(int value)

{

Node\* node = new Node();

node->data = value;

node->next = NULL;

if (head == NULL)

{

head = node;

}

else

{

Node\* ptr = head;

while (ptr->next != NULL)

ptr = ptr->next;

ptr->next = node;

}

}

void ADT\_LinkedList::SetHead(Node\* head)

{

this->head = head;

}

Node\* ADT\_LinkedList::GetHead()

{

return this->head;

}

void ADT\_LinkedList::ConnectLists(Node\* otherList)

{

if (head == NULL)

{

cout << "1st Linked List must not be Empty..." << endl;

return;

}

Node\* ptr = head;

while (ptr->next != NULL)

ptr = ptr->next;

Node\* temp = otherList;

if (temp->next != NULL)

{

temp = temp->next;

if (temp->next != NULL)

temp = temp->next;

else

{

cout << "There is not 3rd Index in other Linked List..." << endl;

return;

}

}

else

{

cout << "There is not 3rd Index in other Linked List..." << endl;

return;

}

ptr->next = temp; // Connect 1st-LL last node with 3rd index of 2nd-LL

}

void ADT\_LinkedList::FindCommonItems(Node\* otherList)

{

Node\* ptr1 = head;

Node\* ptr2;

cout << "Common Elements\n[ ";

while (ptr1 != NULL)

{

ptr2 = otherList;

while (ptr2 != NULL)

{

if (ptr1->data == ptr2->data)

{

cout << ptr1->data << " ";

}

ptr2 = ptr2->next;

}

ptr1 = ptr1->next;

}

cout << "]" << endl;

}

void ADT\_LinkedList::DisplayList()

{

Node\* ptr = head;

cout << "[ ";

while (ptr != NULL)

{

cout << ptr->data << " ";

ptr = ptr->next;

}

cout << "]" << endl;

}

int main()

{

ADT\_LinkedList obj1;

obj1.Insert(1);

obj1.Insert(3);

obj1.Insert(5);

obj1.Insert(7);

obj1.Insert(9);

obj1.Insert(15);

obj1.Insert(20);

cout << "List Number-1" << endl;

obj1.DisplayList();

cout << endl;

ADT\_LinkedList obj2;

obj2.Insert(2);

obj2.Insert(4);

obj2.Insert(6);

obj2.Insert(8);

obj2.Insert(10);

obj2.Insert(12);

obj2.Insert(24);

cout << "List Number-2" << endl;

obj2.DisplayList();

cout << endl;

obj1.ConnectLists(obj2.GetHead());

cout << "\nAfter Connecting Linked List" << endl;

obj1.DisplayList();

cout << endl;

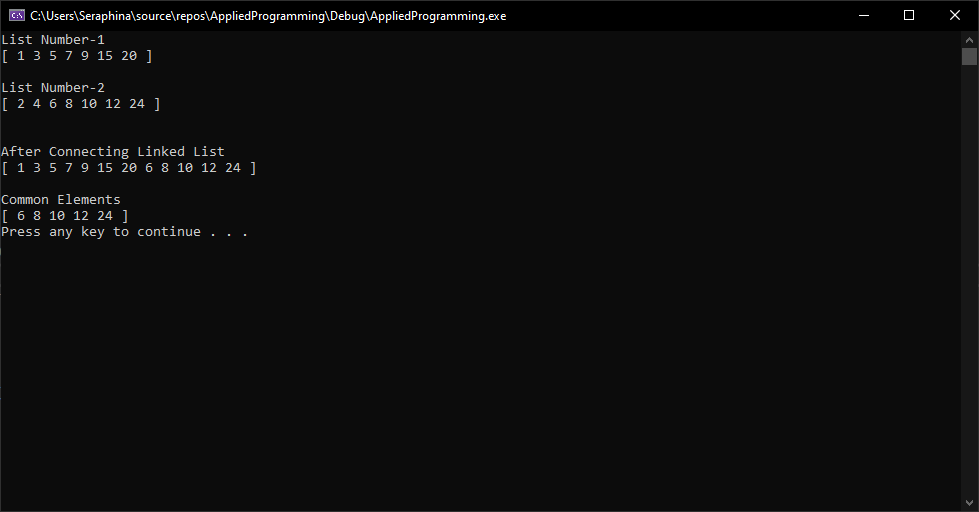
obj1.FindCommonItems(obj2.GetHead());

system("PAUSE");

return 0;

}

## **Screenshot**



# **Question#5**

Make a doubly linked list and done all the things mention below. The node will have an int variable in the data part. Your LinkedList will have a head and a tail pointer.

## **Code**

#include <iostream>

#include <string>

using namespace std;

struct Node

{

int data;

Node\* next, \* prev;

};

class DoublyLinkedList

{

private:

Node\* head = NULL, \* tail = NULL;

public:

void insertNodeAtBeginning(int data);

void insertNodeInMiddle(int key, int data);

void insertNodeAtEnd(int data);

bool deleteFirstNode();

bool deleteNode(int key);

bool deleteLastNode();

bool Search(int key);

void Display();

};

void DoublyLinkedList::insertNodeAtBeginning(int data)

{

Node\* node = new Node();

node->data = data;

node->next = head;

if (head != NULL)

head->prev = node;

node->prev = NULL;

head = node;

}

void DoublyLinkedList::insertNodeInMiddle(int key, int data)

{

Node\* node = new Node();

node->data = data;

Node\* temp = head;

while (temp != NULL && temp->data != key)

temp = temp->next;

if (temp == NULL) // It means we are at the end;

{

printf("Unable insert based on key value...\n");

return;

}

node->next = temp->next;

if (temp->next != NULL)

temp->next->prev = node;

temp->next = node;

node->prev = temp;

}

void DoublyLinkedList::insertNodeAtEnd(int data)

{

Node\* node = new Node;

node->data = data;

if (head == NULL)

{

head = node;

return;

}

Node\* temp = head;

while (temp->next != NULL)

temp = temp->next;

temp->next = node;

node->next = NULL;

}

bool DoublyLinkedList::deleteFirstNode()

{

Node\* node = head;

if (node == NULL)

return false;

head = head->next;

delete node;

return true;

}

bool DoublyLinkedList::deleteNode(int key)

{

Node\* prev = NULL;

Node\* ptr = head;

while (ptr != NULL)

{

if (ptr->data == key)

{

if (ptr == head)

{

head = head->next;

delete ptr;

ptr = head;

return true;

}

else

{

prev->next = ptr->next;

delete ptr;

ptr = prev->next;

return true;

}

}

else

{

prev = ptr;

ptr = ptr->next;

return true;

}

}

return false;

}

bool DoublyLinkedList::deleteLastNode()

{

if (head == NULL)

return false;

if (head->next == NULL)

{

delete head;

return true;

}

Node\* temp = head;

while (temp->next->next != NULL)

temp = temp->next;

delete temp->next;

temp->next = NULL;

return true;

}

bool DoublyLinkedList::Search(int key)

{

Node\* ptr = head;

int iterations\_taken = 1;

bool flag = false;

while (ptr != NULL)

{

if (ptr->data == key)

{

printf("Value[%d] found in list...\n"

"Total Iterations Taken[%d]...\n", key, iterations\_taken);

flag = true;

break;

}

ptr = ptr->next;

iterations\_taken++;

}

return flag;

}

void DoublyLinkedList::Display()

{

Node\* ptr = head;

printf("[ ");

while (ptr != NULL)

{

printf("%d ", ptr->data);

ptr = ptr->next;

}

printf("]\n");

}

void InsertMenu()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t\t1. Insert At Head" << endl;

cout << "\t\t2. Insert On Key" << endl;

cout << "\t\t3. Insert At Tail" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void DeleteMenu()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t\t1. Delete At Head" << endl;

cout << "\t\t2. Delete On Key" << endl;

cout << "\t\t3. Delete At Tail" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void MainMenu()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t\t1. Insert" << endl;

cout << "\t\t2. Delete" << endl;

cout << "\t\t3. Search" << endl;

cout << "\t\t4. Display" << endl;

cout << "\t\t5. Exit" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

int main()

{

DoublyLinkedList\* doubly\_linked\_list = new DoublyLinkedList();

int choice, choice2;

int key, value;

do

{

MainMenu();

cout << "Enter your choice: ";

cin >> choice;

while (choice < 0 || choice > 5)

{

cout << "Invalid choice..." << endl;

cout << "Enter your choice b/w (1-5): ";

cin >> choice;

}

switch (choice)

{

case 1:

InsertMenu();

cout << "Enter your choice: ";

cin >> choice2;

while (choice2 < 0 || choice2 > 3)

{

cout << "Invalid choice..." << endl;

cout << "Enter your choice b/w (1-5): ";

cin >> choice2;

}

switch (choice2)

{

case 1:

cout << "Enter Value: ";

cin >> value;

doubly\_linked\_list->insertNodeAtBeginning(value);

break;

case 2:

cout << "Enter Value: ";

cin >> value;

cout << "Enter Key: ";

cin >> key;

doubly\_linked\_list->insertNodeInMiddle(key, value);

break;

case 3:

cout << "Enter Value: ";

cin >> value;

doubly\_linked\_list->insertNodeAtEnd(value);

break;

}

break;

case 2:

DeleteMenu();

cout << "Enter your choice: ";

cin >> choice2;

while (choice2 < 0 || choice2 > 3)

{

cout << "Invalid choice..." << endl;

cout << "Enter your choice b/w (1-5): ";

cin >> choice2;

}

switch (choice2)

{

case 1:

if (!doubly\_linked\_list->deleteFirstNode())

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

else

cout << "Element Removed from head" << endl;

break;

case 2:

cout << "Enter key to delete: ";

cin >> key;

if (!doubly\_linked\_list->deleteNode(key))

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

else

cout << "Element Removed from head" << endl;

break;

case 3:

if (!doubly\_linked\_list->deleteLastNode())

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

else

cout << "Element Removed from tail" << endl;

break;

}

break;

case 3:

cout << "Enter Key for searching: ";

cin >> key;

if (!doubly\_linked\_list->Search(key))

cout << "No Node found..." << endl;

break;

case 4:

doubly\_linked\_list->Display();

break;

default:

cout << "Program Terminated..." << endl;

break;

}

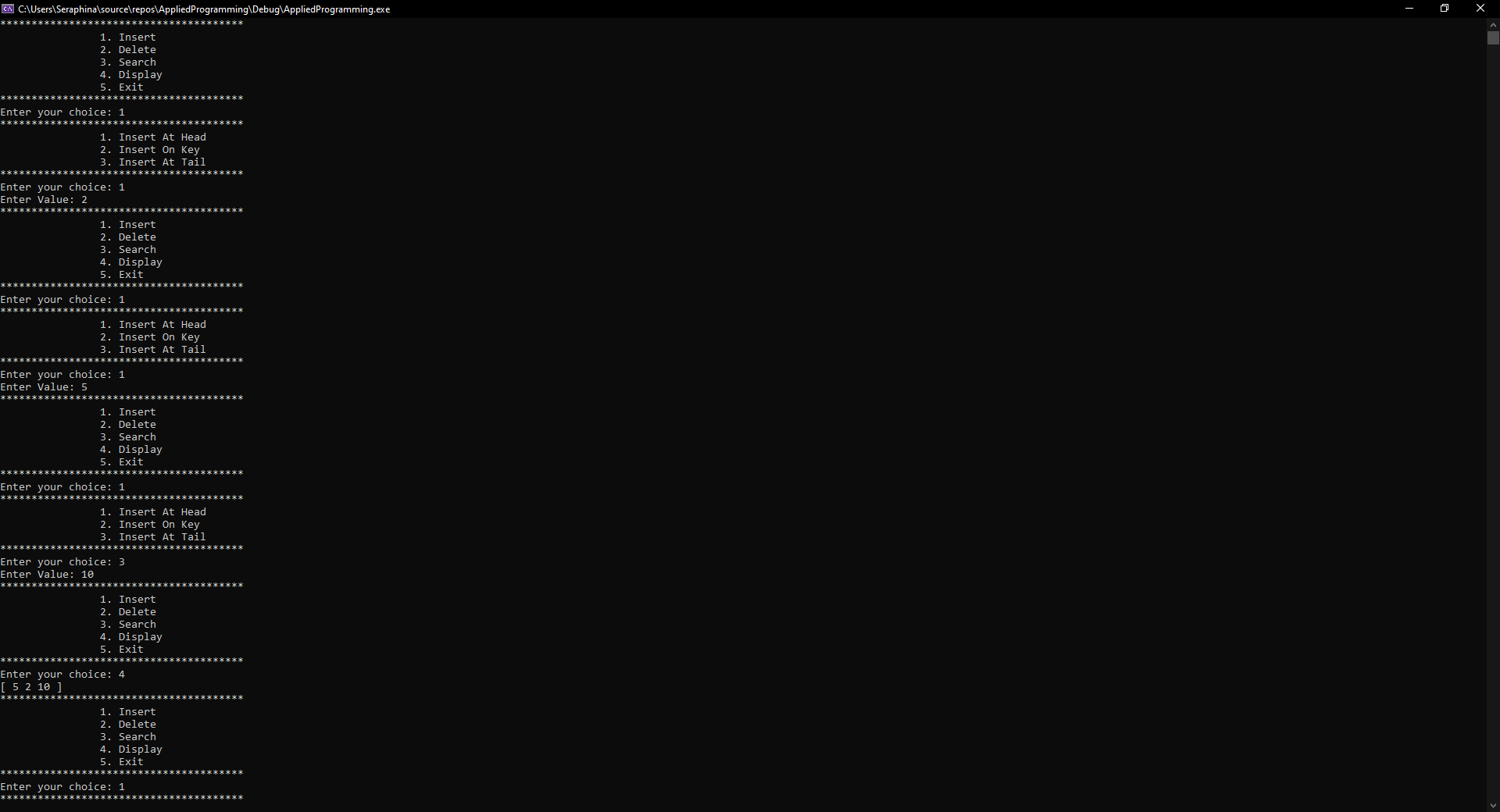
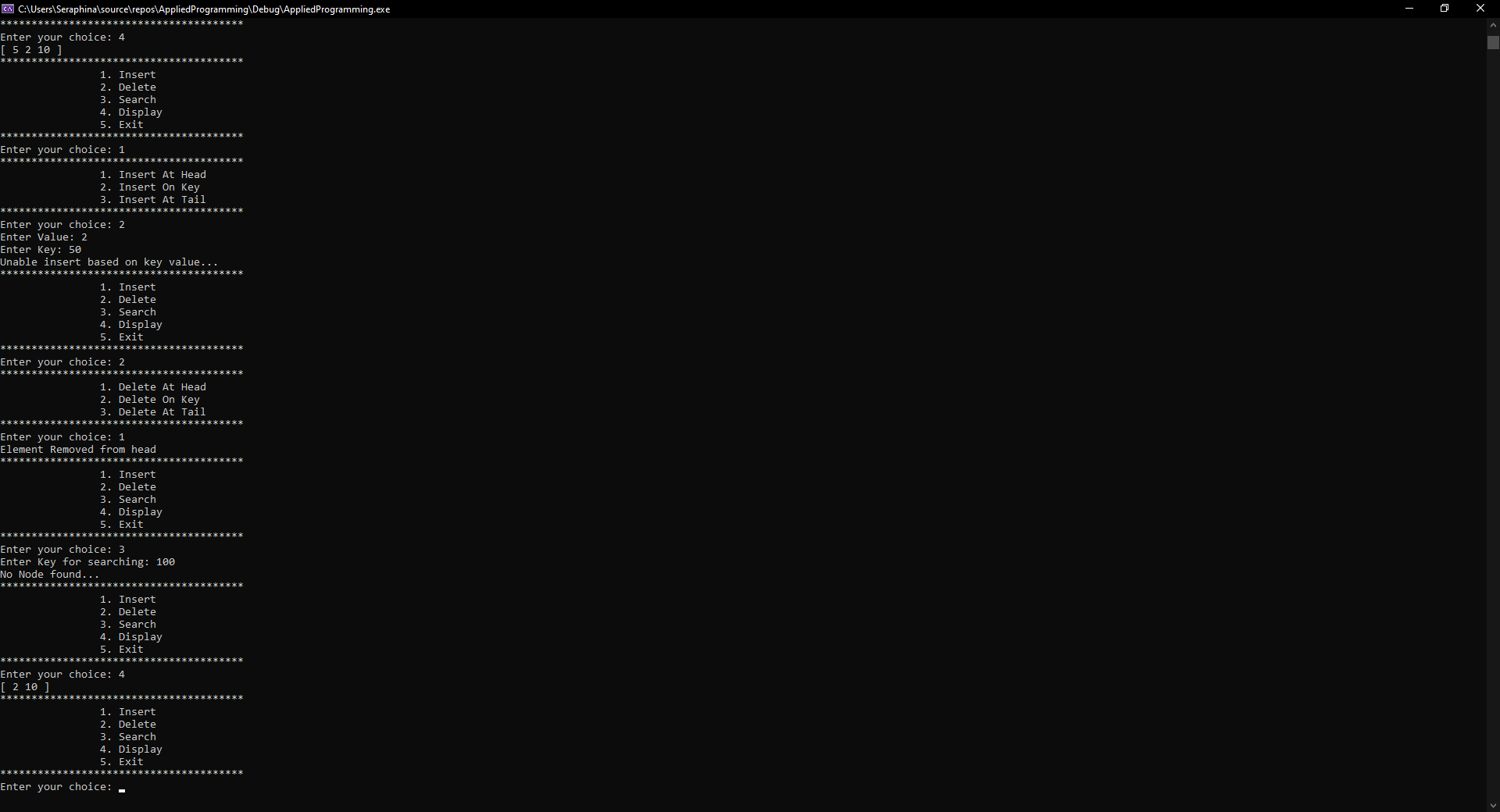
} while (choice != 5);

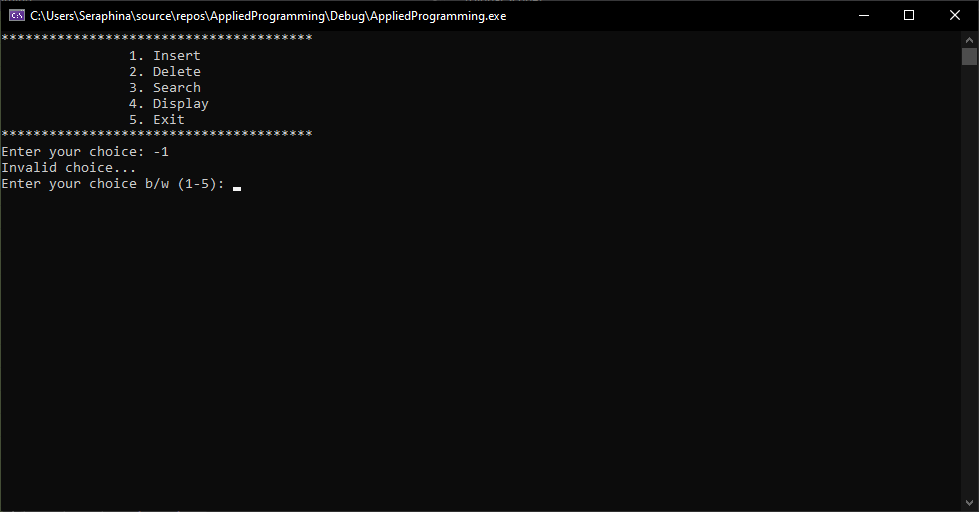
system("PAUSE");

return 0;

}

## **Screenshot**



# **Question#6**

Write a program that prompts the user to input a string and then outputs the string in the pigLatin form. Input the string in a doubly circular link-list. The rules for converting a string into pig Latin form are as follows: If the string begins with a vowel, add the string "-way" at the end of the string. For example, the pig Latin form of the string "eye" is "eye-way". If the string does not begin with a vowel, first add "-" at the end of the string. Then rotate the string one character at a time; that is, move the first character of the string to the end of the string until the first character of the string becomes a vowel. Then add the string "ay" at the end. For example, the pig Latin form of the string "There" is "ere-Thay Strings such as "by" contain no vowels. In cases like this, the letter y can be considered a vowel. So, for this program the vowels are a, e, i, o, u, y, A, E, I, O, U, and Y. Therefore, the pig Latin form of "by" is "y-bay". Strings such as "1234" contain no vowels. The pig Latin form of the string "1234" is"1234- way". That is, the pig Latin form of a string that has no vowels in it is the string followed by the string "-way".

## **Code**

#include <iostream>

#include <string>

using namespace std;

#define IS\_VOWEL(c) ( \

c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' || c == 'y' || \

c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U' || c == 'Y' )

struct Node

{

char data;

Node\* next, \* prev;

};

void Insert(Node\*& head, char value)

{

Node\* node = new Node();

node->data = value;

if (head == NULL)

{

node->next = node;

node->prev = node;

head = node;

}

else

{

Node\* last = head->prev;

node->next = head;

head->prev = node;

node->prev = last;

last->next = node;

}

}

void Insert(Node\*& head, string str)

{

for (int i = 0; i < str.length(); i++)

{

Insert(head, str[i]);

}

}

void DisplayList(Node\*& head)

{

Node\* ptr = head;

cout << "[ ";

if (ptr != NULL)

{

do

{

cout << ptr->data;

ptr = ptr->next;

} while (ptr != head);

}

cout << " ]" << endl;

}

void PigLatin(Node\*& head, string str)

{

Insert(head, str);

char c = head->data;

if (IS\_VOWEL(c))

{

Insert(head, "-way");

return;

}

Node\* ptr = head;

bool has\_vowel = false;

do

{

c = ptr->data;

if (IS\_VOWEL(c))

{

has\_vowel = true;

break;

}

if (has\_vowel)

break;

ptr = ptr->next;

} while (ptr != head);

if (!has\_vowel)

{

Insert(head, "-way");

return;

}

Insert(head, '-');

ptr = head;

do

{

c = ptr->data;

if (IS\_VOWEL(c))

{

head = ptr;

Insert(head, "ay");

return;

}

ptr = ptr->next;

} while (ptr != head);

}

int main()

{

Node\* node = NULL;

cout << "Running Test case for: eye\nResult: ";

PigLatin(node, "eye");

DisplayList(node);

cout << endl;

Node\* node2 = NULL;

cout << "Running Test case for: There\nResult: ";

PigLatin(node2, "There");

DisplayList(node2);

cout << endl;

Node\* node3 = NULL;

cout << "Running Test case for: 1234\nResult: ";

PigLatin(node3, "1234");

DisplayList(node3);

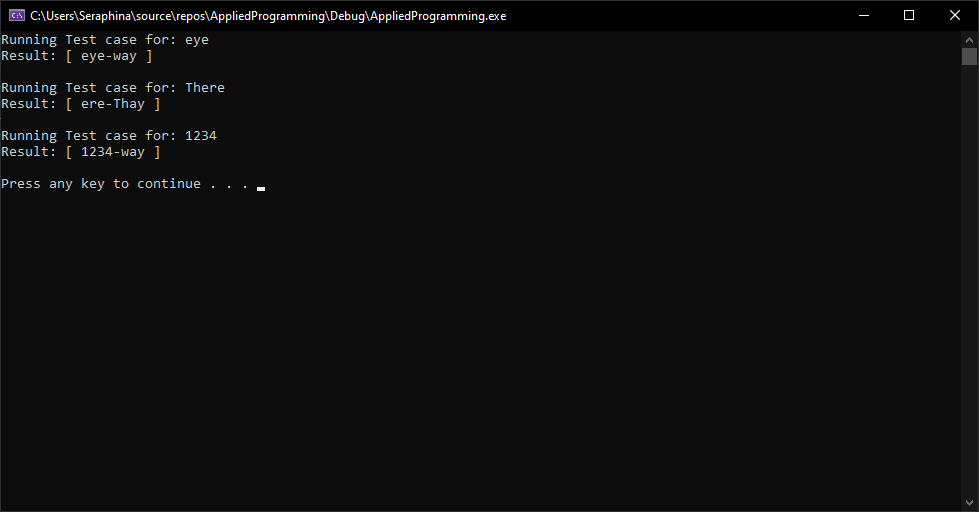
cout << endl;

system("PAUSE");

return 0;

}

## **Screenshot**



# **Question#7**

Round-Robin algorithm is an algorithm used in CPU Scheduling. In this algorithm, all processes take turns and run for a specific interval, until they are all completely executed. You are required to make a process that will consist of two things,

## **Code**

#include <iostream>

#include <string>

using namespace std;

struct Node

{

int ProcessorID, Executiontime;

Node\* next;

};

class RoundRobin

{

private:

Node\* head = NULL;

public:

void CreateProcess(int ProcessorID, int Executiontime);

void DeleteProcess(int ProcessorID);

void CurrentRunningProcess();

void Process(int TimeSlice);

};

void RoundRobin::CreateProcess(int ProcessorID, int Executiontime)

{

Node\* node = new Node;

node->ProcessorID = ProcessorID;

node->Executiontime = Executiontime;

if (head == NULL)

{

head = node;

head->next = head;

}

else

{

Node\* ptr = head;

while (ptr->next != head)

{

ptr = ptr->next;

}

ptr->next = node;

node->next = head;

}

}

void RoundRobin::DeleteProcess(int ProcessorID)

{

if (head == NULL)

return;

if (head->ProcessorID == ProcessorID && head->next == head)

{

delete head;

head = NULL;

return;

}

Node\* last = head, \* temp;

if (head->ProcessorID == ProcessorID)

{

while (last->next != head)

last = last->next;

last->next = head->next;

delete head;

head = last->next;

return;

}

while (last->next != head && last->next->ProcessorID != ProcessorID)

last = last->next;

if (last->next->ProcessorID == ProcessorID)

{

temp = last->next;

last->next = temp->next;

delete temp;

}

}

void RoundRobin::CurrentRunningProcess()

{

Node\* ptr = head;

cout << "Processes in Linked List: ";

do

{

cout << ptr->ProcessorID << " ";

ptr = ptr->next;

} while (ptr != head);

}

void RoundRobin::Process(int TimeSlice)

{

int ExecutionTime = 0;

Node\* ptr = head;

while (head != NULL)

{

do

{

CurrentRunningProcess();

if (ptr->Executiontime <= TimeSlice && (ptr->Executiontime - TimeSlice <= 0))

{

cout << "Process #" << ptr->ProcessorID << " executed for " << ptr->Executiontime << " seconds" << endl;

ExecutionTime += ptr->Executiontime;

cout << "Process #" << ptr->ProcessorID << " has been executed. Time Taken: " << ExecutionTime << endl;

DeleteProcess(ptr->ProcessorID);

ptr = head;

continue;

}

else

{

cout << "Process #" << ptr->ProcessorID << " executed for " << TimeSlice << " seconds" << endl;

ptr->Executiontime -= TimeSlice;

ExecutionTime += TimeSlice;

}

ptr = ptr->next;

} while (ptr != head);

}

}

int main()

{

RoundRobin round\_robin;

int TotalProcessors, ExecutionTime;

cout << "Enter number of processes: ";

cin >> TotalProcessors;

for (int i = 1; i <= TotalProcessors; i++)

{

cout << "Enter Process #" << i << " Execution Time: ";

cin >> ExecutionTime;

round\_robin.CreateProcess(i, ExecutionTime);

}

int TimeSlice;

cout << "Enter Time Slice: ";

cin >> TimeSlice;

round\_robin.Process(TimeSlice);

cout << "Execution Completed" << endl;

system("PAUSE");

return 0;

}

## **Screenshot**

