TASK 1

Write a Program to Enter 5 nodes in circular linked list using insert at start. Now delete first and   
last node.

#include <iostream>

#include <string.h>

#pragma warning(disable : 4996)

using namespace std;

class node {

public:

node\* next;

int roll;

};

node\* head = NULL;

class list {

public:

node\* create\_node() {

node\* newnode = new node;

return newnode;

}

node\* last\_node() {

node\* temp = head;

while (temp->next != head) {

temp = temp->next;

}

node\* last = temp;

return last;

}

void read\_node(node\* newnode) {

cout << "ENTER THE ROLL NO : "; cin >> newnode->roll; cout << endl;

}

void insert\_at\_head(node\* newnode) {

node\* last;

if (head == NULL) {

head = newnode;

last = newnode;

newnode->next = last;

last->next = head;

}

else {

last = last\_node();

newnode->next = head;

last->next = newnode;

head = newnode;

}

}

void delete\_at\_head() {

node\* last = last\_node();

if (head == NULL) {

cout << "NOTHING TO DELETE" << endl;

}

else {

node\* temp = head;

head = head->next;

last->next = head;

delete(temp);

}

}

void delete\_at\_end() {

node\* last = last\_node();

node\* temp = head;

if (head == NULL) {

cout << "NOTHING TO DELETE" << endl;

}

else {

while (temp->next != last) {

temp = temp->next;

}

node\* sec\_last = temp;

node\* temporary = last;

sec\_last->next = head;

last = sec\_last;

delete(temporary);

}

}

void display() {

node\* last = last\_node();

if (head == NULL) {

cout << "LIST IS EMPTY" << endl;

}

else {

node\* temp = head;

while (temp != last) {

cout << temp->roll << endl;

temp = temp->next;

}

cout << last->roll << endl;

}

}

};

int main() {

list l;

int opt;

while (1) {

cout << "ENTER 1 TO INSERT [AT HEAD]" << endl;

cout << "ENTER 2 TO DELETE AT HEAD" << endl;

cout << "ENTER 3 TO DELETE AT END" << endl;

cout << "ENTER 9 TO DISPLAY" << endl;

cout << "ENTER 10 TO EXIT" << endl;

cin >> opt; cout << endl;

switch (opt) {

case 1:

{

node\* newnode = l.create\_node();

l.read\_node(newnode);

l.insert\_at\_head(newnode);

break;

}

case 2:

{

l.delete\_at\_head();

break;

}

case 3:

{

l.delete\_at\_end();

break;

}

case 9:

{

l.display();

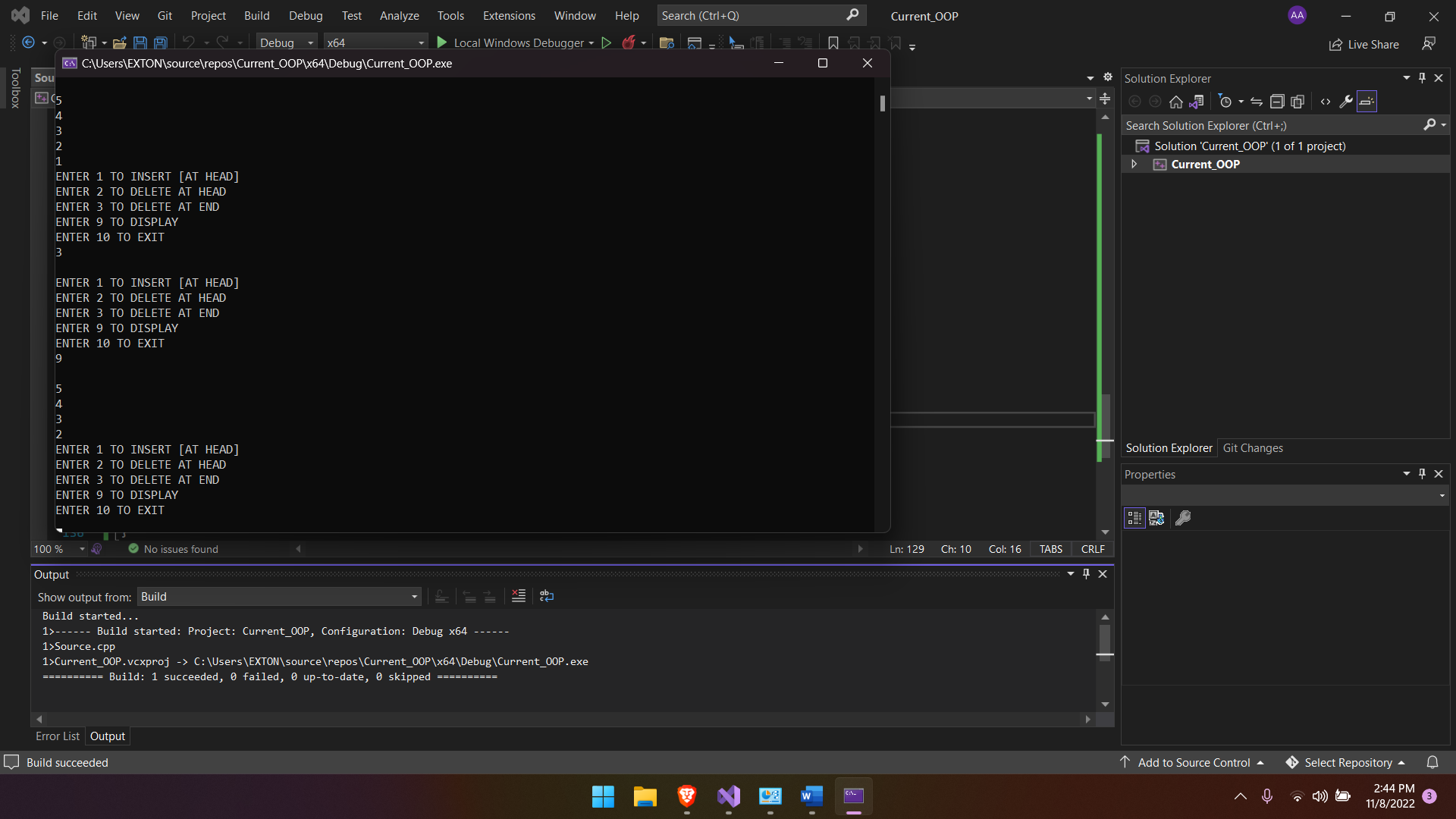
break;

}

}

}

}



TASK 2  
Part A:  
Write a Program to Enter 5 nodes in circular doubly linked list using insert at start and 5 nodes in   
circular doubly linked list using insert at last and 5 node using insert at specific point in circular   
doubly linked list. Do these tasks using 3 functions.  
Part B:  
Display above circular doubly linked list.

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node\* next;

Node\* prev;

Node()

{

data = 0;

next = NULL;

prev = NULL;

}

};

class CDL

{

public:

Node\* head;

Node\* last;

CDL()

{

head = NULL;

last = NULL;

}

Node\* create\_node() {

Node\* newnode = new Node;

return newnode;

}

void read\_data(Node\* newnode) {

cout << "ENTER THE ROLL : "; cin >> newnode->data; cout << endl;

}

void insertAtLast(Node\* newnode)

{

if (head == NULL)

{

head = newnode;

head->prev = newnode;

head->next = newnode;

last = newnode;

}

else if (head->next == head)

{

newnode->next = head;

head->next = newnode;

head->prev = newnode;

last = newnode;

}

else

{

last->next = newnode;

newnode->next = head;

head->prev = newnode;

last = newnode;

}

}

void displayList()

{

if (head == NULL)

{

cout << "NOTHING TO DISPLAY" << endl;

}

else if (head->next == head)

{

cout << head->data << endl;

}

else

{

Node\* temp = head;

do

{

cout << temp->data << " ";

temp = temp->next;

} while (temp != head);

}

}

void insertAtHead(Node\* newnode)

{

if (head == NULL)

{

head = newnode;

newnode->next = head;

newnode->prev = head;

last = head;

}

else

{

newnode->next = head;

head->prev = newnode;

head = newnode;

last->next = head;

}

}

void insertAtPoint(int value)

{

Node\* temp = head;

while(temp->next!=head) {

if (value == temp->data) {

Node\* newnode = create\_node();

read\_data(newnode);

Node\* temp\_next = temp->next;

temp->next = newnode;

newnode->prev = temp;

newnode->next = temp\_next;

temp\_next->prev = newnode;

temp = temp->next;

}

else

{

temp = temp->next;

}

}

}

};

int main()

{

CDL l1;

int opt;

while (1) {

cout << endl;

cout << "ENTER 1 TO INPUT AT HEAD" << endl;

cout << "ENTER 2 TO INPUT AT END" << endl;

cout << "ENTER 3 TO INPUT AT POINT" << endl;

cout << "ENTER 4 TO DISPLAY" << endl;

cin >> opt; system("CLS");

switch (opt) {

case 1:

{

Node\* newnode = l1.create\_node();

l1.read\_data(newnode);

l1.insertAtHead(newnode);

break;

}

case 2:

{

Node\* newnode = l1.create\_node();

l1.read\_data(newnode);

l1.insertAtLast(newnode);

break;

}

case 3:

{

int data\_inp;

cout << "ENTER THE DATA AFTER WHICH YOU WANT TO INSERT A NODE : "; cin >> data\_inp; cout << endl;

l1.insertAtPoint(data\_inp);

}

case 4:

{

l1.displayList();

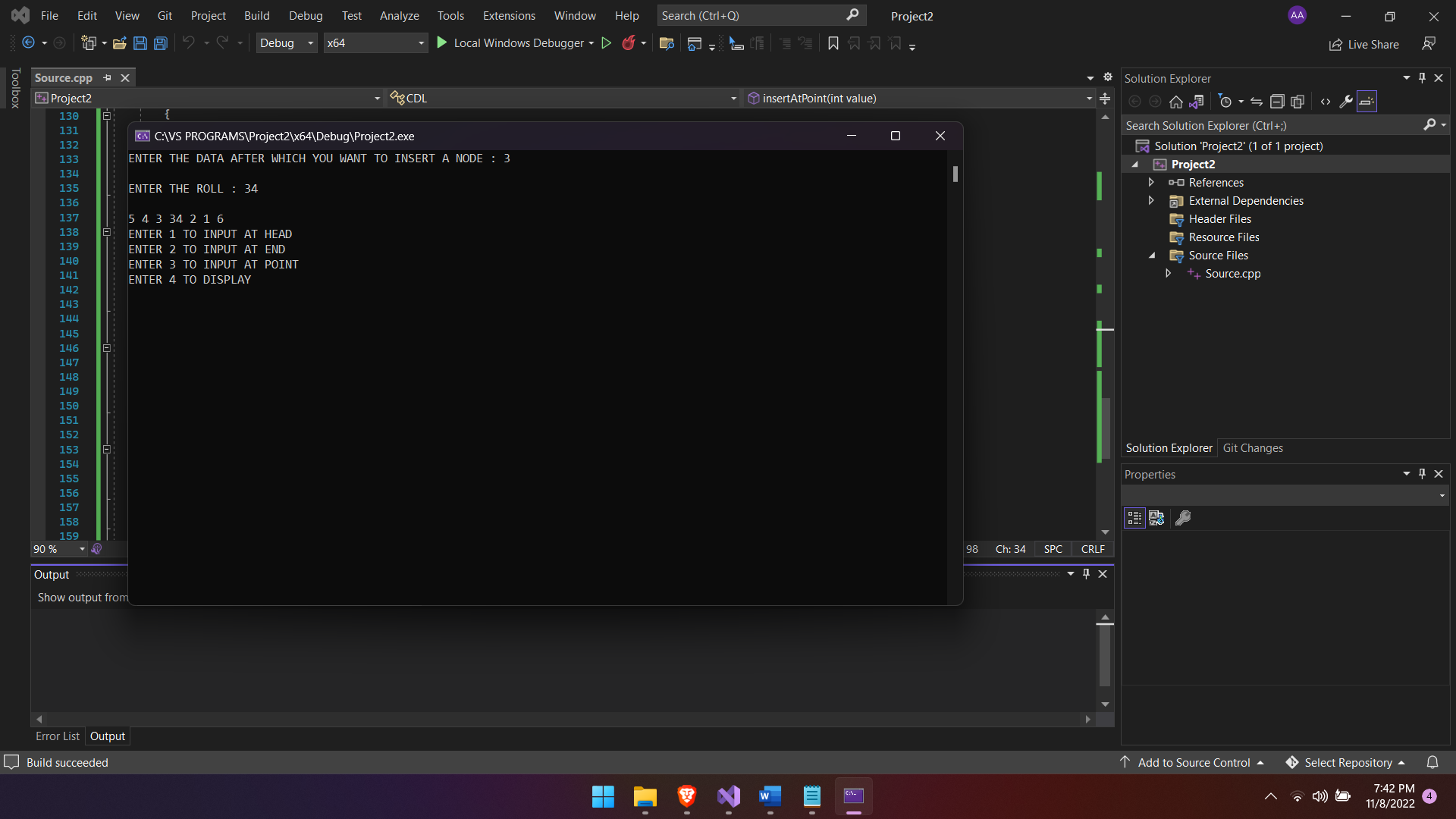
break;

}

}

}

}



Task 3  
Simulate a game, using two circular doubly linked list. The game involves a number of children,   
and you are supposed to read their names from a file. The children whose names are on lines   
numbered by prime numbers should be placed on the first list, and the others on the second list.   
Starting with the child whose name is on the line in the middle (or numberOfChildren/2) of the   
second list, children on that list are counted clockwise. Every mth child, where m is the number   
of elements in the first list is eliminated from that list. Counting goes on with the next child.   
Repeat this counting m times or till the second list gets empty. Your program should output the   
initial lists and the final second list.

#include <iostream>

#include <string.h>

#include <fstream>

//#pragma warning(disable : 4996)

using namespace std;

class node {

public:

node\* next;

node\* prev;

string name;

int roll;

};

node\* head = NULL;

node\* head1 = NULL;

node\* head2 = NULL;

class list {

fstream list;

fstream list\_1;

fstream list\_2;

public:

node\* create\_node() {

node\* newnode = new node;

return newnode;

}

node\* last\_node\_list1() {

node\* temp = head1;

while (temp->next != head1) {

temp = temp->next;

}

node\* last = temp;

return last;

}

node\* last\_node\_list2() {

node\* temp = head2;

while (temp->next != head2) {

temp = temp->next;

}

node\* last = temp;

return last;

}

void read\_node(node\* newnode) {

cout << "ENTER THE NAME : "; cin >> newnode->name; cout << endl;

cout << "ENTER THE ROLL NO : "; cin >> newnode->roll; cout << endl;

}

void insert\_at\_initial(node\* newnode) {

if (head == NULL) {

list.open("initial.txt", ios::app);

list << "NAME : " << "\t" << newnode->name << endl;

list << "ROLL : " << "\t" << newnode->roll << endl;

list.close();

head = newnode;

node\* last = newnode;

head->next = last;

last->next = head;

head->prev = last;

last->prev = head;

}

else {

list.open("initial.txt", ios::app);

list << "NAME : " << "\t" << newnode->name << endl;

list << "ROLL : " << "\t" << newnode->roll << endl;

list.close();

newnode->next = head;

head->prev = newnode;

head = newnode;

}

}

void insert\_at\_head1(node\* newnode) {

fstream list\_1;

//

if (head1 == NULL) {

list\_1.open("f\_first.txt", ios::app);

list\_1 << "NAME : " << "\t" << newnode->name << endl;

list\_1 << "ROLL : " << "\t" << newnode->roll << endl;

list\_1.close();

head1 = newnode;

node\* last = newnode;

head1->next = last;

last->next = head1;

head1->prev = last;

last->prev = head1;

}

else {

list\_1.open("f\_first.txt", ios::app);

list\_1 << "NAME : " << "\t" << newnode->name << endl;

list\_1 << "ROLL : " << "\t" << newnode->roll << endl;

list\_1.close();

node\* last\_node1 = last\_node\_list1();///88/

newnode->next = head1;

head1->prev = newnode;

head1 = newnode;

head1->prev = last\_node1;

last\_node1->next = head1;

}

}

void insert\_at\_head2(node\* newnode) {

if (head2 == NULL) {

list\_2.open("f\_second.txt", ios::app);

list\_2 << "NAME : " << "\t" << newnode->name << endl;

list\_2 << "ROLL : " << "\t" << newnode->roll << endl;

list\_2.close();

//

head2 = newnode;

node\* last = newnode;

head2->next = last;

last->next = head2;

head2->prev = last;

last->prev = head2;

}

else {

node\* last\_node2 = last\_node\_list2();

list\_2.open("f\_second.txt", ios::app);

list\_2 << "NAME : " << "\t" << newnode->name << endl;

list\_2 << "ROLL : " << "\t" << newnode->roll << endl;

list\_2.close();

//

newnode->next = head2;

head2->prev = newnode;

head2 = newnode;

head2->prev = last\_node2;

last\_node2->next = head2;

}

}

void check\_prime() {

}

void display\_list1() {

node\* last = last\_node\_list1();

if (head1 == NULL) {

cout << "LIST IS EMPTY" << endl;

}

else {

node\* temp = head1;

while (temp != last) {

cout << "NAME : " << temp->name << endl;

cout << "ROLL NO : " << temp->roll << endl;

temp = temp->next;

}

cout << "NAME : " << last->name << endl;

cout << "ROLL : " << last->roll << endl;

}

}

void del\_m\_list\_2() { //non

node\* temp = head1;

node\* last = last\_node\_list1();

int count = 0;

while (temp != last) {

count++;

temp = temp->next;

}

count++;

if (count % 2 == 0) {

cout << "EVEN NUMBER OF NODES" << endl;

node\* last = last\_node\_list1();

node\* last\_temp = last; //to be deleted

node\* last\_prev = last->prev;

last = last\_prev;

last->next = head1;

head1->prev = last;

delete(last\_temp);

}

else {

int cnt = 0;

count--;

int half = count / 2;

half++;

node\* head\_temp = head1;

while (head\_temp != last) {

cnt++;

if (cnt == half) {

node\* curr\_prev = head\_temp->prev;

node\* curr\_next = head\_temp->next;

node\* temp\_del = head\_temp;

curr\_prev->next = curr\_next;

curr\_next->prev = curr\_prev;

head\_temp = head\_temp->next;

}

else {

head\_temp = head\_temp->next;

}

}

}

}

void del\_m\_list\_1() {

node\* temp = head2;

node\* last = last\_node\_list2();

int count = 0;

while (temp != last) {

count++;

temp = temp->next;

}

count++;

if (count % 2 == 0) {

node\* temp\_last\_prev = last->prev;

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

cout << "EVEN NUMBER OF NODES" << endl;

node\* temp\_last = last;

temp\_last\_prev->next = head2;

head2->prev = temp\_last\_prev;

last = temp\_last\_prev;

delete(temp\_last);

}

else {

int cnt = 0;

count--;

int half = count / 2;

half++;

node\* head\_temp = head2;

while (head\_temp != last) {

cnt++;

if (cnt == half) {

node\* curr\_prev = head\_temp->prev;

node\* curr\_next = head\_temp->next;

node\* temp\_del = head\_temp;

curr\_prev->next = curr\_next;

curr\_next->prev = curr\_prev;

head\_temp = head\_temp->next;

}

else {

head\_temp = head\_temp->next;

}

}

}

}

void display\_list2() {

node\* last = last\_node\_list2();

if (head2 == NULL) {

cout << "LIST IS EMPTY" << endl;

}

else {

node\* temp = head2;

while (temp != last) {

cout << "NAME : " << temp->name << endl;

cout <<"ROLL NO : " << temp->roll << endl;

temp = temp->next;

}

cout << "NAME : " << last->name << endl;

cout << "ROLL : " << last->roll << endl;

}

}

};

int main() {

list l;

int opt;

while (1) {

cout << "ENTER 1 TO INSERT DATA OF CHILDREN" << endl;

cout << "ENTER 2 TO DELETE M/2th ELEMENT OF CHILDREN i.e non prime" << endl;

cout << "ENTER 3 TO DELETE M/2th ELEMENT OF CHILDREN i.e prime" << endl;

cout << "ENTER 9 TO DISPLAY [LIST 1] i.e non prime" << endl;

cout << "ENTER 10 TO DISPLAY [LIST 2] i.e prime" << endl;

cout << "ENTER 11 TO EXIT" << endl;

cin >> opt; cout << endl;

switch (opt) {

case 1:

{

node\* newnode = l.create\_node();

l.read\_node(newnode);

int j = 0;

//If ID is prime

for (int i = 2; i < newnode->roll; i++)

{

if ((newnode->roll % i) == 0)

{

j++;

}

}

if (j == 0) {

l.insert\_at\_initial(newnode);

l.insert\_at\_head2(newnode);

}

else {

l.insert\_at\_initial(newnode);

l.insert\_at\_head1(newnode);

}

break;

}

case 2:

{

l.del\_m\_list\_2(); //non

break;

}

case 3:

{

l.del\_m\_list\_1();

break;

}

case 9:

{

l.display\_list1();

break;

}

case 10:

{

l.display\_list2();

break;

}

}

}

}

