**ASSIGNMENT #1**

#include<iostream>

using namespace std;

class ListNode

{public:

int data;

ListNode\* next;

};

class LinkList

{public:

ListNode\* start;

ListNode\* last;

ListNode\* loc;

ListNode\* ploc;

LinkList()

{

start=NULL;

last = NULL;

}

bool isEmpty()

{

return (start == nullptr);

}

void insertatFront(int value)

{

ListNode\* newnode = new ListNode();

newnode->data = value;

if (isEmpty())

{

last = newnode;

start = newnode;

}

else

{

newnode->data = value;

newnode->next = start;

start = newnode;

}

}//For printing List in reverse order

**QUESTION # 1**

1. Write a function that prints all nodes of a linked list in the reverse order.

**CODE**

void printListInReverse()

{

ListNode\* temp;

ListNode\* ptemp;

ListNode\* xtemp=NULL;

//temp and ptemps are initialized

temp = start;

ptemp = NULL;

//Loop returns when whole list is accessed

do

{//stores the previous value

ptemp = temp;

//points to next value

temp = temp->next;

} while (temp != xtemp);

//xtemps stores the ptemp previous value where the loop breaks

xtemp = ptemp;

cout << xtemp->data;

//temp is again initialized to start

temp = start;

//loop continues till xtemp reaches to start

while (xtemp != start)

{//loop continues until the value reaches at which the loop breaks

while (temp != xtemp)

{

ptemp = temp;

temp = temp->next;

}

xtemp = ptemp;

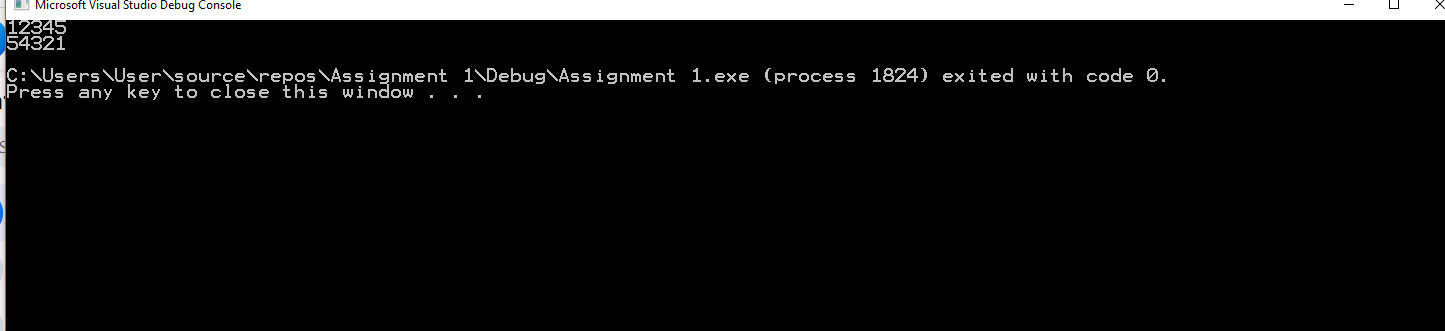
temp = start;

//displays the last accessed element

cout << xtemp->data;

}

}

**OUTPUT**

**QUESTION # 2**

Write a function which reverses the order of a linked list. If the original list is 1-> 2->3->4->5 -> 6, the updated list should be 6 -> 5 -> 4 -> 3 -> 2 ->1. Note that this function should only change the pointer field in all nodes of a list; you are not allowed to modify data field of any node.

**CODE**

void ReversingLinkList()

{

ListNode\* previous = NULL;

ListNode\* temp = start;

ListNode\* iteration = start;

ListNode\* store\_previous=NULL;

//loop continues until whole list is accessed

while (temp->next!=NULL)

{

previous = temp;

temp = temp->next;

}

//

if(temp==last)

{

start = temp;

}

//It will store till where the loop terminates

store\_previous = previous;

//temporary is pointed

temp->next = previous;

//temp is again pointed to start

temp =iteration;

//loop continues till previous element not equals start

while (store\_previous != iteration)

{

while (temp != store\_previous)

{

previous = temp;

temp = temp->next;

}

store\_previous = previous;

//the pointer is pointed to its previous element

//Pointer field changes

temp->next = previous;

temp = iteration;

}

//When loop breaks last equals temp

if (temp == iteration)

{ temp->next=NULL;

last = temp;

}

if (store\_previous !=NULL)

{

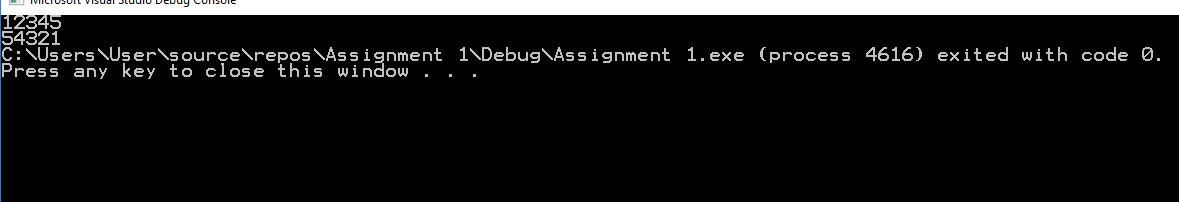
store\_previous->next=last->next;

last = store\_previous;

}

}

**OUTPUT**

 **QUESTION # 3**

1. Write a function that deletes all those nodes from a linked list which have odd numbered value in their data part. For instance, if the original list is 1-> 2->3->4->5 -> 6, then, the updated list should be 2-> 4 -> 6.

void OddNumberDeletion()

{

ListNode\* temp;

ListNode\* xtemp=start;

ListNode\* ptemp=NULL;

temp = start;

//loop continues until whole list is accessed

while (temp != NULL)

{//checks wether a element is even or odd

if (((temp->data) % 2) == 0)

{

ptemp = temp;

temp = temp->next;

//if even pointer points to next field

}

else

{//xtemp is pointing to the start

xtemp = temp;

if (xtemp != NULL)

{

if (ptemp != NULL)

{

//checks if element is last or not

if (temp->next == NULL)

{

ptemp->next = NULL;

last = ptemp;

}

else

{//otherwise prtevious points to next of odd number

ptemp->next = temp->next;

}

}

else

{

start = temp->next;

}

temp = start;

//deletes xtemp in case of odd number

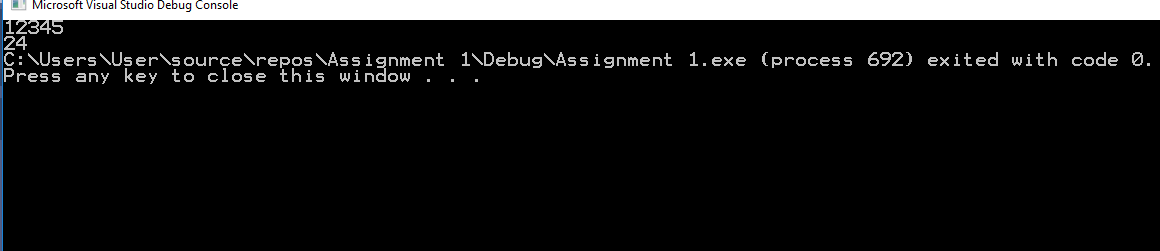
delete xtemp;

}

}

}

printLinkList();

 }

**OUTPUT**

**QUESTION 4**

void saperate\_even()

{

ListNode\* temp = start;

ListNode\* ptemp\_odd=NULL;

ListNode\* ptemp\_even=NULL;

ListNode\* temp\_odd=NULL;

ListNode\* odd\_1 = NULL;

ListNode\* temp\_even = NULL;

ListNode \* even\_1=NULL;

//Loop continues until whole list is accessed

while (temp != NULL)

{//checks if value is even

if (((temp->data) % 2) == 0)

{//if previously even element is found

if (ptemp\_even!=NULL)

{//initialized to current pointer

ptemp\_even = temp\_even;

ptemp\_even->next = temp;

}

else

{//if previously no even element is found

//to store 1st even of list

even\_1 = temp;

ptemp\_even = temp;

}

//current is stored

temp\_even = temp;

//points to next

temp = temp->next;

}

//checks if value is odd

else

{//if previously odd element is found

if (ptemp\_odd!=NULL)

{

//initialized to current pointer

ptemp\_odd = temp\_odd;

ptemp\_odd->next = temp;

}

else

{//if previously no odd element is found

//to store 1st odd of list

odd\_1 = temp;

ptemp\_odd = temp;

}

temp\_odd = temp;

//points to next

temp = temp->next;

}

}

//for connection of odd to even if 1st element is odd

if (((start->data)%2)!=0)

{

if ((temp\_odd != NULL) && (even\_1 != NULL))

{

temp\_odd->next = even\_1;

temp\_even->next = NULL;

last = temp\_even;

}

} //for connection of odd to even if 1st element is even

else

{

if ((temp\_even != NULL) && (odd\_1 != NULL))

{

temp\_even->next =odd\_1;

temp\_odd->next =NULL;

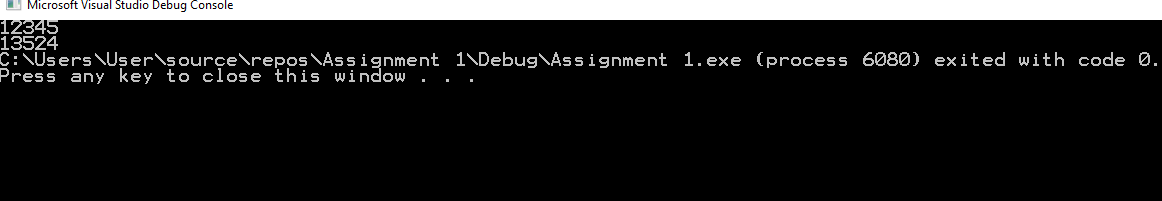
last = temp\_odd;

}

}

printLinkList();

}

**OUTPUT**

**QUESTION 5**

Write a function which takes two values as input from the user and searches them in the list. If both the values are found, your task is to swap both the nodes in which these values are found. Note, that you are not supposed to swap values

**CODE**

void swapping\_values(int value1, int value2)

{

ListNode\* ploc1;

ListNode\* loc1;

ListNode\* ploc2;

ListNode\* loc2;

ListNode\* temp = start;

//Searches value 1 and 2 in list respectively

Search(value1);

//loc and ploc for values are stored in variables like ploc1,loc1 and same for value2

ploc1 = ploc;

loc1 = loc;

//Returns if one of the value is not present in list

if (loc1 == NULL)

{

cout << value1 << "is not found";

return;

}

Search(value2);

ploc2 = ploc;

loc2 = loc;

if (loc2 == NULL)

{

cout << value2 << "is not found";

return;

}

//if user enters same values to be swapped method retuens

if ((ploc1 == NULL) && (ploc2 == NULL))

{

cout << "both are same values not allowed in linklist";

return;

}

//if one is head bode and the other is not last

if ((ploc1==NULL) && (loc2 != last))

{

//if both value are not consecutive

if (loc1->next != loc2)

{

//swaps the pointers

temp = loc2->next;

loc2->next = loc1->next;

loc1->next = temp;

//start is assigned to the first of list after swapping

start = loc2;

// loc1 is pointed by previous of loc2

ploc2->next = loc1;

}

//if both value are consecutive

else

{ //temp stores next to value2

temp = loc2->next;

//start is assigned to the first of list after swapping

start = loc2;

//swaps the pointers

loc2->next = loc1;

loc1->next = temp;

}

}

//if none is head bode and none is last

if ((ploc1 != NULL) && (loc2 != last))

{//if both value are not consecutive

if (loc1->next!= loc2)

{

//swaps the pointers

temp = loc2->next;

loc2->next = loc1->next;

loc1->next = temp;

if (ploc1 != NULL)

{//connections for rest of list

ploc1->next = loc2;

ploc2->next = loc1;

}

}

//if both value are consecutive

else

{//swaps the pointers

temp = loc2->next;

loc2->next = loc1;

loc1->next = temp;

ploc1->next = loc2;

}

}

//if none is head bode and one is last

if ((ploc1!=NULL) && (loc2->next==NULL))

{

//if both value are not consecutive

if (loc1->next!=loc2)

{

temp = loc1->next;

ploc1->next = loc2;

loc2->next = loc1->next;

ploc2->next = loc1;

//for making value1 the last of list

loc1->next = NULL;

//updates last pointer

last = loc1;

}

//if both value are not consecutive

else

{

loc2->next = loc1;

ploc1->next = loc2;

//for making value1 the last of list

loc1->next = NULL;

//updates last pointer

last = loc1;

printLinkList();

}

}

}

};

**OUTPUT**

