**Assignment 1: Implement Various Operations of Singly Linked Lists**

**Data Structures & Algorithms**

**Section:** BSCS-10-A&B **Session:** Fall-2021

**Instructor:** Dr. Yasir Faheem **Mapped to**: CLO-1

**start**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  | 2 |  |  | 3 |  |  | 4 |  |  | 5 |  |  | 6 |  |  | 7 | / |

1. Implement a function that prints all nodes of a linked list in the reverse order. A call to the above function should print 7, 6, 5, 4, 3, 2, 1. Note that you are not allowed to copy the contents of the list in any other structure.

int main()

{

linkedlisk \*list = new linkedlisk;

list->insertnode(2);

list->insertnode(5);

list->insertnode(8);

list->insertnode(13);

list->insertnode(17);

list->printline(list->start);

}

#include<iostream>

using namespace std;

class linknode

{

public:

int data;

linknode\* next;

};

class linkedlisk

{

public:

linknode\* start;// starting node of list

linknode\* last;// last node of list

linknode\* cptr;// current node ptr

linknode\* pptr;// previous node ptr

linknode\* temp\_ = new linknode;

int length = 0;

//---------------------------------------------------------------------------------------------------

linkedlisk()

{

start = NULL;

last = NULL;

cptr = NULL;

pptr = NULL;

}

bool isempty()

{

return start == NULL;

}

void insertnode(int value)

{

linknode\* newnode = new linknode;

linknode\* temp = new linknode;

newnode->data = value;

temp = start;

if (isempty())

{

start = newnode;

last = newnode;

}

else

{

while (temp != last)

temp = temp->next;

temp->next = newnode;

last = newnode;

last->next = temp\_;

}

length++;

}

void printline(linknode\* temp)

{

int x = 0;

if (temp == NULL)

return;

if (temp->next != temp\_)

printline(temp->next);

cout << temp->data << endl;

}

};

Output Screenshot:

Text

Description automatically generated

1. Implement 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 anynode.

void reverse()

{

linknode\* temp = new linknode;

linknode\* temp\_ = new linknode;

temp\_->data = 0;

temp = start;

cptr = start;

pptr = temp\_;

last->next = temp\_;

int c = 0;

while (c != (length))

{

temp = temp->next;

cptr->next = pptr;

pptr = cptr;

cptr = temp;

c++;

}

temp = last;

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

{

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

temp = temp->next;

}

}

int main()

{

linkedlisk\* list = new linkedlisk;

cout << " Original List" << endl;

list->insertnode(2);

list->insertnode(5);

list->insertnode(8);

list->insertnode(13);

list->insertnode(17);

list->printline();

cout << " Reversed List" << endl;

list->reverse();

}

Output:

Text

Description automatically generated

1. Implement 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->5->6 -> 7, then, the updated list should be 2 -> 6.

**Note:** Include all checks for various input cases like: a) all elements are odd numbers 1->3->5, b) only the last element is odd numbered 2->4->5, c) the list has only value which is odd e.g. 1, d) both first and last values are odd 1->2->4->5 etc.

void delete\_oddnode()

{

linknode\* temp = new linknode;

linknode\* temp\_\_ = new linknode;

temp->data = 0;

cptr = start;

pptr = temp;

int c = 0;

if (!isempty())

{

while (c <= length+1)

{

if ((cptr->data) % 2 == 1)//means it is odd number in current node

{

if (cptr == start)

{

start = cptr->next;

delete cptr;

cptr = start;

}

else if (cptr == last)

{

pptr->next = temp;

delete cptr;

last = pptr;

}

else

{

pptr->next = cptr->next;

delete cptr;

cptr = pptr->next;

}

}

else

{

pptr = cptr;

cptr = cptr->next;

}

c++;

length--;

}

linknode\* temp\_ = new linknode;

last->next = temp\_;

linknode\* temp = new linknode;

temp = start;

for (int i = 0; i != length+1; i++)

{

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

temp = temp->next;

}

}

else

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

}

};

Output:  


Text

Description automatically generatedText

Description automatically generatedText

Description automatically generated

1. Write code of a function which rearranges the linked list by separately grouping nodes having **even numbered** and **odd numbered value** in their data part. If the input list is 1-> 2->3->4->5 -> 6, the updated list may look like 1-> 3 -> 5->2-> 4-> 6 or 2-> 4-> 6 -> 1-> 3 -> 5.

void seperation\_evenodd()

{

int x = 0, y = 0;

linknode\* temp = new linknode;

linknode\* temp1 = new linknode;

linknode\* newnode = new linknode;

linknode\* newnode1 = new linknode;

int\* temp\_ = new int[length];

newnode->next = newnode1;

temp = start;

temp1 = start;

x = 0;

while (x != length)

{

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

{

pptr = temp;

temp = temp->next;

}

else

{

temp\_[y] = temp->data;

pptr->next = temp->next;

delete temp;

temp = pptr->next;

y++;

}

x++;

}

x = 0;

pptr->next = newnode;

temp = newnode;

//cout << temp->data << endl;

while (x < y)

{

temp->data = temp\_[x];

temp = temp->next;

x++;

}

temp = start;

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

{

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

temp = temp->next;

}

}

Text

Description automatically generated Text

Description automatically generated

1. Implement a function that rearranges a linked list by separating the **odd positioned** nodes and even **positioned nodes**. Moreover, it should also reverse the sub-list of the even positioned nodes and then append it at the end of odd positioned nodes sub-list. If the original list is 2->4->5->7->8->9 , the updated list should be 2->5->8 ->9->7->4.

Output:

void seperation\_reverse()

{

linknode\* temp = new linknode;

linknode\* temp\_ = new linknode;

linknode\* odd = new linknode;

linknode\* even = new linknode;

linknode\* ptr = new linknode;

linknode\* c = new linknode;

ptr = start;

temp = ptr->next;

even = temp;

int x = 2;

int j = 1;

odd = c;

while (x != length)

{

if ((x % 2) == 0)

{

odd->next = temp->next;

odd = temp->next;

temp\_ = odd;

if (temp->next->next != last->next)

{

temp->next = temp->next->next;

ptr->next = temp\_;

}

else

last = temp;

temp\_->next = even;

j++;

}

ptr = ptr->next;

temp = ptr->next;

if (temp == last->next)

{

odd->next = even;

}

x++;

}

last->next = odd->next;

odd->next = last;

c = start;

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

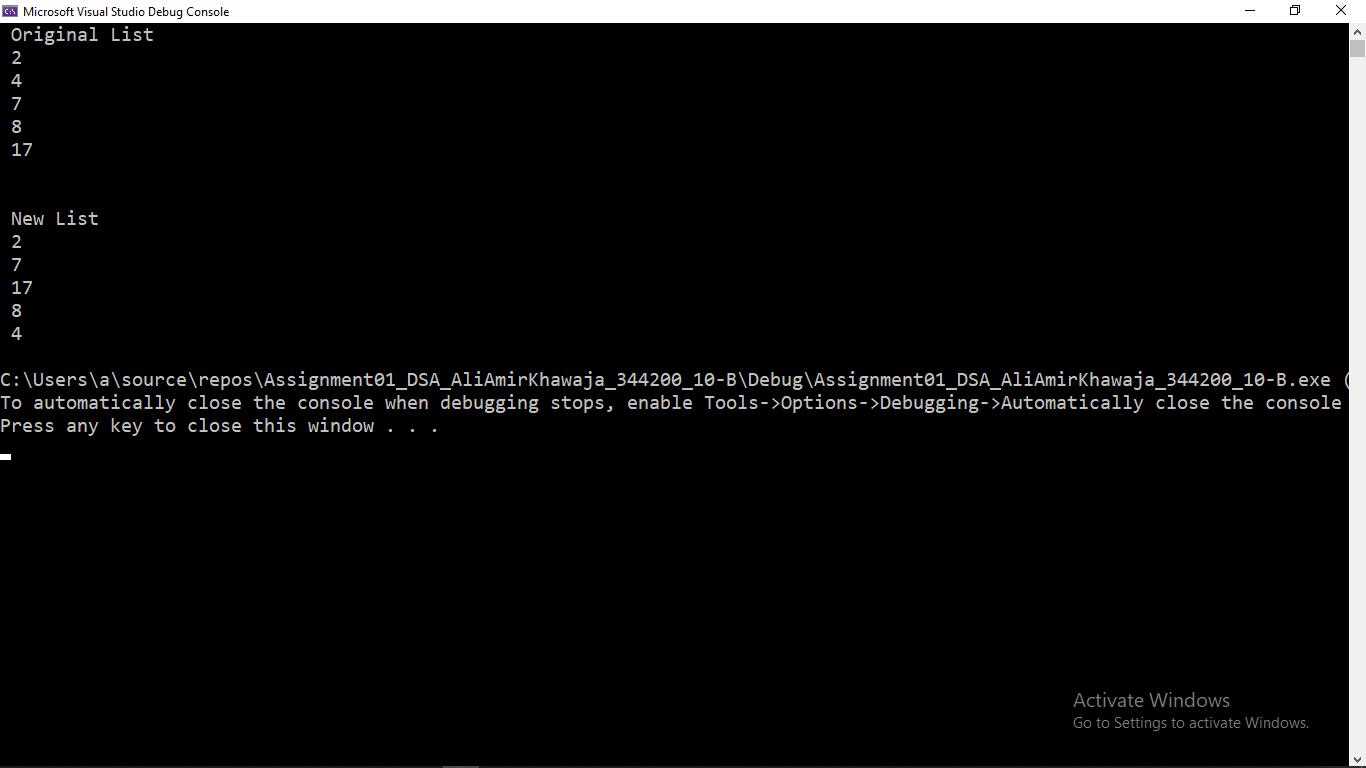
{

cout << " " << c->data << "->";

c = c->next;

}

}



1. Implement 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. You are supposed to change next pointer fields in the concerned nodes. Your function should handle all possible cases:
   * The two nodes are not adjacent and none of them is the first or the last node.
   * The two nodes are not adjacent to each other and one of them is the first/last node.
   * Both nodes are adjacent and none of them is the first or the last one.
   * Both nodes are adjacent and one of them is the first node.
   * Both are adjacent, one is first and the other is last. This is the case in which the length of a list is two.

void swap()

{

int x, y;

int c = 1;

linknode\* xctr = new linknode;

linknode\* yctr = new linknode;

linknode\* xptr = new linknode;

linknode\* yptr = new linknode;

linknode\* temp\_ = new linknode;

x = search(2);

y = search(17);

if (x != -1 && y != -1)

{

xctr = start;

yctr = start;

xptr->next = start;

yptr->next = start;

while (c != x)

{

xptr = xctr;

xctr = xctr->next;

c++;

}

c = 1;

while (c != y)

{

yptr = yctr;

yctr = yctr->next;

c++;

}

if (xctr != start && yctr != start && xctr != last && yctr != last)

{

xptr->next = yctr;

yptr->next = xctr;

temp\_ = yctr->next;

yctr->next = yptr;

xctr->next = temp\_;

}

if (xctr == start && yctr != last)

{

yptr->next = xctr;

temp\_ = yctr->next;

yctr->next = xctr->next;

xctr->next = temp\_;

start = yctr;

}

else if (xctr != start && yctr == last)

{

yptr->next = xctr;

temp\_ = yctr->next;

xptr->next = yctr;

yctr->next = xctr->next;

xctr->next = temp\_;

last = xctr;

}

else if (last == start->next)

{

temp\_ = start;

start = last;

start->next = temp\_;

}

linknode\* temp = new linknode;

temp = start;

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

{

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

temp = temp->next;

}

}

else

{

cout << " Not Found in List." << endl;

}

}

Search Values 4 and 8



Description automatically generated

Search Values 2 and 7

Text

Description automatically generated

Search Values 4 and 7

Text

Description automatically generated

Search Values 2 and 4

Text

Description automatically generated

Text

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidenceSearch Values 2 and 17 ( List has only 2 and 17 )

**Deliverables:**

* You should include screenshots of the outputs of all the implemented functions in this file.
* Moreover, you should upload your source code in a separate .cpp file.