|  |  |  |
| --- | --- | --- |
| American University of Sharjah  College of Engineering  Department of Computer Science & Engineering  P. O. Box 26666, Sharjah, UAE |  | **Instructors:** Dr. Michel Pasquier  **Lab Instructor:** Praveena Kolli  **Office:** EB2-126  **Phone**: 971-6-5152352  **e-mail**: pkolli@aus.edu  **Semester**: Spring 2021 |

**CMP 305 L – Data Structures and Algorithms**

**Lab. Assignment 4 – Doubly Linked List**

***Objectives:***

* To understand the implementation of doubly linked lists.
* To practice list processing and using doubly linked lists.

***Note:***

***Lab:*** Exercises 1 and 2 (10 marks)

***Bonus*:** Exercise 3 (1 mark)

**Exercise1:**

You are required to implement the following functions for *doubly linked lists.*

Note that code for creating a doubly linked list from an array of values is given to you, as well as testing code (driver program) to make sure that all functions work as expected.

1. find: Returns true if the given value is found in the list, and false otherwise.

template <typename Object>

bool findInDLL(DoubleNode<Object>\* head, Object value);

1. insertBefore: Finds a given value and inserts a new node before it with a new value. Returns true if the insertion is successful, else false.

template <typename Object>

bool insertBeforeDLL(DoubleNode<Object>\*& head, Object value, Object newValue);

1. erase: Finds a given value and deletes the matching node from the linked list. Returns true if the insertion is successful, else false.

template <typename Object>

bool eraseInDLL(DoubleNode<Object>\*& head, Object value)

Sample output:

Printing Double linked list:

1 2 4 7 6 8

Finding 7 in the List:

7 exist in the list

Inserting 3 before 4 :

1 2 3 4 7 6 8

Inserting 0 before 1 (inserting at the start of the list):

0 1 2 3 4 7 6 8

Deleting 3:

0 1 2 4 7 6 8

Deleting 9 (deleting last element in the list):

Value not Found!

Deleting 0 (deleting first element in the list):

1 2 4 7 6 8

Deleting -99 (trying to delete an element which is not present in the list):

Value not Found!

Code:

#include <iostream>

#include <string>

using namespace std;

//Doubly Linked List structure

template <typename Object>

struct DoubleNode

{

Object data;

DoubleNode \*prev;

DoubleNode \*next;

DoubleNode(const Object & d = Object{}, DoubleNode \* p = nullptr, DoubleNode \* n = nullptr)

: data{ d }, prev{ p }, next{ n } { }

};

//function to create Doubly Linked List with values

template <typename Object>

DoubleNode<Object>\* createDLL(Object ary[], int size)

{

DoubleNode<Object>\* first = new DoubleNode<Object>(ary[0]);

DoubleNode<Object>\* temp = first;

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

{

DoubleNode<Object>\* node = new DoubleNode<Object>(ary[i]);

temp->next = node;

temp = node; //same as single linked list

}

temp = first; //make temp point back to first element

DoubleNode<Object>\* x = first; //create instance x and make it point to first element

temp = temp->next; //shift temp to next node

while (temp != NULL)

{

temp->prev = x; //prev points to x

x = temp; //x points to temp

temp = temp->next; //temp points one step ahead

} //overall structure is therefore prev --> x --> temp(new position) and this loop stops when temp goes out of the array

return first;

}

template <typename Object>

void printDLL(DoubleNode<Object>\* head)

{

while (head != nullptr)

{

cout << head->data << "\t";

head = head->next;

}

cout << endl;

}

template <typename Object>

bool findInDLL(DoubleNode<Object>\* head, Object value)

{

while (head != nullptr) {

if (head->data == value) {

return true;

}

head = head->next;

}

return false;

}

template <typename Object>

bool insertBeforeDLL(DoubleNode<Object>\* & head, Object givenValue, Object newValue) {

DoubleNode<Object>\* newnode= new DoubleNode<Object>(newValue);

DoubleNode<Object>\* thisnode = head;

while (thisnode != nullptr) {

if (thisnode->data == givenValue) {

//Insert the node before it:

if (thisnode == head) {

newnode->next = thisnode;

newnode->prev = thisnode->prev;

head = newnode;

thisnode->prev = newnode;

return true;

}

else {

newnode->next = thisnode;

newnode->prev = thisnode->prev;

thisnode->prev->next = newnode;

thisnode->prev = newnode;

return true;

}

}

thisnode = thisnode->next;

}

return false;

}

template <typename Object>

bool eraseInDLL(DoubleNode<Object>\*& head, Object givenValue) {

DoubleNode<Object>\* thisnode = head;

while (thisnode != nullptr) {

if (thisnode->data == givenValue) {

if (thisnode == head) {

thisnode->next->prev = nullptr;

head = thisnode->next;

thisnode->next = nullptr;

delete thisnode;

return true;

}

else {

thisnode->next->prev = thisnode->prev;

thisnode->prev->next = thisnode->next;

delete thisnode;

return true;

}

}

thisnode = thisnode->next;

}

return false;

}

int main()

{

int ary[] = { 1,2,4,7,6,8 }, size = 6;

DoubleNode<int>\* head = createDLL<int>(ary, size);

cout << "Printing Double linked list: \n";

printDLL(head);

cout << "\nFinding 7 in the List: \n";

if (findInDLL(head, 7))

cout << "7 exist in the list\n";

//Testing insertBefore function

cout << "\nInserting 3 before 4 :\n";

bool success = insertBeforeDLL(head, 4, 3);

if (success)

printDLL(head);

else

cout << "Value not Found! \n";

cout << "\nInserting 0 before 1 (inserting at the start of the list):\n";

success = insertBeforeDLL(head, 1, 0);

if (success)

printDLL(head);

else

cout << "Value not Found! \n";

cout << "\nDeleting 3:\n ";

success = eraseInDLL(head, 3);

if (success)

printDLL(head);

else

cout << "Value not Found! \n";

cout << "\nDeleting 9 (deleting last element in the list):\n ";

success = eraseInDLL(head, 9);

if (success)

printDLL(head);

else

cout << "Value not Found! \n";

cout << "\nDeleting 0 (deleting first element in the list):\n ";

success = eraseInDLL(head, 0);

if (success)

printDLL(head);

else

cout << "Value not Found! \n";

cout << "\nDeleting -99 (trying to delete an element which is not present in the list):\n ";

success = eraseInDLL(head, -99);

if (success)

printDLL(head);

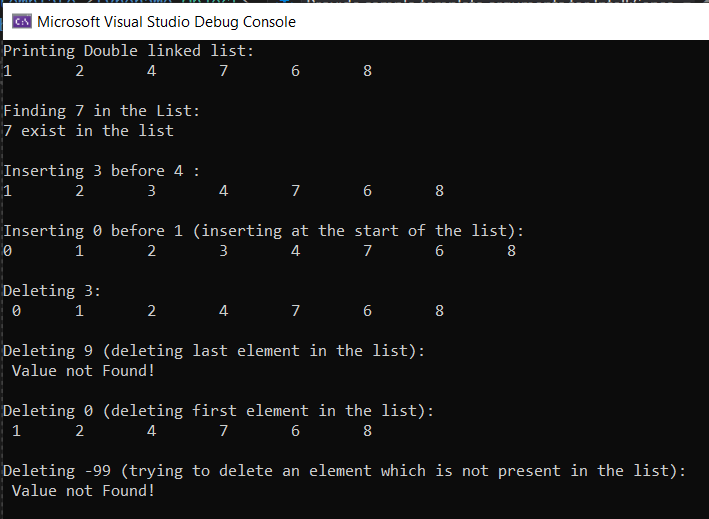
else

cout << "Value not Found! \n";

return 0;

}

Screenshot:



**Exercise 2:**

You are to code some simple music player application making appropriate use of doubly linked lists, as described hereafter.

Create a doubly linked list for a music playlist.

* 1. Provide a Song class that has a title and a singer as data members.
  2. Create a doubly linked list, with a head node (musicPlaylist) pointing to the first song on the list.
  3. Display a menu for the user to choose an option, as follows:
     + 1. Add a song
       2. Delete a song
       3. Play a song
       4. Skip forward
       5. Skip backward
       6. Exit

Adding and deleting songs (#1-2) should update the playlist accordingly. Adding a song places it at the end of the list.

Playing a song (#3) will prompt the user to enter the title of the song and print both the title and singer (e.g., “Now playing … by …”)

Skipping forward or backward (#4-5) will jump to the previous song or the next song, respectively, and display its title and singer (as per above). Skip forward on the last song takes you to the beginning of the list. Skip backward on the first song takes you to the last song of the list.

Exit will quit the program.

\*You may use provided main function in Ex2Driver.cpp file to continue.

**Bonus**

**Exercise3:**

You are to use doubly linked lists to implement mathematical sets. Note that in a set each element is unique.Write the below functions,

1. insert: Inserts a new node with a newValue if it does not exist in the list (set). Returns true if the insertion is successful, else false.

template <typename Object>

bool insertSet(DoubleNode<Object>\*& head, Object newValue);

1. setUnion: Returns a pointer to a newly created list (set) that is the union of two lists: head1 and head2. The union contains all elements that are either in list1 or in list2.

template <typename Object>

DoubleNode<Object>\* setUnion(DoubleNode<Object>\* head1,DoubleNode<Object>\* head2);

1. setIntersection: Returns a pointer to newly created list (set) that is the intersection of two lists (sets): head1 and head2. The intersection containing all elements that are both in list1 and in list2.

template <typename Object>

DoubleNode<Object>\* setIntersection(DoubleNode<Object>\* head1,DoubleNode<Object>\* head2);

Code:

//Bonus

template <typename Object>

bool setInsert(DoubleNode<Object>\* & head, Object newValue)

{

if (findInDLL(head, newValue)) {

return false;

}

else {

if (head == nullptr) {

DoubleNode<Object>\* newnode = new DoubleNode<Object>(newValue);

head = newnode;

newnode->next = nullptr;

newnode->prev = nullptr;

return true;

}

//Insert the new node!

DoubleNode<Object>\* thisnode = head;

while(thisnode->next != nullptr) {

thisnode = thisnode->next;

}

//Now thisnode points to the last node in the list

DoubleNode<Object>\* newnode = new DoubleNode<Object>(newValue);

thisnode->next = newnode;

newnode->next = nullptr;

newnode->prev = thisnode;

return true;

}

}

template <typename Object>

DoubleNode<Object>\* setUnion(DoubleNode<Object>\* head1, DoubleNode<Object>\* head2)

{

DoubleNode<Object>\* anshead = new DoubleNode<Object>(head1->data);

DoubleNode<Object>\* anstail = anshead;

//append whole of list1, and then output the elements in list2 that are not in list1

//list 1 output:

DoubleNode<Object>\* thisnode = head1->next;

while (thisnode != nullptr) {

//insert into answer list

DoubleNode<Object>\* temp = new DoubleNode<Object>(thisnode->data);

anstail->next = temp;

temp->next = nullptr;

temp->prev = anstail;

anstail = temp;

thisnode = thisnode->next;

}

//Output list2 elements not in list1:

thisnode = head2;

while (thisnode != nullptr) {

//search list1 for this element:

bool found = false;

DoubleNode<Object>\* innerptr = head1;

while (innerptr != nullptr) {

if (thisnode->data == innerptr->data) {

//cout << thisnode->data << " ";

found = true;

break;

}

innerptr = innerptr->next;

}

if (!(found)) {

//insert into answer list:

DoubleNode<Object>\* temp = new DoubleNode<Object>(thisnode->data);

anstail->next = temp;

temp->next = nullptr;

temp->prev = anstail;

anstail = temp;

}

thisnode = thisnode->next;

}

return anshead;

}

template <typename Object>

DoubleNode<Object>\* setIntersection(DoubleNode<Object>\* head1, DoubleNode<Object>\* head2)

{

DoubleNode<Object>\* anshead = new DoubleNode<Object>(head1->data);

DoubleNode<Object>\* anstail = anshead;

DoubleNode<Object>\* thisnode = head2->next;

while (thisnode != nullptr) {

//search list1 for this element:

bool found = false;

DoubleNode<Object>\* innerptr = head1;

while (innerptr != nullptr) {

if (thisnode->data == innerptr->data) {

found = true;

break;

}

innerptr = innerptr->next;

}

if (found) {

DoubleNode<Object>\* temp = new DoubleNode<Object>(thisnode->data);

anstail->next = temp;

temp->next = nullptr;

temp->prev = anstail;

anstail = temp;

}

thisnode = thisnode->next;

}

return anshead;

}

int main()

{

//Bonus

cout<<"\n\n Testing Bonus \n";

DoubleNode<int>\* bonushead1=nullptr;

setInsert(bonushead1,2);

setInsert(bonushead1,3);

setInsert(bonushead1,4);

setInsert(bonushead1,4);

setInsert(bonushead1,5);

setInsert(bonushead1,3);

printDLL(bonushead1);

DoubleNode<int>\* bonushead2=nullptr;

setInsert(bonushead2,2);

setInsert(bonushead2,3);

setInsert(bonushead2,6);

setInsert(bonushead2,5);

setInsert(bonushead2,7);

setInsert(bonushead2,4);

cout<<"Lists are : \n";

printDLL(bonushead1);

printDLL(bonushead2);

cout<<"Testing Union: \n";

DoubleNode<int>\* bonusUnionhead = setUnion(bonushead1, bonushead2);

printDLL(bonusUnionhead);

cout<<"Testing Intersection: \n";

DoubleNode<int>\* bonusIntersection= setIntersection(bonushead1, bonushead2);

printDLL(bonusIntersection);

return 0;

}

Screenshot:

