|  |
| --- |
| **Data Structures and Algorithms**  ***Section*: BSCE2021 Assignment # 5 *Total marks*: 100**  ***Name*** : ***\_NIMRA MAQBOOL\_ Roll number* : \_bsce21012\_** |

***Submission:***

• *Email instructor or TA if there are any questions. You cannot look at others’ solutions or use others’ solutions, however, you can discuss it with each other. Plagiarism will be dealt with according to the course policy.*

*• Submission after due time will not be accepted.*

**There should be a Report explaining your code and highlighting the results. Follow this naming convention for your report RollNumber\_Assignment#.pdf e.g BSCE21001\_Assignment3.pdf.**

**TASK: 1**

Make an link list class which contain following functionalities

append(int data);

prepend(int data);

insertAtIndex(int data, int index)

deleteAtEnd();

deleteAtStart();

deleteAtIndex(int index);

display();

Program should be menu driven. the program should continue to run until the exit option is selected. Handel all corner cases.

|  |
| --- |
| **FUNCTION.H:**  #include <iostream> using namespace std;  class Node { //made a class of node public:  int data; //declared data in public  Node \*nextPtr;   Node(int val) {  data = val; //made a constructor to set values  nextPtr = NULL;  }   void setNextPtr(Node \*n) {  nextPtr = n; //set the next ptr  }   Node \*getNextPtr() {  return nextPtr; //get the value of next ptr  } };  class linkList { //making 1 other class of linklist public:  Node \*tail;  Node \*head; //made some pointers  linkList() {  head = nullptr; //declared them to zero  tail = nullptr;  }  void append(int value){  Node \*temp = new Node(value); //declaring  temp->data = value;  temp->nextPtr = nullptr; //initializing the next ptr in the next of the new initialized node  if (head == nullptr) {  head = temp; //if the head is null then store the temp in head  } else {  Node \*temp1 = head; //else make a new node  while (temp1->nextPtr != nullptr) //iterate it till the node is not null  temp1 = temp1->nextPtr; //store the temp to next ptr address  temp1->nextPtr = temp; //store pointer to the last one  }    }  void prepend(int value){  Node \*temp=new Node( value); //make a new node  temp->data=value; //store the value  temp->nextPtr= head; //point the pointer to head  head=temp; //making the head that new temp  }  void insertAtIndex(int index,int value){  Node \*temp=new Node( value); //making a new node  temp->data=value; //storing value  temp->nextPtr= nullptr; //pointing to null  if(index<=0){  cout<<"INDEX MUST BE POSITIVE."<<endl; //checking that index is not zero or negative  }  if(index==1){  temp->nextPtr=head; //is index is 1 then store the head in next ptr and make the temp that head  temp=head;  return;  }  else{  Node \*newTemp=head; //making a new node  int i=1;  while(i<index-1){ //iterating to the place before the index  newTemp=newTemp->nextPtr;  i++;  }  if(newTemp!= nullptr){ //after that checking that node is not null  temp->nextPtr=newTemp->nextPtr; //storing the new temp to the previous temp  newTemp->nextPtr=temp; //making the new temp that previous temp  }  else{  cout<<"THE PREVIOUS NODE IS NOT NULL."<<endl;  }  }  }  void deleteAtEnd(){  if(head== nullptr){  cout<<"ARRAY IS EMPTY."<<endl; //checking is the head is null  }  if (head != nullptr) { //if head is not nul  if (head->nextPtr == nullptr) { //if the next to head is null  head = nullptr; //make the head null  } else {  Node \*temp = head; //making a new node  while (temp->nextPtr->nextPtr != NULL) //if the next to next address is not null  temp = temp->nextPtr; //then store the next address to temp  Node \*temp1 = temp->nextPtr; //store the temp address to new node  temp->nextPtr = nullptr; //make it null and delete the new node  delete temp1;  }  }  }  void deleteAtStart(){  if(head== nullptr){ //is head is null the array is empty  cout<<"THE ARRAY IS EMPTY."<<endl;  }  else{  Node \*newTemp=head; //making a new node and giving it address of head  head=head->nextPtr; //giving head the address the head is pointing to  delete newTemp; // and deleting the temp  }  }  void deleteAtIndex(int index){  if(index<=0){  cout<<"THE INDEX MUST BE POSITIVE."<<endl; //checking the index  }  if(index==1 && head!= nullptr){ //if head is 1 or not null then make a new node  Node \*temp=head;  temp=temp->nextPtr; //and store the address of next ptr to that new node  delete temp; //delete new node  }  else{  Node \*temp1=head; //make a new node give it the address of head  int i = 1;  while (i < index - 1) { //iterate till 1 less position of the index  if (temp1 != nullptr) {  temp1 = temp1->nextPtr; //and storing the addresses  }  i++;  }  Node \*toDelete=temp1->nextPtr; //making new node and storing the address of the ptr of next ptr  temp1->nextPtr=temp1->nextPtr->nextPtr; //storing the value of next to next value to delete the middle one  delete toDelete; //deleting the ptr  }  }  void display() {  Node \*temp = head; //declaring  while (temp != NULL) {  cout << temp->data << "\t"; //displaying the data  temp = temp->nextPtr; //storing the next address  cout << " "; //displaying space  }  } };  **MAIN.CPP:**  #include <iostream> #include "Functions.h"  using namespace std;  // int main() {  Node n(5); //making object  n.setNextPtr(0);  n.getNextPtr(); //calling  linkList l; //making object  l.display();  int opt1; //declaring  int opt;  do {   cout << "\nCHOOSE OPTIONS." << endl;  cout << "1.APPEND." << endl;  cout << "2.PREPEND." << endl;  cout << "3.ADD AT SPECIFIC INDEX." << endl; //displaying options  cout << "4.DELETE AN ELEMENT FROM THE START OF THE LIST." << endl;  cout << "5.DELETE AN ELEMENT FROM THE LAST OF THE LIST." << endl;  cout << "6.DELETE AN ELEMENT FROM THE SPECIFIC INDEX OF THE LIST." << endl;  cout << "7.EXIT." << endl;  cin >> opt1;  if (opt1 == 1) {  int value;  cout << "YOU WANT TO ENTER 1 VALUE OR MULTIPLE VALUES?" << "\nENTER 0 FOR 1 VALUE AND 1 FOR MULTIPLE VALUES"  << endl; //taking choices from user then calling the functions  cin >> opt;  if (opt == 0) {  cout << "ENTER THE NUMBER YOU WANT TO ADD IN LIST = " << endl;  cin >> value;  l.append(value);  l.display();  }  if (opt == 1) {  int val;  cout << "ENTER THE NUMBER OF TIMES YOU WANT TO ENTER THE VALUE." << endl; //taking the choice to enter the number  cin >> val;   for (int i = 0; i < val; i++) {  cout << "ENTER THE NUMBER YOU WANT TO ADD IN LIST = " << endl;  cin >> value;  l.append(value); //calling  }  l.display();  } if(opt!=0 && opt!=1) {  cout << "YOU HAVE ENTERED INVALID ARGUMENT." << endl;  }   }  if(opt1==2){  int value;  cout << "ENTER THE NUMBER YOU WANT TO ADD IN LIST = " << endl;  cin >> value; //taking value to add in list  l.prepend(value); //calling  l.display(); //calling  }  if(opt1==3){  int index;  int value;  cout<<"ENTER THE INDEX AT WHICH YOU WANT TO ADD THE VALUE."<<endl;  cin>>index;  cout << "ENTER THE NUMBER YOU WANT TO ADD IN LIST = " << endl;  cin >> value; //taking value  l.insertAtIndex(index,value); //calling  l.display();  }  if(opt1==4){  l.deleteAtStart(); //calling  l.display();  }  if(opt1==5){  l.deleteAtEnd(); //calling  l.display();  }  if(opt1==6){  int index;  cout<<"ENTER THE INDEX FROM WHICH YOU WANT TO DELETE THE VALUE."<<endl;  cin>>index;  l.deleteAtIndex(index);  l.display(); //calling  }  if(opt1==7){  cout<<"YOU CHOOSE TO EXIT."<<endl; //displaying  exit(3);  }  } while (opt1 >= 1 && opt1 <= 7); //condition to terminate the do while loop   return 0; }  **THE CODES ARE EXPLAINED IN THE COMMENTING.** |

**TASK: 2**

**Write down the time complexity(in Big O) of the following Codes**

int sum = 0 ;

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

sum++;

for ( int j =0; j<N; j++)

sum++;

for ( int k=0; k<N; k++)

sum++;

for ( int m=0; m<N; m++)

sum++;

for ( int n=0; n<N; n++)

sum++;

for ( int p=0; p<N; p++)

sum++;

**Write down the time complexity(in Big O) of the following Codes**

int sum=0;

for ( int i =1; i<N∗N; i ∗=2)

for ( int j =1; j<N∗N; j ∗=2)

sum++;

|  |
| --- |
| **Task 2:**  **Cost repetition total**  **1 1 1**  **1 n-1 n-1**  **1 n-1 n-1**  **1 n-1 n-1**  **1 n-1 n-1**  **1 n-1 n-1**  **1 n-1 n-1**  **Time complexity is = O(n)**  **Part 2:**  **1 1 1**  **1 1 1**  **1 log(n)/2 log(n)/2**  **1 log(n)/2 log(n)/2**  **1 log(n)/2\*log(n)/2 log(n)/2\*log(n)/2**  **Time complexity is = O(log(n)\*log(n)).** |