Faculty of computers and artificial intelligence,

Cairo university,

Data Structures

Final Project

Data Structures

(PM-2182)

Team Members

Name	ID
عبد الرحمن محمد رمضان	**20180158**
توفيق ياسر توفيق ابوسيف	**20180075**

Faculty of Computers and Artificial Intelligence

Cairo University

CS214

Part I

A. Library management system:

Problem Source Code:

```
#include<iostream>
#include<string>
using namespace std;
// this class QueueNode
class QueueNode
{
public:
	string CopyDate;
	bool status;
	string Borrower;
	string BorrowDate;
	int NumberOfDays;
	QueueNode* Next;
	// constractor of the Queue Ndoe
	// make Next point to Null
	QueueNode()
	{
		Next = 0;
	}
};
class Queue
{
public:
	// the node that points to the first node
	QueueNode* front,
		// this node that points to the last node
		* rear;
	// this represt size of the copies in each book
	int size;
	// the queue constructor
	// make front and rear point to null
	Queue()
	{
		front = rear = 0;
```

```
	size = 0;
}
// this method take pamaremters and insert them into the Queue
void Insert(string cop,bool stat,string user,string broDate,int Number)
{
	QueueNode* new_node = new QueueNode();
	new_node->Next = NULL;
	new_node->CopyDate = cop;
	new_node->status = stat;
	new_node->Borrower = user;
	new_node->NumberOfDays = Number;
	// if the rear is equal null
	// the queue if empty
	if (!this->rear)
	{
		// make the front points to the the new_node
		this->front = new_node;
	}
	else
		// else make the last node->next points to new_node
		this->rear->Next = new_node;
	// make the rear point to the last node
	this->rear = new_node;
	// increase size by one
	this->size++;
}
// this method do delete node the queue
void serve()
{
	QueueNode *Tem = front;
	this->front = Tem->Next;
	delete Tem;
	//
	if (!this->front)
	{
		this->rear = NULL;
	}
	this->size--;
}
// this method to print the queue
int print()
{
	QueueNode *Tem = front;
	// if the rear and front are equal null
```

```
                // the qeueue if empty
                if (this->rear == 0 && this->front == 0)
                {
                        return 0;
                }
                // if the front is not null
                while (Tem!=NULL)
                {

                        cout << "CopyDate is  :" << Tem->CopyDate << endl;
                        cout << "Avaliable :" << Tem->status << endl;
                        cout << "Days to return  :" << Tem->NumberOfDays << endl;
                        cout << "Borrowed_Date" << Tem->BorrowDate << endl;
                        cout << "User  :" <<Tem->Borrower<< endl;
                        Tem = Tem->Next;
                }
        }
};
// linkedListNate class
class linkedListNode
{
public:
        string Book;
        int ISBN;
        int NumberOfCopies;
        Queue* object;
        linkedListNode* Next;
        // empty constructor to inialize next and object
        linkedListNode()
        {
                Next = 0;
                object = new Queue();
        }
        //parameterize constructor
        linkedListNode(string name,int IB,int number)
        {
                this->Book = name;
                this->Book = IB;
                this->NumberOfCopies = number;
        }
};

class LinkedList
{
```

```
public:
	linkedListNode *root;
	LinkedList()
	{
		root = new linkedListNode();
	}
	//this method to add a new book
	void AddNew_Book(string Name, int IS, int number)
	{
		linkedListNode* new_node = new linkedListNode();
		linkedListNode* last = root;
		new_node->Book = Name;
		new_node->ISBN = IS;
		new_node->NumberOfCopies = number;
		new_node->Next = NULL;
		// if the root equal -> null
		// the linked list is empty
		if (root == NULL)
		{
			root = new_node;
			return;
		}
		// loop to get the last node in insered
		while (last->Next != NULL)
			last = last->Next;
		last->Next = new_node;
		return;
	}
	// this method to print books elements
	void dis()
	{

		while (root->Next != 0)
		{
			root = root->Next;
			cout << "Book_Name " << root->Book << "\nBOOk ISBN "
<< root->ISBN
				<< "\nBOOK number of copies " << root-
>NumberOfCopies << endl;

		}
	}
	void AddNew_borrow(string copy, bool status, string borrower,string
Date_return, int number)
```

```
	{

		root->object->Insert(copy, status, borrower, Date_return,number);
	}
	void Add_Book_Return()
	{
		root->object->serve();
	}
	void DisBorrow()
	{
		root->object->print();
	}
	void Library_Inventory()
	{
		linkedListNode *Current = root;
		while (Current->Next != 0)
		{
			Current =Current->Next;
			cout << "Book_Name :" << Current->Book << endl;
			cout << "BOOK_ISBN :" << Current->ISBN << endl;
			cout << "original_Copies :" << Current->NumberOfCopies <<
endl;
			cout << "borrowed_Copies :" << Current->object->size <<
endl;
			cout << "////////////////////////////////////////////////" << endl;

		}
	}
	bool Search_About_Book(string Book_Name)
	{
		linkedListNode* Current = root;
		while (Current->Next != 0)
		{
			if (Current->Book == Book_Name)
			{
				return true;
			}
		}
		return false;
	}
};
int main()
{
	Queue object;
```

```
        //object.Insert("1/1/2020", true, "Ahmed", "2/2/2020", 23);
        //object.Insert("2/1/2020", false, "Ali", "5/2/2020", 23);
        //object.print();
        //object.serve();
        cout << "After" << endl;
        //object.print();
        cout << "//////////////" << endl;
        LinkedList object2;
        LinkedList object3;
        object2.AddNew_Book("DS", 123, 34);
        object2.AddNew_Book("FA", 23, 100);
        //object2.dis();
        object2.AddNew_borrow("1/1/2020", true, "Ahmed", "10/10/2010", 23);
        object2.AddNew_borrow("1/1/2060", true, "Abdelrhman", "100/10/2010",
80);
        object2.Add_Book_Return();
        cout << "Three" << endl;
        object2.DisBorrow();
        cout << "After_2" << endl;
        object2.DisBorrow();
        cout << "Libaray: " << endl;
        object2.Library_Inventory();
        return 0;
}
```

B. Linked List Example:

Problem Source Code:

```
 #include<iostream>
#include<string>
using namespace std;
template<class Tem>
// class node
class Node
{
public:
   // info
         Tem Data;
   // oints to the next node
         Node* Next;
         Node()
         {
      // next points to null
                  Next = 0;
         }
};
template<class Tem>
// linked in class
class Linked_List
{
public:
   // head of the linked list
         Node<Tem>* head;
         Linked_List()
         {
      head = new Node<Tem>();
         }
         void Insert(Tem element)
         {
      // current equal head to move with Current
                  Node<Tem>* Current = head;
      // create the new node
                  Node<Tem>* new_node = new Node<Tem>();
      // if the current is null
```

```
    // the linked list is empty
    if (Current == NULL)
    {
       // make head point the new_node is consider the first node
       head = new_node;
    }
    // move to get the last node inserted
                while (Current->Next != NULL)
                {
                        Current = Current->Next;
                }
    // new node points to null
                new_node->Next = NULL;
                new_node->Data = element;
    // make current(pointer of the last node is inserted) point the new node
                Current->Next = new_node;
       }
void InsertPos(Tem element,int Position)
{
    int counter=0;
    Node<Tem>* Current = head;
    Node<Tem>* new_node = new Node<Tem>();
    // pointer to points to the node (after the new node)
    Node<Tem>* ptr = new Node<Tem>();
    new_node->Data = element;
    int i;
    while (Current != NULL)
    {
       Current = Current->Next;
       counter++;
    }
    // if the poistion is 1
    // insert element to the first
    if (Position == 1)
    {
       // if the Link_list is empty
       if (Current == NULL)
       {
          // Current points to new_node
          Current = new_node;
          Current->Next = NULL;
       }
       else
       {
```

```
            ptr = Current;
            Current = new_node;
            Current->Next = ptr;
        }
    }
    else if (Position > 1 && Position <= counter)
    {
        Current = head;
        for (i = 1; i < Position; i++)
        {
            ptr = Current;
            Current =  Current->Next;
        }
        ptr->Next = new_node;
        new_node->Next = Current;
    }
    else
    {
        cout << "Positon out of range" << endl;
    }
}
void Delete(Tem element)
{
    Node<Tem>* Prev = new Node<Tem>();
    Node<Tem>* Current = head;
    while (Current != NULL) {
        if (Current-> Data == element) {
            break;
        }
        else {

            Prev = Current;
            Current = Current->Next;
        }
    }
    if (Current == NULL) {
        return;
    }
    else {
        Prev->Next = Current->Next;
        delete Current;
    }
}
void Print()
```

```
{
   Node<Tem>* Cur = head;
   if (Cur->Next == NULL)
   {

      return;
   }
   cout << "Elements of list are: " << endl;
   while (Cur!= 0)
   {
      cout << Cur->Data<<"   ";
      Cur = Cur->Next;
   }
}
void DeletePos(int Position)
{


      int i, counter = 0;
      if (head == NULL)
      {
         return;
      }
      Node<Tem>* Current=head;
      Node<Tem>* ptr = new Node<Tem>();
      if (Position == 1)
      {
         Current = Current->Next;
      }
      else
      {
         while (Current != NULL)
         {
            Current = Current->Next;
            counter++;
         }
         if (Position > 0 && Position <= counter)
         {
            Current = head;
            for (i = 1; i < Position; i++)
            {
               ptr = Current;
               Current= Current->Next;
            }
```

```
                ptr->Next = Current->Next;
            }
            else
            {
                cout << "Position out of range" << endl;
            }
            delete Current;
            cout << "Element Deleted" << endl;
        }
    }
    void Concatenate(Node<Tem>*T1,Node<Tem>*T2)
    {
        Node<Tem>* Current = T2;
        if (T2 == 0)
        {
            return;
        }
        while (Current != 0)
        {
            this->Insert(Current->Data);
            Current = Current->Next;
        }
    }
};
int main()
{
        Linked_List<int> object;
        object.Insert(3);
        object.Insert(2);
        object.Insert(5);
        object.Insert(7);
    object.InsertPos(23, 4);
    object.Print();
    object.Delete(7);
    object.Print();
    object.DeletePos(3);
    object.Print();
    Linked_List<int> object2;
    object2.Insert(28);
    object2.Insert(3000);
    object.Concatenate(object.head, object2.head);
    object.Print();
        return 0;
}
```

C.Sorting Array Example:

Problem Source Code:

```
#include<iostream>


using namespace std;


//First Algorithm ( Quick Sort AL )
// Time O(n) - no space required
int quickSortArray(int a[], int n){
    int pivot = 1;
    int h = 0;
    for(int i=0; i<n ; i++){
        if(a[i] < pivot){
            swap(a[i],a[h]);
            h++;
        }
    }
}


//Second Algorithm is to count the number of 0's and fill the array with the number of 0's then fill the rest by 1's
// Time O(n) - space O(1)
void countingAndSorting(int a[],int n){
    int zeros = 0;
    for(int i=0;i<n;i++){
```

```
        if(a[i]==0)
            zeros++;
    }
    int i=0;
    while(zeros--){
        a[i++] = 0;
    }
    while(i<n){
        a[i++] = 1;
    }
}

//Third Algorithm using insertion sort
// Time O(n) [worst - best] - space O(1)
void sortingByInsertion(int a[],int n){
    int h = 0,j=0;
    for(int i=1;i<n;i++){
        h = a[i];
        j = i - 1;
        while( j>=0 && a[j] > h)
        {
            a[j+1] = a[j];
            j = j - 1;
        }
        a[j+1] = h;
    }
}
```

```
int main(){

    int arr[] = {1,0,1,0,1,0,0,1,0};
    int Size = sizeof(arr) / sizeof(int);
    //quickSortArray(arr,Size);
    //countingAndSorting(arr,Size);
    //sortingByInsertion(arr,Size);
    for(int i=0;i<Size;i++)
        cout<<arr[i]<<" ";
    return 0;
}
```