# Name: Ayesha Sindu

# Roll No: 22F-3773

**Section: MS-1A**

# Submitted to: Sir. Muhammad Yousaf

# Q-1

**Take inputs in a user defined array. After that if the input is even, place it at an even index and if the input is odd, place it an odd index. If the user puts in an even integer and all even indexes are occupied, ask user to enter an odd integer and vice versa. If all the even and odd entries are filled, then notify the user that the program has ended.**

**Inputs: 9 8 5 7 4 2 1**

**Output: 8 9 4 5 2 7 – 1**

**Index: 0 1 2 3 4 5 6 7**

**#include <iostream>**

**using namespace std;**

**void arrangeWithInd(int arr[], int n)**

**{**

**int oddInd = 1;**

**int evInd = 0;**

**while(true)**

**{**

**while(evInd < n && arr[evInd] % 2 == 0 )**

**evInd += 2;**

**while(oddInd < n && arr[oddInd] % 2 == 1 )**

**oddInd += 2;**

**if (evInd < n && oddInd < n )**

**swap(arr[evInd], arr[oddInd] );**

**else**

**break;**

**}**

**}**

**void printArray(int arr[], int n)**

**{**

**for (int i = 0; i < n; i++)**

**cout << arr[i] << " ";**

**}**

**int main()**

**{**

**int arr[10] = {};**

**cout << "Enter 10 values" << endl;**

**for (int i = 0; i < 10; i++)**

**{**

**cin >> arr[i];**

**}**

**int n = sizeof(arr) / sizeof(arr[0]);**

**cout << "Input:";**

**printArray(arr, n);**

**arrangeWithInd(arr, n);**

**cout << "\nOutput:";**

**printArray(arr, n);**

**return 0;**

**}**

# Q- 2

**Create a user defined array and then take inputs. Print all the Divisors of every number user gave as input. Inputs: Size: 3 Elements: 4 5 6 Output: 4: 1 2 5: 1 6: 1 2 3**

#include <iostream>

using namespace std;

void diviser(int arr[], int n)

{

int i;

int w = 0;

for (w = 0; w < arr[w]; w++)

{

cout << "\nAll the divisor: " << arr[w] << "\n";

for (i = 1; i <= arr[i]; i++)

{

if ((arr[w] % i) == 0)

{

cout <<i<< " ";

cout << "\n";

}

}

}

}

void print(int arr[], int n)

{

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

cout << arr[i] << " ";

}

int main()

{

int arr[3] = {};

cout <<"Put 3 values" << endl;

for (int i = 0; i < 3; i++)

{

cin >> arr[i];

}

int n = sizeof(arr) / sizeof(arr[0]);

cout << "Diviser Array:";

print(arr, n);

diviser(arr, n);

return 0;

}

# Q- 3

**Your task is to input two sorted singly linked lists in descending order. Change the next pointers to obtain a single, merged linked list which also has data in descending order. Note that either head pointer given may be null meaning that the corresponding list is empty. You must use the following function prototype, node\* merge\_lists(node \*, node \*);**

**#include <iostream>**

**using namespace std;**

**struct Node**

**{int data;**

**struct Node \*next;**

**};**

**Node \*merge\_lists(Node \*a, Node \*b)**

**{**

**if (a == NULL && b == NULL)**

**return NULL;**

**Node \*result = NULL;**

**while (a != NULL && b != NULL)**

**{**

**if (a->data <= b->data)**

**{**

**Node \*temp = a->next;**

**a->next = result;**

**result = a;**

**a = temp;**

**}**

**else**

**{**

**Node \*temp = b->next;**

**b->next = result;**

**result = b;**

**b = temp;**

**}**

**}**

**while (a != NULL)**

**{**

**Node \*temp = a->next;**

**a->next = result;**

**result = a;**

**a = temp;**

**}**

**while (b != NULL)**

**{**

**Node \*temp = b->next;**

**b->next = result;**

**result = b;**

**b = temp;**

**}**

**return result;**

**}**

**void printList(struct Node \*Node)**

**{while (Node != NULL)**

**{**

**cout << Node->data << " ";**

**Node = Node->next;**

**}**

**}**

**Node \*newNode(int data)**

**{**

**Node \*temp = new Node;**

**temp->data = data;**

**temp->next = NULL;**

**return temp;**

**}**

**int main()**

**{**

**struct Node \*result = NULL;**

**Node \*a = newNode(5);**

**a->next = newNode(9);**

**a->next->next = newNode(25);**

**Node \*b = newNode(3);**

**b->next = newNode(1);**

**b->next->next = newNode(50);**

**cout << "list A: \n";**

**printList(a);**

**cout << "\n list B: \n";**

**printList(b);**

**result = merge\_lists(a, b);**

**cout << "\nmerged linked: \n";**

**printList(result);**

**return 0;**

**}**

# Q- 4

**Your task is to take two singly linked lists let suppose A and B. The last index of A is connected to 3rd index of the B. Find the common elements in both linked list**

**#include <iostream>**

**using namespace std;**

**class Node**

**{ public:**

**int data;**

**Node \*next;**

**};**

**void insert(Node \*\*, int);**

**void print(Node \*);**

**void merge(Node \*, Node \*\*);**

**int main()**

**{**

**Node \*first = NULL, \*second = NULL;**

**insert(&first, 3);**

**insert(&first, 2);**

**insert(&first, 1);**

**cout<<"First List:\n";**

**print(first);**

**insert(&second, 10);**

**insert(&second, 9);**

**insert(&second, 8);**

**insert(&second, 7);**

**insert(&second, 6);**

**insert(&second, 5);**

**insert(&second, 4);**

**cout<<"\nSecond List:\n";**

**print(second);**

**merge(first, &second);**

**cout<<"\nMerged list is: ";**

**print(first);**

**return 0;**

**}**

**void insert(Node \*\* head\_ref, int new\_data)**

**{**

**Node\* new\_node = new Node();**

**new\_node->data = new\_data;**

**new\_node->next = (\*head\_ref);**

**(\*head\_ref) = new\_node;**

**}**

**void print(Node \*head)**

**{**

**Node \*temp = head;**

**while (temp != NULL )**

**{**

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

**temp = temp->next;**

**}**

**cout<<endl;**

**}**

**void merge(Node \*first, Node \*\*second)**

**{**

**Node \*firstRef = first;**

**Node \*secondRef=\*second;**

**while (firstRef->next != NULL)**

**{**

**firstRef = firstRef->next;**

**}**

**firstRef->next = secondRef->next->next;**

**}**

# Q- 5

**Make a doubly linked list and done all the things mention below. The node will have an int variable in the data part. Your LinkedList will have a head and a tail pointer. Your LinkedList class must have the following functions: 1) Insert a node void insertNodeAtBeginning(int data); void insertNodeInMiddle(int key, int data); //will search for key and insert node after the node where a node’s data==key void insertNodeAtEnd(int data); 2) Delete a node bool deleteFirstNode(); //will delete the first node of the LL bool deleteNode(int key); //search for the node where its data==key and delete that particular node bool deleteLastNode(); //will delete the last node of the LL 3) Searching 4) Display**---

**#include<iostream>**

**using namespace std;**

**struct Node**

**{int data;**

**Node\* next,\*prev;**

**};**

**class doubly**

**{ private:**

**Node\*head, \*temp,\*tail;**

**public:**

**void insertAtStart(Node\*& head, int value){**

**{**

**Node\* node = new Node();**

**node->data = value;**

**node->next = head;**

**if (head != NULL)**

**head->prev = node;**

**node-> prev = NULL;**

**head = node;**

**}**

**}**

**void insertAtTail(Node\*& head, int value)**

**{**

**if (head == NULL)**

**{**

**insertAtStart(head, value);**

**return;**

**}**

**Node\* node = new Node();**

**node->data = value;**

**Node\* temp = head;**

**while (temp->next != NULL)**

**temp = temp->next;**

**temp->next = node;**

**node->prev = temp;**

**node->next = NULL;**

**}**

**void insertInMid(Node\*& head, int value, int after)**

**{**

**Node\* node = new Node();**

**node->data = value;**

**Node\* temp = head;**

**while (temp != NULL && temp->data != after)**

**temp = temp->next;**

**if (temp == NULL)**

**{**

**cout << "Key not found. Cannot insert value.\n";**

**return;**

**}**

**node->next = temp->next;**

**if(temp->next != NULL)**

**temp->next->prev = node;**

**temp->next = node;**

**node->prev = temp;**

**}**

**void deleteAtFirst(Node\*& head){**

**Node\* node = head;**

**head = head->next;**

**head->prev = NULL;**

**delete node;**

**}**

**void deleteAtLast(Node\*& head){**

**if (head == NULL)**

**return;**

**if (head->next == NULL)**

**{**

**delete head;**

**return;**

**}**

**Node\* temp = head;**

**while (temp->next->next != NULL)**

**temp = temp->next;**

**delete temp->next;**

**temp->next = NULL;**

**}**

**void deleteNode(Node\*& head, int key){**

**Node\* temp = head;**

**if (temp->data == key)**

**{**

**deleteAtFirst(head);**

**temp = temp->next;**

**return;**

**}**

**while (temp != NULL && temp->data != key)**

**temp = temp->next;**

**if (temp == NULL)**

**{**

**cout << "Node Not found..." << endl;**

**return;**

**}**

**if (temp->next == NULL)**

**{**

**deleteAtLast(head);**

**}**

**else**

**{**

**temp->next->prev = temp->prev;**

**temp->prev->next = temp->next;**

**}**

**}**

**bool search(Node \*head, int key)**

**{**

**Node \*temp = head;**

**int pos = 0;**

**if (temp != NULL)**

**{**

**while (temp != NULL)**

**{**

**pos++;**

**if (temp->data == key)**

**{**

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

**break;**

**}**

**temp = temp->next;**

**}**

**}**

**}**

**Node \*reverve(Node \*&head)**

**{**

**Node \*prevptr = NULL;**

**Node \*currptr = head;**

**Node \*nextptr;**

**while (currptr != NULL)**

**{**

**nextptr = currptr->next;**

**currptr->next = prevptr;**

**prevptr = currptr;**

**currptr = nextptr;**

**}**

**return prevptr;**

**}**

**void display(Node\* head)**

**{**

**Node\* temp = head;**

**cout << " ";**

**while (temp != NULL)**

**{**

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

**temp = temp->next;**

**}**

**cout << "\n";**

**}**

**};**

**int main()**

**{**

**Node\* head = NULL;**

**doubly obj;**

**obj.insertAtStart(head,1);**

**obj.display(head);**

**obj.insertAtStart(head,2);**

**obj.display(head);**

**obj.insertAtTail(head,3);**

**obj.display(head);**

**obj.insertAtTail(head,4);**

**obj.display(head);**

**obj.insertAtTail(head,5);**

**obj.display(head);**

**obj.insertAtTail(head,6);**

**obj.display(head);**

**obj.insertInMid(head,8,1);**

**obj.display(head);**

**Node \*newhead = obj.reverve(head);**

**obj.display(newhead);**

**obj.deleteAtFirst(head);**

**obj.display(head);**

**obj.deleteAtLast(head);**

**obj.display(head);**

**obj.search(head, 1);**

**obj.display(head);**

**return 0;**

**}**

# Q- 6

**Write a program that prompts the user to input a string and then outputs the string in the pigLatin form. Input the string in a doubly circular link-list. The rules for converting a string into pig Latin form are as follows: If the string begins with a vowel, add the string "-way" at the end of the string. Forexample, the pig Latin form of the string "eye" is "eye-way". If the string does not begin with a vowel, first add "-" at the end of the string. Then rotate the string one character at a time; that is, move the first character of the string tothe end of the string until the first character of the string becomes a vowel. Then add the string "ay" at the end. For example, the pig Latin form of the string "There" is "ere-Thay Strings such as "by" contain no vowels. In cases like this, the letter y can be considered a vowel. So, for this program the vowels are a, e, i, o, u, y, A, E, I, O, U,and Y. Therefore, the pig Latin form of "by" is "y-bay". Strings such as "1234" contain no vowels. The pig Latin form of the string "1234" is"1234- way". That is, the pig Latin form of a string that has no vowels in it is the string followed by the string "-way"**

# Q- 7

**Round-Robin algorithm is an algorithm used in CPU Scheduling. In this algorithm, all processes take turns and run for a specific interval, until they are all completely executed.You are required to make a process that will consist of two things, ID (must be unique) Execution time Each process will take a certain amount of time to execute completely, this time will bestored in the “execution time” variable. You are required to implement a circular linked list that will store user-entered number of processes (say n). You will also input a time slice from the user, this will be the time every process will get in one turn. You will then create a function that will execute every process in the circular linked list using Round-Robin algorithm and store the total time taken by each process to execute. Your final output will be the time required by each process to completely execute**