



**INSTRUCTIONS TO CANDIDATES:**



**SECTION A: Attempt all Questions./55pts**

1. Define the following terms in linked list data structure.**/4p**ts
2. Node in a linked list
3. Tail in a linked list

**Answer:**

**Node**

A node is a collection of two sub-elements or parts. A data part that stores the element and a next part that stores the link to the next node.  The next part is a pointer that stores the address of the second link.

**Tail**

The tail is the last node of the linked list.It is because no memory address is specified in the last node, the final node of the linked list will have a null next pointer.

1. A linked list is a linear dynamic data structure.**/2pts**
2. Yes
3. No

**Answer:**

Yes

1. A structure can have more than one data members.**/2pts**
2. Yes
3. No

**Answer:**

Yes

1. Differentiate a class to a structure in C++./**4pts**

**Answer:**

1. *A class is a blueprint for creating objects while.*

***A structure*** *is a user-defined data type in C/C++. A structure is a collection of variables of different data types under a single name.  A structure creates a data type that can be used to group items of possibly different types into a single type*

1. *By default, structure data members are similar to a class where all data members are made public.*
2. Correct the following program so that it can compile.**/4pts**

#include <iostream>

using namespace std;

class Rectangle {

  int width, height;

public:

  Rectangle(int x, int y) : width(x), height(y) {}

  int area(void) {

return width \* height;

}

};

int main() {

Rectangle rec(2,4);

  Rectangle \*rect;

  rect = new Rectangle (5, 6);

  cout << "Rec's area: " << rec->area() << '\n';

  cout << "\*rect's area: " << rect.area() << '\n';

  delete rect;

  return 0;

}

**Answer:**

#include <iostream>

using namespace std;

class Rectangle {

  int width, height;

public:

  Rectangle(int x, int y) : width(x), height(y) {}

  int area(void) {

return width \* height;

}

};

int main() {

Rectangle rec(2,4);

  Rectangle \*rect;

  rect = new Rectangle (5, 6);

**cout << "Rec's area: " << rec.area() << '\n';**

**cout << "\*rect's area: " << rect->area() << '\n';**

  delete rect;

  return 0;

}

1. Differentiate the insertion sort and selection sort.**/4pts**

**Answer:**

**The insertion sort**

 The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

**Algorithm**

* To sort an array of size n in ascending order:
* Iterate from arr[1] to arr[n] over the array.
* Compare the current element (key) to its predecessor. It starts by comparing the second element with the first element. If the second element is smaller than the first, then we will swap it.
* If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

**Selection Sort**

Selection sort is a sorting technique that works by **finding the smallest value in the array and placing it in the first position.**

After that, it was then  .

Thus, the selection sort works by finding the smallest unsorted element remaining in the entire array and then swapping it with the element in the next position to be filled.

It is a very simple technique, and it is also easier to implement than other sorting techniques. Selection sort is used for sorting files with large records.

Let’s take the same example as in the selection sort of playing cards.

The algorithm maintains two subarrays in a given array.

1. The subarray is already sorted.
2. The remaining subarray is unsorted.

**In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.**

1. Differentiate pointer to the array. **/4pts**

**Answer:**

*A pointer is a variable that store the address of another variable,*

*The memory size of the pointer is the same*

***While***

*The array is a variable that stores elements of the same type.*

*The memory size of the array depends on the number of elements*

*The name of the array is the pointer to the first element in the array.*

1. Which one is correct about linear search? To perform the linear search, the array must be sorted.**/2pts**
2. Yes
3. No

Answer:

b

1. Explain the following  terminologies used in a Queue data structure.**/6pts**
   1. Rear
   2. Front
   3. Circular Queue

**Answer:**

* 1. **Rear:** The top of the Queue used to add data
  2. **Front:** The position of the first element in the Queue used to remove data.
  3. **Circular Queue:** A queue where the first element is linked to the last element.

1. Assume x=12, y=6, z=4, t=4, s=7 what is the output of the following expression. **/3pts**

 p=y\*(x/t%z)\*t-s\*x/t+y/(x\*t)

**Answer: 51**

1. Can a Structure contain a pointer to itself? **/2pts**

(A) Yes

(B) No

(C) Compilation Error

Answer:

**Yes**

1. Given a singly linked list of numbers identified by a globally accessible head node “**head”** ,develop a function **deleteNode** that would Delete the head of the linked list.

The function signature is **Node\* deleteNode(Node \*head);**  **~~void deleteNode(Node \*head);~~ /5pts**

Node\* deleteFirstNode(Node\* head){

if (head == NULL)

        return NULL;

    // Move the head pointer to the next node

    Node\* temp = head;

    head = temp->next;

    delete temp;

return head;

}

1. Given a singly linked list, write a function that add a node at the end. **/5pts**

**Sample Answer:**

Node\* insertTail(Node \*head, int val){

if(head==NULL){

Node\* n = new Node(val);

n->next = head;

head = n;

return head;

}

Node \*temp = head;

while(head->next!=NULL){

head = head->next;

}

if(head){

Node \*n = new Node(val);

n->next = head->next;

head->next = n;

}

return temp;

}

1. Given an array **arr** of size **N** and an element **E,** using the binary search searching algorithm; write a function **findP** which finds and returns the position **P** of **E** in the same array **arr** . **/5pts**

int BinarySearch(int a[],int n,int key){

      int s = 0, e = n-1;

      while(s<=e){

         int mid = s + (e-s)/2;

         if(a[mid] == key){

             return mid;

         }else if(a[mid] > key){

             e = mid-1;

         }else{

             s = mid + 1;

         }

      }

      return -1;

}

1. Using recursion find the second index of an element if available in the array.**/5pts**

a[8]={5,5,6,20,5,6,7,6}

if x=6, the second index of x is 5.

int secondIndex(int a[],int n, int x, int i, int counter)

**Answer:**

#include<iostream>

using namespace std;

//a array of Integers, n as n numbers in the array, x as an element and i as index, counter to count times x is occurring

int secondIndex(int a[],int n, int x, int i, int counter){

if(i==n){

return -1;

}

if(a[i]==x){

counter++;

}

if(counter==2){

return i;

}

//For iterating in recursion, use i+1

return secondIndex(a,n,x,i+1,counter);

}

int main(){

int a[8]={5,5,6,20,5,6,7,6} ;

cout<<secondIndex(a,8,6,0,0);

return 0;

}

int secondIndex(int a[],int n, int x, int i, int counter)

**SECTION B: Choose only 3 Questions./30pts**

1. Using array implementation of the queue data structure, Develop a functions that would perform the enqueue, dequeue and display operations on any given queue: “**Q”** identified by rear: “**R”** and front: “**F”** . **/10pts**

**Answer:**

void enqueue() {

   int val;

   if (rear == n - 1)

   cout<<"Queue Overflow/Full"<<endl;

   else {

      if (front == - 1)

      front = 0;

      cout<<"Insert the element in queue : "<<endl;

      cin>>val;

      rear++;

      queue[rear] = val;

   }

}

void dequeue() {

   if (front == - 1 || front > rear) {

      cout<<"Queue Underflow/Empty ";

      return ;

   }

      cout<<"Element deleted from queue is : "<< queue[front] <<endl;

      // No need of shifting left , but this is not a better implementation

      front++;

}

void display() {

   if (front == - 1)

   cout<<"Queue is empty"<<endl;

   else {

      cout<<"Queue elements are : ";

      for (int i = front; i <= rear; i++)

      cout<<queue[i]<<" ";

cout<<endl;

   }

}

1. Write a function to sort the array using insertion sort of the array. **/10pts**

**Answer:**

***void******insertionSort****(****int*** *arr[],* ***int*** *n)  
{* ***int*** *i, key, j;* ***for*** *(i = 1; i < n; i++)  
 {  
 key = arr[i];  
 j = i - 1;  
 /\* Move elements of arr[0..i-1], that are greater than temp, to one position ahead of their current position \*/* ***while*** *(j >= 0 && arr[j] > key)  
 {  
 arr[j + 1] = arr[j];  
 j = j - 1;  
 }  
 arr[j + 1] = key;  
 }  
}  
}*

1. Given an array **arr** of size **N** and an element **E,** using the binary search searching algorithm; write a function **findP** which finds and returns the position **P** of **E** in the same array **arr** . **/10pts**

**Answer:**

int BinarySearch(int a[],int n,int key){

      int s = 0, e = n-1;

      while(s<=e){

         int mid = s + (e-s)/2;

         if(a[mid] == key){

             return mid;

         }else if(a[mid] > key){

             e = mid-1;

         }else{

             s = mid + 1;

         }

      }

      return -1;

}

1. Write a function to reverse a linked list**./10pts**

z

**Answer:**

Node\*  reverse(Node\* head){

Node \*current=head;

Node \*previous=NULL;

Node \*n=NULL;

while (current!=NULL){

n=current->next;

*//Instead of pointing to next element, point to previous*

                    current->next=previous;

previous=current;

current=n;

}

      //The last prev is our head

return previous;c

}

1. Write the program to implement the circular Queue using array.**/10pts**

**Cfr Summary**

**SECTION C: Choose one Question./15pts**

1. Write a program to sort the linked list.**/15pts**

Answer:

//Key functions

Node \*mid\_point(Node \*head)

{

    // base case

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

        return head;

    // recursive case

    Node \*fast = head;

    Node \*slow = head;

    while(fast != NULL && fast->next != NULL)

    {

        fast = fast->next;

        if(fast->next == NULL)

            break;

        fast = fast->next;

        slow = slow->next;

    }

    return slow;

}

Node\* merge\_sort(Node \*head)

{

    // base case

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

        return head;

    // recursive case

    // Step 1: divide the linked list into

    // two equal linked lists

    Node \*mid = mid\_point(head);

    Node \*a = head;

    Node \*b = mid->next;

    mid->next = NULL;

    // Step 2: recursively sort the smaller

    // linked lists

    a = merge\_sort(a);

    b = merge\_sort(b);

    // Step 3: merge the sorted linked lists

    Node \*c = merge(a, b);

    return c;

}

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

{

    // base case

    if(a == NULL)

        return b;

    if(b == NULL)

        return a;

    // recursive case

    // take a head pointer

    Node \*c;

    if(a->data < b->data)

    {

        c = a;

        c->next = merge(a->next, b);

    }

    else

    {

        c = b;

        c->next = merge(a, b->next);

    }

    return c;

}

1. Write program using a linked list with a function to insert at any position **p** in a linked list./**15pts**

Node\* insertAtPos(Node \*head, int i, int data){

      if(i<0){

        return head;

      }

      if(i==0){

        Node\* n = new Node(data);

        n->next = head;

        head = n;

        return head;

      }

      Node \*temp = head;

      int count = 1;

      while(count<=i-1 && head!=NULL){

        head = head->next;

        count++;

      }

      if(head){

        Node \*n = new Node(data);

        n->next = head->next;

        head->next = n;

      }

      return temp;

}

1. Write a program that implement the circular linked list with functions.**/15pts**
2. Insert at the beginning
3. Insert at the end function
4. Delete at the beginning
5. Delete at the end
6. Prints element of the linked list

Add a main function to call these functions.

**Use array or Linked list**