A black text on a white background

Description automatically generated

**Data Structures & Algorithms (DSA)**

Year 2, Semester 4

## SCHOOL OF INFOCOMM TECHNOLOGY

Diploma in Information Technology

Diploma in Cybersecurity & Digital Forensics

**COMMON TEST Sample paper**

Weightage: 20%

INSTRUCTIONS TO CANDIDATES:

1. This paper consists of 5 pages including this cover page. Check carefully to make sure your set is complete.
2. There are **4** questions in this paper. Answer ALL questions.
3. Write your answers in the Answer Booklet.
4. Download the files from Brightspace as instructed by the invigilator.
5. Computer laptops/notebooks (with no internet access) are ALLOWED. Language translators are NOT allowed.

There are 4 questions. Answer ALL questions (100 marks).

**Question 1 (25 marks)**

A pointer-based List ADT (Linked list) is given in the folder Q1.

A new element may be inserted into the correct position in the list sorted in ascending order using the sortedInsert() or sortedInsertR() function.

For example, if the linked list has the following values:

0 2 3 4 6 8 9

using the sortedInsert() function to insert 5 would result in the following linked list:

0 2 3 4 5 6 8 9

Duplicate values are allowed in the list.

1. Implement the **iterative** void List::sortedInsert(Itemtype item) function.

void List::sortedInsert(ItemType item) {

    Node\* newNode = new Node{item, nullptr}; // Create a new node

    if (!firstNode || firstNode->item >= item) {

        // Insert at the beginning if the list is empty or the item is smaller than the first node

        newNode->next = firstNode;

        firstNode = newNode;

    } else {

        // Find the correct position for insertion

        Node\* current = firstNode;

        while (current->next && current->next->item < item) {

            current = current->next;

        }

        // Insert the new node

        newNode->next = current->next;

        current->next = newNode;

    }

    size++; // Increment the size of the list

}

(10 marks)

1. Implement the **recursive** void List::sortedInsertR(Itemtype item) function.

void List::sortedInsertR(ItemType item) {

    sortedInsertHelper(firstNode, item);

    size++; // Increment the size of the list

}

void List::sortedInsertHelper(Node\* current, ItemType item) {

    if (!current || current->item >= item) {

        // Insert at the current position if end is reached or item is smaller

        Node\* newNode = new Node{item, current};

        current = newNode;

    } else {

        // Recursive call for the next node

        sortedInsertHelper(current->next, item);

    }

}

(10 marks)

1. State the time complexity of the function in 1(a) and 1(b). Justify your answers.

Time complexity for both functions are O(n), as you have to traverse through the list to the desired node, worst case scenario will be O(n) where the desired node is at the end of the list.

Space complexity is O(n) for recursive, O(1) for iterative, this is due to every time the recursive function is called, it is another frame on the stack.

(5 marks)

**Question 2 (25 marks)**

A simple console application to simulate a short messaging system (SMS) is to be implemented using the Stack ADT given in folder Q2.

Some sample outputs of the application are shown below.

Main Menu

|  |
| --- |
| Main Menu  -------------------  [1] Compose  [2] Undo  [3] Clear  [0] Exit  -------------------  Enter your option:\_ |

Option 1 : Compose

|  |
| --- |
| Option 1 : Compose  Hello how are yoiu |

Note : text underlined are entered by user.

Option 2 : Undo

|  |
| --- |
| Option 2 : Undo  Hello how are yoiu  Enter 1 to undo, 0 to quit: 1  Hello how are yoi  Enter 1 to undo, 0 to quit: 1  Hello how are yo  Enter 1 to undo, 0 to quit: 0 |

Note : user can repeatedly undo by choosing option 1 (or 0 to quit).

Option 1 : Compose

|  |
| --- |
| Option 1 : Compose  Hello how are you? |

Note : new text are underlined.

Option 3 : Clear

|  |
| --- |
| Option 3 : Clear  The text is cleared. |

(a) Implement the compose() function that allows the user to enter text and save to the stack. The function prototype is given below.

void compose(Stack& s)

void compose(Stack& s) {

    cout << "Option 1: Compose" << endl;

    cout << "Enter your text (end with Enter): ";

    string input;

    cin.ignore(); // Clear input buffer

    getline(cin, input); // Get user input as a string

    for (char c : input) {

        if (!s.push(c)) {

            cout << "Stack overflow. Could not add: " << c << endl;

            break;

        }

    }

    cout << "Text added: " << input << endl;

}

(10 marks)

(b) Implement the undo() function that allows the user to undo changes. The function prototype is given below.

void undo(Stack& s)

void undo(Stack& s) {

    cout << "Option 2: Undo" << endl;

    if (s.isEmpty()) {

        cout << "No text to undo." << endl;

        return;

    }

    char undoneChar;

    do {

        s.getTop(undoneChar); // Get the character at the top of the stack

        cout << "Undoing: " << undoneChar << endl;

        s.pop(); // Remove the character from the stack

        cout << "Enter 1 to undo again, 0 to quit: ";

        int choice;

        cin >> choice;

        if (choice == 0) break;

    } while (!s.isEmpty());

    if (s.isEmpty()) {

        cout << "The text is now empty." << endl;

    } else {

        cout << "Remaining text: ";

        s.printInOrderOfInsertion(); // Display the remaining text

        cout << endl;

    }

}

(10 marks)

(c) Implement the clear() function that allows the user to clear the text. The function prototype is given below.

void clear(Stack& s)

void clear(Stack& s) {

    cout << "Option 3: Clear" << endl;

    while (!s.isEmpty()) {

        s.pop(); // Continuously pop until the stack is empty

    }

    cout << "The text is cleared." << endl;

}

(5 marks)

**Question 3 (25 marks)**

The customer service centre of an IT company has decided to implement a queue system to process the service requests of its customers. The specification of the queue is given below. A queue ADT is given in folder Q3.

1. Implement a function, void registerCustomer(Queue& serviceQueue, int& queueNumber), to allow the staff to register a customer. The function should prompt the customer to enter his/her name and add a customer to the queue, serviceQueue with the queueNumber, followed by incrementing the queueNumber for next potential customer

void registerCustomer(Queue& serviceQueue, int& queueNumber) {

    string customerName;

    cout << "Enter customer's name: ";

    cin.ignore(); // Clear input buffer

    getline(cin, customerName); // Get the customer's name

    // Create a new customer object

    Customer newCustomer(queueNumber, customerName);

    serviceQueue.enqueue(newCustomer); // Add the customer to the queue

    cout << "Customer registered successfully: " << customerName

         << " (Queue Number: " << queueNumber << ")" << endl;

    // Increment the queue number for the next customer

    queueNumber++;

}

(10 marks)

1. Implement a function, void nextCustomer(Queue& serviceQueue), to allow the staff to get the next customer in the queue to serve. The function should remove the customer object from the queue and display the queue number of that customer.

void nextCustomer(Queue& serviceQueue) {

    if (serviceQueue.isEmpty()) {

        cout << "No customers in the queue." << endl;

    } else {

        // Get the next customer

        Customer next;

        serviceQueue.dequeue(next); // Remove the customer from the queue and retrieve their data

        cout << "Serving next customer:" << endl;

        next.print(); // Display customer details

    }

}

(5 marks)

1. Implement a function, void displayCount(Queue& serviceQueue),to allow the staff to compute and display the total number of customers left in the queue.

void displayCount(Queue& serviceQueue) {

    int count = 0;

    // Temporarily store the current queue

    Queue tempQueue;

    // Traverse the queue to count elements

    while (!serviceQueue.isEmpty()) {

        Customer tempCustomer;

        serviceQueue.dequeue(tempCustomer); // Dequeue each customer

        tempQueue.enqueue(tempCustomer);   // Add to a temporary queue

        count++;

    }

    // Restore the original queue

    while (!tempQueue.isEmpty()) {

        Customer tempCustomer;

        tempQueue.dequeue(tempCustomer);

        serviceQueue.enqueue(tempCustomer);

    }

    cout << "Total number of customers in the queue: " << count << endl;

}

(10 marks)

**Question 4 (25 marks)**

1. Consider the following 2 hash functions:

public int hash1(String str)

{

int hash = 0;

for (int i=0; i< str.length(); i++)

hash = hash \* 31 + int(str.at(i));

return hash;

}

public int hash2(String str)

{

int hash = 0;

int skip = Math.max(1, str.length()/4);

for (int i=0; i< str.length(); i+=skip)

hash = hash \* 31 + int(str.at(i));

return hash;

}

Which hash function is better? Explain why.

Hash1

Hash1 computes the hash value using all characters in the string, multiplies current hash value by 31 for each character and adds ASCII value of each character

Hash 2 computes the hash value by skipping over characters, meaning the function only processes a subset of characters

(6 marks)

(b) Given the following information for a hash table:

The size of the hash table is 7 and chaining is used to resolve collisions.

The hash function used is hash(k) = k % 7.

(i) Draw the hash table after the following sequence of insertions.

10, 36, 17, 19, 24

I am NOT DRAWING

Index 0: 10

Index 1: 36

Index 2: 17

Index 3: 24

Index 4: 19

(8 marks)

(ii) Consider the table size of 7. Give one good reason for picking 7 and one bad reason for picking 7 for the table size.

It is a prime number, therefore it reduces the possibility of collisions, minimizing the chances of keys hashing to same index when using modulo operations.

However, it is a small size, limits what it can hold without excessive collisions or rehashing

(6 marks)

(iii) What is the maximum number of entries that can be placed in the hash table? Justify your answer.

Infinite, assuming separate chaining is used for collision resolution. Separate chaining allows each slot in the hash table to contain a linked list, storing multiple entries. There is no size limit on each chain, the only limiting factor is sufficient memory to support the linked lists and nodes.

(5 marks)

**– End of Paper –**