NAME SHAHID ZAMAN

SAP ID 55123

COURSE CS(3-2)

SUBJECT DSA

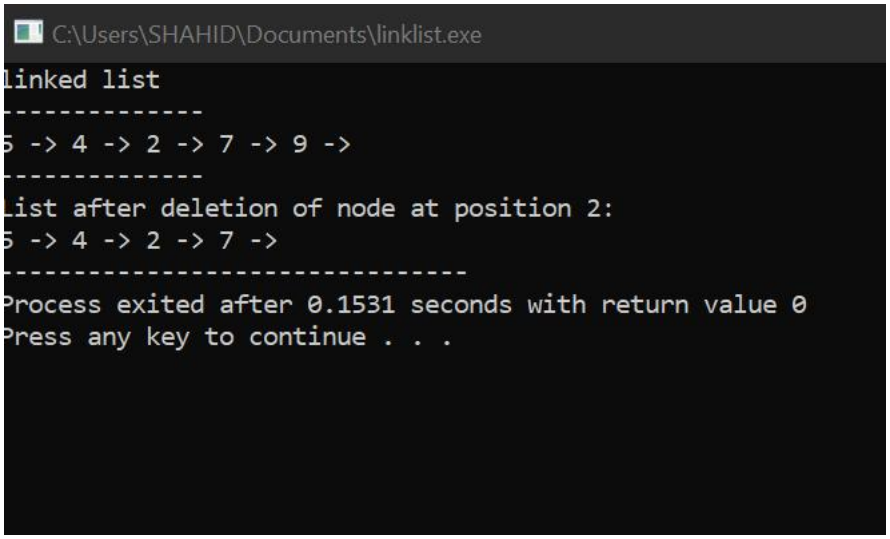
QUESTION NO=01

```
#include <iostream>
using namespace std;
// Node class to represent a node of the linked list.
class Node{
public:
	int data;
	Node* next;
	// Default constructor
	Node()
	{
		data = 0;
		next = NULL;
	}
	// Parameterised Constructor
	Node(int data)
	{
		this->data = data; //  -> is used to access the members (variables and
functions) of an object that is pointed to by a pointer
		this->next = NULL;
	}
};
// Linked list class.
class Linkedlist {
	Node* head;
public:
	// Default constructor
	Linkedlist() { head = NULL; }
	// Insert a node at the end of the linked list.
	void insertNode(int);
	// Print all the nodes in the linked list.
```

```
        void printList();
        // Delete the node at given position
        void deleteNode(int);
};
// Insert a new node.
//  :: scope resolution operator, is used to access members of a class.
void Linkedlist::insertNode(int data)
{
        // Create the new Node.
        Node* newNode = new Node(data);
        // Assign to head
        if (head == NULL) {
                head = newNode;
                return;
        }
        // Traverse till end of list
        Node* temp = head;
        while (temp->next != NULL) {
                // Update temp
                temp = temp->next;
        }
        // Insert at the last.
        temp->next = newNode;
}
// Print all the nodes of the linked list.
void Linkedlist::printList()
{
        Node* temp = head;
        // Check for empty list.
        if (head == NULL) {
        //      cout << "List is empty" << endl;
                return;
        }
        // Traverse the list.
        while (temp != NULL) {
                cout << temp->data << " -> ";
                temp = temp->next;
        }
}
// Delete the node at given position
void Linkedlist::deleteNode(int nodepos)
{
        Node *temp1 = head, *temp2 = NULL;
        int ListLen = 0;
        if (head == NULL) {
                cout << "LIST IS EMPTY deletion not performed" << endl;
                return;
        }
        // Find length of the linked-list.
        while (temp1 != NULL) {
                temp1 = temp1->next;
```

```
                ListLen++;
        }
        // Check if the position to be deleted is greater than the length of the linked list.
        if (ListLen < nodepos) {
                cout << "Index is out of range"<< endl;
                return;
        }
        // Declare temp1
        temp1 = head;
        // Deleting the head.
        if (nodepos == 1) {
                // Update head
                head = head->next;
                delete temp1;
                // cout<<"In head next value must be NULL at this stage: "<<head-
>next<<endl;
                return;
        }
        // Traverse the list to  find the node to be deleted.
        while (nodepos-- > 1) {
                // Update temp2
                temp2 = temp1;
                // Update temp1
                temp1 = temp1->next;
        }
        // Change the next pointer of the previous node.
        temp2->next = temp1->next;
        // Delete the node (deallocate memory)
        delete temp1;
}
// Driver Code
int main()
{
        Linkedlist list; // Object creation , Node Head is created.
        cout<<"linked list"<<endl;
        cout<<"--------------"<<endl;
        list.insertNode(5);
        list.insertNode(4);
        list.insertNode(2);
        list.insertNode(7);
        list.insertNode(9);

        list.printList();
        cout<<"\n--------------";
                // Delete node at position 5.
        list.deleteNode(5);
                cout<<"\nList after deletion of node at position 2:"<<endl;
                        list.printList();
        return 0;
}
```



QUESTION NO= 02

```
#include <iostream>
using namespace std;

// Define the structure for a node in the linked list
struct Student {
   char name[50];
   int sap_id;
   Student* next;
};

// Static array to store student nodes (manual memory management)
Student studentArray[100];
int studentCount = 0;  // Keeps track of the number of students created

// Function to create a new student node
Student* createStudent(const char* name, int sap_id) {
   // Manual memory allocation by using the static array
   if (studentCount >= 100) {
```

```
        cout << "Error: Cannot add more than 100 students." << endl;
        return NULL;
    }

    Student* newStudent = &studentArray[studentCount++];

    // Manually copy the name (without using strcpy)
    int i = 0;
    while (name[i] != '\0' && i < 49) {  // Limit to 49 characters to leave space for '\0'
        newStudent->name[i] = name[i];
        i++;
    }
    newStudent->name[i] = '\0';  // Null-terminate the string

    newStudent->sap_id = sap_id;
    newStudent->next = NULL;
    return newStudent;
}

// Recursive function to insert a student into the linked list
void insertStudent(Student** head, const char* name, int sap_id) {
    if (*head == NULL) {
        *head = createStudent(name, sap_id);
    } else {
        insertStudent(&((*head)->next), name, sap_id);
    }
}

// Recursive function to delete a student at a specific position
void deleteStudent(Student** head, int position, int current = 1) {
    if (*head == NULL) {
        return;  // Base case: empty list
    }

    // Special case: delete the head (1st position)
    if (position == current) {
        Student* temp = *head;
        *head = (*head)->next;
        // No need to free memory, as we're using a static array
        return;
    }

    // Recursive case: go to the next node if the list is long enough
    if ((*head)->next != NULL) {
        deleteStudent(&((*head)->next), position, current + 1);
    }
}

// Recursive function to display the linked list
void displayList(Student* head) {
    if (head == NULL) {
```

```
        return;  // Base case: end of the list
    }

    // Display current student
    cout << "Name: " << head->name << ", SAP ID: " << head->sap_id << endl;

    // Move to the next node
    displayList(head->next);
}

int main() {
    Student* head = NULL;
    int n, sap_id;
    char name[50];

    // Input number of students
    cout << "Enter the number of students: ";
    cin >> n;

    // Check if the number of students exceeds the limit
    if (n > 100) {
        cout << "Error: Cannot add more than 100 students." << endl;
        return 1;
    }

    cin.ignore();  // Clear the input buffer before getline

    // Input name and SAP_ID for each student
    for (int i = 0; i < n; i++) {
        cout << "Enter the name of student " << i + 1 << ": ";
        cin.getline(name, 50);
        cout << "Enter the SAP ID of student " << i + 1 << ": ";
        cin >> sap_id;
        cin.ignore();  // Clear the input buffer again
        insertStudent(&head, name, sap_id);
    }

    // Display the list before deletion
    cout << "\nStudent List:\n";
    displayList(head);

    // Delete the 2nd and 5th student (only if enough students are present)
    if (n >= 2) {
        cout << "\nDeleting the 2nd student...\n";
        deleteStudent(&head, 2);
    }
    if (n >= 5) {
        cout << "Deleting the 5th student...\n";
        deleteStudent(&head, 5);
    }
```

```
    // Display the updated list after deletion
    cout << "\nUpdated Student List:\n";
    displayList(head);

    return 0;
}
```