CAUC201 – Data Structures and Algorithms

Assignment Submission

Assignment : 4

Student ID: 24BCA095

1.

```
#include <iostream>
#include <malloc.h>
using namespace std;
int count;
struct Linked {
   int data;
   char arr[20];
   struct Linked * next;
}*node, *head, *last, *temp;

void counti(){
   temp=head;
   count = 0;
   while (temp != NULL){
      count++;
      temp=temp->next;
   }
}

void create_at_begin (){
   node = (struct Linked * )malloc (sizeof(struct Linked));
   cout<< "Enter data & arr : ";
   cin >> node->data ;
   cin >> node->arr;
   node->next=NULL;
   if(head==NULL)
      head=last=node;
   else{
      node->next=head;
      head=node;
   }
}

void create_at_last(){
   node = (struct Linked * ) malloc (sizeof(struct Linked));
   cout << "Enter data & arr : ";
   cin >> node->data;
   cin >> node->arr;
   node->next=NULL;
   if (head==NULL)
      head=last=node;
   else{
      last->next=node;
      last=node;
```

```
    }
}

void create_at_pos(){
    int pos;
    cout << "Enter position : ";
    cin >> pos;
    counti();
    if ( pos == 0 ){
        create_at_begin();
        return;
    }
    else if (pos == count){
        create_at_last();
        return;
    }
    else if (pos > 0 && pos < count){
        node=(struct Linked * ) malloc (sizeof(struct Linked));
        cout << "Enter data & arr : ";
        cin >> node->data;
        cin >> node->arr;
        node->next=NULL;
        if (head == NULL)
            head = last= node;
        else {
            temp = head;
            for(int i = 0 ; i < pos - 1 ; i++){
                temp=temp->next;
            }
            node->next = temp->next;
            temp->next = node;
        }
    }
}

void delete_at_begin(){
    if (head == NULL) {
        cout << "List is empty!" << endl;
        return;
    }
    temp = head;
    head = head->next;
    free(temp);
    if (head == NULL) last = NULL;
}

void delete_at_end(){
    if (head == NULL) {
        cout << "List is empty!" << endl;
        return;
```

```
    }
    if (head->next == NULL) {
      free(head);
      head = last = NULL;
      return;
    }
    temp = head;
    while (temp->next->next != NULL) {
      temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
    last = temp;
}

void delete_at_pos(){
    int pos;
    cout << "Enter position : ";
    cin >> pos;
    counti();
    if (pos == 0 ){
      delete_at_begin();
      return;
    }
    else if (pos == count ){
      delete_at_end();
      return;
    }
    else if (pos > 0 && pos < count){
      temp = head;
      for(int i = 0 ; i < pos - 1 ; i ++){
        temp=temp->next;
      }
      struct Linked *t1 = temp->next;
      temp->next = t1->next;
      free(t1);
    }
}
void display(){
    temp=head;
    while (temp!=NULL){
      cout << "----------"<<endl;
      cout << "Data : " << temp->data<<endl;
      cout << "Arr : " << temp->arr<<endl;
      temp=temp->next;
    }
}

int main() {
    int choice;
```

```
    while (1) {
      cout << "\n--- Singly Linked List Operations ---\n";
      cout << "1. Insert at Begin\n2. Insert at Last\n3. Insert at Position\n";
      cout << "4. Delete at Begin\n5. Delete at Last\n6. Delete at Position\n";
      cout << "7. Display\n8. Count Nodes\n9. Exit\n";
      cout << "Enter your choice: ";
      cin >> choice;

      switch (choice) {
        case 1: create_at_begin(); break;
        case 2: create_at_last(); break;
        case 3: create_at_pos(); break;
        case 4: delete_at_begin(); break;
        case 5: delete_at_end(); break;
        case 6: delete_at_pos(); break;
        case 7: display(); break;
        case 8: counti(); cout << "Total Nodes: " << count << endl; break;
        case 9: exit(0);
        default: cout << "Invalid choice!" << endl;
      }
    }
  }
```

2.

```
#include <iostream>
#include <malloc.h>

using namespace std;

int counto=-1;

struct Double_Linkedlist{
  int data;
  struct Double_Linkedlist * next;
  struct Double_Linkedlist * prev;
}*node,*head,*last,*temp , *t1;

void display(){
  temp=head;
  if(head == NULL){
    cout << "List is NULL ";
    return;
  }
  int i = -1;
  while(temp!=NULL){
    i++;
    cout <<endl << "------" << i << "-----" << endl;
    cout << "Data is : " << temp->data;
    cout << endl << "------------" <<endl;
    temp=temp->next;
  }
```

```
}
void counti(){
   temp=head;
   counto=-1;
   while(temp!=NULL ){
      counto++;
      temp=temp->next;
   }
}
void create_at_begin(){
   node=(struct Double_Linkedlist*) malloc (sizeof(struct Double_Linkedlist));
   cout << "Enter data : ";
   cin >> node->data;
   node->next = node->prev = NULL;
   if ( head == NULL )
      head = last = node;
   else{
      node->next = head;
      head->prev=node;
      head=node;
   }
}

void create_at_last(){
   node=(struct Double_Linkedlist* ) malloc (sizeof(struct Double_Linkedlist));
   cout << "Enter data : ";
   cin >> node->data;
   node->prev=node->next=NULL;
   if (head == NULL)
      head=last=node;
   else{
      last->next=node;
      node->prev=last;
      last=node;
   }
}

void create_at_position(){
   int pos;
   cout << "Enter position : ";
   cin >> pos;
   counti();
   if(pos == 0 ){
      create_at_begin();
      return;
   }
   else if (pos == counto){
      create_at_last();
      return;
   }
```

```
    else if (pos > 0 && pos < counto ) {
      node=(struct Double_Linkedlist* ) malloc (sizeof(struct Double_Linkedlist));
      cout << "Enter data : ";
      cin >> node->data;
      node->next=node->prev=NULL;
      if(head==NULL)
        head=last=NULL;
      else{
        temp=head;
        for(int i = 0 ; i < pos - 1 ; i ++){
          temp=temp->next;
        }
        node->next=temp->next;
        node->prev=temp;
        temp->next->prev = node;
        temp->next=node;
      }
    }
}

void delete_at_begin(){
    if (head == NULL ){
      cout << "List is empty \n" ;
      return;
    }
    else if (head->next == NULL){
      free (head);
      head =last = NULL;
    }
    else {
      temp=head;
      head=head->next;
      head->prev=NULL;
      free(temp);
    }
}

void delete_at_last(){
    if (head == NULL ){
      cout << "List is empty \n" ;
      return;
    }
    else if (head->next == NULL){
      free (head);
      head =last = NULL;
    }
    else {
    temp=last;
    last=last->prev;
    last->next=NULL;
```

```
    free(temp);
    }
}

void delete_at_pos(){
    int pos;
    cout << "Enter position : ";
    cin >> pos;
    counti();
    if (pos == 0){
        delete_at_begin();
        return;
    }
    else if (pos == counto){
        delete_at_last();
        return;
    }
    else if (pos > 0 && pos < counto) {
        temp = head;
        for(int i = 0 ; i < pos -1 ; i++)
            temp=temp->next;
        t1 = temp->next;
        temp->next=temp->next->next;
        if(t1->next!=NULL)
            temp->next->prev = temp;
        free(t1);
    }
    else
        cout << "Invalid Positon"<<endl;
}

int main(){

    int choice;
    head = last = NULL;

     while(1){
        cout << "\n1. Insert at beginning\n";
        cout << "2. Insert at end\n";
        cout << "3. Insert at position\n";
        cout << "4. Delete at beginning\n";
        cout << "5. Delete at end\n";
        cout << "6. Delete at position\n";
        cout << "7. Display\n";
        cout << "8. Count nodes\n";
        cout << "9. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch(choice){
```

```
            case 1:
               create_at_begin();
               break;
            case 2:
               create_at_last();
               break;
            case 3:
               create_at_position();
               break;
            case 4:
               delete_at_begin();
               break;
            case 5:
               delete_at_last();
               break;
            case 6:
               delete_at_pos();
               break;
            case 7:
               display();
               break;
            case 8:
               counti();
               cout << "Total number of nodes : "<<counto<<endl;
               break;
            case 9:
               return 0;
            default:
               cout << "Invalid choice!" << endl;
         }
      }
}
```

3.

```
#include <iostream>
#include <malloc.h>
#include <algorithm>
using namespace std;

int counto = -1;

struct SLL
{
   int data;
   struct SLL *next;
} *node, *head, *last, *temp, *headfin, *lastfin;

void display(struct SLL *h)
{
```

```
    if (h == NULL)
    {
        cout << "List is empty" << endl;
        return;
    }
    int i = -1;
    while (h != NULL)
    {
        cout << "-----" << ++i << "-----" << endl;
        cout << "Data : " << h->data;
        cout << endl
             << "------------" << endl;
        h = h->next;
    }
}

void counti()
{
    temp = head;
    counto = -1;
    while (temp != NULL)
    {
        counto++;
        temp = temp->next;
    }
}
void create_at_begin()
{
    node = (struct SLL *)malloc(sizeof(struct SLL));
    cout << "Enter data : ";
    cin >> node->data;
    node->next = NULL;
    if (head == NULL)
        head = last = node;
    else
    {
        node->next = head;
        head = node;
    }
}

void create_at_last()
{
    node = (struct SLL *)malloc(sizeof(struct SLL));
    cout << "Enter data : ";
    cin >> node->data;
    node->next = NULL;
    if (head == NULL)
        head = last = node;
    else
```

```
    {
      last->next = node;
      last = node;
    }
}

void create_at_pos()
{
   int pos;
   cout << "Enter position : ";
   cin >> pos;
   counti();
   if (pos == 0)
      create_at_begin;
   else if (pos == counto)
      create_at_last;
   else if (pos < 0 && pos > counto)
      cout << "Invalid position\n";
   else
   {
      node = (struct SLL *)malloc(sizeof(struct SLL));
      cout << "Enter data : ";
      cin >> node->data;
      node->next = NULL;
      if (head == NULL)
         head = last = node;
      else
      {
         temp = head;
         for (int i = 0; i < pos - 1; i++)
            temp = temp->next;
         node->next = temp->next;
         temp->next = node;
      }
   }
}

void delete_at_front()
{
   if (head == NULL)
   {
      cout << "List is empty" << endl;
      return;
   }
   else if (head->next == NULL)
   {
      head = last = NULL;
      return;
   }
   else
```

```
    {
        temp = head;
        head = head->next;
        free(temp);
    }
}

void sum_of_list()
{
    int sum = 0;
    temp = head;
    while (temp != NULL)
    {
        sum += temp->data;
        temp = temp->next;
    }
    cout << "Sum of List : " << sum << endl;
}

void search_data()
{
    int key;
    cout << "Enter key : ";
    cin >> key;
    temp = head;
    while (temp != NULL)
    {
        if (key == temp->data)
        {
            cout << "Element is present" << endl;
            return;
        }
        else
        {
            temp = temp->next;
        }
    }
    cout << "Data isn't present" << endl;
}

void reverse_linkedlist()
{
    struct SLL *temp1, *temp2;
    while (head != NULL)
    {
        temp2 = head->next;
        head->next = temp;
        temp = head;
        head = temp2;
    }
```

```
    head = temp;
}

void sorting(struct SLL *h)
{
    if (h == NULL || h->next == NULL)
    {
        cout << "List is too small to be sorted " << endl;
        return;
    }
    struct SLL *i, *j;
    bool swapped;
    for (i = h; i->next != NULL; i = i->next)
    {
        swapped = false;
        for (j = i->next; j != NULL; j = j->next)
        {
            if (i->data > j->data)
            {
                swapped = true;
                swap(i->data, j->data);
            }
        }
        if (!swapped)
            break;
    }
}
SLL *head2, *last2;
void second_linked_list()
{
    int n;
    cout << "How many nodes in second list? ";
    cin >> n;

    for (int i = 0; i < n; i++)
    {
        node = (struct SLL *)malloc(sizeof(struct SLL));
        cout << "Enter data " << i + 1 << ": ";
        cin >> node->data;
        node->next = NULL;

        if (head2 == NULL)
            head2 = last2 = node;
        else
        {
            last2->next = node;
            last2 = node;
        }
    }
}
```

```
void merge()
{
   second_linked_list();
   sorting(head);
   sorting(head2);
   if (head == NULL)
   {
      head = head2;
      return;
   }
   temp = head;
   while (temp->next != NULL)
      temp = temp->next;
   temp->next = head2;
   cout << "Successfully Merged!!!" << endl;
   display(head);
}

void concatenate()
{
   if (head == NULL)
   {
      head = head2;
      return;
   }
   temp = head;
   while (temp->next != NULL)
      temp = temp->next;
   temp->next = head2;
   cout << "Concatenation Successfull" << endl;
}
void Union_SLL()
{
   second_linked_list();
   sorting(head);
   sorting(head2);
   SLL *t1 = head, *t2 = head2;
   SLL *headfin = NULL, *lastfin = NULL;

   while (t1 != NULL && t2 != NULL)
   {
      int val;
      if (t1->data < t2->data)
      {
         val = t1->data;
         t1 = t1->next;
      }
      else if (t1->data > t2->data)
      {
```

```
        val = t2->data;
        t2 = t2->next;
      }
      else
      {
        val = t1->data;
        t1 = t1->next;
        t2 = t2->next;
      }

      node = (SLL *)malloc(sizeof(SLL));
      node->data = val;
      node->next = NULL;

      if (headfin == NULL)
        headfin = lastfin = node;
      else
      {
        lastfin->next = node;
        lastfin = node;
      }
    }

    while (t1 != NULL)
    {
      node = (SLL *)malloc(sizeof(SLL));
      node->data = t1->data;
      node->next = NULL;
      if (headfin == NULL)
        headfin = lastfin = node;
      else
      {
        lastfin->next = node;
        lastfin = node;
      }
      t1 = t1->next;
    }

    while (t2 != NULL)
    {
      node = (SLL *)malloc(sizeof(SLL));
      node->data = t2->data;
      node->next = NULL;
      if (headfin == NULL)
        headfin = lastfin = node;
      else
      {
        lastfin->next = node;
        lastfin = node;
      }
```

```
      t2 = t2->next;
   }
   cout << "Union of Linked Lists:" << endl;
   display(headfin);
}
void Intersection_SLL()
{
   second_linked_list();
   sorting(head);
   sorting(head2);
   SLL *t1 = head, *t2 = head2;
   SLL *headfin = NULL, *lastfin = NULL;

   while (t1 != NULL && t2 != NULL)
   {
      if (t1->data < t2->data)
         t1 = t1->next;
      else if (t1->data > t2->data)
         t2 = t2->next;
      else
      {
         SLL *node = (SLL *)malloc(sizeof(SLL));
         node->data = t1->data;
         node->next = NULL;

         if (headfin == NULL)
            headfin = lastfin = node;
         else
         {
            lastfin->next = node;
            lastfin = node;
         }
         t1 = t1->next;
         t2 = t2->next;
      }
   }

   if (headfin == NULL)
      cout << "No common elements found." << endl;
   else
   {
      cout << "Intersection of Linked Lists:" << endl;
      display(headfin);
   }
}
int main()
{
   int choice;
   head = last = head2 = last2 = NULL;
```

```
while (true)
{
   cout << "\n========= LINKED LIST OPERATIONS =========\n";
   cout << "1.  Insert element at front (List 1)\n";
   cout << "2.  Delete element from front (List 1)\n";
   cout << "3.  Sum of elements (List 1)\n";
   cout << "4.  Count number of nodes (List 1)\n";
   cout << "5.  Search element in List 1\n";
   cout << "6.  Reverse List 1\n";
   cout << "7.  Display Lists\n";
   cout << "8.  Create/Modify Second Linked List\n";
   cout << "9.  Concatenate Lists (List1 + List2)\n";
   cout << "10. Merge Lists (Sorted)\n";
   cout << "11. Union of Lists\n";
   cout << "12. Intersection of Lists\n";
   cout << "13. Exit\n";
   cout << "Enter your choice: ";
   cin >> choice;

   switch (choice)
   {
   case 1:
      create_at_begin();
      break;
   case 2:
      delete_at_front();
      break;
   case 3:
      sum_of_list();
      break;
   case 4:
      counti();
      cout << "Total number of elements: " << counto + 1 << endl;
      break;
   case 5:
      search_data();
      break;
   case 6:
      cout << "Before Reversing:\n";
      display(head);
      reverse_linkedlist();
      cout << "After Reversing:\n";
      display(head);
      break;
   case 7:
      cout << "\n--- First List ---\n";
      display(head);
      cout << "\n--- Second List ---\n";
      display(head2);
      break;
```

```
        case 8:
            second_linked_list();
            break;
        case 9:
            concatenate();
            display(head);
            break;
        case 10:
            merge();
            break;
        case 11:
            Union_SLL();
            break;
        case 12:
            Intersection_SLL();
            break;
        case 13:
            cout << "Exiting program..." << endl;
            exit(0);
        default:
            cout << "Invalid choice. Try again." << endl;
            break;
        }
    }
}
```

4.

5.

```
#include <iostream>
#include <malloc.h>
using namespace std;

struct SLL
{
    int data;
    struct SLL *next;
} *node, *head, *last, *temp;

void display()
{
    if (head == NULL)
    {
        cout << "List is empty." << endl;
        return;
    }
    int i = 1;
    temp = head;
    while (temp != NULL)
    {
        cout << "Node " << i++ << ": " << temp->data << endl;
        temp = temp->next;
```

```
    }
}

void insert_element()
{
    node = (struct SLL *)malloc(sizeof(struct SLL));
    cout << "Enter data: ";
    cin >> node->data;
    node->next = NULL;

    if (head == NULL)
        head = last = node;
    else
    {
        last->next = node;
        last = node;
    }
    cout << "Element inserted successfully!" << endl;
}

void delete_from_end()
{
    if (head == NULL)
    {
        cout << "List is empty." << endl;
        return;
    }

    if (head->next == NULL)
    {
        free(head);
        head = last = NULL;
        cout << "Last element deleted successfully!" << endl;
        return;
    }

    temp = head;
    while (temp->next->next != NULL)
        temp = temp->next;

    free(temp->next);
    temp->next = NULL;
    last = temp;
    cout << "Last element deleted successfully!" << endl;
}

void remove_duplicates()
{
    if (head == NULL)
    {
```

```
    cout << "List is empty." << endl;
    return;
  }

  struct SLL *temp1, *temp2;
  temp1 = head;

  while (temp1 != NULL && temp1->next != NULL)
  {
    temp2 = temp1;
    while (temp2->next != NULL)
    {
      if (temp1->data == temp2->next->data)
      {
        temp = temp2->next;
        temp2->next = temp2->next->next;
        free(temp);
      }
      else
        temp2 = temp2->next;
    }
    temp1 = temp1->next;
  }

  cout << "Duplicates removed successfully!" << endl;
}

void count_duplicates()
{
  if (head == NULL)
  {
    cout << "List is empty." << endl;
    return;
  }

  int count = 0;
  struct SLL *temp1, *temp2;
  temp1 = head;

  while (temp1 != NULL)
  {
    int duplicate_found = 0;
    temp2 = temp1->next;
    while (temp2 != NULL)
    {
      if (temp1->data == temp2->data)
      {
        duplicate_found = 1;
        break;
      }
```

```
            temp2 = temp2->next;
        }
        if (duplicate_found)
            count++;
        temp1 = temp1->next;
    }

    cout << "Total number of duplicate elements: " << count << endl;
}

int main()
{
    int choice;
    while (true)
    {
        cout << "\n========== SINGLE LINKED LIST MENU ==========\n";
        cout << "1. Insert an element\n";
        cout << "2. Delete last element\n";
        cout << "3. Display all elements\n";
        cout << "4. Remove duplicates (unsorted list)\n";
        cout << "5. Count total number of duplicate elements\n";
        cout << "6. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            insert_element();
            break;
        case 2:
            delete_from_end();
            break;
        case 3:
            display();
            break;
        case 4:
            remove_duplicates();
            break;
        case 5:
            count_duplicates();
            break;
        case 6:
            cout << "Exiting program..." << endl;
            exit(0);
        default:
            cout << "Invalid choice. Try again." << endl;
        }
    }
}
```

6.	#include <iostream>
	#include <malloc.h>
	using namespace std;
	
	struct SLL
	{
	int data;
	struct SLL *next;
	} *head = NULL, *last = NULL, *node, *temp;
	
	void insert_element()
	{
	node = (struct SLL *)malloc(sizeof(struct SLL));
	cout << "Enter data: ";
	cin >> node->data;
	node->next = NULL;
	
	if (head == NULL)
	head = last = node;
	else
	{
	last->next = node;
	last = node;
	}
	
	cout << "Element inserted successfully!" << endl;
	}
	
	void delete_from_end()
	{
	if (head == NULL)
	{
	cout << "List is empty." << endl;
	return;
	}
	
	if (head->next == NULL)
	{
	free(head);
	head = last = NULL;
	cout << "Last element deleted successfully!" << endl;
	return;
	}
	
	temp = head;
	while (temp->next->next != NULL)
	temp = temp->next;

```
    free(temp->next);
    temp->next = NULL;
    last = temp;
    cout << "Last element deleted successfully!" << endl;
}

void display()
{
    if (head == NULL)
    {
        cout << "List is empty." << endl;
        return;
    }

    int i = 1;
    temp = head;
    while (temp != NULL)
    {
        cout << "Node " << i++ << ": " << temp->data << endl;
        temp = temp->next;
    }
}

void print_and_count_primes()
{
    if (head == NULL)
    {
        cout << "List is empty." << endl;
        return;
    }

    temp = head;
    int count = 0;
    cout << "Prime numbers in the list: ";

    while (temp != NULL)
    {
        int num = temp->data;
        int isPrime = 1;

        if (num <= 1)
            isPrime = 0;
        else
        {
            for (int i = 2; i < num; i++)
            {
                if (num % i == 0)
                {
                    isPrime = 0;
                    break;
```

```
            }
        }
    }

    if (isPrime)
    {
        cout << num << " ";
        count++;
    }

    temp = temp->next;
  }

  if (count == 0)
    cout << "None";

  cout << "\nTotal prime numbers: " << count << endl;
}

int main()
{
  int choice;
  while (1)
  {
    cout << "\n========== SINGLE LINKED LIST MENU ==========\n";
    cout << "1. Insert an element\n";
    cout << "2. Delete element from end\n";
    cout << "3. Display all elements\n";
    cout << "4. Print all Prime numbers and count total\n";
    cout << "5. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice)
    {
    case 1:
      insert_element();
      break;
    case 2:
      delete_from_end();
      break;
    case 3:
      display();
      break;
    case 4:
      print_and_count_primes();
      break;
    case 5:
      cout << "Exiting program..." << endl;
      exit(0);
```

```
        default:
            cout << "Invalid choice. Try again." << endl;
        }
    }
}
```