**Sri Sri University, Cuttack, Odisha.**

**Faculty of Science**

|  |  |
| --- | --- |
| **Program: B.Sc. – Computer Science, Data Science, & Environmental Science**  **(2020-23 Batch)**  **Subject Code/Subject Name: Data Structure Laboratory**  **Assignment –X** | |
| **Full Name of the Student:** | VINAYAK SANJAY CHAVAN |
| **Full Roll Number:** | BCS-011 |
| **Program:** | B.Sc. (Computer Sc.) / B.Sc. (Data Sc.) / B.Sc. (Env. Sc.) |
| **Date:** | 24th – 26th April, 2021 (10.00 AM – 12.00 Noon) |
| **Signature** |  |

|  |  |
| --- | --- |
| **All Questions are compulsory** | **Total Marks: 120** |

|  |  |
| --- | --- |
| **Question (s)** | **Maximum Marks** |
| 1. Create a singly linked list of integers; write functions to add elements at different places of the linked list. Write functions to delete a node from different positions of the linked list. Write recursive and non-recursive functions to print the list in its usual order.   **Your code:** **#include<iostream>**  **#include<cstdlib>**  **typedef struct node{**  **int data;**  **struct node\* next;**  **}NODE;**  **using namespace std;**  **NODE\* create\_node(int);**  **NODE\* insert\_beg(NODE\*,int);**  **NODE\* insert\_loc(NODE\*,int,int);**  **NODE\* del\_loc(NODE\*,int);**  **int count\_nodes(NODE\*);**  **void rec\_frwd\_display(NODE\*);**  **int main(){**  **NODE \*mylinkedlist=NULL;**  **int count=0,choice;**  **do{**  **cout<< "Main menu:"<<endl;**  **cout<< "1. INSERT AT BEGINNING"<<endl;**  **cout<< "2. INSERT AT GIVEN LOCATION"<<endl;**  **cout<< "3. DELETE AT GIVEN LOCATION"<<endl;**  **cout<< "4. NO\_OF\_ELEMENTS"<<endl;**  **cout<< "5. RECURSIVE DISPLAY"<<endl;**  **cout<< "6. EXIT"<<endl;**  **cout<<"Enter the option :";**  **cin >>choice;**  **switch(choice){**  **case 1:**  **int val;**  **cout<<"Enter the elements to be inserted: ";**  **cin>>val;**  **mylinkedlist=insert\_beg(mylinkedlist,val);**  **break;**  **case 2:**  **int value,pos;**  **cout<<"Enter the elements to be inserted: ";**  **cin>>value;**  **cout<<"Enter the position to be inserted :";**  **cin>>pos;**  **mylinkedlist=insert\_loc(mylinkedlist,pos,value);**  **break;**  **case 3:**  **int position;**  **cout<<"Enter the position: ";**  **cin>>position;**  **mylinkedlist=del\_loc(mylinkedlist,position);**  **break;**  **case 4:**  **count=count\_nodes(mylinkedlist);**  **cout<<"The total number of nodes is: "<<count;**  **break;**  **case 5:**  **cout<<"The display in linkedlist are: ";**  **rec\_frwd\_display(mylinkedlist);**  **break;**  **case 6:**  **exit(0);**  **}**  **}**  **while(choice!=6);**  **return 0;**  **}**  **NODE\* create\_node(int val){**  **NODE \*newnode;**  **newnode=(NODE\*)malloc(sizeof(NODE));**  **newnode->data=val;**  **newnode->next=NULL;**  **return newnode;**  **}**  **NODE\* insert\_beg(NODE \*head,int n){**  **NODE \*linked;**  **linked=create\_node(n);**  **if(head==NULL){**  **head=linked;**  **}else{**  **linked->next=head;**  **head=linked;**  **}**  **return head;**  **}**  **NODE\* insert\_loc(NODE \*head,int pos,int val){**  **NODE \*t,\*p;**  **if(pos<0 && pos>count\_nodes(head)){**  **cout<<"Cannot insert the value";**  **}else{**  **t=create\_node(val);**  **p=head;**  **for(int i=0;i<pos-1 && p!=NULL ;i++){**  **p=p->next;**  **}**  **t->next=p->next;**  **p->next=t;**  **}**  **return head;**  **}**  **void rec\_frwd\_display(NODE \*head){**  **if(head!=NULL){**  **cout<<head->data<<" ";**  **rec\_frwd\_display(head->next);**  **}**  **}**  **NODE\* del\_loc(NODE \*head,int index){**  **NODE \*q,\*p=head;**  **if(index<1 && index>count\_nodes(head)){**  **cout<<"invalid";//invalid index**  **}else{**  **for(int i=0;i<index-1;i++){**  **q=p;//to get the previous node**  **p=p->next;//at the end of while loop it will get the node in the index**  **}**  **q->next=p->next;//previous node next will get assigned by the index node next**  **p->next=NULL;//null the index node next**  **free(p);//free the memory**  **}**  **return head;**  **}**  **int count\_nodes(NODE \*head){**  **int result=0;**  **NODE\* copy=head;**  **while(copy!=NULL){**  **copy=copy->next;**  **result++;**  **}**  **return result;**  **}**  **Screenshot of output:** | 20 |
| 1. Write a function that deletes each even numbered node from a null terminated singly list. Write function to print the list in a recursive manner.   **Your code:** **#include<iostream>**  **#include<cstdlib>**  **typedef struct node{**  **int data;**  **struct node\* next;**  **}NODE;**  **using namespace std;**  **NODE\* create\_node(int);**  **NODE\* insert\_beg(NODE\*,int);**  **int count\_nodes(NODE\*);**  **void del\_alt(NODE\*);**  **void rec\_frwd\_display(NODE\*);**  **int main(){**  **NODE \*mylinkedlist=NULL,\*p;**  **mylinkedlist=insert\_beg(mylinkedlist,40);**  **mylinkedlist=insert\_beg(mylinkedlist,50);**  **mylinkedlist=insert\_beg(mylinkedlist,60);**  **mylinkedlist=insert\_beg(mylinkedlist,70);**  **mylinkedlist=insert\_beg(mylinkedlist,80);**  **mylinkedlist=insert\_beg(mylinkedlist,90);**  **mylinkedlist=insert\_beg(mylinkedlist,100);**  **rec\_frwd\_display(mylinkedlist);//100,90,80,70,60,50,40**  **cout<<endl;**  **del\_alt(mylinkedlist);//100,80,60,40**  **rec\_frwd\_display(mylinkedlist);//100 90 70 50**  **cout<<endl;**  **}**  **NODE\* create\_node(int val){**  **NODE \*newnode;**  **newnode=(NODE\*)malloc(sizeof(NODE));**  **newnode->data=val;**  **newnode->next=NULL;**  **return newnode;**  **}**  **NODE\* insert\_beg(NODE \*head,int n){**  **NODE \*linked;**  **linked=create\_node(n);**  **if(head==NULL){**  **head=linked;**  **}else{**  **linked->next=head;**  **head=linked;**  **}**  **return head;**  **}**  **int count\_nodes(NODE \*head){**  **int result=0;**  **NODE\* copy=head;**  **while(copy!=NULL){**  **copy=copy->next;**  **result++;**  **}**  **return result;**  **}**  **void rec\_frwd\_display(NODE \*head){**  **if(head!=NULL){**  **cout<<head->data<<" ";**  **rec\_frwd\_display(head->next);**  **}**  **return;**  **}**  **void del\_alt(NODE \*head){**  **NODE \*previous,\*current;**  **previous=head;//give the first to previous**  **current=head->next;//second node address in current node**  **while(previous!=NULL && current!=NULL){**  **previous->next=current->next;//as the previous will give to third node**  **free(current);**  **//current->next=previous->next;//then current to fourth node**  **//changed**  **//we are deleting current node, they are even and prev node are odd**  **//identify the next even number**  **previous = previous->next;**  **if (previous != NULL){ //condition required for prev reaches last node**  **current = previous->next;**  **}**  **//changed**    **}**      **}**  **Screenshot of output:** | 20 |
| 1. Cumulative list: Given a null-terminated linked list “input”, create a new null-terminated linked, list “output”, of the same length, such that the ith node of output contains the sum of the data in the input nodes up to and including the ith node of the first linked list.   **Your code:** **#include<iostream>**  **#include<cstdlib>**  **typedef struct node{**  **int data;**  **struct node\* next;**  **}NODE;**  **using namespace std;**  **NODE\* create\_node(int);**  **NODE\* insert\_beg(NODE\*,int);**  **int count\_nodes(NODE\*);**  **void cum\_sum\_nodes(NODE\*);**  **void rec\_frwd\_display(NODE\*);**  **int main(){**  **NODE \*mylinkedlist=NULL,\*p;**  **mylinkedlist=insert\_beg(mylinkedlist,40);**  **mylinkedlist=insert\_beg(mylinkedlist,50);**  **mylinkedlist=insert\_beg(mylinkedlist,60);**  **mylinkedlist=insert\_beg(mylinkedlist,70);**  **mylinkedlist=insert\_beg(mylinkedlist,80);**  **mylinkedlist=insert\_beg(mylinkedlist,90);**  **mylinkedlist=insert\_beg(mylinkedlist,100);**  **cout<<"The original linkedlist is: ";**  **rec\_frwd\_display(mylinkedlist);**  **cum\_sum\_nodes(mylinkedlist);**  **cout<<"\nNow the linkedlist is: ";**  **rec\_frwd\_display(mylinkedlist);**  **}**  **NODE\* create\_node(int val){**  **NODE \*newnode;**  **newnode=(NODE\*)malloc(sizeof(NODE));**  **newnode->data=val;**  **newnode->next=NULL;**  **return newnode;**  **}**  **NODE\* insert\_beg(NODE \*head,int n){**  **NODE \*linked;**  **linked=create\_node(n);**  **if(head==NULL){**  **head=linked;**  **}else{**  **linked->next=head;**  **head=linked;**  **}**  **return head;**  **}**  **int count\_nodes(NODE \*head){**  **int result=0;**  **NODE\* copy=head;**  **while(copy!=NULL){**  **copy=copy->next;**  **result++;**  **}**  **return result;**  **}**  **void cum\_sum\_nodes(NODE \*head){**  **NODE \*p=head;**  **int sum=0;**  **//NODE \*previous=NULL;**  **while(p!=NULL){**  **sum = sum + p->data;**  **p->data = sum;**  **//p->data=(previous->data)+(p->data);//it will sum the data**  **//previous=p;//give the previous node**  **p=p->next;//give the next node**  **}**  **//return head;**  **}**  **void rec\_frwd\_display(NODE \*head){**  **if(head!=NULL){**  **cout<<head->data<<" ";**  **rec\_frwd\_display(head->next);**  **}**  **return;**  **}**  **Screenshot of output:** | 20 |
| 1. Write a function makesparse that stores a sparse matrix in (k+1) x 3 form where k is the number of non-zero elements. Write functions to add two sparse matrices where all sparse matrices will be stored in (k+1) x 3 form. (*Implement using Linked List*)   **Your code:**  **Screenshot of output:** | 20 |
| 1. Implement STACK using linked list.   **Your code:** **#include<iostream>**  **#include<cstdlib>**  **using namespace std;**  **//creating the node structure**  **typedef struct node{**  **int data;**  **struct node\* next;**  **}STACK\_NODE;**  **//creating the stack structure for linkedlist**  **typedef struct stack{**  **int count;**  **struct node \*top;**  **}NEW\_STACK;**  **NEW\_STACK\* create\_stack();**  **NEW\_STACK\* push(NEW\_STACK\*,int);**  **int pop(NEW\_STACK\*);**  **void display(NEW\_STACK\*);**  **int peek(NEW\_STACK\*);**  **int main(){**  **NEW\_STACK \*s=NULL;**  **int choice;**  **s=create\_stack();**  **do{**  **cout<< "Main menu:"<<endl;**  **cout<< "1. PUSH"<<endl;**  **cout<< "2. POP"<<endl;**  **cout<< "3. PEEK"<<endl;**  **cout<< "4. DISPLAY"<<endl;**  **cout<<"Enter the option :";**  **cin >>choice;**  **switch(choice){**  **case 1:**  **int val;**  **cout<<"Enter the value: ";**  **cin>>val;**  **s=push(s,val);**  **break;**  **case 2:**  **int popped;**  **popped=pop(s);**  **cout<<"The popped value is: "<<popped;**  **break;**  **case 3:**  **int top\_element;**  **top\_element=peek(s);**  **cout<<"The top element in the stack is: "<<top\_element;**  **break;**  **case 4:**  **display(s);**  **break;**  **case 5:**  **exit(0);**  **}**  **}**  **while(choice!=4);**  **return 0;**  **}**  **NEW\_STACK\* create\_stack(){**  **NEW\_STACK \*s;**  **s=(NEW\_STACK\*)malloc(sizeof(NEW\_STACK));**  **s->count=0;**  **s->top=NULL;**  **return s;**  **}**  **NEW\_STACK\* push(NEW\_STACK \*newstack,int val){**  **STACK\_NODE \*newnode;**  **newnode=(STACK\_NODE\*)malloc(sizeof(STACK\_NODE));**  **newnode->data=val;**  **newnode->next=NULL;**  **if(newstack==NULL){**  **newstack->top=newnode;**  **}else{**  **newnode->next=newstack->top;**  **newstack->top=newnode;**  **}**  **newstack->count++;**  **return newstack;**  **}**  **int pop(NEW\_STACK \*newstack){**  **STACK\_NODE \*p;**  **int x;**  **p=newstack->top;**  **if(newstack==NULL){**  **x=-1;**  **}else{**  **newstack->top=p->next;**  **x=p->data;**  **p->next=NULL;**  **free(p);**  **}**  **return x;**  **}**  **int peek(NEW\_STACK \*newstack){**  **int x;**  **x=newstack->top->data;**  **return x;**  **}**  **void display(NEW\_STACK \*newstack){**  **STACK\_NODE \*p;**  **p=newstack->top;**  **while(p!=NULL){**  **cout<<p->data<<" ";**  **p=p->next;**  **}**  **}**  **Screenshot of output:** | 20 |
| 1. Implement linear QUEUE using linked list.   **Your code:** **#include<iostream>**  **#include<cstdlib>**  **using namespace std;**  **typedef struct node{**  **int data\_info;**  **struct node\* next;**  **}QUEUE\_NODE;**  **typedef struct queue{**  **QUEUE\_NODE\* front;**  **QUEUE\_NODE\* rear;**  **int count;**  **}NEW\_QUEUE;**  **NEW\_QUEUE\* create\_queue();**  **NEW\_QUEUE\* insert\_queue(NEW\_QUEUE\*,int);**  **int delete\_queue(NEW\_QUEUE\*);**  **void display(NEW\_QUEUE\*);**  **int main(){**  **NEW\_QUEUE \*q=NULL;**  **int choice;**  **q=create\_queue();**  **do{**  **cout<< "Main menu:"<<endl;**  **cout<< "1. INSERT"<<endl;**  **cout<< "2. DELETE"<<endl;**  **cout<< "3. DISPLAY"<<endl;**  **cout<<"Enter the option :";**  **cin >>choice;**  **switch(choice){**  **case 1:**  **int val;**  **cout<<"Enter the value: ";**  **cin>>val;**  **q=insert\_queue(q,val);**  **break;**  **case 2:**  **int popped;**  **popped=delete\_queue(q);**  **cout<<"The popped value is: "<<popped;**  **break;**  **case 3:**  **display(q);**  **break;**  **case 4:**  **exit(0);**  **}**  **}**  **while(choice!=4);**  **return 0;**  **}**  **//create a new queue and make front rear null and count to zero**  **NEW\_QUEUE\* create\_queue(){**  **NEW\_QUEUE \*queue;**  **queue=(NEW\_QUEUE\*)malloc(sizeof(NEW\_QUEUE));**  **queue->front=NULL;**  **queue->rear=NULL;**  **queue->count=0;**  **return queue;**  **}**  **//inserting queue**  **NEW\_QUEUE\* insert\_queue(NEW\_QUEUE \*queue,int val){**  **QUEUE\_NODE \*newnode;//new node**  **newnode=(QUEUE\_NODE\*)malloc(sizeof(QUEUE\_NODE));//allocate memory**  **newnode->data\_info=val;//to give the value to new node in data info**  **newnode->next=NULL;//give the node address null**  **if(queue->front==NULL){//the queue is empty**  **queue->front=queue->rear=newnode;//the queue is empty so we will make the front and rear to new node**  **}else{**  **//if the queue is not empty then we will give address of newnode to rear**  **//change the rear to newnode as we will insert the queue from rear side.**  **queue->rear->next = newnode;**  **queue->rear=newnode;**  **}**  **queue->count++;//to count the nodes**  **return queue;**  **}**  **int delete\_queue(NEW\_QUEUE \*queue){**  **QUEUE\_NODE \*q;//to store the front node**  **int x;//to store the deleted data**  **if(queue==NULL){//if queue is empty no deletion can happen so x will return 99**  **x=-99;**  **}else{**  **q=queue->front;// q store queue front node**  **x=q->data\_info;//it will store the data**  **queue->front=q->next;//now the front will change to the next node**  **q->next=NULL;//and make the q next to null**  **free(q);//free memory**  **}**  **return x;//return the deleted integer**  **}**  **void display(NEW\_QUEUE \*queue){**  **QUEUE\_NODE \*q;**  **q=queue->front;//q points to front**  **while(q!=NULL){//upto front next with not null the while loop will run**  **cout<<q->data\_info<<" ";//print the queue data**  **q=q->next;//then it will help to point to next node**  **}**  **return;**  **}**  **Screenshot of output:** | 20 |