DIVISION: SE COMPUTERS

FACULTY: IMRAN MIRZA

SUBJECT: DATA STRUCTURES

DATA STRUCTURES

MINI PROJECT AND CASE STUDY 2019-20

PRESENTED BY

NIGEL CRASTO 06

ATHARVA DHANWATE 08

HARIHARAN R IYER 20

COURSE: DATA STRUCTURE	**SEMESTER:**III
COURSE CODE: -CSC 303	ASSESMENT: Group PRESENTATION and Execution
CASE STUDIES	**ORAL:** 25 MARKS
FACULTY INCHARGE: MR. MIRZA ALI IMRAN	

CASE STUDY PARTS	**CRITERIA**
Divide and Conquer **Decomposing the problem into modules.**	Abstraction
Knowledge and Understanding	
Identifying the best suited data structure for solving the sub problems with justification	Decision on the Data Structure to be used Appropriate program structures (arrays, loops, ifs, methods, etc.) Proper use of structures (variables, naming, data types, files, correct usage, etc.) Follows accepted principles of good software design
Define algorithms for various identified functions.	Algorithms
Thinking and Inquiry	
Code: **Implement the modules**	Code Program runs; evidence of testing; catches errors in input-output; “crash-free”; works as expected Graceful exits; logical data input; intuitive; proper and appropriate prompts; directs/controls user interaction Use of comments; proper style (indents/spacing); proper naming of objects/variables
Test Cases: **Test cases to test correctness of the solution**	All required functions are implemented Screen shoots of successful and unsuccessful output

CONTENTS

MINIPROJECT (TO DO LIST)

INTRODUCTION

The program “TO DO LIST” is a C program coded on a Unix platform and gcc compiler. It is a text based program. It is basically used to keep a track of all you to do activities like experiments and assignments. The data structure used in this project is Link List.

WHY THIS PROJECT

The most basic reason for selecting this project is our (student’s) need to keep an ongoing track of all the activities that are to be done within a deadline. We have limited this program to experiment track recording, but this can be extended to a larger scale. As of now this project helps a student to store his experiment details in a .dat file and also he can view them as and when he wants to. It also shows the overdue experiments and hence helps a student to manage his activities.

DATA STRUCTURE USED

The data structure used in our project is link list.

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.

Why Linked List?
Arrays can be used to store linear data of similar types, but arrays have the following limitations.
1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.
2) Inserting a new element in an array of elements is expensive because the room has to be created for the new elements and to create room existing elements have to be shifted.

For example, in a system, if we maintain a sorted list of IDs in an array id[].

id[] = [1000, 1010, 1050, 2000, 2040].

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).
Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved.

Advantages over arrays:
1) Dynamic size
2) Ease of insertion/deletion

Drawbacks:
1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we

cannot do binary search with linked lists efficiently with its default implementation. Read about it here.
2) Extra memory space for a pointer is required with each element of the list.
3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

Representation:
A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL.
Each node in a list consists of at least two parts:
1) data
2) Pointer (Or Reference) to the next node
In C, we can represent a node using structures. Below is an example of a linked list node with integer data.

ADT SPECIFICATIONS

1. struct entry* create():

This function takes void as a parameter. It is used to create a block of memory of the size equal to that of the structure node and sets its parameter – next as NULL. It prompts the user to enter date of submission, subject and Experiment number and stores it.

The function returns the address of the newly created node to the calling function.

2. int NoOfNodes(struct):

This function is used to count the number of nodes(i.e. the number of experiments) the user has entered in order to create that many number of bytes of space in the .dat file.

Return type is integer.

Argument is the address to the first element of the list

3. void add():

This is the driving function of the program. It calls the create() function to create block of memory. Once the create() function returns, the add() function adds the node(ie the experiment) to the already existing list, or creates a new list is one already doesn’t exist. This addition of a node to existing list is done in a sorted manner in ascending order as per the date of submission of the experiments.

The function returns void type.

4. void display():

This function displays all the experiments with their due dates that the user has fed into the program.

Return type of the function is NULL.

5. struct entry* del():

When called, this function automatically deletes those experiments that have already crossed the deadlines.

It compares the current CPU time and the date of submission of the experiments entered by the user and removes and displays those experiments that have already crossed the deadlines.

This function returns the address to the first node.

6. void removemanual():

This function lets the user to manually remove any of the experiment he/she desires from the todo list. It asks

for the name of the subject and the experiment number to the user and deletes that experiment if found in the list.

Return type is void.

PROGRAM SOURCE CODE:

```
#include<stdio.h>
#include<math.h>
#include<time.h>
#include<string.h>
#include<assert.h>

struct entry
{
   struct date
   {
      int day;
      int month;
      int year;
   }dd;
   char sub[20];
   int expno;
   struct entry *next;
}*start=NULL,*tail=NULL;

/*returns address of newly created block of memory*/
struct entry* create()
```

```
{
   struct entry *data;
   data=(struct entry*)malloc(sizeof(struct entry));
   printf("\nenter data of submission dd/mm/yyyy: ");
   scanf("%d%[/]%d%[/]%d",&data->dd.day,&data->dd.month,&data->dd.year);
   printf("\nenter name of subject: ");
   scanf("%s",data->sub);
   printf("\nenter experiment number: ");
   scanf("%d",&data->expno);

   data->next=NULL;
   return data;

}
/*
this function counts no of nodes to allocate that many bytes*number of bytes required for each instance of structure for dat file
*/
int NoOfNodes(struct entry *front)
{
   struct entry *temp=front;
```

```
    int count=0;
    while(temp!=NULL)
    {
        count++;
        temp=temp->next;
    }
    return count;
}
/*
This function is used to add new experiments to to do list
in  a sorted manner as per the submission dates.
An experiment with a nearer due date is stored first
*/

void add()
{
    struct entry *temp=create();
    if(start==NULL)
    {
        start=temp;
        tail=start;
    }
```

```
   else
   {
      struct entry *tempnode; //as a temporary variable to
hold address of start


      struct entry *current,*prev=start;
      current=start;            //to traverse the list


      int nodesize=temp->dd.year*1000+temp-
>dd.month*100+temp->dd.day;   //numerical
representation of due date of the node to be inserted
      while(prev!=NULL&&current!=NULL)
      {
         int currentsize=current->dd.year*1000+current-
>dd.month*100+current->dd.day;  //numerical
representation of the due date of the node iterated
         if(currentsize>=nodesize)      // i.e. if the new node
has due date prior to the current iterated node
         {
            if(current==start)        //if the new node is to be
inserted at the start itself
            {
               tempnode=start;
               start=temp;
```

```
          start->next=tempnode;
        }
        else if(prev->next=NULL)//prev=tail i.e if the new
node has to be inserted at the very end of the list
        {
          prev->next=temp;
          tail=temp;


        }
        else                //here, we insert the new node
between the current and the previous nodes
        {
          prev->next=temp;
          prev=prev->next;
          prev->next=current;
        }
        return;
      }

      else          //if the newly created node has due date
somewhere after the current node
      {
       prev=current;
       current=current->next;
      }


    }
    prev->next=temp;
    tail=prev;
    return;
  }
  int c=NoOfNodes(start);//to count number of nodes for
the fwrite function so as to allocate that many bytes for
the stream dynamically


  FILE *fp;        //file pointer
  temp=start;
  if((fp=fopen("abc.dat","a"))!=NULL)
  {
    fwrite(temp,sizeof(struct entry),c,fp);
    /*
    fwrite is used to write to file
    fwrite takes 4 arguements:the pointer to the
stream,size of each variable,total no of such sizes,file ptr


    */
```

```
        fclose(fp);
    }
    else
    {
     printf("\nerror in opening the File");
     exit(0);
    }
}
void display()
{
    struct entry *temp=start;
    printf("\n---------------------------------------------------------
\n");
    printf("\nDate of Submission\tSubject\tExperiment
Number\n---------------------------------------------------------
\n");

    while(temp!=NULL)
    {

        printf("%d/%d/%d\t\t%s\t%d\n",temp->dd.day,temp-
>dd.month,temp->dd.year,temp->sub,temp->expno);
```

```
        temp=temp->next;
    }
}
/*
this function deletes the experiment numbers that have
crossed due dates of submission.
it takes the current cpu time as input to compare the
current time and date of submission
any experiment with due date before the current system
time is removed automatically
*/
struct entry* del()
{

    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    printf("\ncurrent date and time: %d-%d-%d
%d:%d:%d\n", tm.tm_year + 1900, tm.tm_mon +
1,tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec);
    int currentTime=(tm.tm_year +
1900)*1000+(tm.tm_mon + 1)*100+tm.tm_mday;
//current date of system
    while(start!=NULL)
    {
        int subtime=start->dd.year*1000+start-
>dd.month*100+start->dd.day;  //sub date of start
        printf("\nOverdue Experiment:\n");
        if(subtime<currentTime) //crossed deadline
        {
            /*Print the current node*/
            printf("%d/%d/%d\t\t%s\t%d\n",start-
>dd.day,start->dd.month,start->dd.year,start->sub,start-
>expno);
            struct entry *temp=start;
            //push(temp);
            free(start);
            start=start->next;
        }
        else return start;
    }
}
/* this function lets the user to remove an experiment
manually. it asks the subjetc name and the exp no. as input
to search and delete the experimnet if found*/
void removemanual()
{
        char rmsub[20];
        int rmexp;
        char ch='n';
        printf("\nenter name of subject: ");
        scanf("%s",rmsub);
        printf("\nenter experiment number: ");
        scanf("%d",&rmexp);
        struct entry* curr=start,*prev;
        while(curr!=NULL)
        {
            if(!strcmp(curr->sub,rmsub))
            {
                if(curr->expno==rmexp)
                {
                    printf("\nexperiment found!! Sure to
delete?(y/n)");
                    ch=getche();
                    if(ch=='n')
                    {
                        printf("\ncanceled\n");
                        break;
                    }
                    if(curr==start)
                {
                   start=start->next;
                   free(curr);
                   curr=NULL;
                   printf("\n\tfreed\n");
                   break;
                }
                prev->next=curr->next;
                printf("\n\tfreed\n");
                free(curr);
             }
          }
          prev=curr;
          curr=curr->next;
       }
printf("\nexperiment not found!\n");
}



main()
{
   int ch=1;
```

```
    while(ch!=0){
        printf("\nPress 1 to add");
        printf("\nPress 2 to display");
        printf("\nPress 3 to remove experiments
automatically");
        printf("\npress 4 to remove experiments manually");
        printf("\npress 0 to exit\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:add();
                    break;
            case 2: display();
                    break;
            case 3: del();
                    break;
            case 4: removemanual();
                    break;
            case 0:
                    exit(0);
            default: printf("\ninvalid choice");
}}}
```

PROGRAM OUTPUT

```
Press 1 to add
Press 2 to display
Press 3 to remove experiments automatically
press 4 to remove experiments manually
press 0 to exit
1

enter data of submission dd/mm/yyyy: 10/10/2019

enter name of subject: dlda

enter experiment number: 1

Press 1 to add
Press 2 to display
Press 3 to remove experiments automatically
press 4 to remove experiments manually
press 0 to exit
1

enter data of submission dd/mm/yyyy: 15/10/2019
enter name of subject: dlda

enter experiment number: 2

Press 1 to add
Press 2 to display
Press 3 to remove experiments automatically
press 4 to remove experiments manually
press 0 to exit
1

enter data of submission dd/mm/yyyy: 20/10/2019

enter name of subject: dlda

enter experiment number: 3

Press 1 to add
Press 2 to display
Press 3 to remove experiments automatically
press 4 to remove experiments manually
press 0 to exit
2

-------------------------------------------------------

Date of Submission      Subject Experiment Number
-------------------------------------------------------
10/10/2019            dlda    1
15/10/2019            dlda    2
20/10/2019            dlda    3

Press 1 to add
Press 2 to display
Press 3 to remove experiments automatically
press 4 to remove experiments manually
press 0 to exit
3

current date and time: 2019-10-12 13:44:13

Overdue Experiment:
10/10/2019            dlda    1
```

Overdue Experiment:

```
Press 1 to add
Press 2 to display
Press 3 to remove experiments automatically
press 4 to remove experiments manually
press 0 to exit
2

---------------------------------------------------------

Date of Submission      Subject Experiment Number
---------------------------------------------------------
15/10/2019              dlda    2
20/10/2019              dlda    3

Press 1 to add
Press 2 to display
Press 3 to remove experiments automatically
press 4 to remove experiments manually
press 0 to exit
```

```
4
enter name of subject: dlda

enter experiment number: 2

experiment found!! Sure to delete? (y/n) y
        freed

Press 1 to add
Press 2 to display
Press 3 to remove experiments automatically
press 4 to remove experiments manually
press 0 to exit
0

Process returned 0 (0x0)   execution time: 125.297 s
Press any key to continue.
```

PROBLEMS FACED

The first and the most difficult problem that with dealt with is the difference in the way each of us coded.

For example, we dint understand the variables that the other group member had declared.

So as a part of a solution to this we started adding comments and slowly the problem came down.

One of the technical problems that we faced was with the file handling.

First we used abc.txt, but then with shifted to abc.dat because the txt files gave lot of glitches, which were difficult to handle.

Similarly, there were many more problems but finally we did it.

CONCLUSION

Hence we learnt to effectively used linked lists in data management with the help of a real life example.

CASE STUDY (CASE STUDY 1)

INTRODUCTION

Given two numbers represented by two lists, write a function that returns sum list. The sum list is list representation of addition of two input numbers.

Example 1 Input: First List: 5->6->3 // represents number 365

Second List: 8->4->2 // represents number 248

Output Resultant list: 3->1->6 // represents number 613

Example 2 Input: First List: 7->5->9->4->6 // represents number 64957

Second List: 8->4 // represents number 48

Output Resultant list: 5->0->0->5->6 // represents number 65005

ADT SPECIFICATIONS

1. struct node* create():

This function creates a node that stores an integer info and returns the address of the created node to the calling function

2. struct node* insert():

this function takes NULL as parameter. It prompts the user to enter the number in string format; iterates through the entered string character by character

converting char data type to int and calling the create function with the converted integer as parameter

it then links all the nodes together to form a singly linked list.

3. void display(struct node*):

This function takes address of the first node as the argument. Starting from the first node, it iterates through each node, displaying the data of each node in the process.

4. int getsum(struct node*):

This function returns the reverse of sum of all elements in the list such that an element at any node starting from head of a linked list of length 'n' has a place value of 10n-1

Eg. If the list is 3->4->5, it returns 543

5. struct node* makelist(int):

This function accepts an integer as a parameter and creates a linked list of each digit of that integer such that the least significant digit becomes the head of the list and so on.

PROGRAM SOURCE CODE:

```
#include<stdio.h>

#include<string.h>

#include<assert.h>


struct node

{

   int data;

   struct node* next;

};

/*This function creates a node that stores an integer info
and returns the address of the created node to the calling
function*/

struct node* create(int info)

{

   struct node* temp=(struct node*)malloc(sizeof(struct
node));

   temp->next=NULL;

   temp->data=info;


   return temp;


}
```

```
/*this function takes NULL as parameter. It prompts the
user to enter the number in string format;iterates through
the entered string character by character

converting char data type to int and calling the create
function with the converted integer as parameter

it then links all the nodes together to form a singly linked
list*/

struct node* insert()

{

   struct node *tail,*head=NULL;

   char a[100];

   printf("\nenter a number: ");

   scanf("%s",a);

   strrev(a);

   //printf("\n%s",a);

   int i=0;

   int len=strlen(a);

   for(i=0;i<len;i++)

   {

      a[i]=(int)a[i]-48;  ///character to integer

      struct node* temp=create(a[i]); ///temp stores the
address of new node

      if(head==NULL)
```

```
        {
            head=tail=temp;
        }
        else
        {
            tail->next=temp;
            tail=tail->next;
        }
    }
    return head;

}
/*to display the list*/
void display(struct node* head)
{
    printf("\nThe list is:\n");
    while(head!=NULL)
    {
        printf("%d ",head->data);
        if(head->next !=NULL)
            printf("-> ");
        head=head->next;
    }
    printf("\n");
}
/*this function returns the sum of all the elements in the
list
*/
int getsum(struct node* head)
{
    int rem=0,sum=0;
    //get sum of all elements in the node. as we start from
the head, sum is in reversed order
    while(head!=NULL)
    {
        sum=sum*10+head->data;
        head=head->next;


    }
    //reverse the obtained sum
    int temp=0;
    while(sum>0)
    {
        rem=sum%10;
        temp=temp*10+rem;
        sum=sum/10;
    }
    return temp;
}
/*This function is used to create a lined list of the obtained sum of the two linked lists from the getsum() method*/
struct node* makelink(int sum)
{
    struct node* head=NULL,*tail,*temp;
    int rem;
    while(sum>0)
    {
        rem=sum%10;
        sum=sum/10;
        temp=create(rem);
        if(head==NULL)
        {
            head=tail=temp;
        }
        else
        {
            tail->next=temp;
            tail=tail->next;
        }
    }
    return head;
}
int main()
{
    struct node *r1,*r2,*r3;
    int sum,sum1,sum2,temp;
    r1=insert();
    display(r1);
    r2=insert();
    display(r2);
    sum1=getsum(r1);///obtain sum of elements in LL1
    sum2=getsum(r2);
    sum=sum1+sum2;///obtain sum of both LL
    printf("\nsum= %d\n",sum);
    r3=makelink(sum);///create linked list with digits of
'sum' as nodes->data
    display(r3);
}
```

PROGRAM OUTPUT

```
enter a number: 123


The list is:

3 -> 2 -> 1


enter a number: 30


The list is:

0 -> 3


sum= 126


The list is:

6 -> 2 -> 1

-------------------------------

Process exited after 4.937 seconds with return value 10

Press any key to continue
```

REFERENCES

WEBSITES

- https://www.learn-c.org/
- https://www.programiz.com/c-programming
- https://www.w3schools.in/c-tutorial/
- https://www.codecademy.com/learn/learn-c

BOOKS

- Data Structures Using C (Book by Reema Taneja)
- Data Structures with C (Schaum's Outline Series)