DATA STRUCTURE NOTES

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

BASIC CONCEPTS :

1.What is an ARRAY:-

An array is a collection of similar data type. It contain homogeneous data;
Code to initiate an array in c:

```
Void main(){
    Int arr[5];
    Int i;
    Printf("ENTER ELEMENT IN ARRAY :")
    For (i=0;i<5;i++){
        Scanf("%d",&a[i]);              //to put data in array
        }
        Printf("YOUR ENTERED ELEMENTS ARE :")
    For (i=0;i<5;i++){
        Printf("%d",a[i]);              //to fetch data from array
        }
    }
```

NOTE: The data created in array allocate memory in contiguous manner it means that the memory is fixed. we can also set the size of array at rum time but by doing so it can't behave like dynamic memory allocation.

2.What is a POINTER:-

A pointer is an address variable which used to store the address of variable or data .it interacts with the **heap memory.**
Code to initiate a Pointer in c:

```
Void main(){
    Int a=10;
    Int *p;
    p=&a;                   //address of a is assigned to the pointer variable

    printf("%d",a)          // it will print the value of a =10
    printf("%d",&a)         // it will print the address of a =67850;
    printf("%d",p)          // it will print the address value which is stored in p =67850
    printf("%d",&p)         // it will print the address of p =87850
    printf("%d",*p)         // it will print the data inside the p=10;
}
```

3.What is a Structure:-

It is a user defined data type which hold more than one element of different data types. It contain heterogenous data like(it include int data ,char data ,float data).it can't allocate memory directly until we assign a variable to indicate it in the main memory. It is ideal to define your structure outside the function in global level.
Code to initiate an array in c:

```
#include<stdio.h>
    Struct employee{
        Int eno;
        Char name[30];
        Float salary;
        };
    Void main(){
        Struct employee  e={1001,'samundar',50000} //directly putting data into structure
        Prinf("employee id : %d   employee name: %s  salary: %f",e.eno,e.name,e.salary);}
```

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

4.Parameter passing to a function techniques :-

a. Pass by value:

In this method value is passed to function and value is only carried to the function as a photo copy in other word we can say that we can perform only read operation not the write operation now whatever task we performed inside the function where value is called is remained only inside that function we can't able to modify data of the original data variable this happens because we create a another variable and stored the modified data on that not in the original that's why it reflected on the same function not globally . For example we can perform addition of three number using call by value but can't perform swapping of two numbers using this method

Code to perform pass by value operation in c:

```
        #include<stdio.h>
        Void main(){
                Int a ,b
                Printf('enter any 2 number');
                Scanf('%d %d' ,&a,&b);      // initiate a =10 and b=20
                Printf("before swapping values of  a:%d  and  b=%d",a,b );          //    a= 10  b=10
                swap( a,b);
                Printf("after swapping values of  a:%d  and  b=%d",a,b );           //a=10   b=20
}
Void swap(int a ,int b){
        Int temp;
        temp = a;
        a=b;
        b=temp
Printf("inside function  values of  a:%d  and  b=%d",a,b );                        //a=20   b=10
}
```

! PROGRAM FAILED

a. Pass by reference:

In this method address of data is passed to function and it means it directly interact with the address stored in heap. Whatever task is performed it is going to stored in the same variable address that's why the change takes place globally and reflected on the same block as well as another block.

Code to perform pass by value operation in c:

```
#include<stdio.h>
        void main(){
                int a ,b;
                printf("enter any 2 no");
                scanf("%d %d" ,&a,&b);              // initiate a =10 and b=20
                printf("before swapping values of  a:%d  and  b=%d \n",a,b );          //    a= 10  b=10
                swap( &a,&b);                       //here we passing the address to the function.
                printf("after swapping values of  a:%d  and  b=%d \n",a,b );           //a=10   b=20
}
void swap(int *a ,int *b){
        int temp;
        temp = *a;
        *a=*b;
        *b=temp;
printf("inside function  values of  a:%d  and  b=%d \n",*a,*b );                    //a=20   b=10
}
```

! PROGRAM SUCESSFULLY EXECUTED

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

1.What is a RECURSION or a RECURSIVE function: -

A recursion is a function which call itself. There must be a base condition which helps to terminate the function otherwise it will call itself infinitely.
Code to perform Recursive function in c:

```
# #include<stdio.h>
        void main(){
                int x;
                x=5;
                fun1(x);
                }
        void fun1(int x){
              if(x>0){
                printf("%d ",x);
                fun1(x-1);
}
}
```

OUTPUT:- **5 4 3 2 1**

In this case first the print statement executes and then the function called itself until the base condition false so it will start execution from first and print the statement then function loop get open and open .

```
# #include<stdio.h>
        void main(){
                int x;
                x=5;
                fun1(x);
                }
        void fun1(int x){
              if(x>0){
                fun1(x-1);
                printf("%d ",x);
}
}
```

OUTPUT:- **1 2 3 4 5**

In this case first the function call itself and then it print the value. So at first the function loop get open and open until base condition false and when base condition false then it come back and printing the value.

e.g:1. Factorial of a number using recursion:

```
#include<stdio.h>
void  main(){
        int i,f ;
        printf("Enter a number");
        scanf("%d",&i);
        f=fact(i);
                printf("factorial of your number : %d",f);
}
int fact (int i ){
        int f=1;
        if(i==0){
                return (1);
        }
        return(i*fact(i-1));
}
```

OUTPUT:- **Enter a number 5;**
Factorial of Your number 120

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

e.g:2. Prime numbers for n elements:

Prime number without recursion :-

```
#include<stdio.h>
int max_val(int array[],int size);
int lcm(int array[],int size);

void  main(){
        int size ,i;
        printf("Enter the size of array : ");
        scanf("%d",&size);
        int array[size];
        for(i=0;i<size;i++){
                printf("enter value : ");
                scanf("%d",&array[i]);
        }
   printf("\nNUMBERS : ");
   for(i=0;i<size;i++){
                printf("%d  ",array[i]);
        }
        printf("LCM is %d ",lcm(array,size));
}

int lcm(int array[],int size){
        int result,max,i,j;
        j = max = max_val(array,size);
        i = 0;
   while(i<size){
      if(j % array[i] == 0){
         i=i+1;
         max = result = j;
      }
      else
         j = j+ max;

   }
   return result;
}
int max_val(int array[],int size){
        int i,max= array[0];
        for (i=0;i<size;i++){
        if (array[i]>max)
          max = array[i];
        }
        return max;
}
```

//Used to find the lcm

Used to find max value

Prime number with recursion :-

```
#include<stdio.h>
```

pending

Written by:
SAMUNDAR SINGH

LINKED LIST

What is a linked list ?

A linked list, in simple terms, is a linear collection of data elements. A linked list does not store its elements in consecutive memory locations and the user can add any number of elements to it. linked list does not allow random access of data. Elements in a linked list can be accessed only in a sequential manner

Structure of linked list :-

A linked list in which every node contains two parts, an integer and a pointer to the next node. The left part contains the data and the right part contains pointer to next node. The last node address pointer is set as NULL. Linked lists contain a pointer variable START that stores the address of the first node in the list. We can traverse the entire list using START which contains the address of the first node; the next part of the first node in turn stores the address of its succeeding node.

Code which is base of the linked list : -

```
struct node
{
int data;
struct node *next;
};
```

This code says that we are creating a customized data structure which has one data type integer and other one is a pointer which is named as next ,which hold the address of the next node to which a connection is going to established;

NOTE :- This code must be mention outside main function. To avoid writing it multiple times.

1.Single Linked List:

A single linked list contain a set of data and a pointer (*next) which point the next address of the data. There is **start** or **head** pointer in which the starting address is stored .The first node contains a pointer to the second node. The second node contains pointer to the third node, ... and so on. The last node in the list has its next field set to NULL to mark the end of the list.

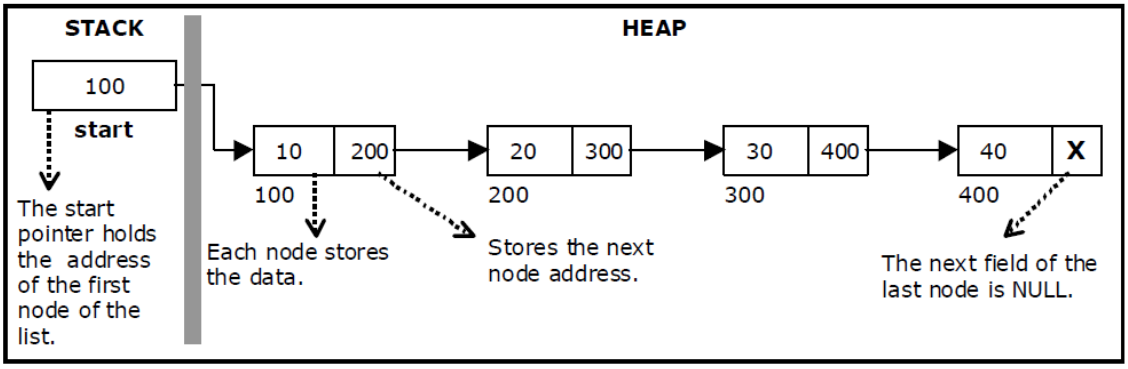


Figure 3.2.1. Single Linked List

CODING:-

We have to perform coding by creating function for each case. The various modules of function are stated here:-

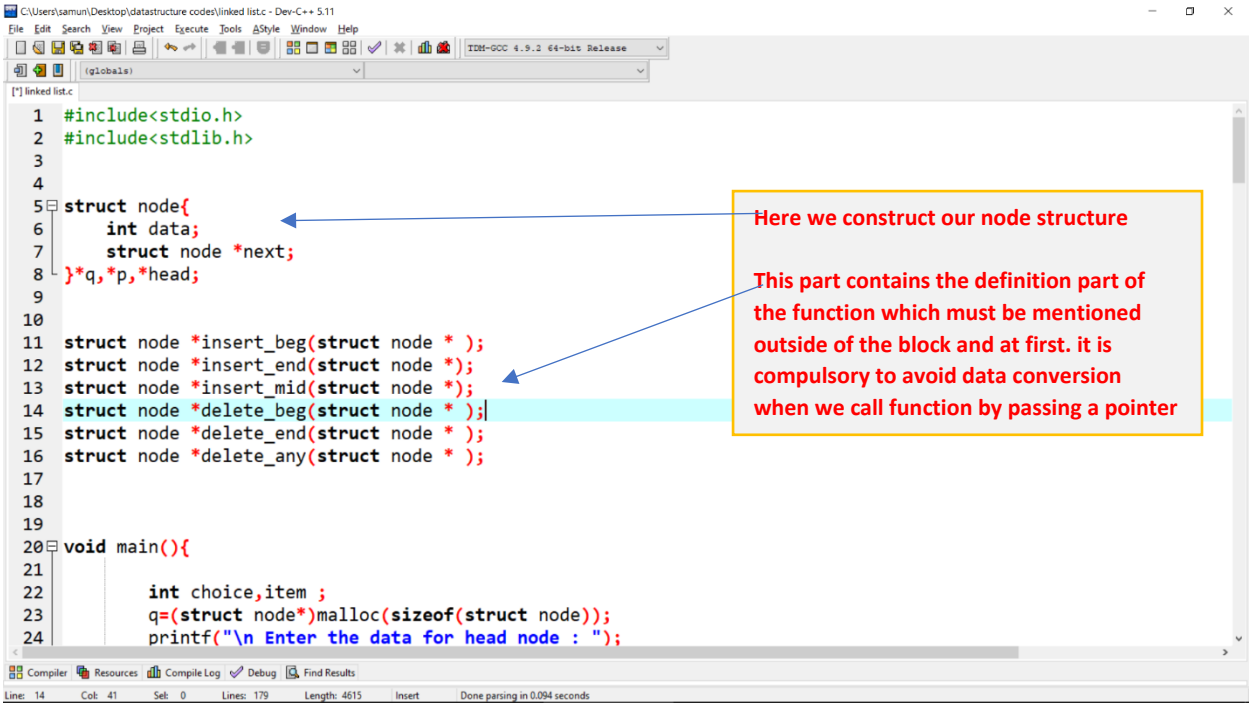
1.**Void main () :-** This is main function in which we call our function and create a menu driven program.
2. **Insert_beg() :-** In this function data is inserted at the first.
3. **Insert_end() :-** In this function data is inserted at the end.
4. **Insert_mid() :-** In this function data is inserted at the middle as the position mentioned by user.

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

5. **delete_beg() :-** In this function data is deleted at the first.
6. **delete_end() :-** In this function data is deleted at the end.
7. **delete_any() :-** In this function data is deleted from any where user want.

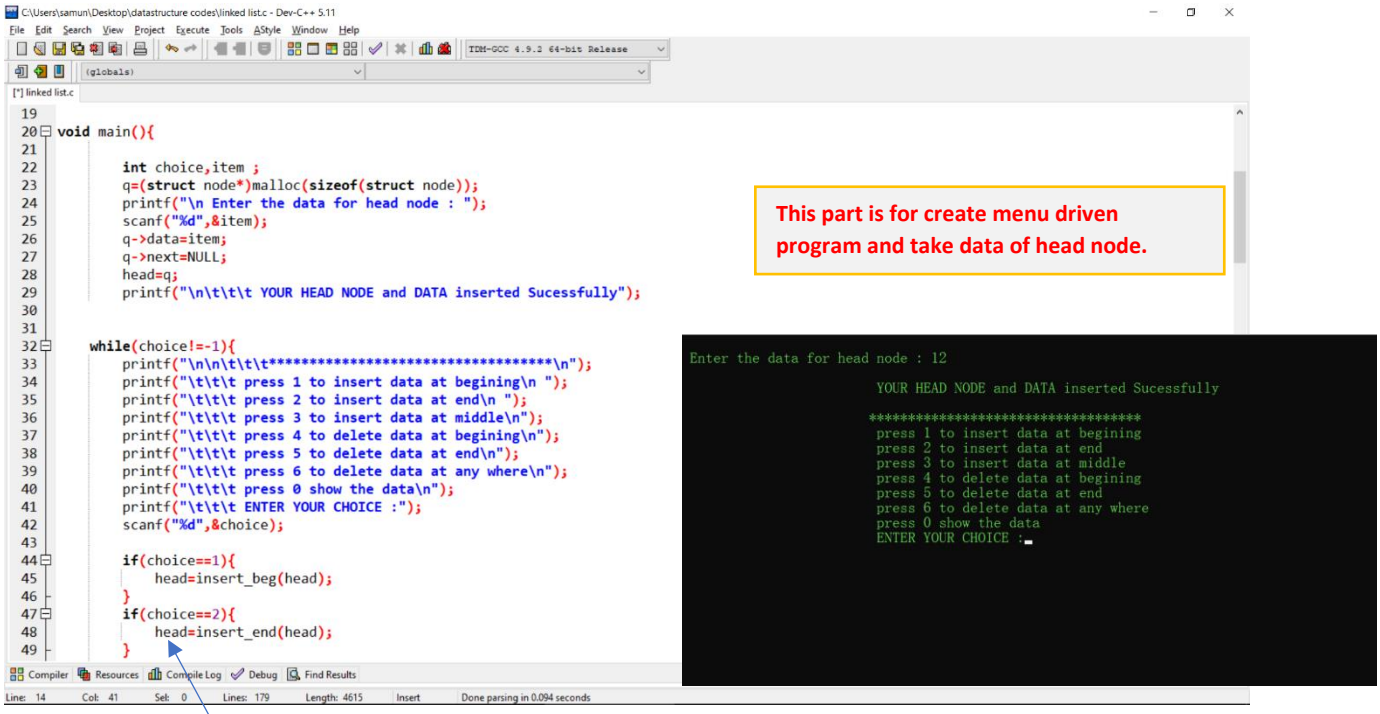
First I create a assumed list and now by using these function I add data dynamically, There is one another option that is first check whether the list is empty or not then insert the data from beginning for that we have to create a anther function IS_EMPTY() . so I skipped that and start coding by assumption :



```c
#include<stdio.h>
#include<stdlib.h>


struct node{
    int data;
    struct node *next;
}*q,*p,*head;


struct node *insert_beg(struct node * );
struct node *insert_end(struct node *);
struct node *insert_mid(struct node *);
struct node *delete_beg(struct node * );
struct node *delete_end(struct node * );
struct node *delete_any(struct node * );


void main(){

        int choice,item ;
        q=(struct node*)malloc(sizeof(struct node));
        printf("\n Enter the data for head node : ");
```

Here we construct our node structure

This part contains the definition part of the function which must be mentioned outside of the block and at first. it is compulsory to avoid data conversion when we call function by passing a pointer



```c
void main(){

        int choice,item ;
        q=(struct node*)malloc(sizeof(struct node));
        printf("\n Enter the data for head node : ");
        scanf("%d",&item);
        q->data=item;
        q->next=NULL;
        head=q;
        printf("\n\t\t\t YOUR HEAD NODE and DATA inserted Sucessfully");


    while(choice!=-1){
        printf("\n\n\t\t\t**********************************\n");
        printf("\t\t\t press 1 to insert data at begining\n ");
        printf("\t\t\t press 2 to insert data at end\n ");
        printf("\t\t\t press 3 to insert data at middle\n");
        printf("\t\t\t press 4 to delete data at begining\n");
        printf("\t\t\t press 5 to delete data at end\n");
        printf("\t\t\t press 6 to delete data at any where\n");
        printf("\t\t\t press 0 show the data\n");
        printf("\t\t\t ENTER YOUR CHOICE :");
        scanf("%d",&choice);

        if(choice==1){
            head=insert_beg(head);
        }
        if(choice==2){
            head=insert_end(head);
        }
```

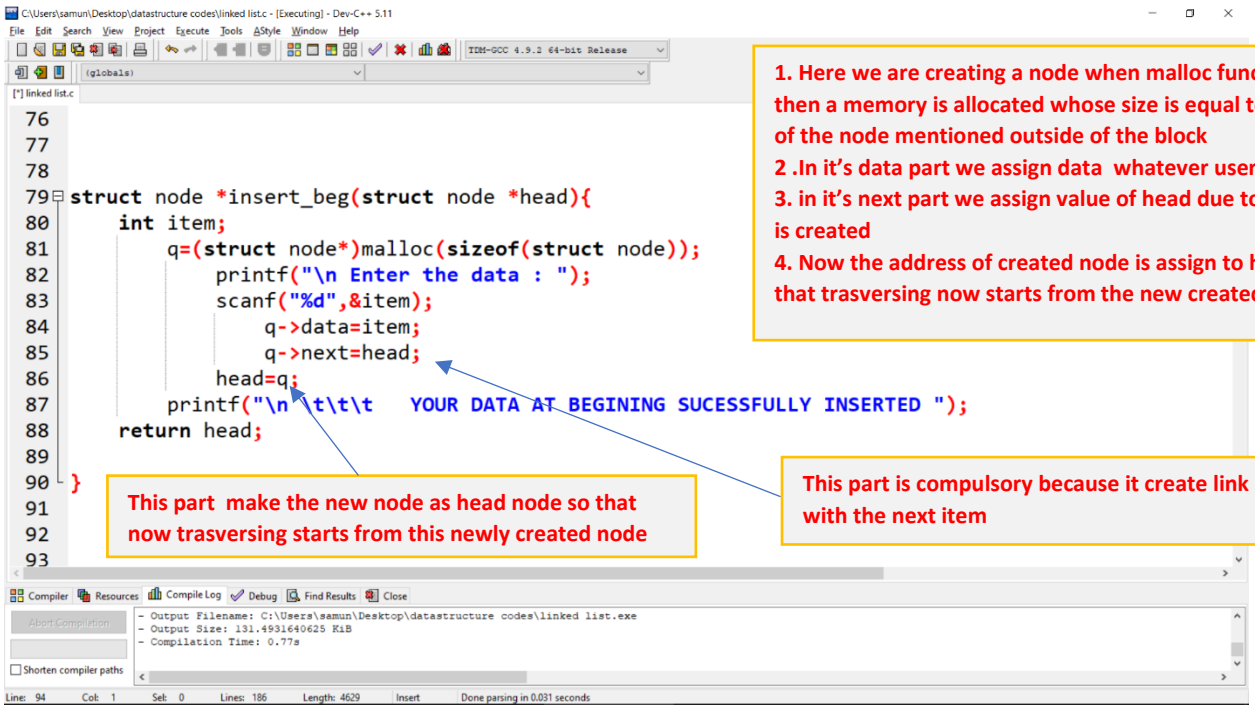
This part is for create menu driven program and take data of head node.

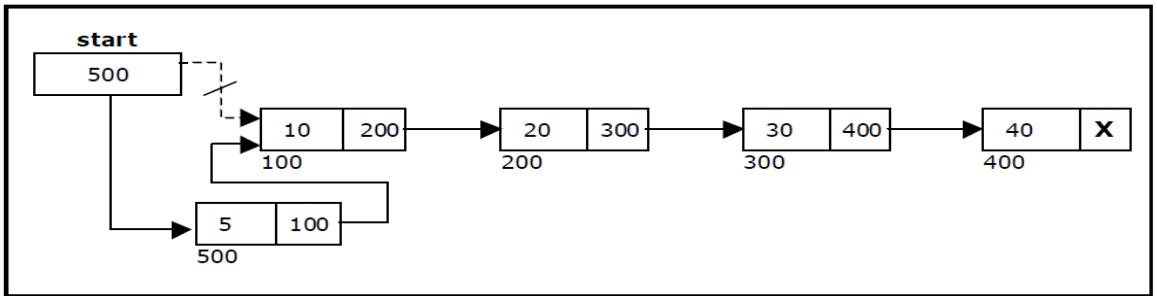
```
Enter the data for head node : 12

                        YOUR HEAD NODE and DATA inserted Sucessfully

                       **********************************
                        press 1 to insert data at begining
                        press 2 to insert data at end
                        press 3 to insert data at middle
                        press 4 to delete data at begining
                        press 5 to delete data at end
                        press 6 to delete data at any where
                        press 0 show the data
                        ENTER YOUR CHOICE :
```

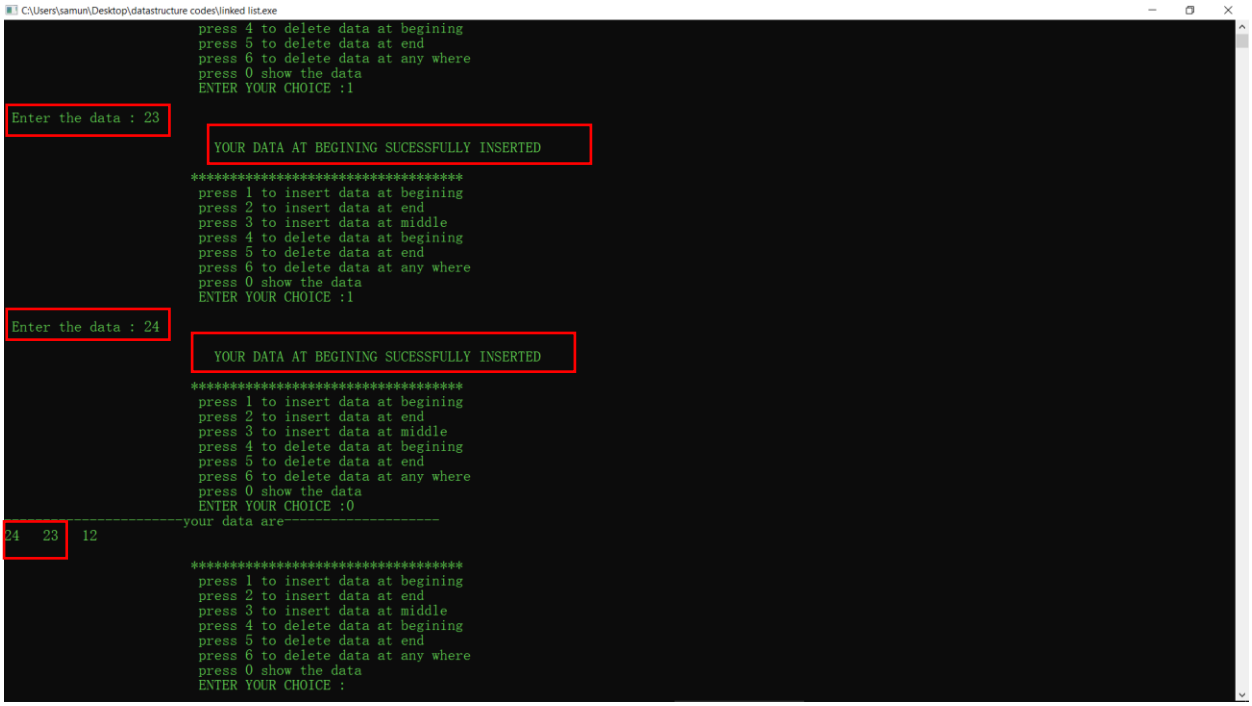
NOTE: Here function are called and whatever the function return will store in head which is a pointer type variable. This is important because of during traversing we need the head pointer shown below

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES







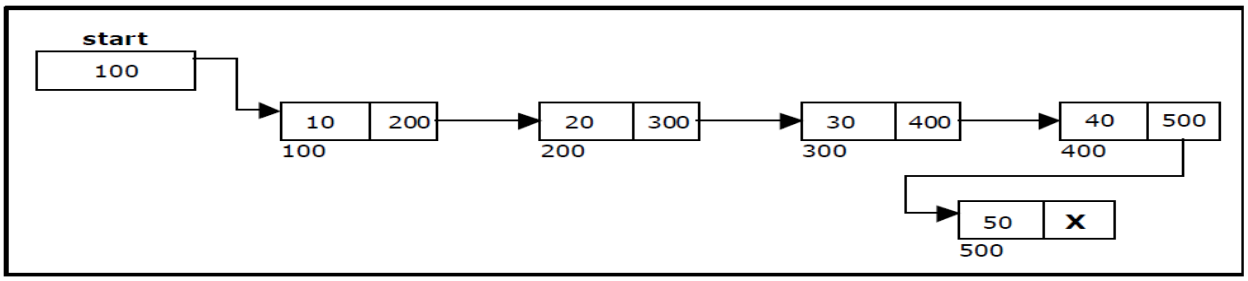
Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

```c
struct node *insert_end(struct node *head){
    int item;
        q=(struct node*)malloc(sizeof(struct node));
            printf("\n Enter the data : ");
            scanf("%d",&item);
                q->data=item;

            p=head;
            while(p->next!=NULL){
                p=p->next;
                }
                p->next=q;
                q->next=NULL;
        printf("\n \t\t\t   YOUR DATA AT END SUCESSFULLY INSERTED ");
    return head;
}
```

1.To insert data in end first we have to reached its last node for this we use a while loop .
2.we create a node and allocate space for it then assign user input data to its data part.
3.when we reached to the last node then in it's next part we assign the value(address) which we created in step 2
4.we have to assign NULL at the newely created node so that trasversing made easy;

Note : The while loop is different from the while loop we use in trasversing to display data.
while(p->next!=NULL)
means ye current node ke value ko hold karega




```
-----------------your data are----------
24   23   12

                  **********************************
                   press 1 to insert data at begining
                   press 2 to insert data at end
                   press 3 to insert data at middle
                   press 4 to delete data at begining
                   press 5 to delete data at end
                   press 6 to delete data at any where
                   press 0 show the data
                   ENTER YOUR CHOICE :2

Enter the data : 40

                     YOUR DATA AT END SUCESSFULLY INSERTED
                  **********************************
                   press 1 to insert data at begining
                   press 2 to insert data at end
                   press 3 to insert data at middle
                   press 4 to delete data at begining
                   press 5 to delete data at end
                   press 6 to delete data at any where
                   press 0 show the data
                   ENTER YOUR CHOICE :0
-----------------your data are----------
24   23   12   40

                  **********************************
                   press 1 to insert data at begining
                   press 2 to insert data at end
                   press 3 to insert data at middle
                   press 4 to delete data at begining
                   press 5 to delete data at end
                   press 6 to delete data at any where
                   press 0 show the data
                   ENTER YOUR CHOICE :
```

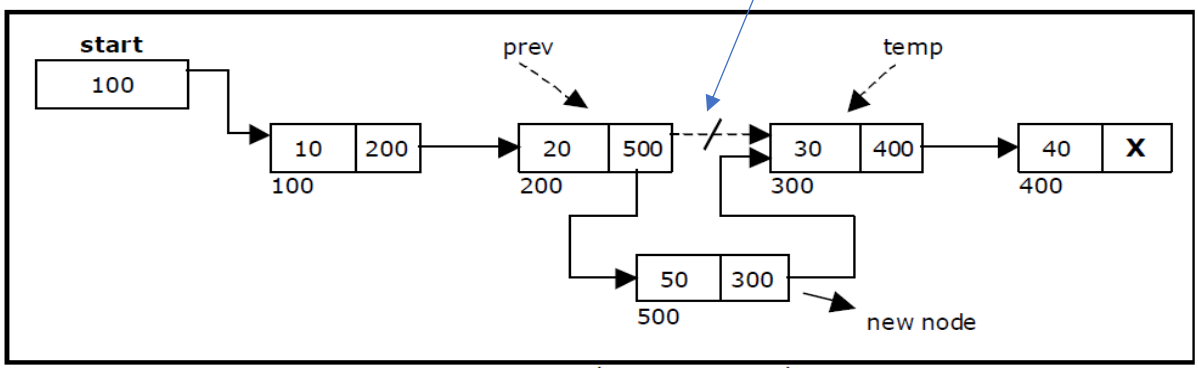
Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

```c
struct node *insert_mid(struct node *head){
    int item,pos,i;
        q=(struct node*)malloc(sizeof(struct node));
            printf("\n Enter the data : ");
            scanf("%d",&item);
            printf("\n Enter the position : ");
            scanf("%d",&pos);
                q->data=item;

                p=head;
                for(i=1;i<pos-1&&p!=NULL;i++){
                    p=p->next;
                }
                q->next = p->next;
                p->next=q;
        printf("\n \t\t\t   YOUR DATA AT  %d postion SUCESSFULLY INSERTED ",pos);
    return head;
}
```

1. In this program the position and data is asked from user.
2. Here we used a FOR LOOP which executed one less then the position i.e(pos-1) and also p!=NULL is also important.
3. Now first we assign the next variable of new node equal to the next variable of (pos-1) node.
4. And then in the (pos-1) node's next pointer we assign the address of new node.

NOTE :- The for loop must be executed upto (pos-1) & p!=NULL must do



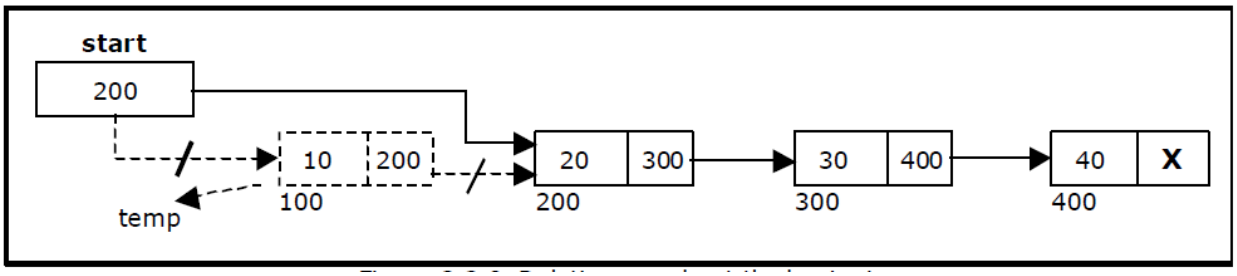

```
C:\Users\samun\Desktop\datastructure codes\linked list.exe
                    ENTER YOUR CHOICE :0
--------------------your data are--------------------
24   23   12   40

                  **********************************
                   press 1 to insert data at begining
                   press 2 to insert data at end
                   press 3 to insert data at middle
                   press 4 to delete data at begining
                   press 5 to delete data at end
                   press 6 to delete data at any where
                   press 0 show the data
                   ENTER YOUR CHOICE :3

Enter the data : 700

Enter the position : 3

                    YOUR DATA AT  3 postion SUCESSFULLY INSERTED

                  **********************************
                   press 1 to insert data at begining
                   press 2 to insert data at end
                   press 3 to insert data at middle
                   press 4 to delete data at begining
                   press 5 to delete data at end
                   press 6 to delete data at any where
                   press 0 show the data
                   ENTER YOUR CHOICE :0
--------------------your data are--------------------
24   23   700   12   40

                  **********************************
                   press 1 to insert data at begining
                   press 2 to insert data at end
                   press 3 to insert data at middle
                   press 4 to delete data at begining
                   press 5 to delete data at end
                   press 6 to delete data at any where
                   press 0 show the data
                   ENTER YOUR CHOICE :
```


Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

```
130 }
131
132
133
134
135 struct node *delete_beg(struct node *head){
136     struct node *temp;
137     temp=head;
138     head=temp->next;
139     free(temp);
140         printf("\n \t\t\t    YOUR DATA from BEGINING SUCESSFULLY DELETED ");
141     return head;
142 }
143
144
145
146
147
148
```

1.To delete a element we just need a pointer variable(say temp)
2. The head node is assign to the temp pointer so that the head become empty
3.Now the next pointer of head is assign to head (it means when we remove the first node then the just next node of head node become the head node**).**
4.Then we have to free the temp node which hold data of head node.




```
press 5 to delete data at end
press 6 to delete data at any where
press 0 show the data
ENTER YOUR CHOICE :0
----------your data are----------
24   23   700   12   40

**********
press 1 to insert data at begining
press 2 to insert data at end
press 3 to insert data at middle
press 4 to delete data at begining
press 5 to delete data at end
press 6 to delete data at any where
press 0 show the data
ENTER YOUR CHOICE :4

    YOUR DATA from BEGINING SUCESSFULLY DELETED

**********
press 1 to insert data at begining
press 2 to insert data at end
press 3 to insert data at middle
press 4 to delete data at begining
press 5 to delete data at end
press 6 to delete data at any where
press 0 show the data
ENTER YOUR CHOICE :0
----------your data are----------
23   700   12   40

**********
press 1 to insert data at begining
press 2 to insert data at end
press 3 to insert data at middle
press 4 to delete data at begining
press 5 to delete data at end
press 6 to delete data at any where
press 0 show the data
ENTER YOUR CHOICE :
```

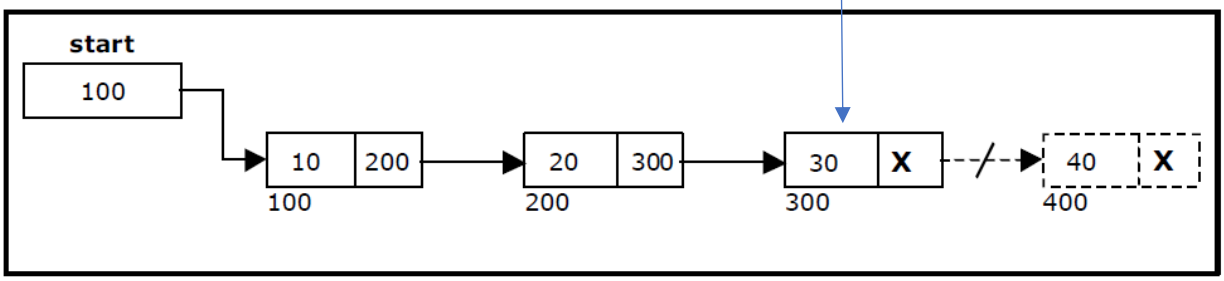

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

1.we need 3 pointer variable (say p,q,temp)
2.first we traversed and reached to the last node then store the value in the temp variable.(means traversed upto NULL)
3.Again we traversed but this time we traversed one before the last node(for this the while condition execute until it reached to the state where the next node equal to the temp)
4.now the next part of the one before node is assigned with NULL.

```c
struct node *delete_end(struct node *head){
    struct node *temp;
        p=head;
        while(p->next!=NULL){
            p=p->next;
            }
            temp=p;
        q=head;
        while(q->next!=temp){
            q=q->next;
        }
        q->next=NULL;
        free(temp);
    printf("\n \t\t\t    YOUR DATA AT END SUCESSFULLY INSERTED ");
    return head;
}
```

Logic by :-SAMUNDAR SINGH



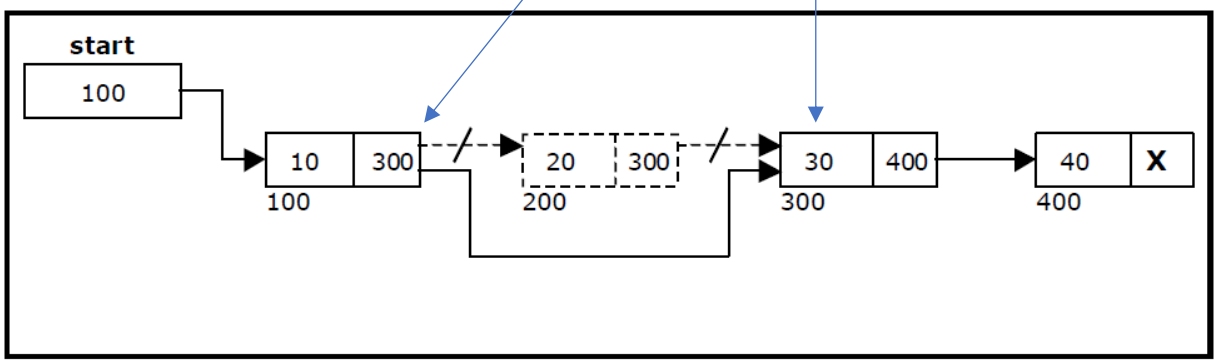

```
C:\Users\samun\Desktop\datastructure codes\linked list.exe
----------your data are----------
23   700   12   40

**********
 press 1 to insert data at begining
 press 2 to insert data at end
 press 3 to insert data at middle
 press 4 to delete data at begining
 press 5 to delete data at end
 press 6 to delete data at any where
 press 0 show the data
 ENTER YOUR CHOICE :5

   YOUR DATA AT END SUCESSFULLY deleted
**********
 press 1 to insert data at begining
 press 2 to insert data at end
 press 3 to insert data at middle
 press 4 to delete data at begining
 press 5 to delete data at end
 press 6 to delete data at any where
 press 0 show the data
 ENTER YOUR CHOICE :0
----------your data are----------
23   700   12

**********
 press 1 to insert data at begining
 press 2 to insert data at end
 press 3 to insert data at middle
 press 4 to delete data at begining
 press 5 to delete data at end
 press 6 to delete data at any where
 press 0 show the data
 ENTER YOUR CHOICE :
```

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

```c
struct node *delete_any(struct node *head){
    int item;
    struct node *temp;
            printf("\n Enter the data : ");
            scanf("%d",&item);
            p=head;
            while(p->next!=NULL){
                if(p->data==item){
                    temp=p;

                    q=head;
                        while(q->next!=temp){
                            q=q->next;
                            }
                            q->next=temp->next;
                }
                p=p->next;
            }
            free(temp);

        printf("\n \t\t\t    YOUR DATA %d AT postion SUCESSFULLY DELETED ",item);
    return head;
}
```

1.we ask user to which data have to be deleted
2.A while loop is used to traversed until NULL encounter .
3. A if condition used to match the data element if it true then the value stored in temp variable and a another while loop is used which traversed until the another pointer address equal to the temp.
4.we have to do this because have to remove the element so the next pointer of previous node must be altered .




```
your data are
23   700   12

**********
 press 1 to insert data at begining
 press 2 to insert data at end
 press 3 to insert data at middle
 press 4 to delete data at begining
 press 5 to delete data at end
 press 6 to delete data at any where
 press 0 show the data
 ENTER YOUR CHOICE :6

Enter the data : 700

 YOUR DATA 700 AT postion SUCESSFULLY DELETED
**********
 press 1 to insert data at begining
 press 2 to insert data at end
 press 3 to insert data at middle
 press 4 to delete data at begining
 press 5 to delete data at end
 press 6 to delete data at any where
 press 0 show the data
 ENTER YOUR CHOICE :0
your data are
23   12

**********
 press 1 to insert data at begining
 press 2 to insert data at end
 press 3 to insert data at middle
 press 4 to delete data at begining
 press 5 to delete data at end
 press 6 to delete data at any where
 press 0 show the data
 ENTER YOUR CHOICE :
```


Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

2.Circular Linked List:

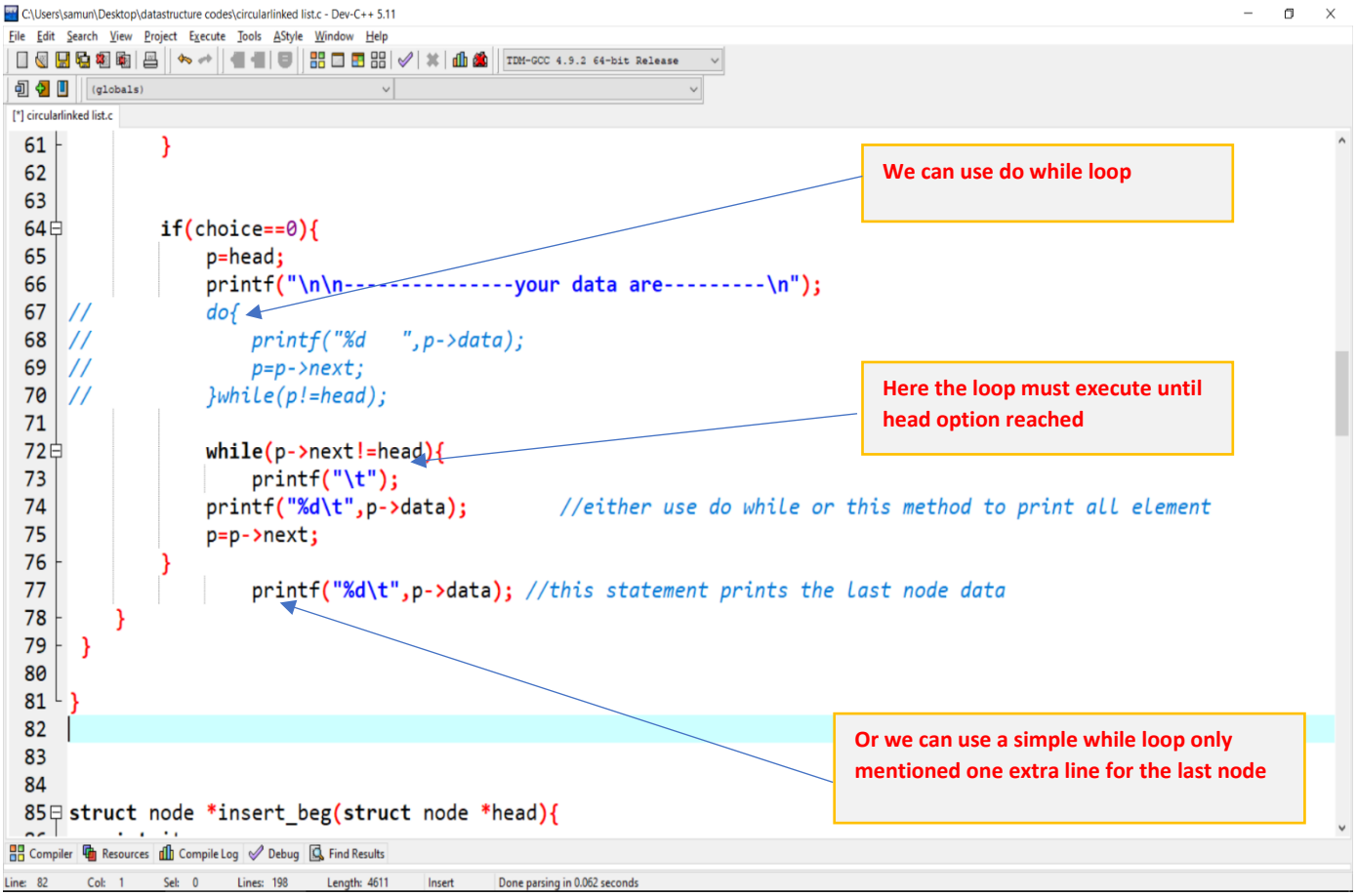
In a circular linked list, the last node contains a pointer to the first node of the list. In circular linked list we can traversed in both forward as well as backward direction. But for backward traverse we have to go to the last node once

NOTE : Since it is similar to the linked list except the last node contain address of head node instead of NULL so codes are quite similar so briefly explanation is skipped clear the linked list concept for better understanding

CODING:-

NOTE: For detailed coding see the above coding

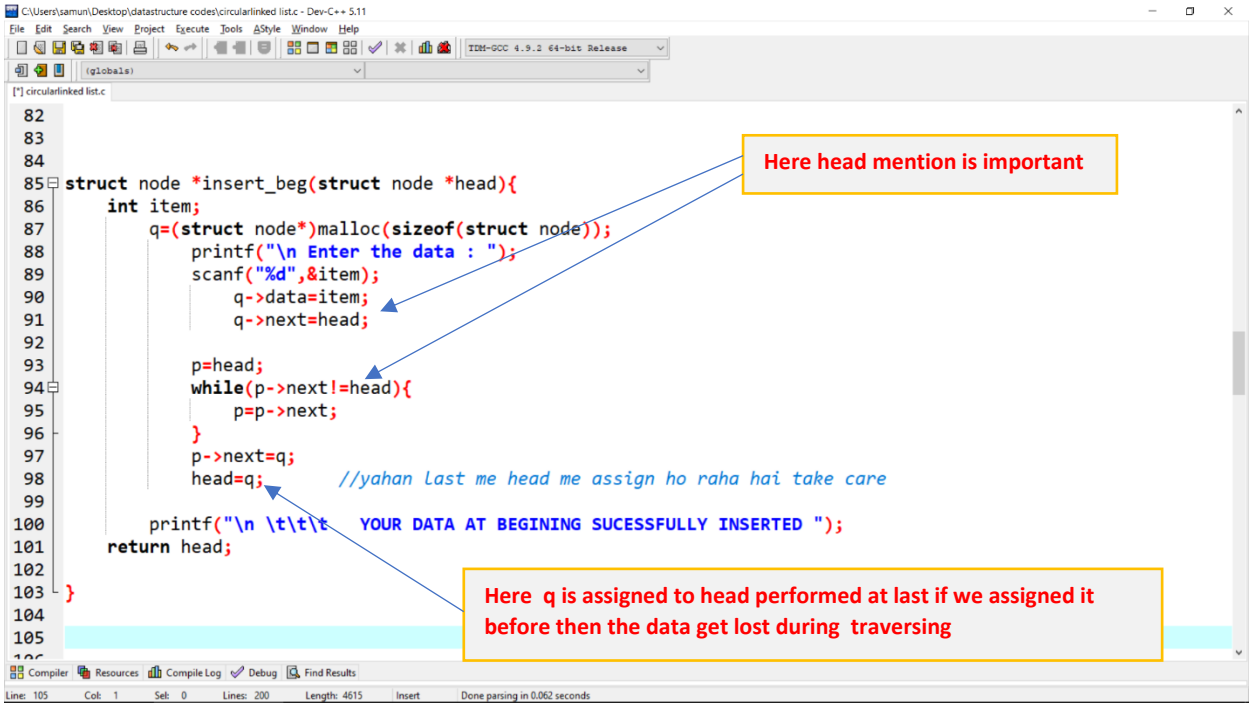
Code for traversing or display the data



Written by:
SAMUNDAR SINGH

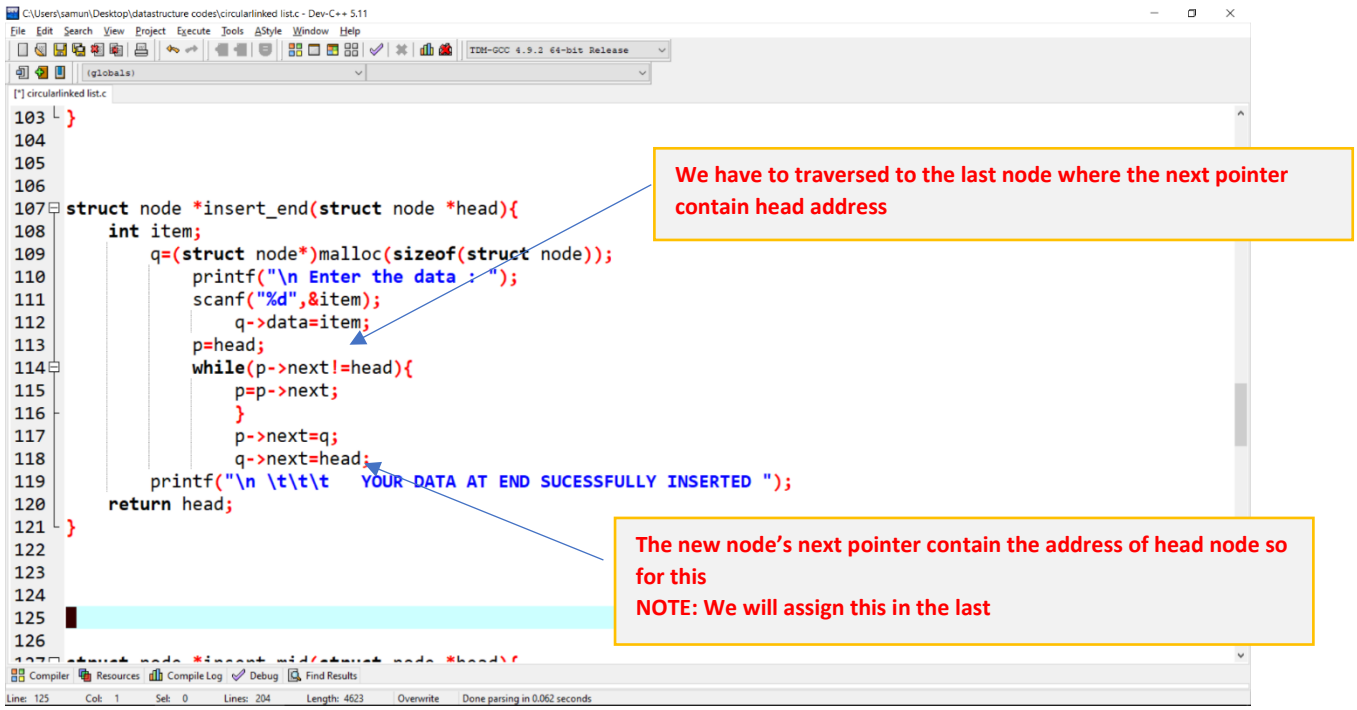
DATA STRUCTURE NOTES

Code for insert data at the beginning:-




```
struct node *insert_beg(struct node *head){
    int item;
        q=(struct node*)malloc(sizeof(struct node));
            printf("\n Enter the data : ");
            scanf("%d",&item);
                q->data=item;
                q->next=head;

            p=head;
            while(p->next!=head){
                p=p->next;
            }
            p->next=q;
            head=q;        //yahan last me head me assign ho raha hai take care

        printf("\n \t\t\t   YOUR DATA AT BEGINING SUCESSFULLY INSERTED ");
    return head;

}
```

Code for insert data at the end :-

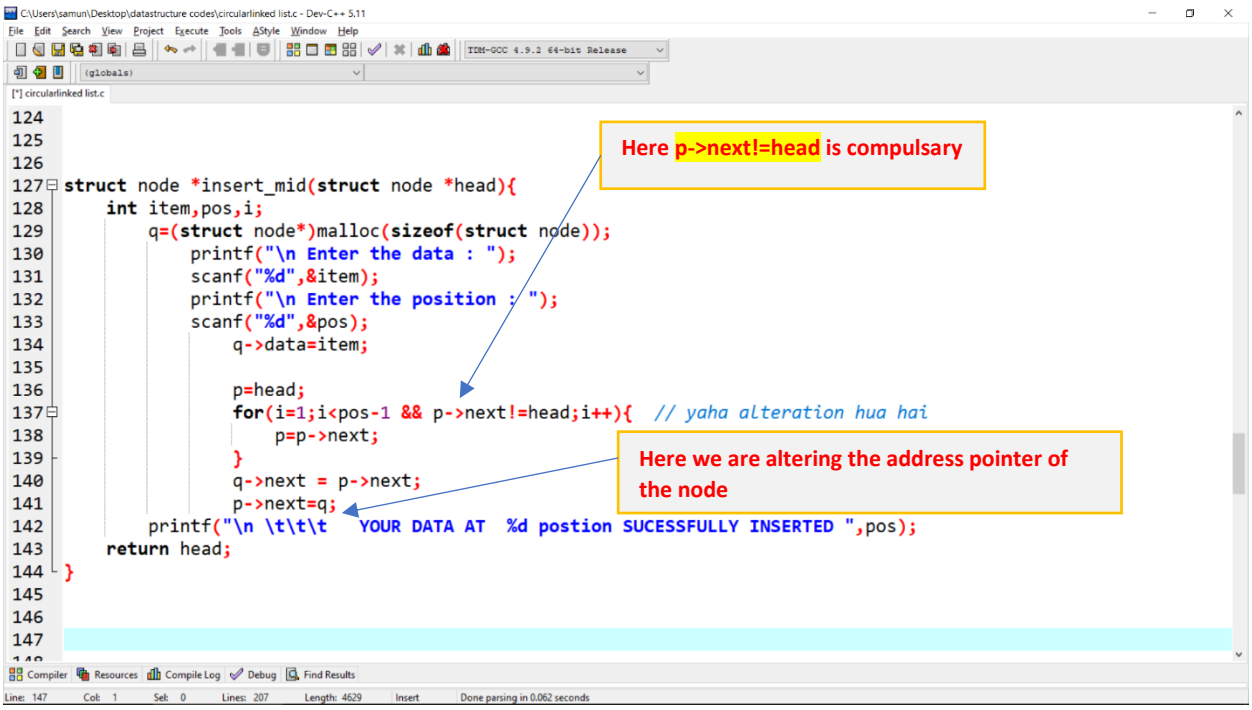



```
}


struct node *insert_end(struct node *head){
    int item;
        q=(struct node*)malloc(sizeof(struct node));
            printf("\n Enter the data : ");
            scanf("%d",&item);
                q->data=item;
            p=head;
            while(p->next!=head){
                p=p->next;
                }
                p->next=q;
                q->next=head;
        printf("\n \t\t\t   YOUR DATA AT END SUCESSFULLY INSERTED ");
    return head;
}
```

Written by:
SAMUNDAR SINGH

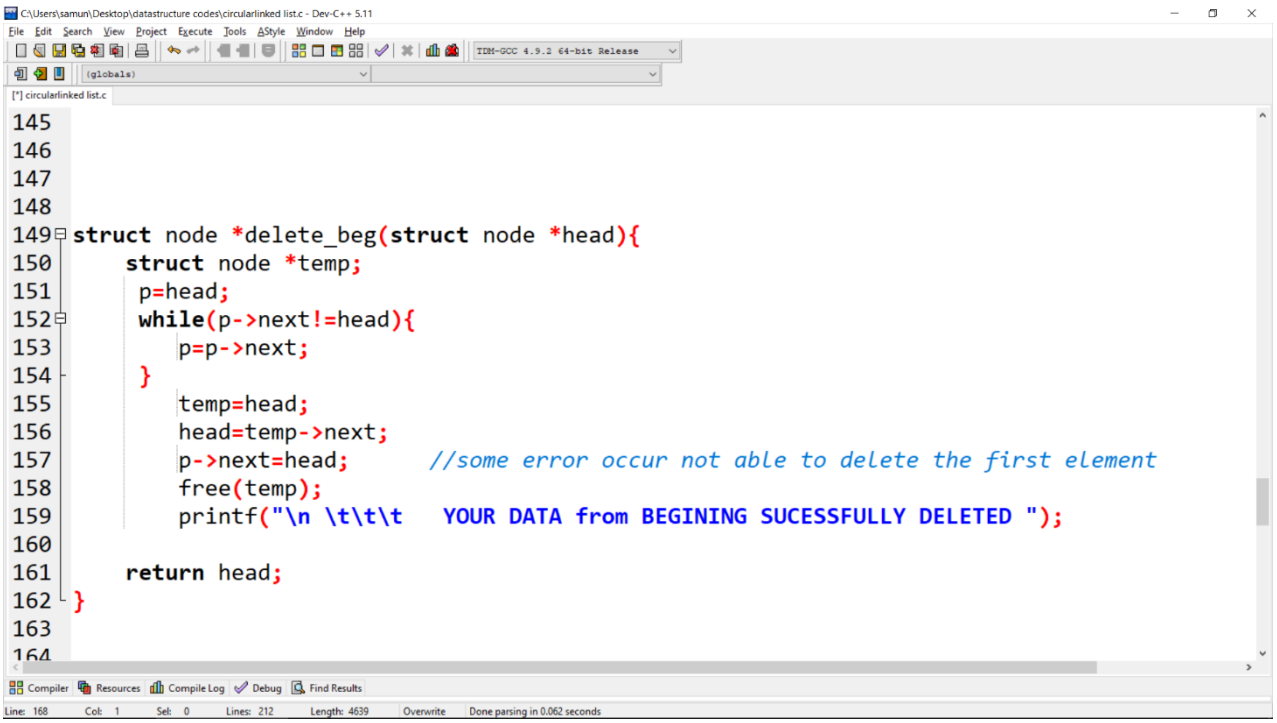
DATA STRUCTURE NOTES

Code for insert data at the Mid node(user choice) :-




```
124
125
126
127 struct node *insert_mid(struct node *head){
128     int item,pos,i;
129         q=(struct node*)malloc(sizeof(struct node));
130             printf("\n Enter the data : ");
131             scanf("%d",&item);
132             printf("\n Enter the position : ");
133             scanf("%d",&pos);
134                 q->data=item;
135
136                 p=head;
137                 for(i=1;i<pos-1 && p->next!=head;i++){  // yaha alteration hua hai
138                     p=p->next;
139                 }
140                 q->next = p->next;
141                 p->next=q;
142         printf("\n \t\t\t   YOUR DATA AT  %d postion SUCESSFULLY INSERTED ",pos);
143     return head;
144 }
145
146
147
```

Code for Delete data from the Beginning :-

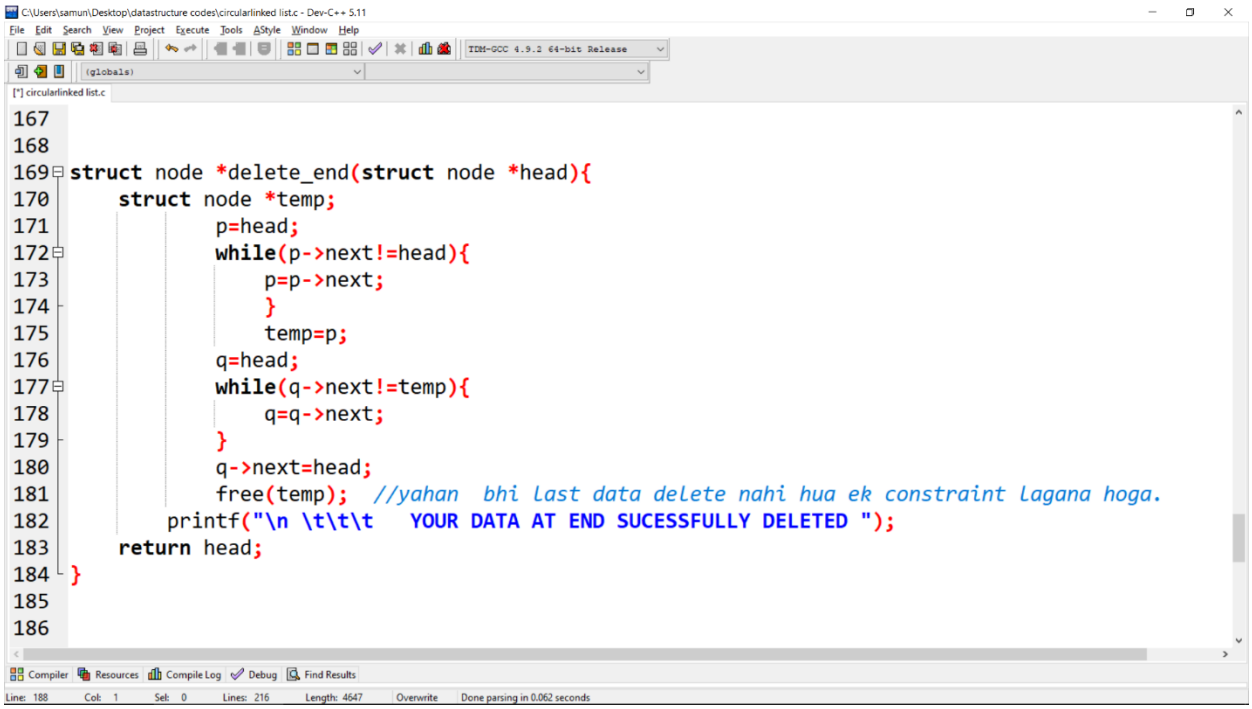


```
145
146
147
148
149 struct node *delete_beg(struct node *head){
150     struct node *temp;
151      p=head;
152      while(p->next!=head){
153          p=p->next;
154      }
155          temp=head;
156          head=temp->next;
157          p->next=head;       //some error occur not able to delete the first element
158          free(temp);
159          printf("\n \t\t\t   YOUR DATA from BEGINING SUCESSFULLY DELETED ");
160
161     return head;
162 }
163
164
```

Written by:
SAMUNDAR SINGH

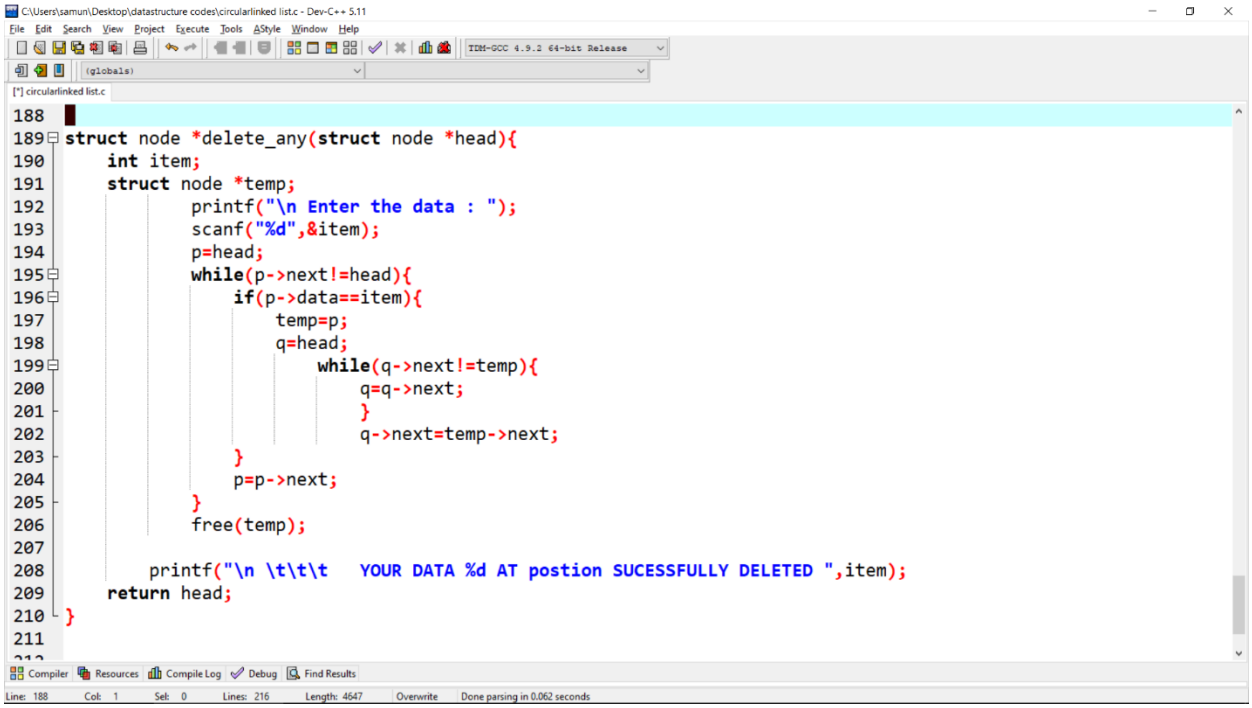
DATA STRUCTURE NOTES

Code for Delete data from the End :-



```
167
168
169 struct node *delete_end(struct node *head){
170     struct node *temp;
171             p=head;
172             while(p->next!=head){
173                 p=p->next;
174                 }
175                 temp=p;
176             q=head;
177             while(q->next!=temp){
178                 q=q->next;
179             }
180             q->next=head;
181             free(temp);  //yahan  bhi last data delete nahi hua ek constraint lagana hoga.
182         printf("\n \t\t\t   YOUR DATA AT END SUCESSFULLY DELETED ");
183     return head;
184 }
185
186
```

Code for Delete data from the Mid(user choice) :-



```
188
189 struct node *delete_any(struct node *head){
190     int item;
191     struct node *temp;
192             printf("\n Enter the data : ");
193             scanf("%d",&item);
194             p=head;
195             while(p->next!=head){
196                 if(p->data==item){
197                     temp=p;
198                     q=head;
199                         while(q->next!=temp){
200                             q=q->next;
201                             }
202                             q->next=temp->next;
203                 }
204                 p=p->next;
205             }
206             free(temp);
207
208         printf("\n \t\t\t   YOUR DATA %d AT postion SUCESSFULLY DELETED ",item);
209     return head;
210 }
211
```

Written by:
SAMUNDAR SINGH

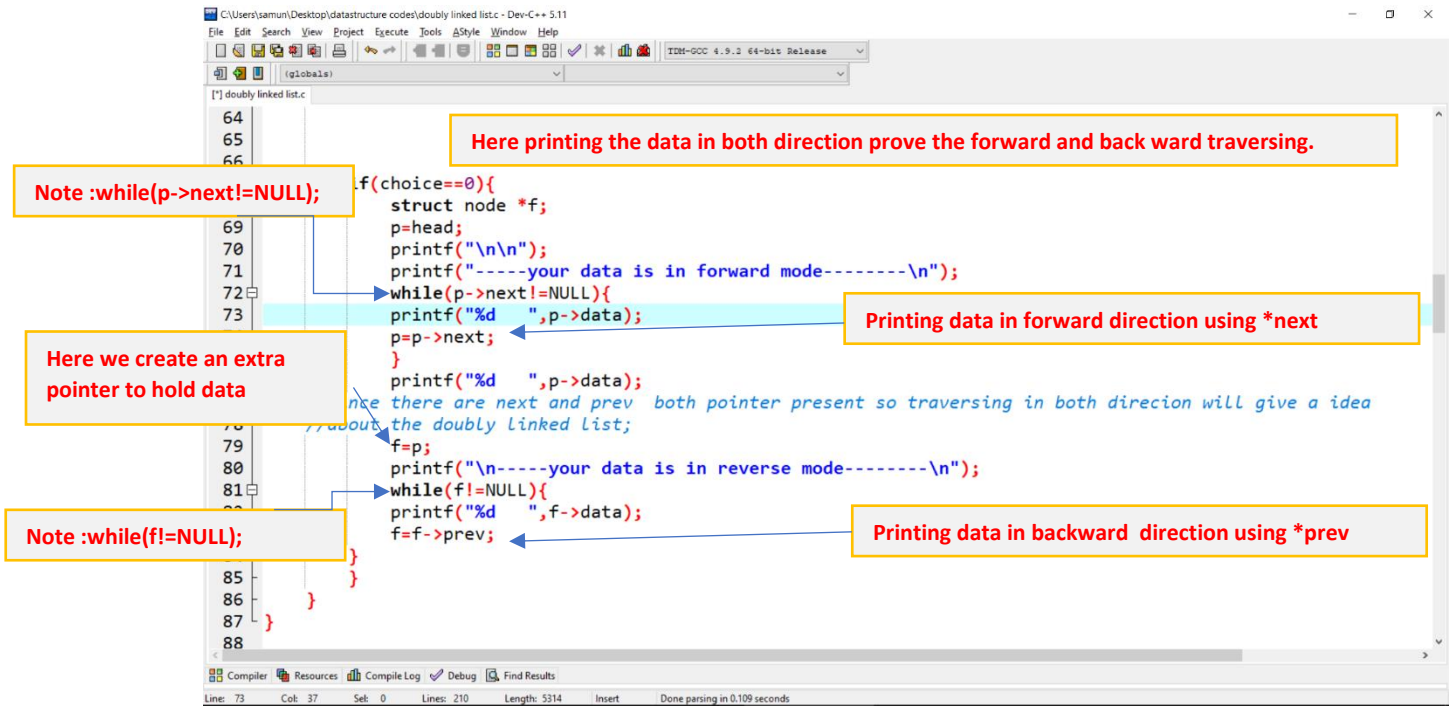
DATA STRUCTURE NOTES

3.Doubly Linked List:

A doubly linked list contains a pointer to the next as well as the previous node in the sequence. The ***prev** and ***next** field of the list contain NULL. In doubly linked list we can traversed in both direction (forward and backward direction). In circular linked list we can't move in backward direction so to traverse the list in backward direction we have to traversed the whole node but in doubly linked list we traversed in both direction. Now Suppose a situation in which we can create function inside function and as the function starts execution it goes on executing the inside function ,now if there a situation occur that we want to execute a function that has already executed then it is not ideal to terminate the process and start the function execution again .simply we use circular lined list to go backward execute that function and traversed to the current execution.

NOTE : The menu driven and modules are quite similar so I am only mentioning the main parts of doubly linked list. Since it is complex then I mention the procedure also for better understanding

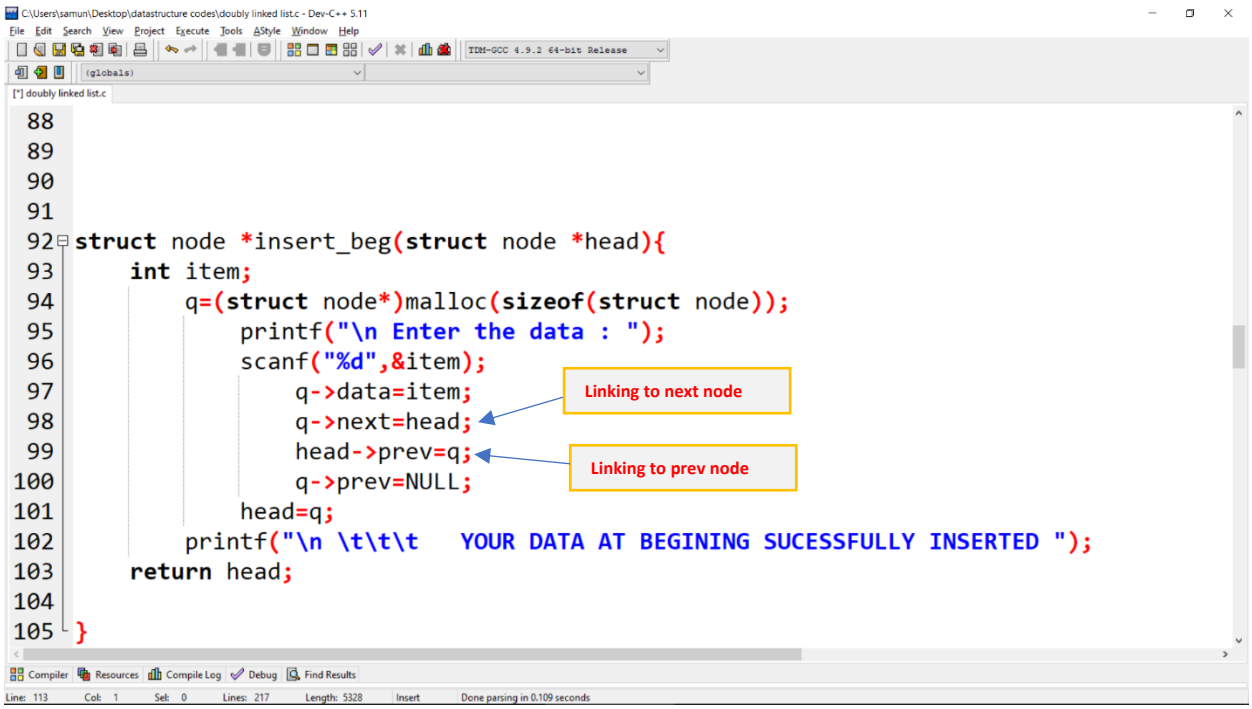
CODING:-



Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Code for insert data at the beginning:-



Step1: First we create the node and allocate space by using malloc function.(see line 94)

Step2: Insert the data of user in it's data part. (see line 97)

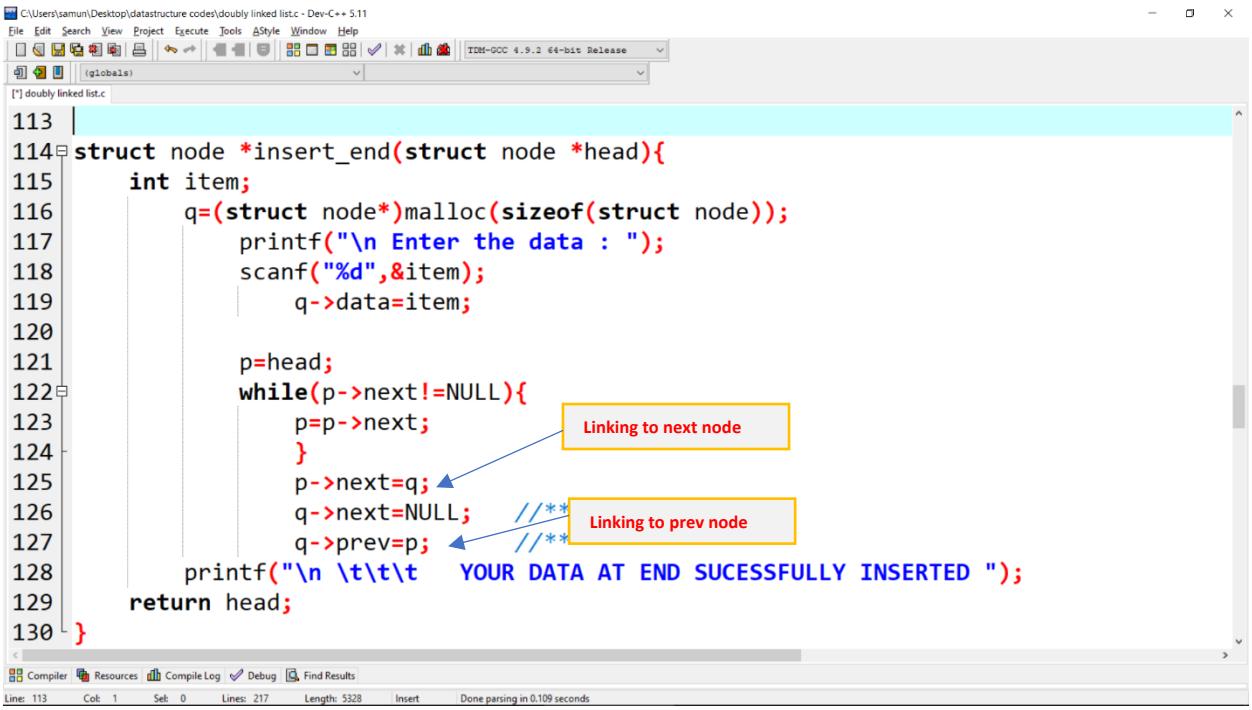
Step3: Now in its next part we allocate the address of head node which is already existed (since we want to make our newly created node as a head node or first node). This step creates the forward link of the new node to the existing node. (see line 98)

Step4: Since we have to traversed in backward direction also then in head (which is the head node before creating the new node) we assign the q(address of new node) to the prev section of the old head node. (see line 99)

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Code for insert data at the end :-



Step1: First we create the node and allocate space by using malloc function.(see line 116)

Step2: Insert the data of user in it's data part. (see line 119)

Step3: we use a loop and traversed to the end node. (see line 122)

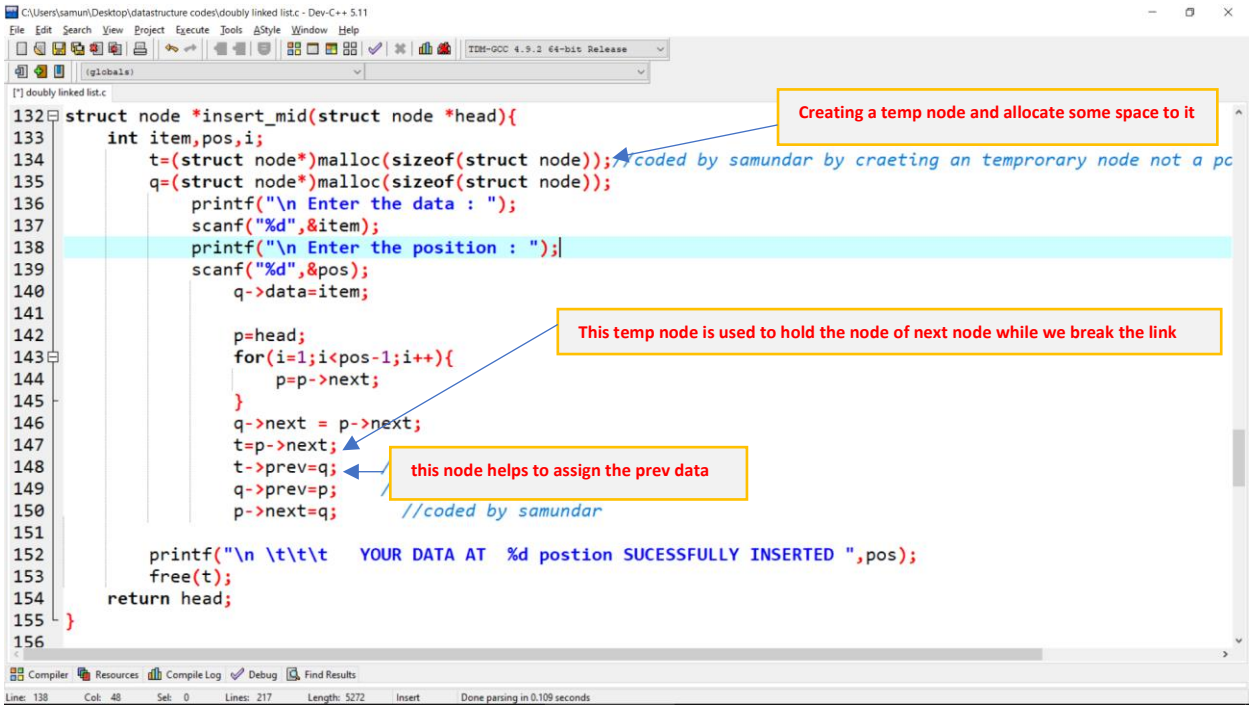
Step4: when we reached to the last node then in next part we mention the address of the newly created node and the newly node next pointer we assign NULL value . (see line 125)

Step5: And in the prev part of newely created node we assign the addres of previous end node which we get during while loop execution terminated (say p in this case) **.** (see line 127)

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Code for insert data at the Mid node(user choice) :-




```
132 struct node *insert_mid(struct node *head){
133     int item,pos,i;
134         t=(struct node*)malloc(sizeof(struct node));//coded by samundar by craeting an temprorary node not a pc
135         q=(struct node*)malloc(sizeof(struct node));
136             printf("\n Enter the data : ");
137             scanf("%d",&item);
138             printf("\n Enter the position : ");
139             scanf("%d",&pos);
140                 q->data=item;
141
142                 p=head;
143                 for(i=1;i<pos-1;i++){
144                     p=p->next;
145                 }
146                 q->next = p->next;
147                 t=p->next;
148                 t->prev=q;
149                 q->prev=p;
150                 p->next=q;       //coded by samundar
151
152         printf("\n \t\t\t   YOUR DATA AT  %d postion SUCESSFULLY INSERTED ",pos);
153         free(t);
154     return head;
155 }
156
```

Step1: First we create the node and allocate space by using malloc function. (see line 116)

Step2: Insert the data of user in it's data part. (see line 119)

Step3: we use a loop and traversed to the end node. (see line 122)

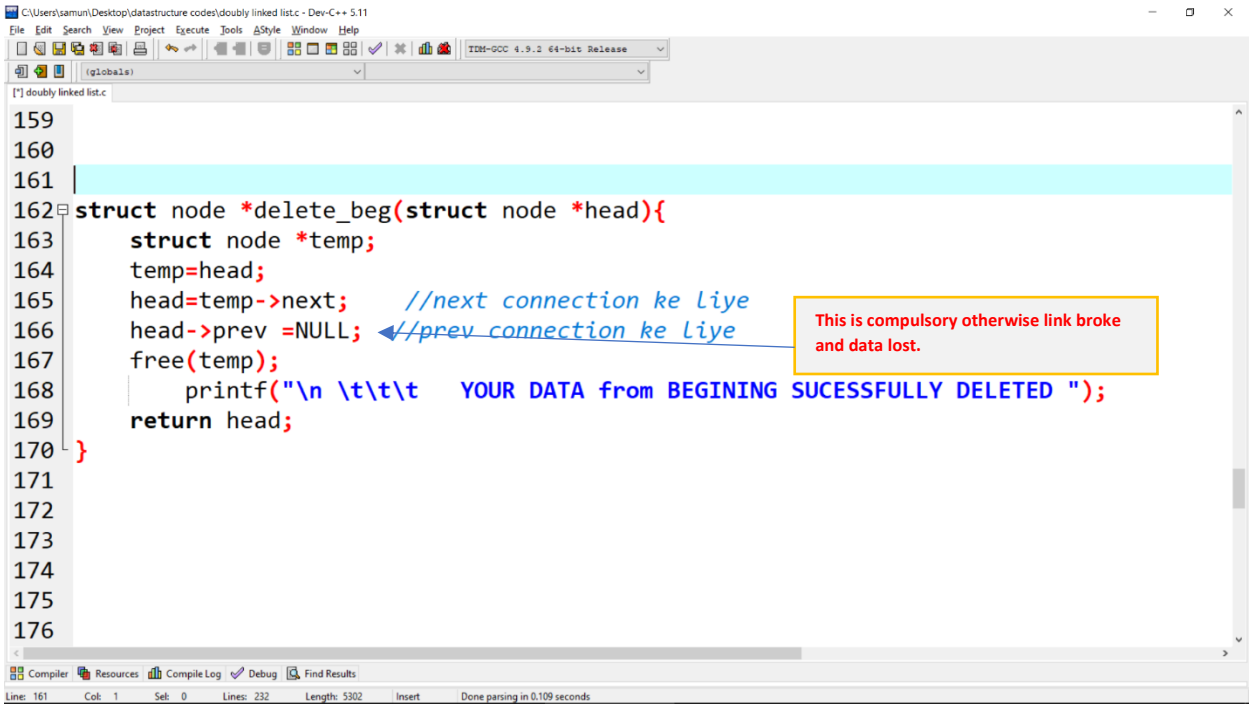
Step4: when we reached to the last node then in next part we mention the address of the newly created node and the newly node next pointer we assign NULL value. (see line 125)

Step5: And in the prev part of newly created node we assign the address of previous end node which we get during while loop execution terminated (say p in this case) **.** (see line 127)

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Code for Delete data from the Beginning :-



Step1: we create a pointer say(temp) to hold address. (see line 163)

Step2: all the data of head is sent to this temp pointer. (see line 164)

Step3: since we are going to delete the first node we have to make the next node as a head node for this purpose the address of the next node is store in the head's next part so we have to assign that address to the head part . (see line 165)

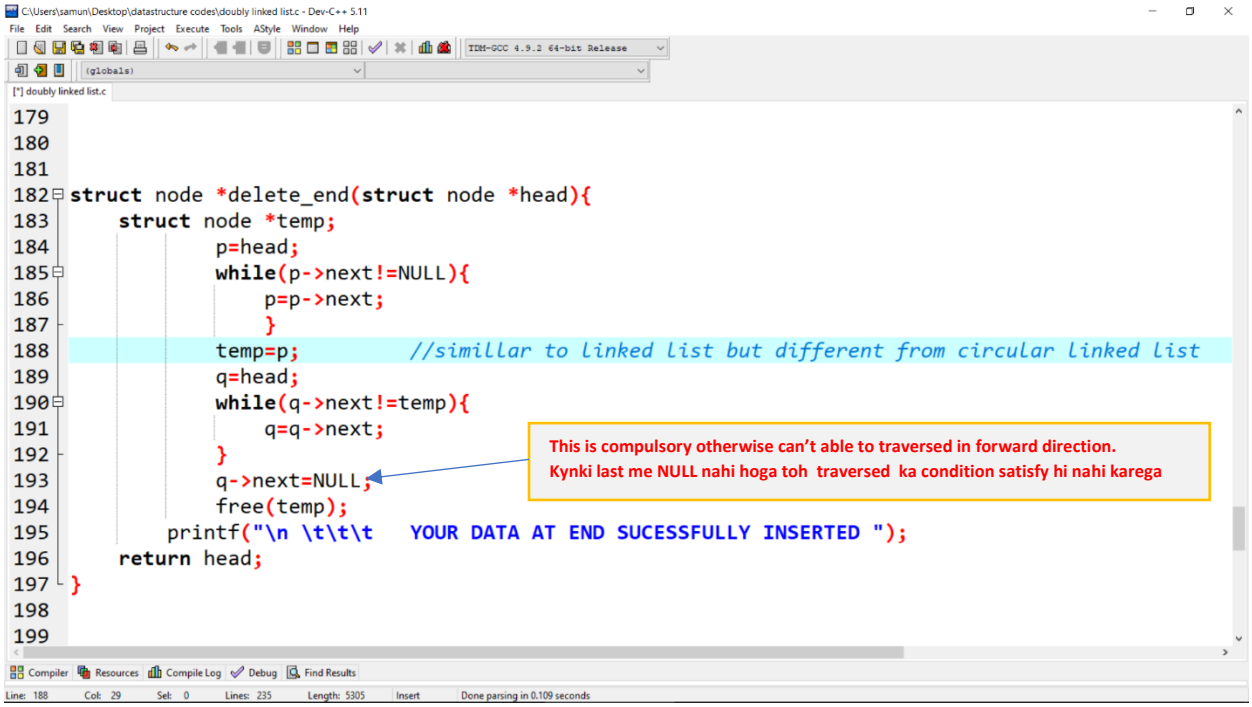
Step4: Now the next node become the head node hence we hane to make it's prev section as NULL. (see line 166)

Step5: The prev section must be make NULL otherwise we can't traversed in backward direction .

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Code for Delete data from the End :-



```
179
180
181
182 struct node *delete_end(struct node *head){
183     struct node *temp;
184             p=head;
185             while(p->next!=NULL){
186                 p=p->next;
187                 }
188             temp=p;          //simillar to linked list but different from circular linked list
189             q=head;
190             while(q->next!=temp){
191                 q=q->next;
192             }
193             q->next=NULL;
194             free(temp);
195         printf("\n \t\t\t    YOUR DATA AT END SUCESSFULLY INSERTED ");
196     return head;
197 }
198
199
```

This is compulsory otherwise can't able to traversed in forward direction.
Kynki last me NULL nahi hoga toh traversed ka condition satisfy hi nahi karega

Step1: We create a pointer say(temp) to hold address. (see line 183)

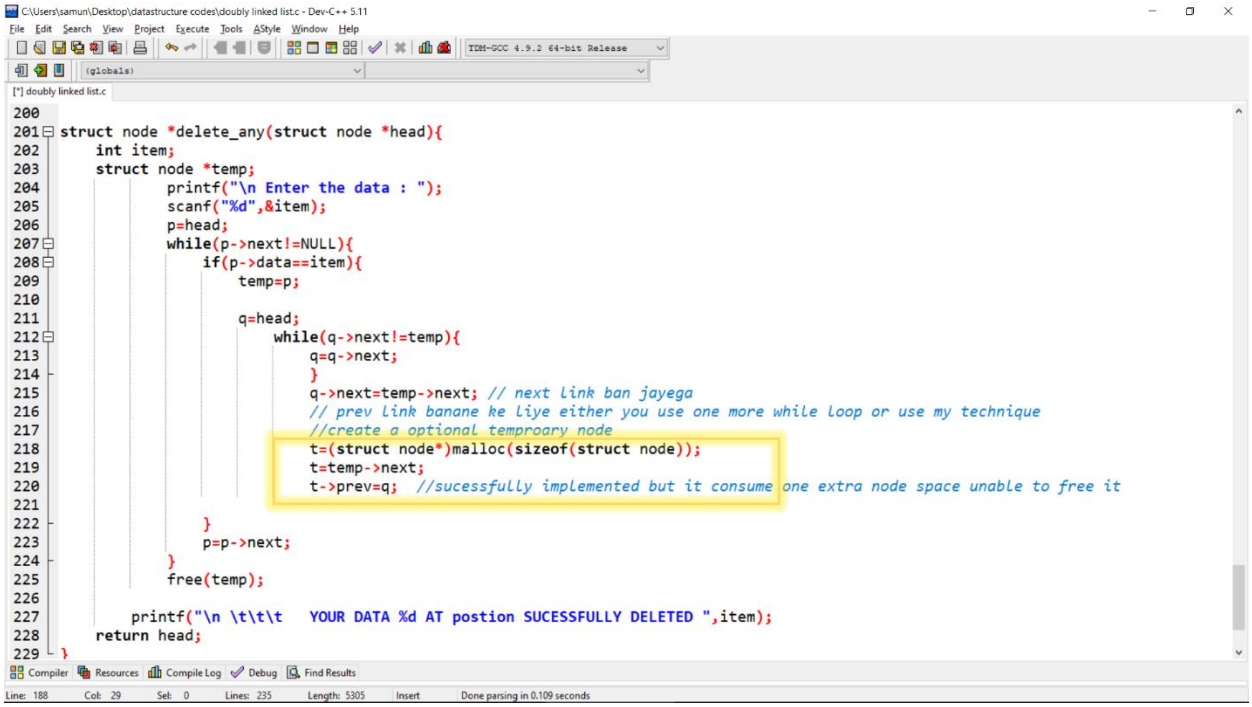
Step2: We traversed and reached to the last node. (see line 185)

Step3: The address of last node is stored to the temp variable . (see line 188)

Step4: Again we traversed but this time we reached one before node to the next node this is so because when we delete the node then this previous node become the last node and in it's prev field address of the next node is stored which we have to make NULL **.** (see line 190 and 194)

Written by:
SAMUNDAR SINGH

Code for Delete data from the Mid (user choice):-



```
200
201 struct node *delete_any(struct node *head){
202     int item;
203     struct node *temp;
204         printf("\n Enter the data : ");
205         scanf("%d",&item);
206         p=head;
207         while(p->next!=NULL){
208             if(p->data==item){
209                 temp=p;
210
211                 q=head;
212                     while(q->next!=temp){
213                         q=q->next;
214                         }
215                         q->next=temp->next; // next link ban jayega
216                         // prev link banane ke liye either you use one more while loop or use my technique
217                         //create a optional temproary node
218                         t=(struct node*)malloc(sizeof(struct node));
219                         t=temp->next;
220                         t->prev=q;  //sucessfully implemented but it consume one extra node space unable to free it
221
222             }
223             p=p->next;
224         }
225         free(temp);
226
227     printf("\n \t\t\t   YOUR DATA %d AT postion SUCESSFULLY DELETED ",item);
228 return head;
229 }
```

Step1: We create a pointer say(temp) to hold address(**say temp**). (see line 203)

Step2: We traversed and reached to that node which have the same data which user want to delete (**say p is that node**). (see line 207)

Step3: The address of that node is stored to the temp variable(**say temp has the address of the same pointer p**) . (see line 209)

Step4: Again we traversed but this time we reached one before node to the next node (**say q**)this is so because when we delete the node then this previous node's next part hold the data of deleted node , so here we have to assign the address of that node which is next to the deleted node for this we know that the deleted node's next part hold the data of the next node so we simply use that knowledge and assign the previous node's next part the address of the node that is in the deleted node's next part**.** (see line 216)

Step5: Now forward link is established and we have to establish the backward link for this purpose we create a node using malloc function. (see line 218).

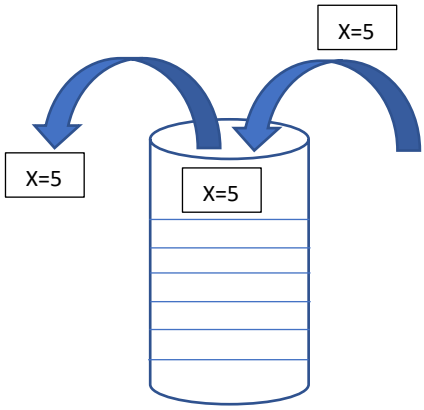
Step6: In this created node we assign the address of node which is after to the deleted node(see line 220 temp mera wo node hai jisko delete karna hai ise ka next part me aage ka node ka address hai jo ki 't' ko allocate kar rahe hai)**.** Now we are in the node which is after the deleted node(means we are in node t jo ki next node ka address hold kiya hai) , to make backward link we assign the address of the node which is before the deleted node in the prev part of it(means 't' ke prev part me 'q' ka address store karna hai). (see line 220)

---------- END ----------

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

STACK



1.Stack using Array:

A stack is a linear data structure which use LIFO (Last-In-First-Out) principle.
We perform three basic operation for stack:

- **Push:** used to insert an element into a stack.
- **Pop:** used to delete an element from a stack.
- **Peek**: used to fetch the last element of stack.

Code for menu driven program :-

C:\Users\samun\Desktop\datastructure codes\stack using array.c - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 64-bit Release

(globals)

[*] stack using array.c

```c
#include<stdio.h>
void main() {
  int size, choice;
  printf("insert the size of array : ");
  scanf("%d", & size);
  int top = -1;
  int arr[size];

  while (choice != -1) {
    printf("\n*************************************\n");
    printf("press 1 for push data in stack \n");
    printf("press 2 for pop data in stack \n");
    printf("press 3 for peek data in stack\n ");
    printf("press -1 to exit \n");
    printf("Enter your choice : ");
    scanf("%d", & choice);

    if (choice == 1) {
      if (top == size - 1) {
        printf("\t\t\tWarning:The stack is full");
        break;
      } else {
        push(arr, size, top);
      }
      top = top + 1;
    }

    if (choice == 2) {
      if (top == -1) {
        printf("\t\t\tWarning:The stack is empty");
        break;
      } else {
        pop(arr, size, top);
      }
      top = top - 1;
    }
    if (choice == 3) {
      printf("the top most element of stack is : %d", arr[top]);
    }
    if (choice == 4) {
      display(arr, top);
    }
  }
}
```

int push (int arr[], int size, int top)

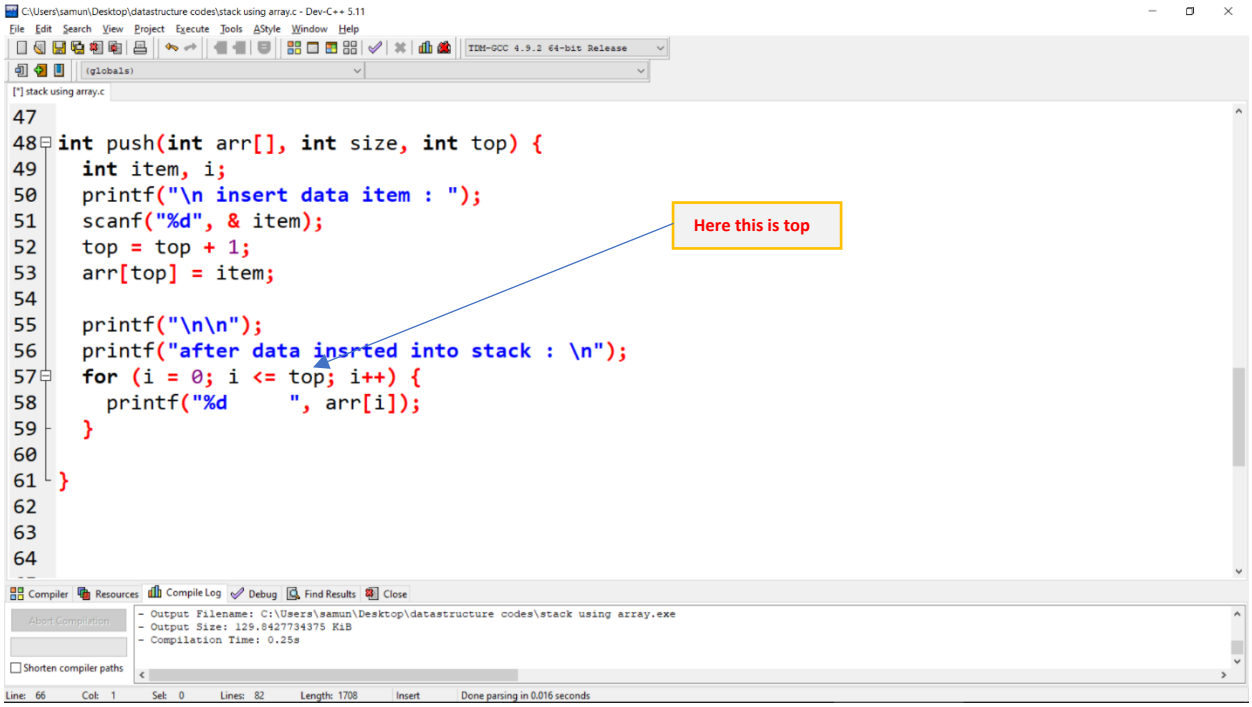
Here the top is initialized as -1

NOTE: here we check the top function in order to determine whether the stack is full or empty

Written by:
SAMUNDAR SINGH

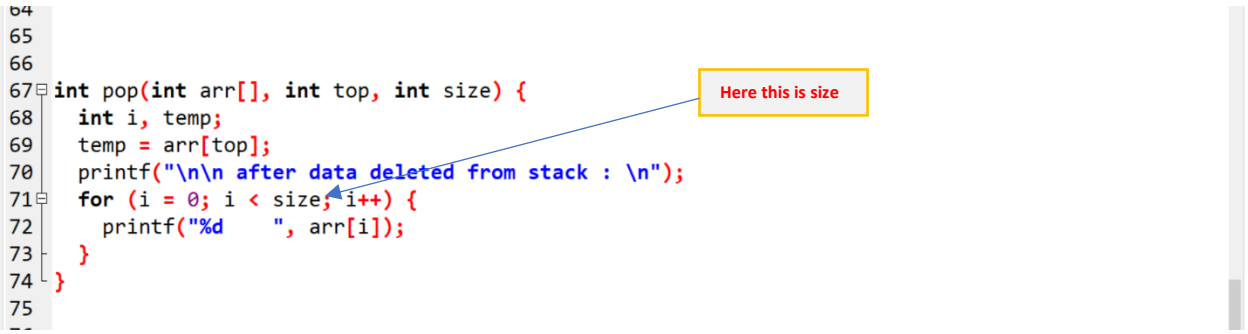
DATA STRUCTURE NOTES

Code to perform PUSH Operation :-




```
47
48 int push(int arr[], int size, int top) {
49    int item, i;
50    printf("\n insert data item : ");
51    scanf("%d", & item);
52    top = top + 1;
53    arr[top] = item;
54
55    printf("\n\n");
56    printf("after data insrted into stack : \n");
57    for (i = 0; i <= top; i++) {
58      printf("%d      ", arr[i]);
59    }
60
61 }
62
63
64
```

Code to perform POP Operation :-




```
64
65
66
67 int pop(int arr[], int top, int size) {
68    int i, temp;
69    temp = arr[top];
70    printf("\n\n after data deleted from stack : \n");
71    for (i = 0; i < size; i++) {
72      printf("%d     ", arr[i]);
73    }
74 }
75
```

Code to perform PEEK Operation :-

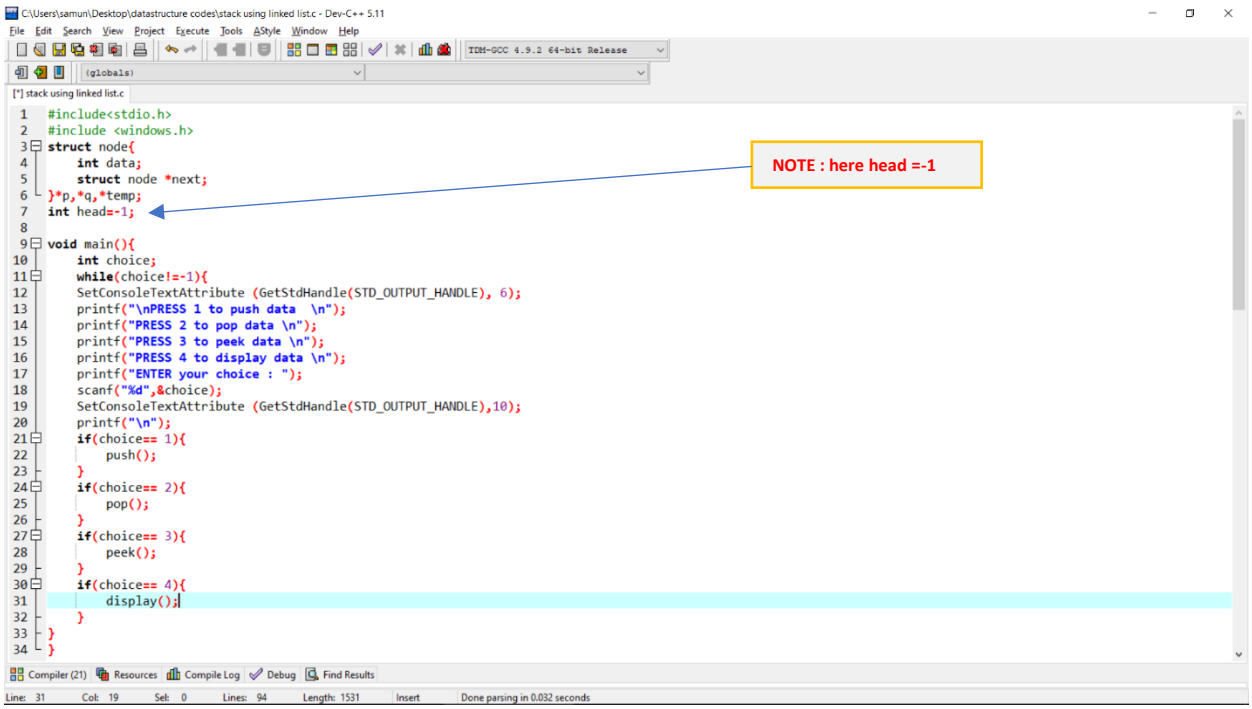
```
80
81
82 int display(int arr[], int top) {
83    int i;
84    for (i = 0; i <= top; i++) {
85      printf("%d    ", arr[i]);
86
87    }
88 }
```

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

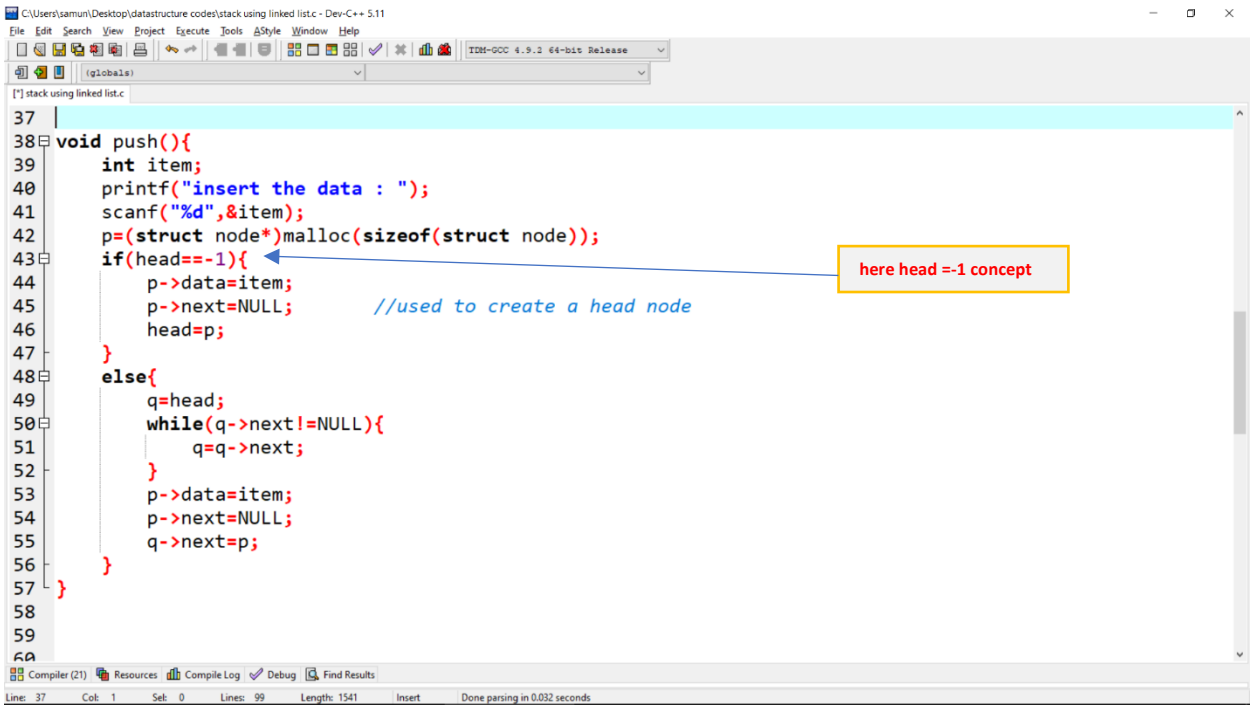
2.Stack using LINKED LIST:

Code for menu driven program :-




```
#include<stdio.h>
#include <windows.h>
struct node{
    int data;
    struct node *next;
}*p,*q,*temp;
int head=-1;

void main(){
    int choice;
    while(choice!=-1){
    SetConsoleTextAttribute (GetStdHandle(STD_OUTPUT_HANDLE), 6);
    printf("\nPRESS 1 to push data  \n");
    printf("PRESS 2 to pop data \n");
    printf("PRESS 3 to peek data \n");
    printf("PRESS 4 to display data \n");
    printf("ENTER your choice : ");
    scanf("%d",&choice);
    SetConsoleTextAttribute (GetStdHandle(STD_OUTPUT_HANDLE),10);
    printf("\n");
    if(choice== 1){
        push();
    }
    if(choice== 2){
        pop();
    }
    if(choice== 3){
        peek();
    }
    if(choice== 4){
        display();
    }
}
}
```

Code to perform PUSH Operation :-

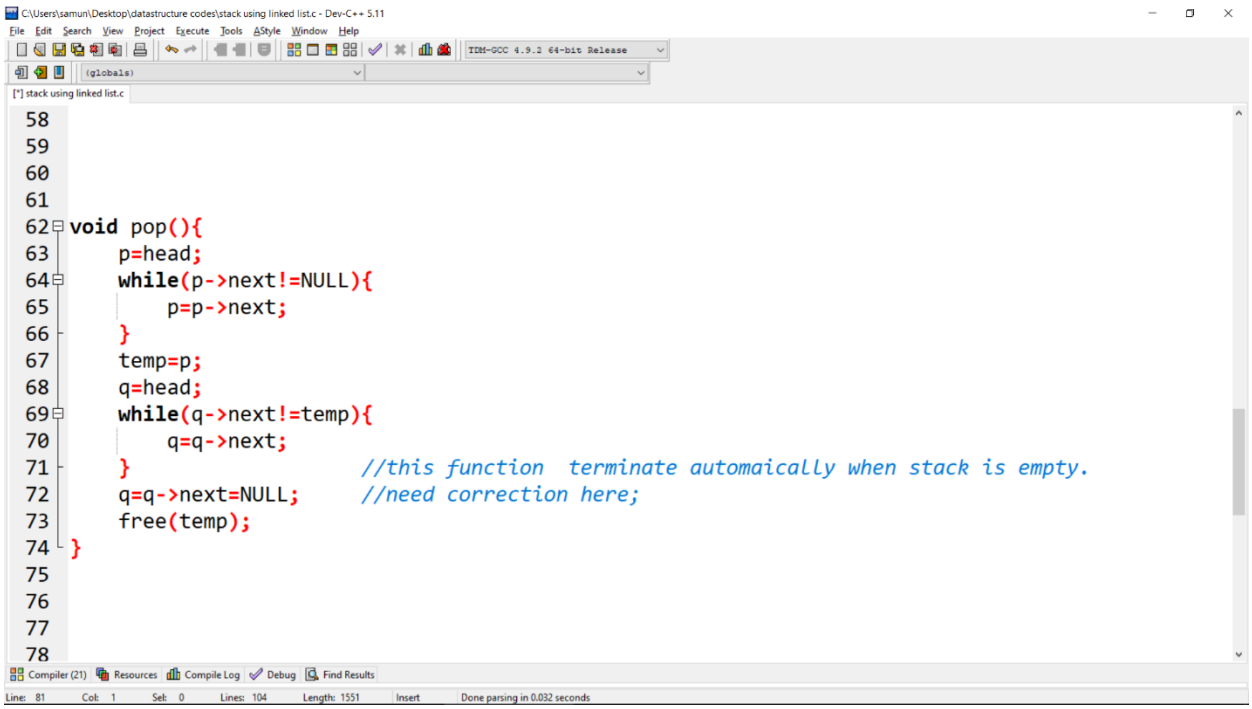



```
void push(){
    int item;
    printf("insert the data : ");
    scanf("%d",&item);
    p=(struct node*)malloc(sizeof(struct node));
    if(head==-1){
        p->data=item;
        p->next=NULL;       //used to create a head node
        head=p;
    }
    else{
        q=head;
        while(q->next!=NULL){
            q=q->next;
        }
        p->data=item;
        p->next=NULL;
        q->next=p;
    }
}
```

Written by:
SAMUNDAR SINGH

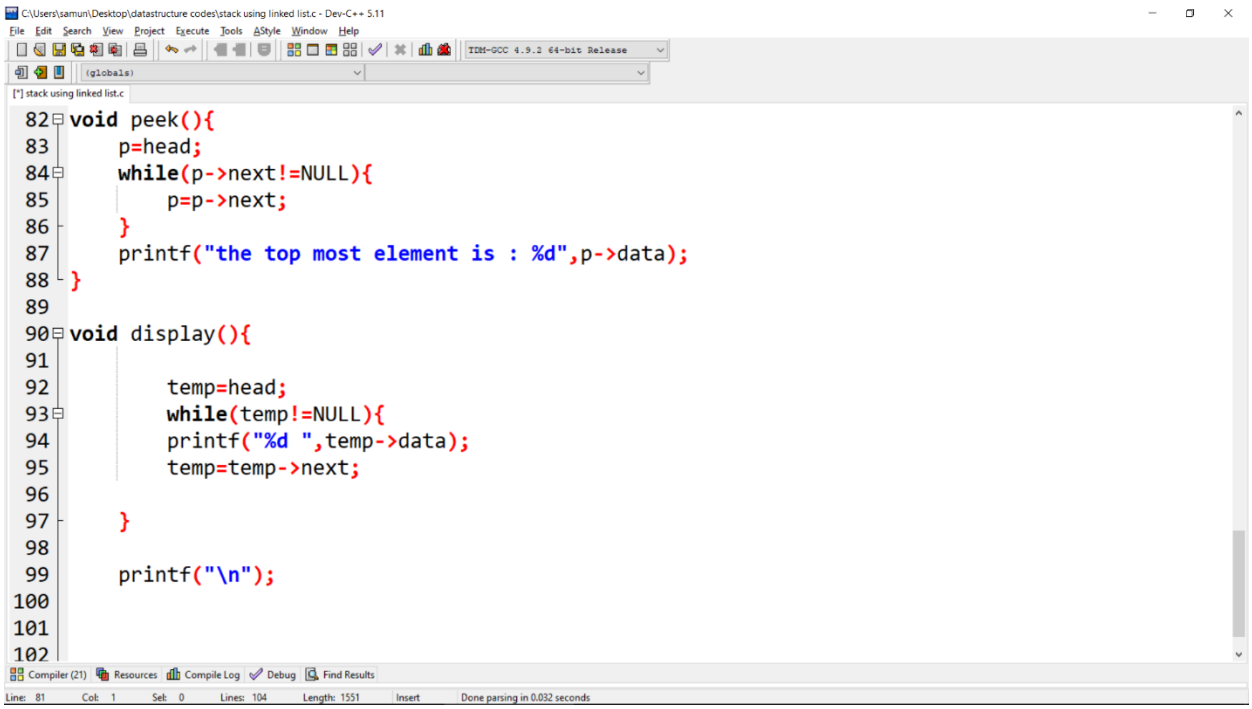
DATA STRUCTURE NOTES

Code to perform POP Operation :-



```
58
59
60
61
62  void pop(){
63      p=head;
64      while(p->next!=NULL){
65          p=p->next;
66      }
67      temp=p;
68      q=head;
69      while(q->next!=temp){
70          q=q->next;
71      }                   //this function  terminate automaically when stack is empty.
72      q=q->next=NULL;     //need correction here;
73      free(temp);
74  }
75
76
77
78
```

Code to perform PEEK Operation :-



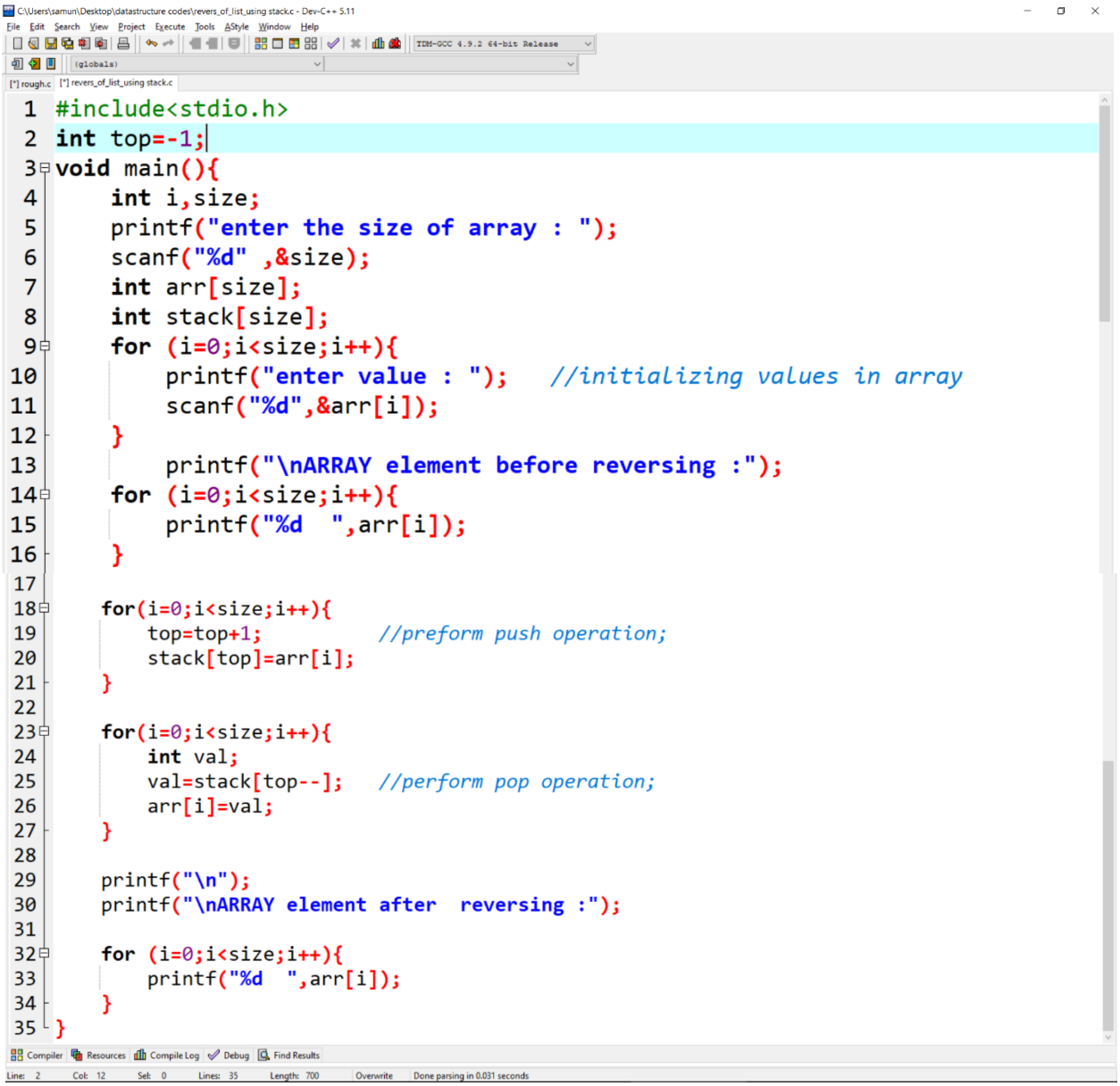
```
82  void peek(){
83      p=head;
84      while(p->next!=NULL){
85          p=p->next;
86      }
87      printf("the top most element is : %d",p->data);
88  }
89
90  void display(){
91
92          temp=head;
93          while(temp!=NULL){
94          printf("%d ",temp->data);
95          temp=temp->next;
96
97      }
98
99      printf("\n");
100
101
102
```

Written by:
SAMUNDAR SINGH

APPLICATIONS OF STACK:

1.Reversing a list: -

A list can be reversed using Stack. Here I am arranging my data item in an array. The logic of reversing the data elements stored in array is that I read each data element from the array and push it into a stack once all the element of array is pushed to the stack then then pop operation is performed by pop a number at a time and then stored in the array starting from first index.



`

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

2. Parentheses checker: -

NOTE:
Before starting this one concept must be clear that in order to insert a line in c we have to take the Input in gets(variable name) and print the output in puts(variable name).

And the character variavle declare as
Char variable[size]

```
#include <stdio.h>
int main()
{
   char name[30];
   printf("Enter name: ");
   gets(name);  // read string
   printf("Name: ");
   puts(name);    // display string
   return 0;
}
```

LOGIC :

Step1: first we create a array in which the expression is going to be stored .

Step2: Next a for loop run which detect the presence of parenthesis in expression whenever a parenthesis is detected (open parenthesis) then it performed push operation by storing that parenthesis in a separate array named “stack[50]”;

Step3: Now if any closed parenthesis detected then the top value is checked it means we check whether any parenthesis is present in the stack or not this is only for if we insert any closed parenthesis in beginning if this happens then we simply print it was a invalid expression.

Step4: Now if closed parenthesis detected and top value checked if the top indicates that there are some value in stack then further process executed.

Step5: Now we know that stack contain some parenthesis and that is the open parenthesis . so we perform a pop operation here .we create a function named pop which return the last value of the stack each time when we call and store that value in the temp variable.

Step6: Simultaneously the for loop is running and it will detect the closed parenthesis if the step 3 is skipped if any closed parenthesis is detected then it check what type of data included in temp variable if it has different open parenthesis then invalid is printed .(don’t forget that the stack only hold open parenthesis and the temp only store the last data of stack)

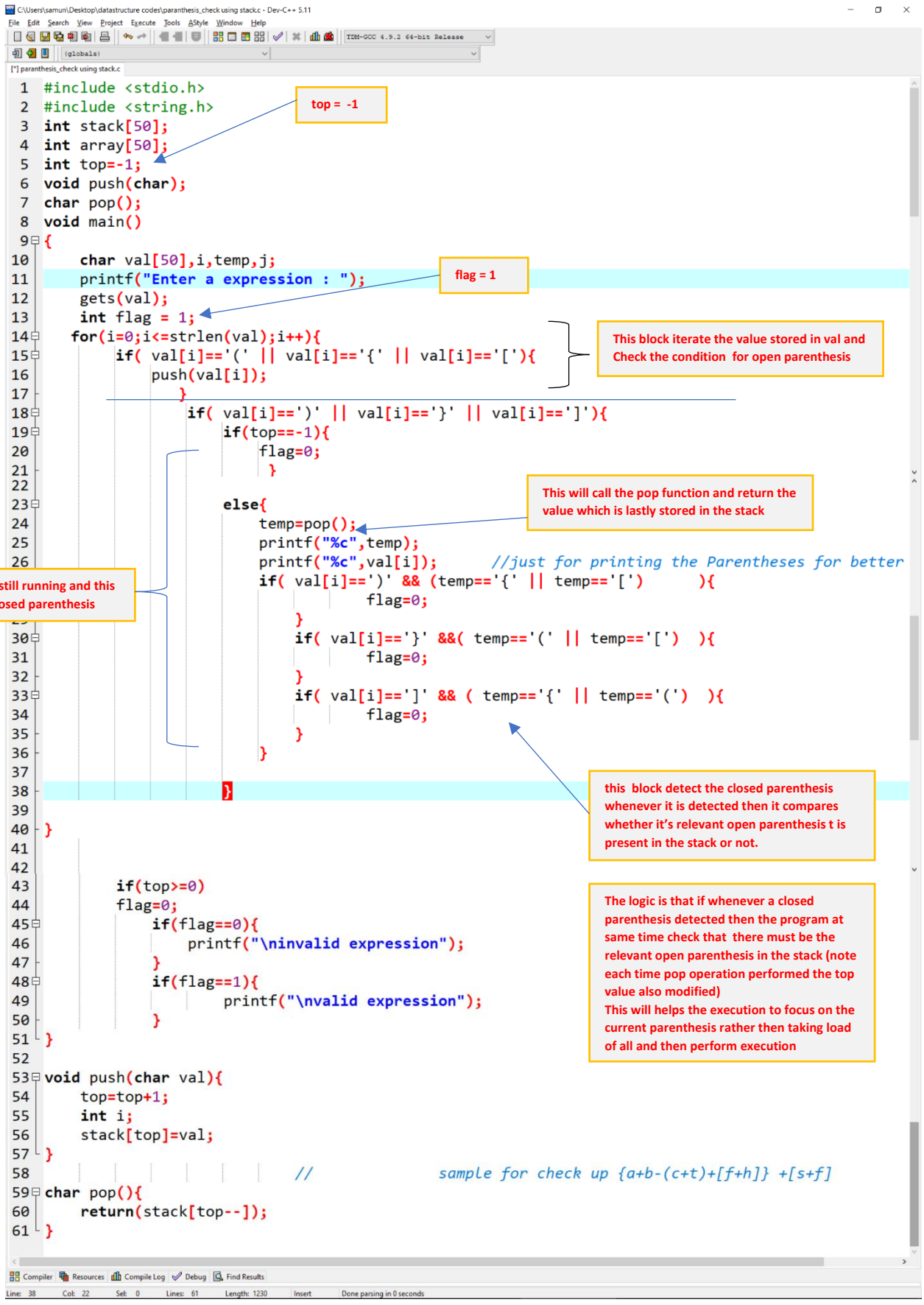
Step7: We provide the possible case for the parenthesis check each time whenever there is any mismatch found the flag become “0” other wise it will be “1” ;

This Area left blank for personal notes :

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Code for Parentheses checker



Written by:
SAMUNDAR SINGH

3. Airthmetic expression Evaluation:

An algebraic expression can be represented using three different notations. They are infix, postfix and prefix notations:

Infix: It is the form of an arithmetic expression in which we fix (place) the arithmetic operator in between the two operands.

Example: (A + B) * (C - D)

Prefix: It is the form of an arithmetic notation in which we fix (place) the arithmetic operator before (pre) its two operands.

Example: * + A B – C D

Postfix: It is the form of an arithmetic expression in which we fix (place) the arithmetic operator after (post) its two operands.

Example: A B + C D - *

Converting expressions using Stack:

1.Conversion from infix to postfix:

Procedure to convert from infix expression to postfix expression is as follows:-

1. Scan the infix expression from **left to right.**

2. a) If the scanned symbol is open parenthesis, push it onto the stack.
   b) If the scanned symbol is an operand, then place directly in the postfix expression (output).
   c) If the symbol scanned is a closed parenthesis, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parenthesis.
   d) If the scanned symbol is an operator, then go on removing all the operators from the stack and place them in the postfix expression, if and only if the precedence of the operator which is on the top of the stack is greater than (or greater than or equal) to the precedence of the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack.

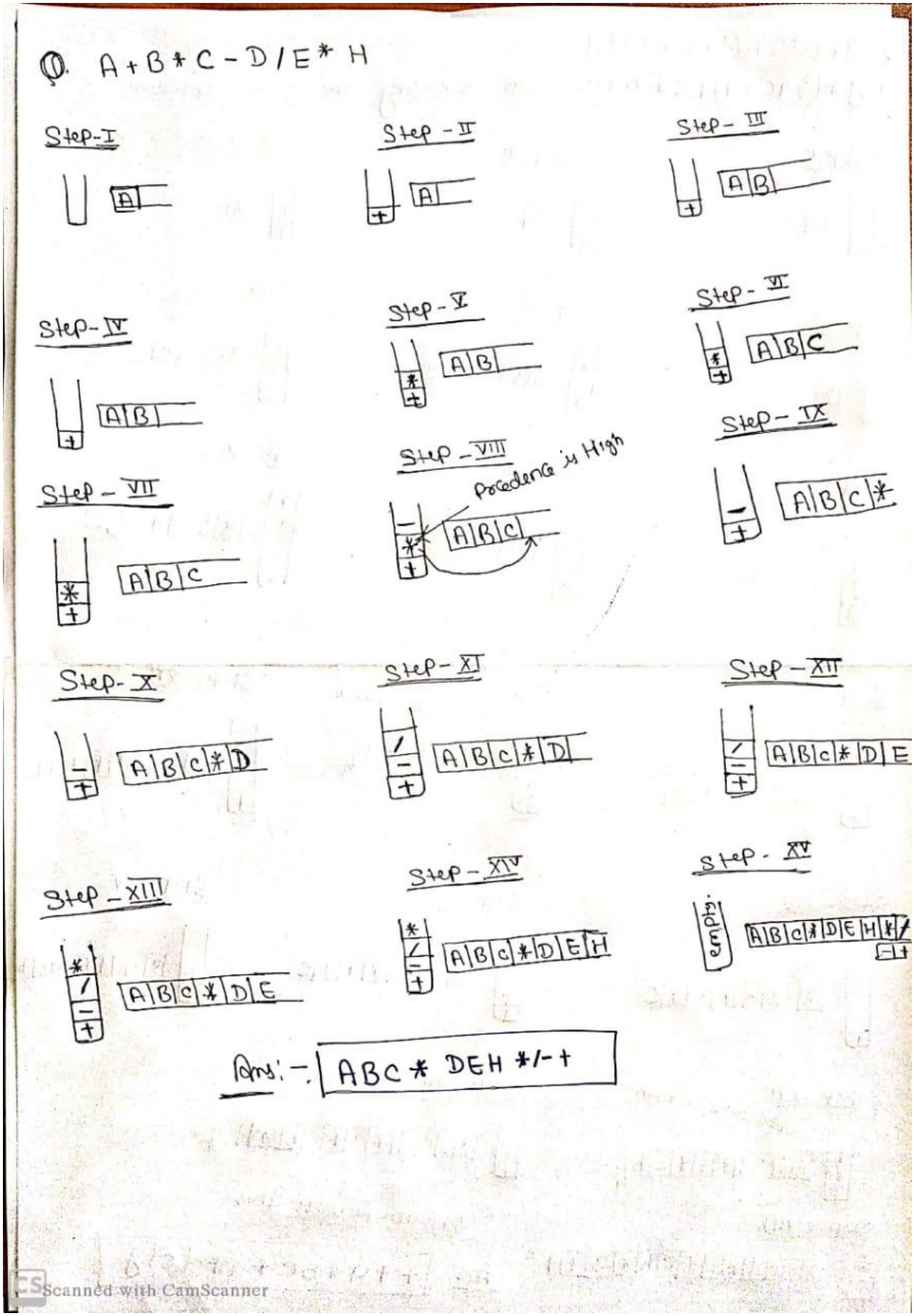
3. At last there may be a situation that there is some operator which is not inside any parenthesis for that simply pop the element from the stack until it became empty and put it to the postfix array.

This Area left blank for personal notes :

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

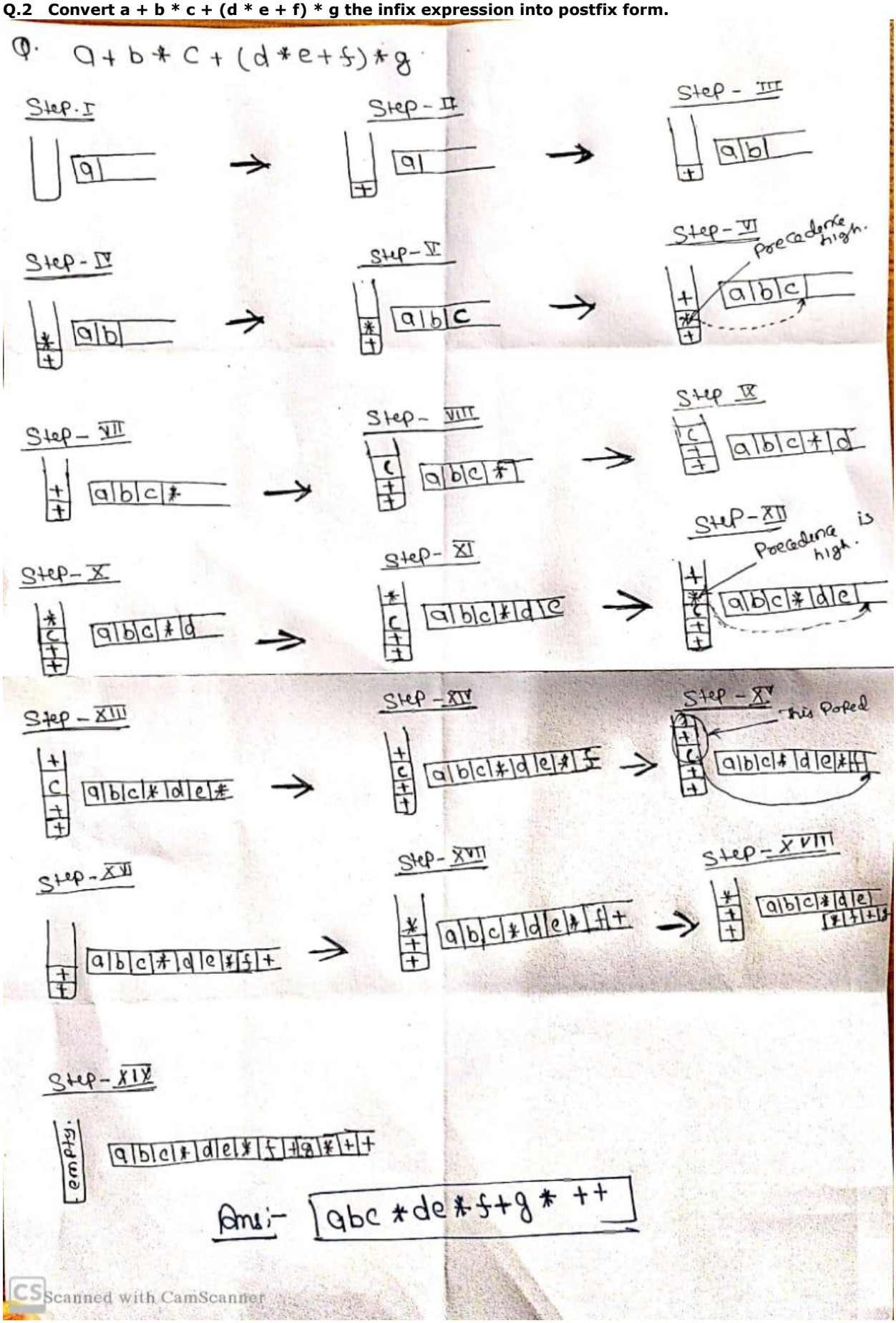
Q. Convert the following infix expression A + B * C – D / E * H into its equivalent postfix expression.



Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

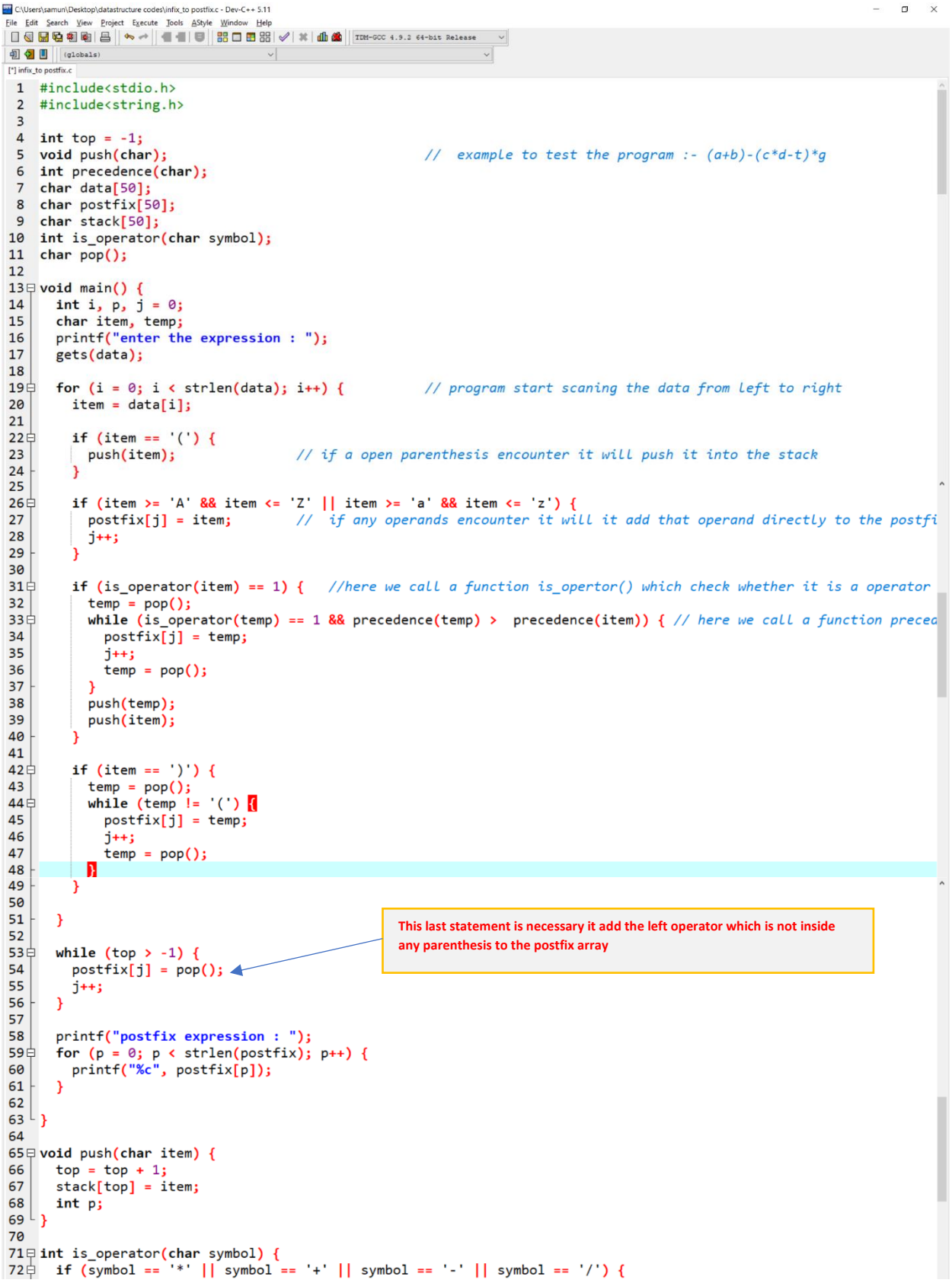
Q.2 Convert a + b * c + (d * e + f) * g the infix expression into postfix form.



Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Program to implement the infix to postfix operation:-



Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

```
73      return 1;
74    } else {
75      return 0;
76    }
77  }
78  char pop() {
79    char item;
80    item = stack[top];
81    top--;
82    return (item);
83  }
84  int precedence(char symbol) {
85    if (symbol == '*' || symbol == '/')
86      return 1;
87    if (symbol == '+' || symbol == '-')
88      return 0;
89
90  }
```

Compiler | Resources | Compile Log | Debug | Find Results

Line: 48 Col: 8 Sel: 0 Lines: 90 Length: 2033 Insert Done parsing in 0.016 seconds

LOGIC :

Step1: first we have to create a basic structure by creating three arrays:

1.**data [50]** which hold the input data or the infix expression

.

2.**postfix [50]** which hold our output or the postfix expression

.

3.**stack [50]** which help us to perform the push and pop operation.

Step2: here we required four functions:

1.**push ():** it simply push the operators or open braces on the stack and each time push operation performed then the top value get incremented.

2.**pop ():**is simple return the last value of the stack and also decrement the top value.

3.**is_operator ():** it simply checks whether there is an operator or not and return true (or 1) if operator found.

4.**precedence ():** it checks the precedence of the operator by comparing operator available on the stack and the operator that is going to insert in the stack. Higher precedence return value 1 and lower precedence return value 0;

EXECUTION FLOW:

1.first user input his data in the data [50] array using gets function.

2.A for loop executed which run to the strlen(data) and store the data in a variable item. This step is important because we Can run our if condition by making item as our base.

3.There are 4 possible case happens:

a.) if it encounters an **open parenthesis** then is simply store it into the stack array.

b.) if it encounters **an operand** then is simply store it into the postfix array.

c.) if it encounters an **operator** then it calls a function is_operator () which return a true value if it is an operator otherwise the if block skipped. Inside this if block if operator is encountered then first, we use a temp variable in which the last element of the stack is stored then again, we check that the last element is an operator or not simultaneously we check precedence of the input and the data in the temp. if the precedence of the temp is greater means that the operator in the stack has the higher precedence then the current inserted data then it will store in the postfix array otherwise it will store the data in the stack again (since it pop out then we have to again store it back). First it will store the popped data then it stores the current upcoming data.

d.) if it encounters a **close parenthesis** then a while loop is run which add the stack's data until a open parenthesis is encountered.

4. at last there is one case which deal with the situation where no parenthesis occurs or there must be some operator which is not inside any parenthesis for that we have to empty our stack and add the operator inside the stack to our postfix array for this we run a while loop which run until top > -1 means to the initial stage of the stack and it will pop each and every data and insert it to the postfix array.

Written by:
SAMUNDAR SINGH

2.Conversion from infix to prefix:

Procedure to convert from infix expression to postfix expression is as follows: -

1.reverse the infix expression and also convert open parenthesis into closed parenthesis and vice versa.

2.Scan the infix expression from **left to right .**

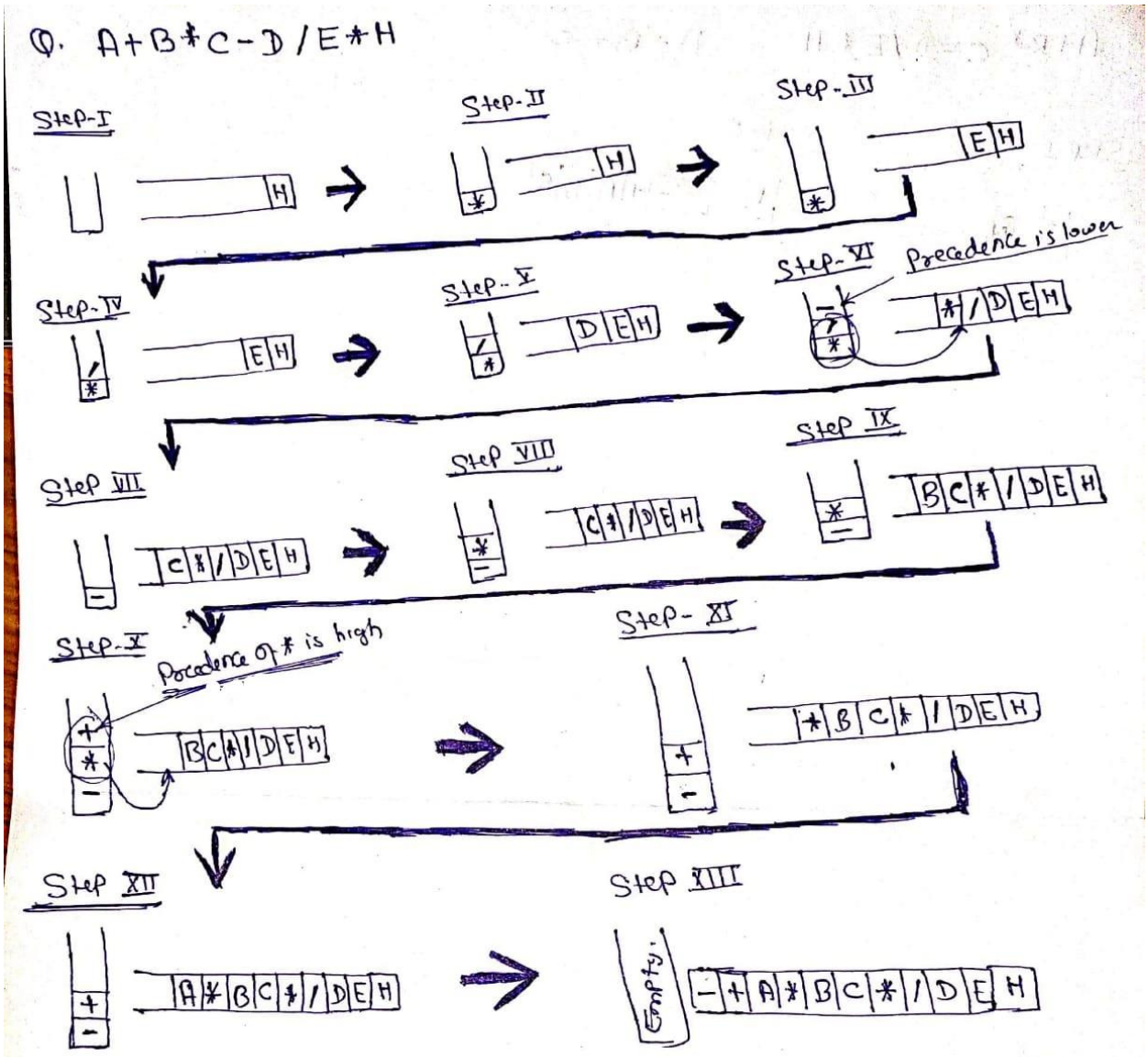
3. a) If the scanned symbol is open parenthesis, push it onto the stack.
   b) If the scanned symbol is an operand, then place directly in the prefix expression (output).
   c) If the symbol scanned is a closed parenthesis, then go on popping all the items from the stack and place them in the prefix expression till we get the matching left parenthesis.
   d) If the scanned symbol is an operator, then go on removing all the operators from the stack and place them in the prefix expression, if and only if the precedence of the operator which is on the top of the stack is greater than (or greater than or equal) to the precedence of the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack.

4. There may be a situation that there is some operator which is not inside any parenthesis for that simply pop the element from the stack until it became empty and put it to the prefix array.

5. At last we have to reverse all output once again to get the result.

Example :

Q.1 Convert the following infix expression A + B * C – D / E * H into its equivalent prefix expression.

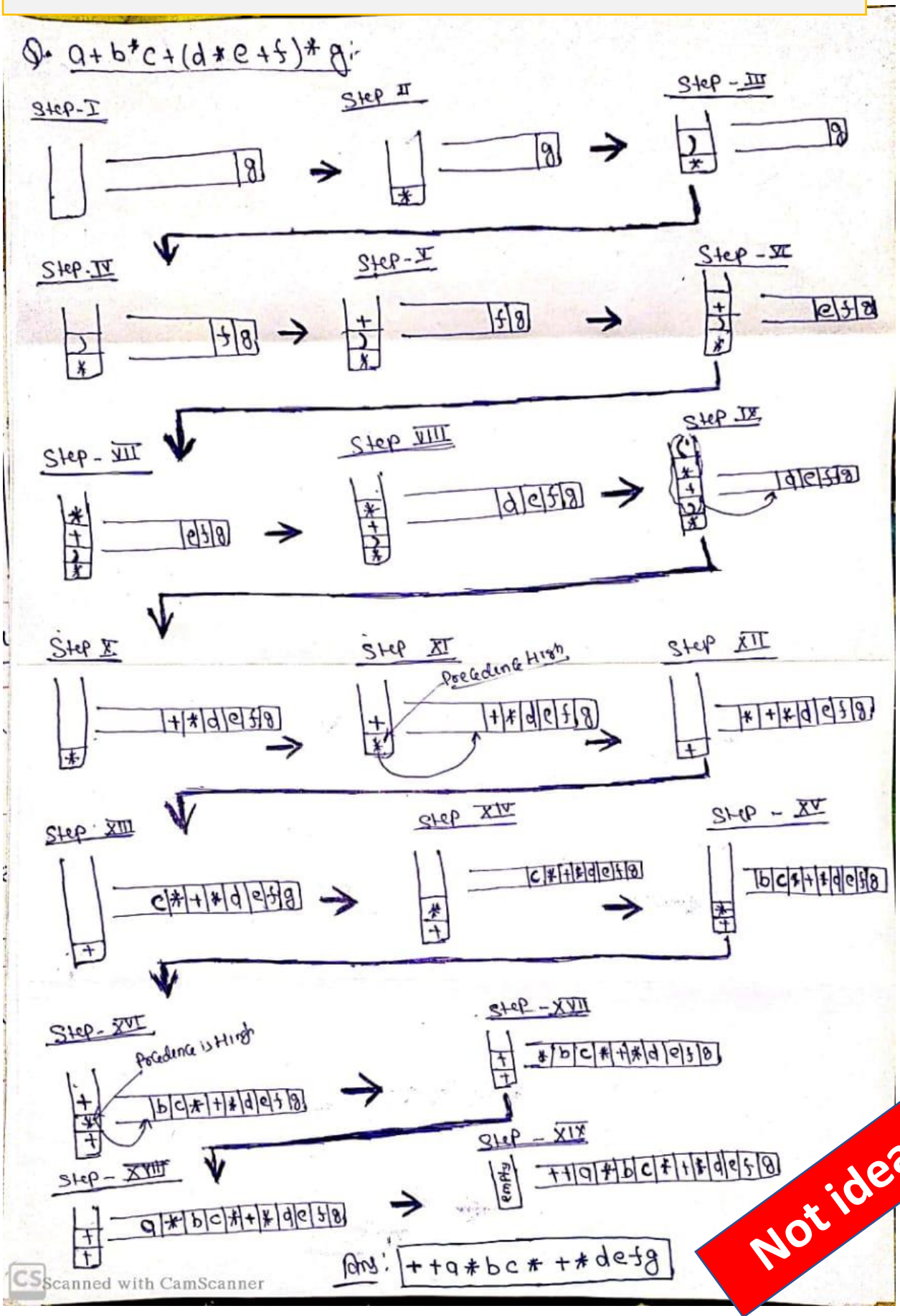


Q.2 Convert a + b * c + (d * e + f) * g the infix expression into prefix form

Ans :- −+A*BC*/DEH

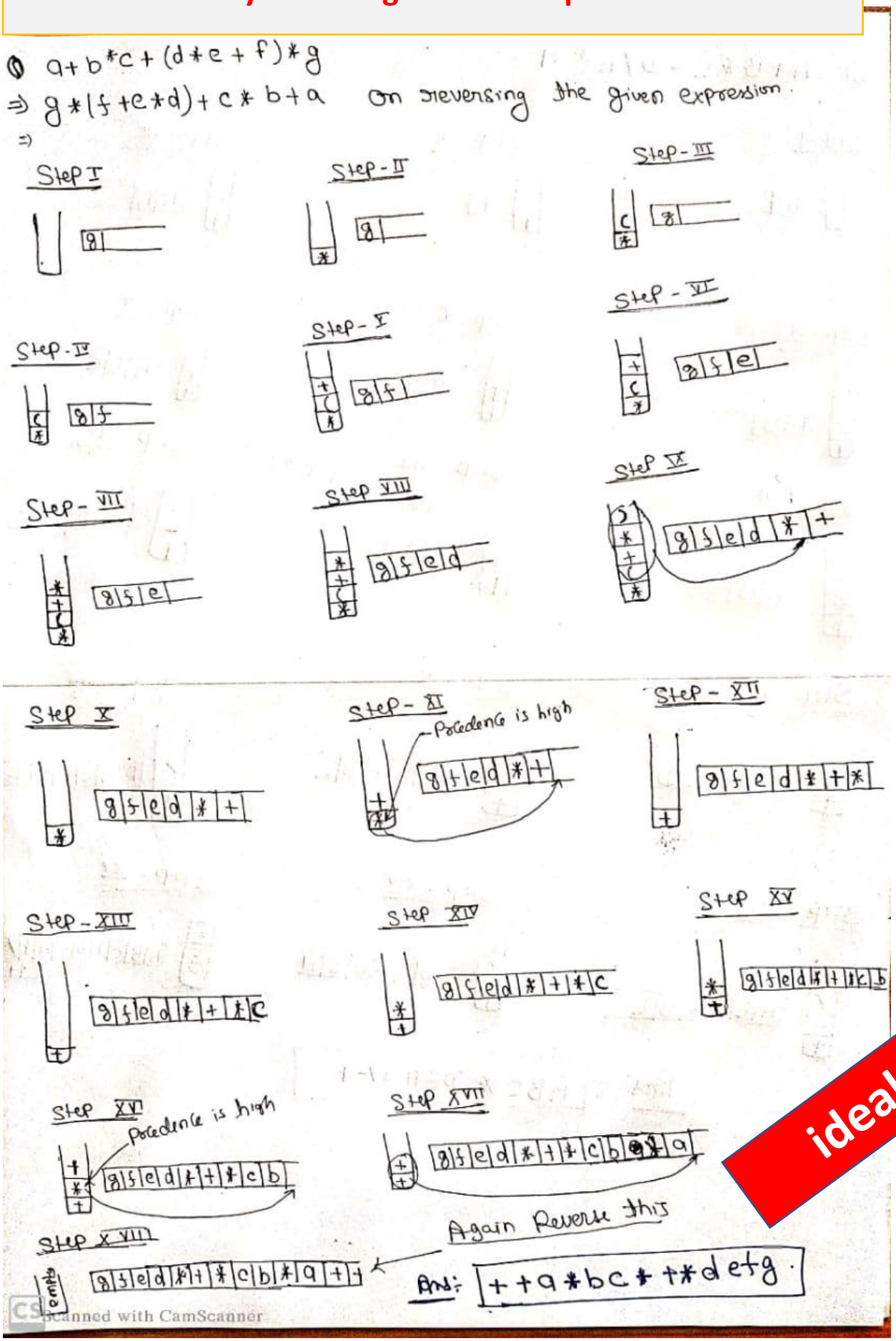
DATA STRUCTURE NOTES

Without reversing the infix expression



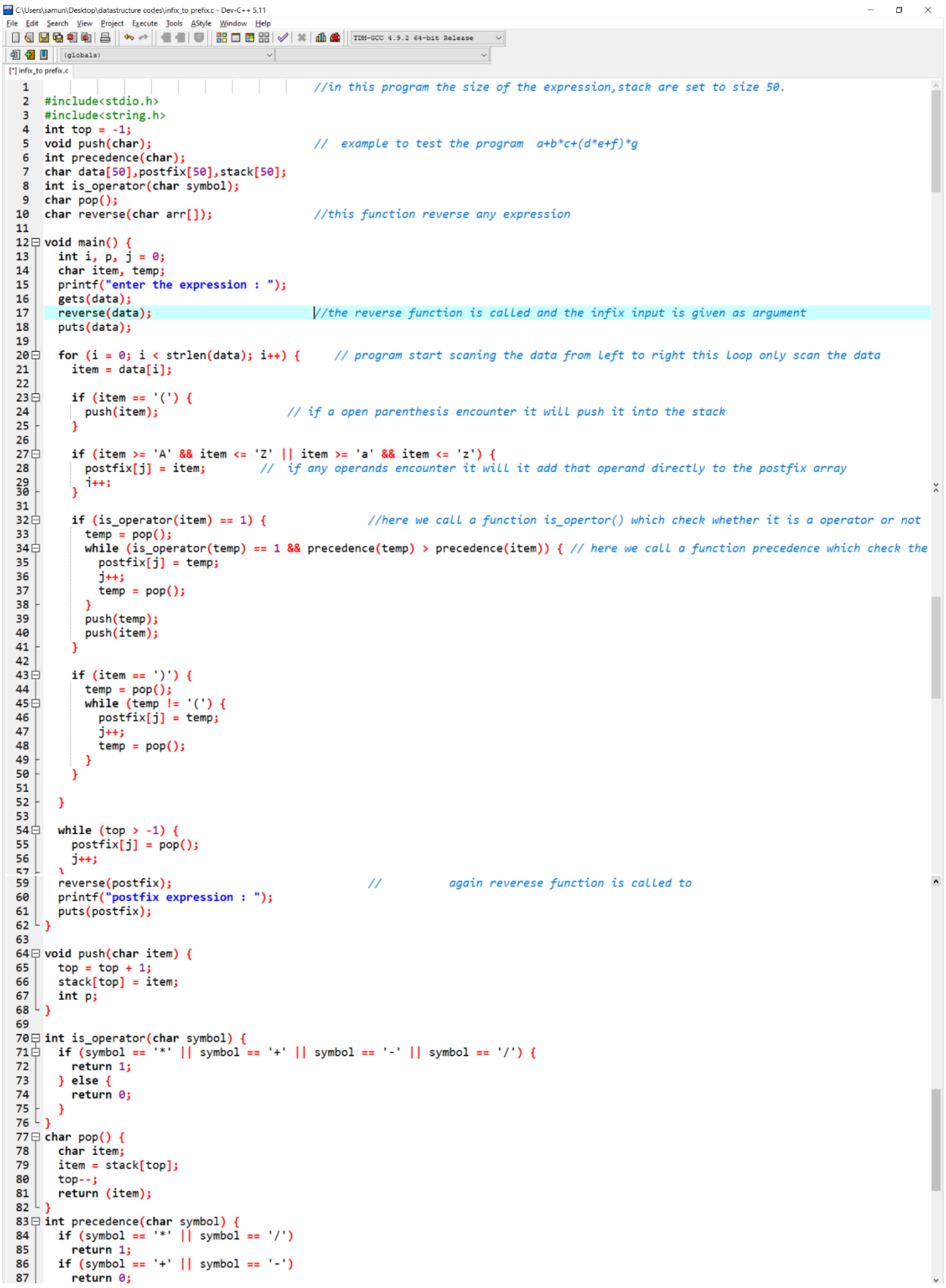
Written by:
SAMUNDAR SINGH

By reversing the infix expression



Written by:
SAMUNDAR SINGH

Program to implement the infix to prefix operation:-



```c
//in this program the size of the expression,stack are set to size 50.
#include<stdio.h>
#include<string.h>
int top = -1;
void push(char);                        //  example to test the program  a+b*c+(d*e+f)*g
int precedence(char);
char data[50],postfix[50],stack[50];
int is_operator(char symbol);
char pop();
char reverse(char arr[]);               //this function reverse any expression

void main() {
  int i, p, j = 0;
  char item, temp;
  printf("enter the expression : ");
  gets(data);
  reverse(data);                        //the reverse function is called and the infix input is given as argument
  puts(data);

  for (i = 0; i < strlen(data); i++) {     // program start scaning the data from left to right this loop only scan the data
    item = data[i];

    if (item == '(') {
      push(item);                   // if a open parenthesis encounter it will push it into the stack
    }

    if (item >= 'A' && item <= 'Z' || item >= 'a' && item <= 'z') {
      postfix[j] = item;        //  if any operands encounter it will it add that operand directly to the postfix array
      j++;
    }

    if (is_operator(item) == 1) {              //here we call a function is_opertor() which check whether it is a operator or not
      temp = pop();
      while (is_operator(temp) == 1 && precedence(temp) > precedence(item)) { // here we call a function precedence which check the
        postfix[j] = temp;
        j++;
        temp = pop();
      }
      push(temp);
      push(item);
    }

    if (item == ')') {
      temp = pop();
      while (temp != '(') {
        postfix[j] = temp;
        j++;
        temp = pop();
      }
    }

  }

  while (top > -1) {
    postfix[j] = pop();
    j++;
  }
  reverse(postfix);                          //          again reverese function is called to
  printf("postfix expression : ");
  puts(postfix);
}

void push(char item) {
  top = top + 1;
  stack[top] = item;
  int p;
}

int is_operator(char symbol) {
  if (symbol == '*' || symbol == '+' || symbol == '-' || symbol == '/') {
    return 1;
  } else {
    return 0;
  }
}
char pop() {
  char item;
  item = stack[top];
  top--;
  return (item);
}
int precedence(char symbol) {
  if (symbol == '*' || symbol == '/')
    return 1;
  if (symbol == '+' || symbol == '-')
    return 0;
```

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

```
88
89  }
90
91  char reverse(char arr[]){
92      int i,q;
93      char temp[50];
94      q=strlen(arr)-1;
95      for(i=0;i<strlen(arr);i++){
96          if(arr[i]=='(')
97              arr[i]=')';
98          else if(arr[i]==')')
99              arr[i]='(';
100         temp[q--]=arr[i];
101
102     }
103     for(i=0;i<strlen(arr);i++){
104         arr[i]=temp[i];
105     }
106     return(arr[50]);
107 }
108
109
110
111
```

Compiler | Resources | Compile Log | Debug | Find Results

Line: 17 Col: 41 Sel: 0 Lines: 111 Length: 2616 Insert Done parsing in 0.015 seconds

This reverse function reverse the input and change the braces of input and at last once this function will reverse the output

LOGIC :

Step1: first we have to create a basic structure by creating three arrays:

1.**data [50]** which hold the input data or the infix expression

.

2.**prefix [50]** which hold our output or the postfix expression

.

3.**stack [50]** which help us to perform the push and pop operation.

This is the extra element from the previous one program

Step2: here we required four functions:

1.**reverse ():**this function will reveres the array and convert open parenthesis to closed parenthesis and vice versa..

2.**push ():** it simply push the operators or open braces on the stack and each time push operation performed then the top value get incremented.

3.**pop ():**is simple return the last value of the stack and also decrement the top value.

4.**is_operator ():** it simply checks whether there is an operator or not and return true (or 1) if operator found.

5.**precedence ():** it checks the precedence of the operator by comparing operator available on the stack and the operator that is going to insert in the stack. Higher precedence return value 1 and lower precedence return value 0;

EXECUTION FLOW:

1.first user input his data in the data [50] array using gets function.

1.then reverse function is called which reverse the elements in the data array .

2.A for loop executed which run to the strlen(data) and store the data in a variable item. This step is important because we Can run our if condition by making item as our base.

3.There are 4 possible case happens:

a.) if it encounters an **open parenthesis** then is simply store it into the stack array.

b.) if it encounters **an operand** then is simply store it into the postfix array.

c.) if it encounters an **operator** then it calls a function is_operator () which return a true value if it is an operator otherwise the if block skipped. Inside this if block if operator is encountered then first, we use a temp variable in which the last element of the stack is stored then again, we check that the last element is an operator or not simultaneously we check precedence of the input and the data in the temp. if the precedence of the temp is greater means that the operator in the stack has the higher precedence then the current inserted data then it will store in the postfix array otherwise it will store the data in the stack again (since it pop out then we have to again store it back). First it will store the popped data then it stores the current upcoming data.

d.) if it encounters a **close parenthesis** then a while loop is run which add the stack's data until a open parenthesis is encountered.

4. at last there is one case which deal with the situation where no parenthesis occurs or there must be some operator which is not inside any parenthesis for that we have to empty our stack and add the operator inside the stack to our postfix array for this we run a while loop which run until top > -1 means to the initial stage of the stack and it will pop each and every data and insert it to the postfix array.

This Step is very important

5.At last once again we reverse the output to get the result

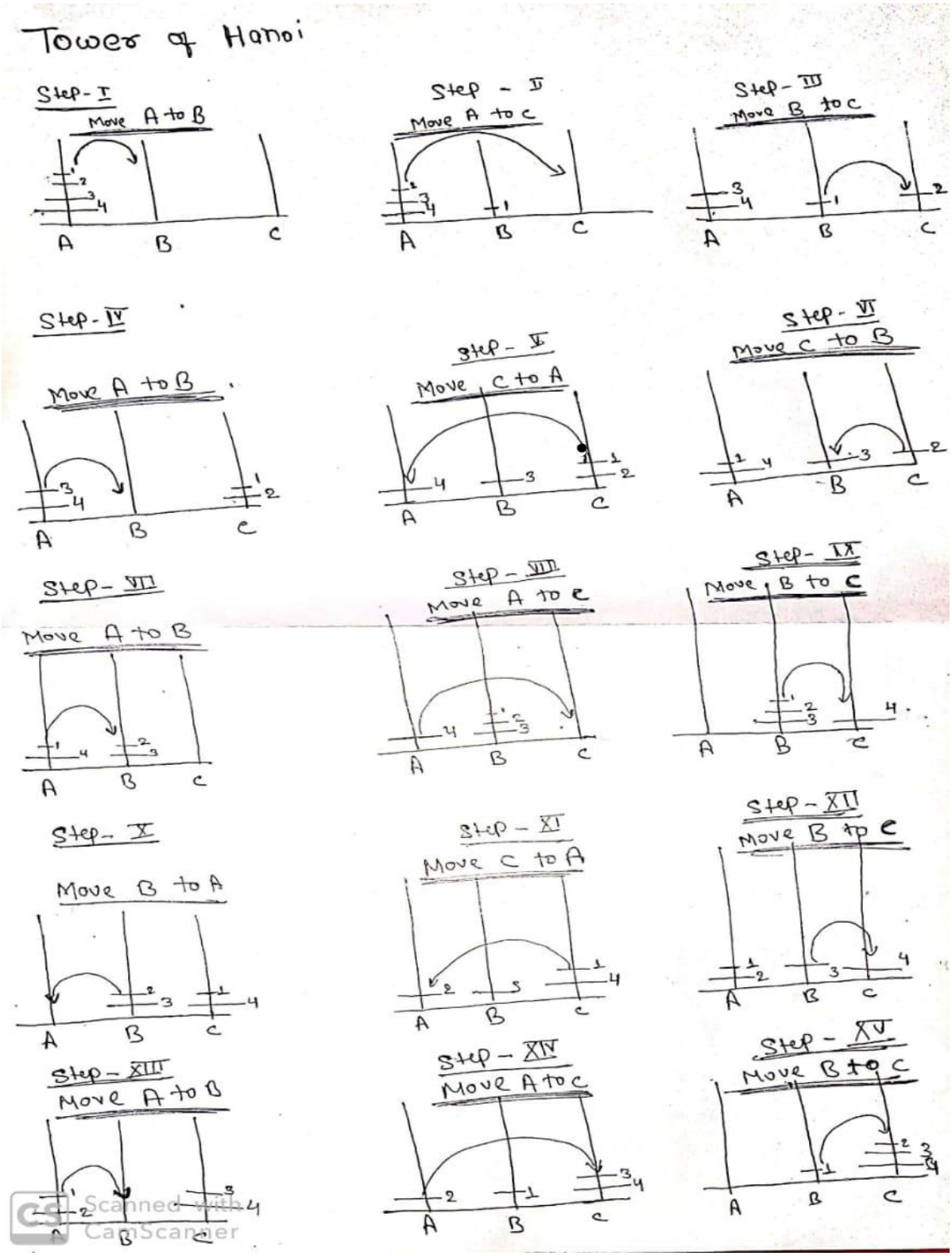
Written by:
SAMUNDAR SINGH

4. Tower of Hanoi:-

In this problem we have to transfer a set of disk from one pole to another pole by following some certain rule :

1. Only one disk moved at a time.
2. Only top of the disk can be moved from one tower to other tower.
3. Larger disk can't be placed on another smaller disk.

Example: Moving 4 disks from pole A to pole B.

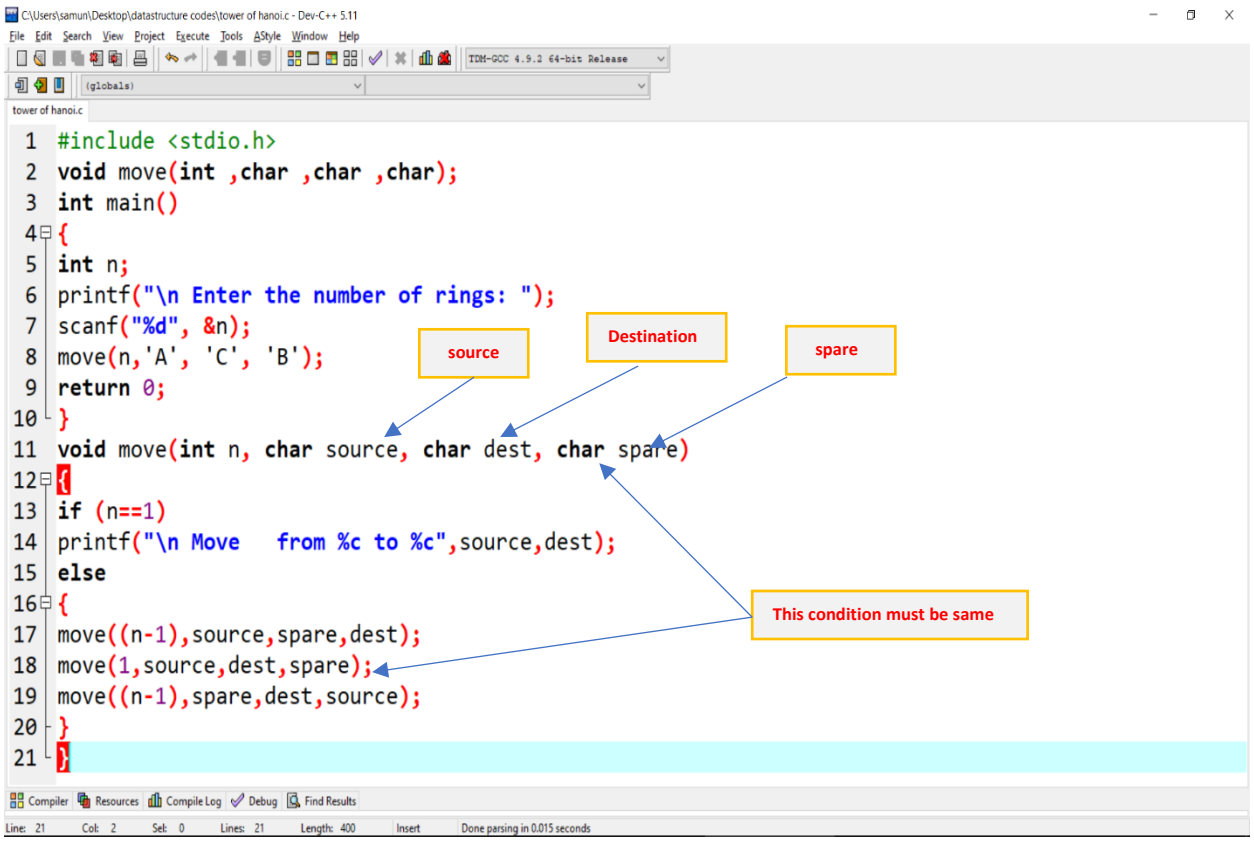


Written by:
SAMUNDAR SINGH

From above example :

A is our source pole , C is our destination pole and since we have a restriction that we can't place a larger disk on smaller disk so we are going to use a another pole named B (spare pole).

Program to implement the Tower Of Hanoi:-



LOGIC :

we use recursion to solve the problem of tower of Hanoi it include following base and recursive case:

Base case: `if n=1`
Move the ring from A to C using B as spare

Recursive case:
Move `n – 1` rings from A to B using C as spare
Move the one ring left on A to C using B as spare
Move `n – 1` rings from B to C using A as spare

Please always maintained the order of the function in which it was called otherwise have to face big error.

When defining the function move(int **n**, char **source**, char **destination** , char **spare**)

Base case : nothing just print out the source and destination ; Trick to remember

Recursive case: move((n-1), source, spare, destination). SO-SP-DE

move(1,source, destination, spare) SO-DE-SP

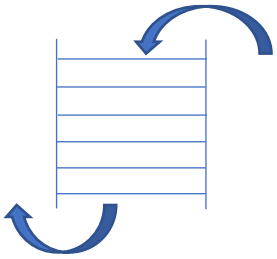
move((n-1), spare, destination, source).. SP-DE-SO

We can also calculate total number of turn required to transfer all disk from one pole to another pole by using the equation: **(2^n -1)** where n = total number of rings

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

QUEUE:



Queue using Array:

A Queue is a linear data structure which use FIFO (First-In-First-Out) principle. The elements in a queue are added at one end called the REAR and removed from the other end called the FRONT.

Code for menu driven program :-

```c
#include<stdio.h>
int front=-1;
int rear=-1;
void enqueue(int [],int);
void dequeue(int []);
void peek(int []);
void display(int [],int);

void main(){
    int size,choice;
    printf("enter the size of array : ");
    scanf("%d",&size);
    printf("\n");
    int arr[size];
    while(choice!=-1){
        printf("press 1 for insert data \n");
        printf("press 2 for delete data \n");
        printf("press 3 for peek data \n");
        printf("press 4 for dispaly data \n");
        printf("press -1 for Quit  \n");
            printf("\t\t\tENTER YOUR CHOICE :");
        scanf("%d",&choice);

    if(choice==1)
        enqueue(arr,size);
        else if(choice==2)
            dequeue(arr);
            else if(choice==3)
                peek(arr);
                else if (choice==4)
                    display(arr,size);
}
}
```

At initial Front and rear ==-1

If we are passing an array in a function then it must be defined

Code for enqueue :

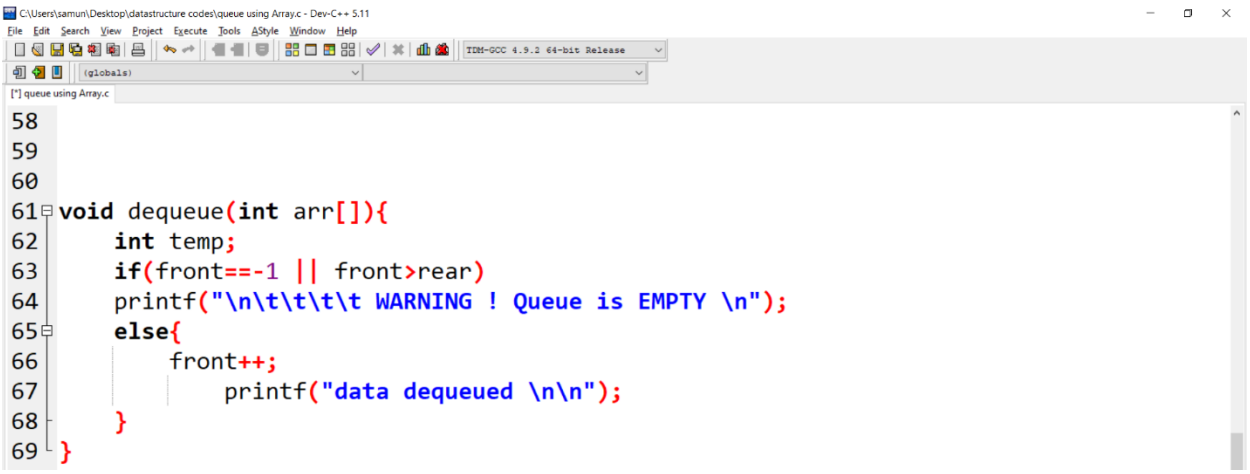
```c
void enqueue(int arr[],int size){
    int item;
    if(rear==size-1){
        printf("\n\t\t\t\t WARNING ! Queue is FULL \n");
    }
    else{
        if (front=-1){
        front=0;    // only for traversing because traversing strat from zero
    }
        printf("enter your item : ");
        scanf("%d",&item);
        rear=rear+1;
        arr[rear]=item;
        printf("data enqueued \n\n");
    }
}
```

This must be performed

Written by:
SAMUNDAR SINGH

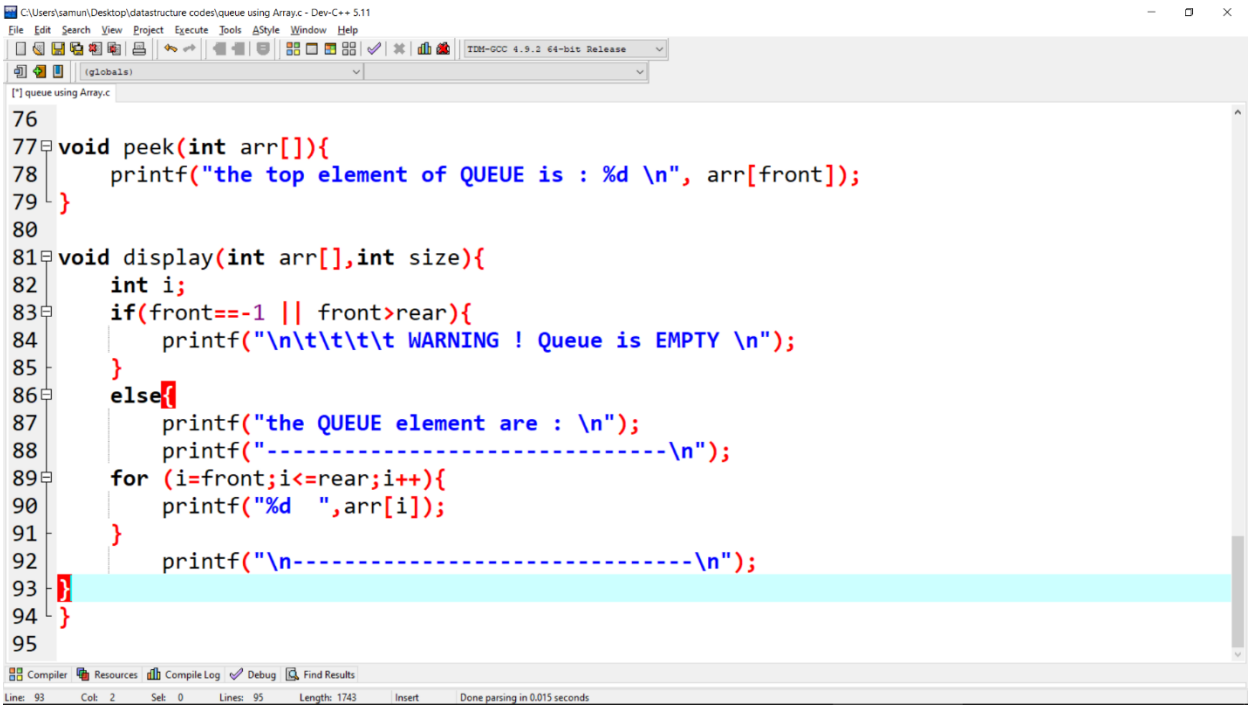
DATA STRUCTURE NOTES

Code for dequeue :



```
58
59
60
61  void dequeue(int arr[]){
62      int temp;
63      if(front==-1 || front>rear)
64      printf("\n\t\t\t\t WARNING ! Queue is EMPTY \n");
65      else{
66          front++;
67              printf("data dequeued \n\n");
68      }
69  }
```

Code for peek and display the data in queue :



```
76
77  void peek(int arr[]){
78      printf("the top element of QUEUE is : %d \n", arr[front]);
79  }
80
81  void display(int arr[],int size){
82      int i;
83      if(front==-1 || front>rear){
84          printf("\n\t\t\t\t WARNING ! Queue is EMPTY \n");
85      }
86      else{
87          printf("the QUEUE element are : \n");
88          printf("--------------------------------\n");
89      for (i=front;i<=rear;i++){
90          printf("%d  ",arr[i]);
91      }
92          printf("\n--------------------------------\n");
93  }
94  }
95
```

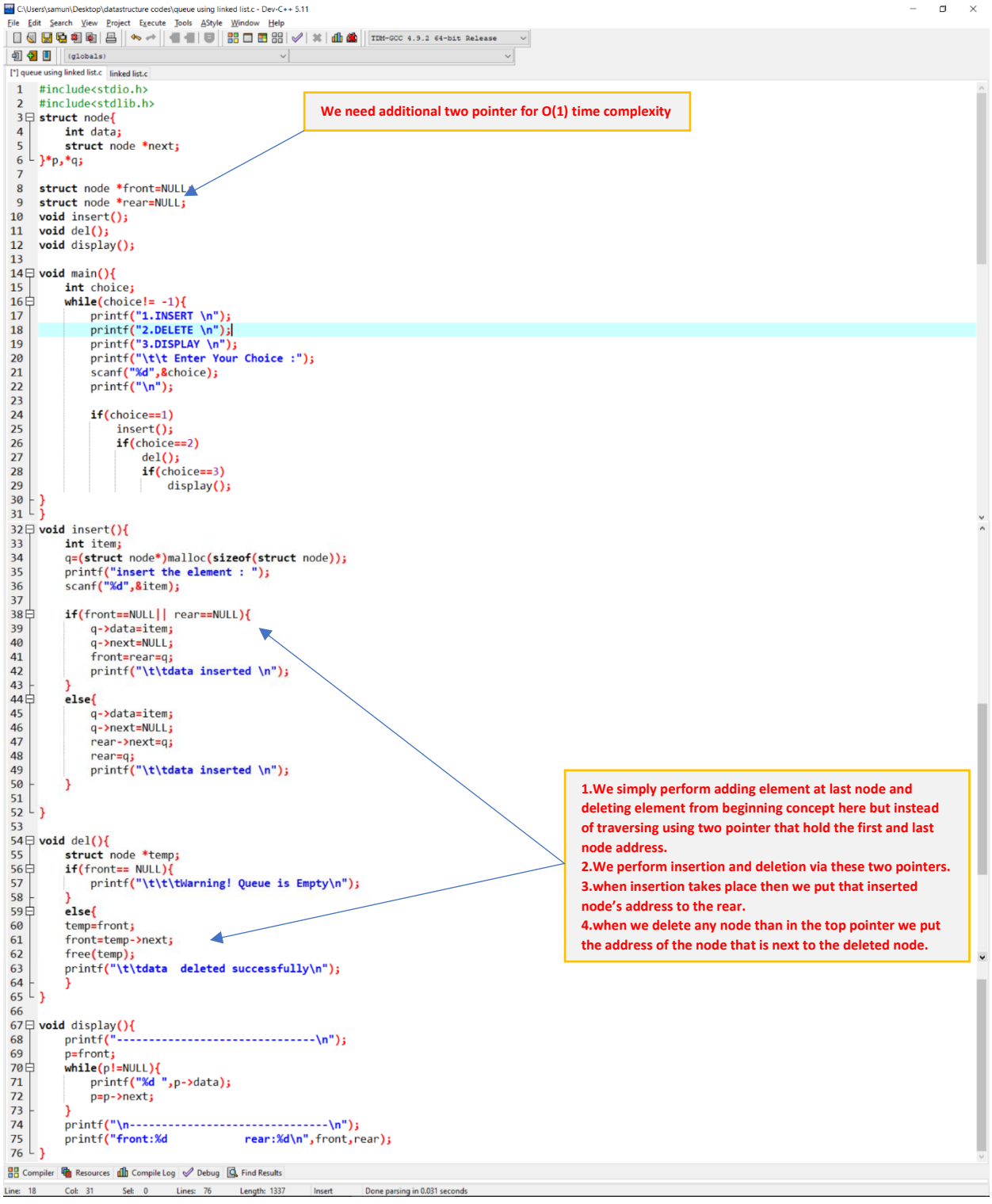
Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Queue using Linked List:

We can simply implement Queue through linked list by simply add data in the beginning and delete data from the end but doing so we have to traversed each time to get our last element. Technically we have time complexity O(n) but we want time complexity O(1). It means we don't want to traversed 'n' element to add or remove any item in the queue. In order to solve this we required two additional pointer which point to the first and the last element. Hence we use ***front** and ***rear** pointer which points the top and the last element.

Code to implement Queue using Linked List:-



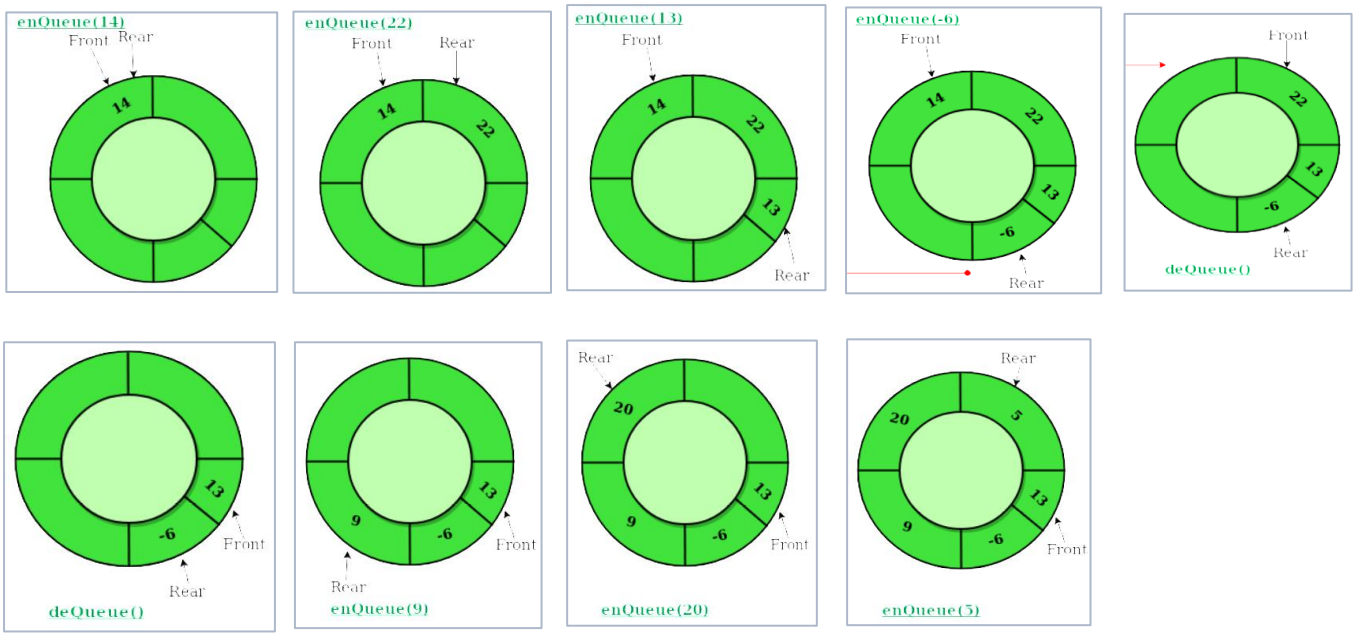

```c
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *next;
}*p,*q;

struct node *front=NULL;
struct node *rear=NULL;
void insert();
void del();
void display();

void main(){
    int choice;
    while(choice!= -1){
        printf("1.INSERT \n");
        printf("2.DELETE \n");
        printf("3.DISPLAY \n");
        printf("\t\t Enter Your Choice :");
        scanf("%d",&choice);
        printf("\n");

        if(choice==1)
            insert();
            if(choice==2)
                del();
                if(choice==3)
                    display();
}
}
void insert(){
    int item;
    q=(struct node*)malloc(sizeof(struct node));
    printf("insert the element : ");
    scanf("%d",&item);

    if(front==NULL|| rear==NULL){
        q->data=item;
        q->next=NULL;
        front=rear=q;
        printf("\t\tdata inserted \n");
    }
    else{
        q->data=item;
        q->next=NULL;
        rear->next=q;
        rear=q;
        printf("\t\tdata inserted \n");
    }

}

void del(){
    struct node *temp;
    if(front== NULL){
        printf("\t\t\tWarning! Queue is Empty\n");
    }
    else{
    temp=front;
    front=temp->next;
    free(temp);
    printf("\t\tdata  deleted successfully\n");
    }
}

void display(){
    printf("-------------------------------\n");
    p=front;
    while(p!=NULL){
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n-------------------------------\n");
    printf("front:%d            rear:%d\n",front,rear);
}
```

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Circular Queue:-

A circular queue is designed to solve the memory reuse problem. In linear queue the memory of deleted process cannot be used by other new process but in circular queue the memory of deleted process can be used by some other new process.



Step1: first we our basic structure by creating a menu driven program and since we are using a queue and require time complexity *O(1)*
So we need the front as well as rear variable.

Step2: To insert data in queue there are several case occur

Case 1: *when there is no element in the queue then front=rear==-1 in that case we put front =rear==0 and insert the item using rear.*

Case 2: *when there are some element then we simply increment the value of rear and assign the item to the array having index equal to the incremented variable.*

Note: The rear value incremented in this way "rear=(rear+1)%size;" not rear++ this is because if we don't This then the rear value always incremented to a greater number it will not repeat.

Case 3: *the queue is full when this condition is true* ***"front== (rear+1)%size"***

Step3: To delete data from the queue there are several case occur

Case 1: *when there is no element in the queue then front=rear==-1 in that case the Queue is Empty.*

Case 2: *if front ==rear occurs then it means that we are in a position that whatever we inserted is now deleted so we need a fresh Restart so we simply put front=rear=-1. Now whatever data is inserted is now stored in a fresh queue.*

. Case 3: *if queue is not empty and front and rear indicating some other data is that case we increment the front and whatever in the Front will be removed. To increment the front variable we have to do the same process as we done in enqueue.*

Note: The front value incremented in this way "front=(front+1)%size;" not front++ this is because if we don't This then the front value always incremented to a greater number it will not repeat.

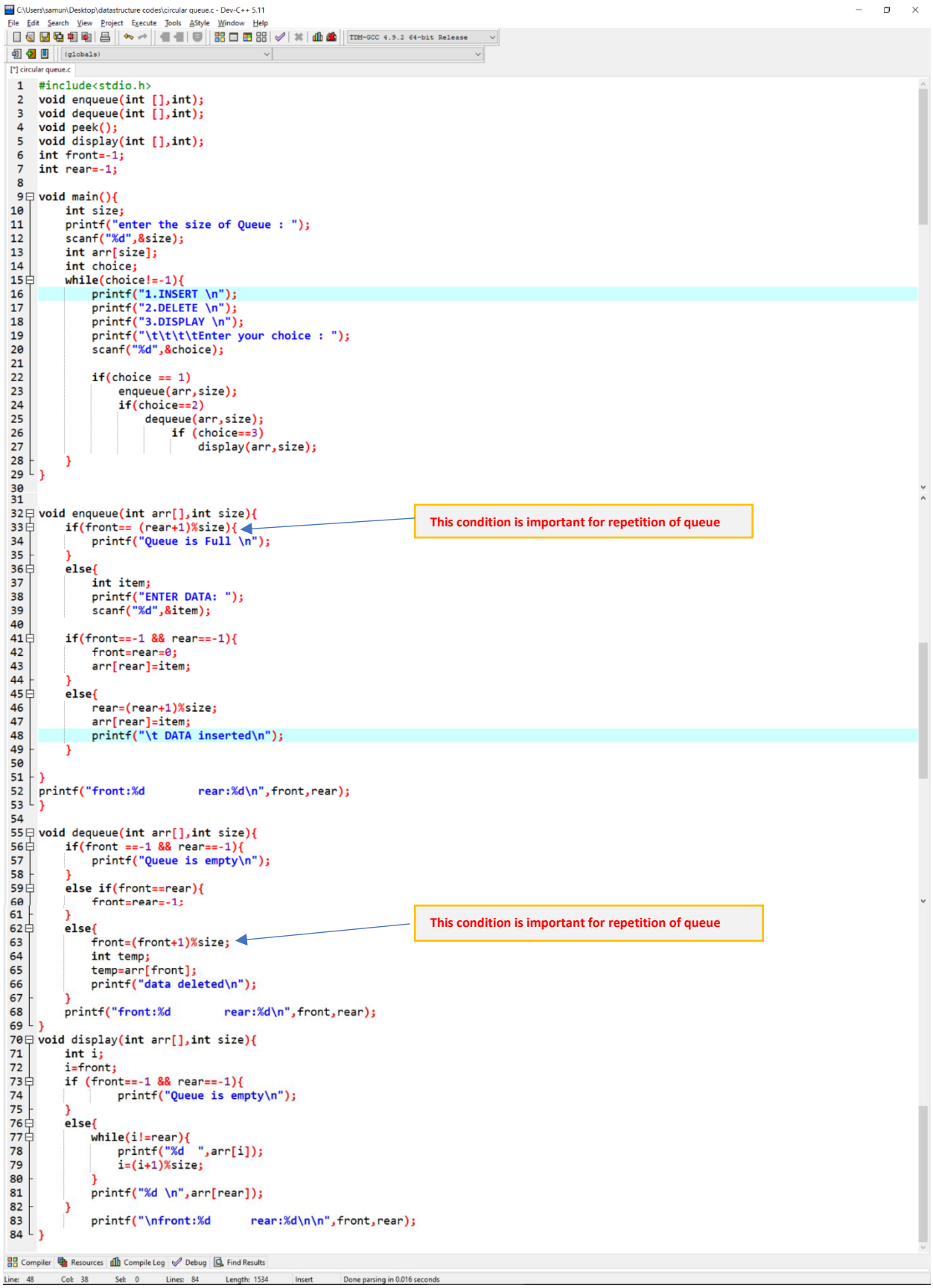
Step4: In order to display the element we have two cases

Case 1: *when there is no element in the queue then front=rear==-1 in that case the Queue is Empty.*

Case 2: *if Queue is not Empty then we use a while loop initializing* ***i=front*** *the loop terminates until its value equals to rear the value Of* ***i*** *also incremented as* ***i=(i+1)%size;***

Written by:
SAMUNDAR SINGH

Code to implement Circular Queue using Array:-



Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

TREES

A tree is a non-linear data structure. It stores data in hierarchical structure. The node that is in the top of hierarchy is called as root node. There are various type of Tree here we discussed some of them.

1.Binary Tree:

Properties:

1. A binary tree has at most 2 children in a node it means it have 0 child ,1 child or 2 children.
2. The maximum number at any level "n" is $\mathbf{2^n}$. (where n is the level of the tree).
3. Calculation of nodes when height is given:
   a.) The maximum number of nodes of height "h" is $\boldsymbol{n = 2^{h+1} - 1}$.
   b.) Minimum number of nodes of height "h" is **n=h+1**.

4. Calculation of height when node are given:
   a.) The minimum height when maximum number of node is given: $\boldsymbol{h = \log_2(n+1)}$ **-1**
   b.) Maximum height when minimum number of node is given: **h=n-1.**

Types of binary tree:

1. Strict Binary Tree: A binary tree is called *strict binary tree* if each node has exactly two children or no children.

2.Full Binary Tree: A binary tree is called *full binary tree* if each node has exactly two children and all leaf nodes are at the same level.

3. Complete Binary Tree: A *complete binary tree* is a binary tree that satisfies two properties. First, in a complete binary tree, every level, except possibly the last, is completely filled. Second, all nodes appear as far left as possible.

Representation of the binary tree in the memory:

```
struct node {
struct node *left;
int data;
struct node *right;
};
```

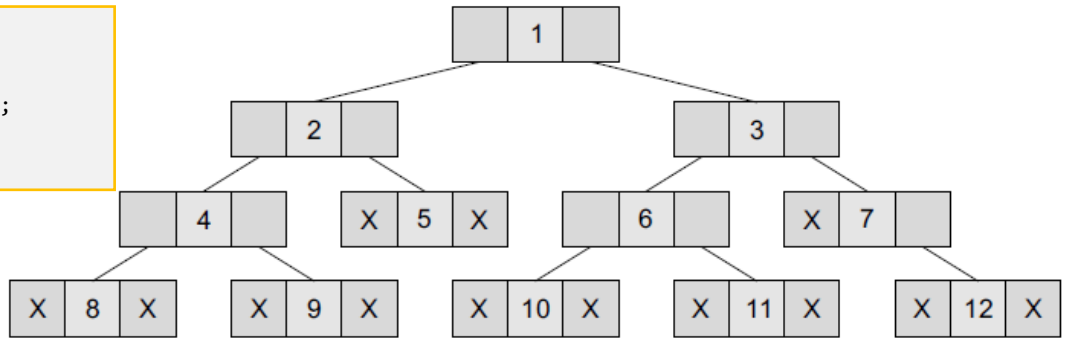


Fig: Linked representation of the binary tree.

Binary Tree Traversal:

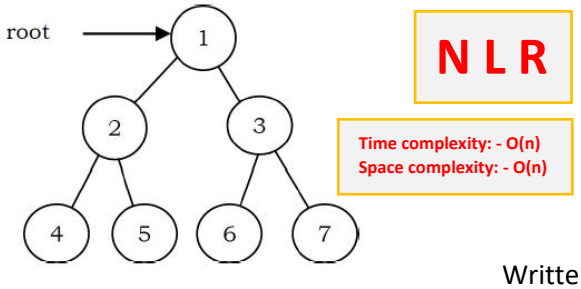
The process of visiting all nodes of a tree is called *tree traversal*. In linear data structure the element are visited in sequential order but in tree there are many different ways that are:-

1.**Pre-order Traversal (*depth-first traversal*):**

In pre-order traversal the following operation performed recursively at each node the algorithm works by:

1. Visiting the root node,
2. Traversing the left sub-tree, and finally
3. Traversing the right sub-tree.

The node will be visited in this order: **1 2 4 5 3 6 7**



Written by:
SAMUNDAR SINGH

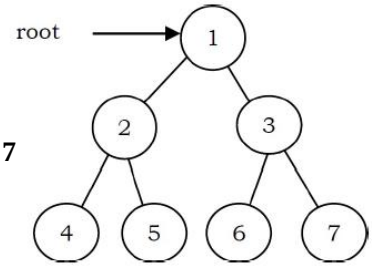
DATA STRUCTURE NOTES

2.In-order Traversal (*symmetric traversal*):

In In-order traversal the following operation performed recursively at each node the algorithm works by:

1. Traversing the left sub-tree,
2. Visiting the root node, and finally
3. Traversing the right sub-tree.

The node will be visited in this order: **4 2 5 1 6 3 7**



L N R

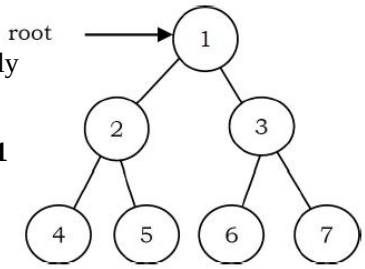
Time complexity: - O(n)
Space complexity: - O(n)

3.Post-order Traversal:

In Post-order traversal the following operation performed recursively at each node the algorithm works by:

1. Traversing the left sub-tree,
2. Traversing the right sub-tree, and finally
3. Visiting the root node.

The node will be visited in this order: **4 5 2 6 7 3 1**



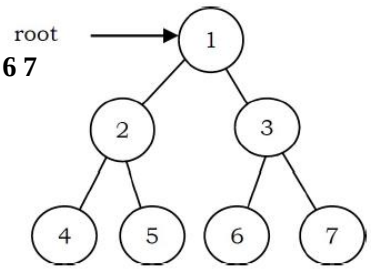
L R N

Time complexity: - O(n)
Space complexity: - O(n)

4.Level-order Traversal (*breadth-first traversal algorithm*):

In level-order traversal all the node at a level are accessed before going to the next level and data are scanned from left to right.

The node will be visited in this order: **1 2 3 4 5 6 7**



Time complexity: - O(n)
Space complexity: - O(n)

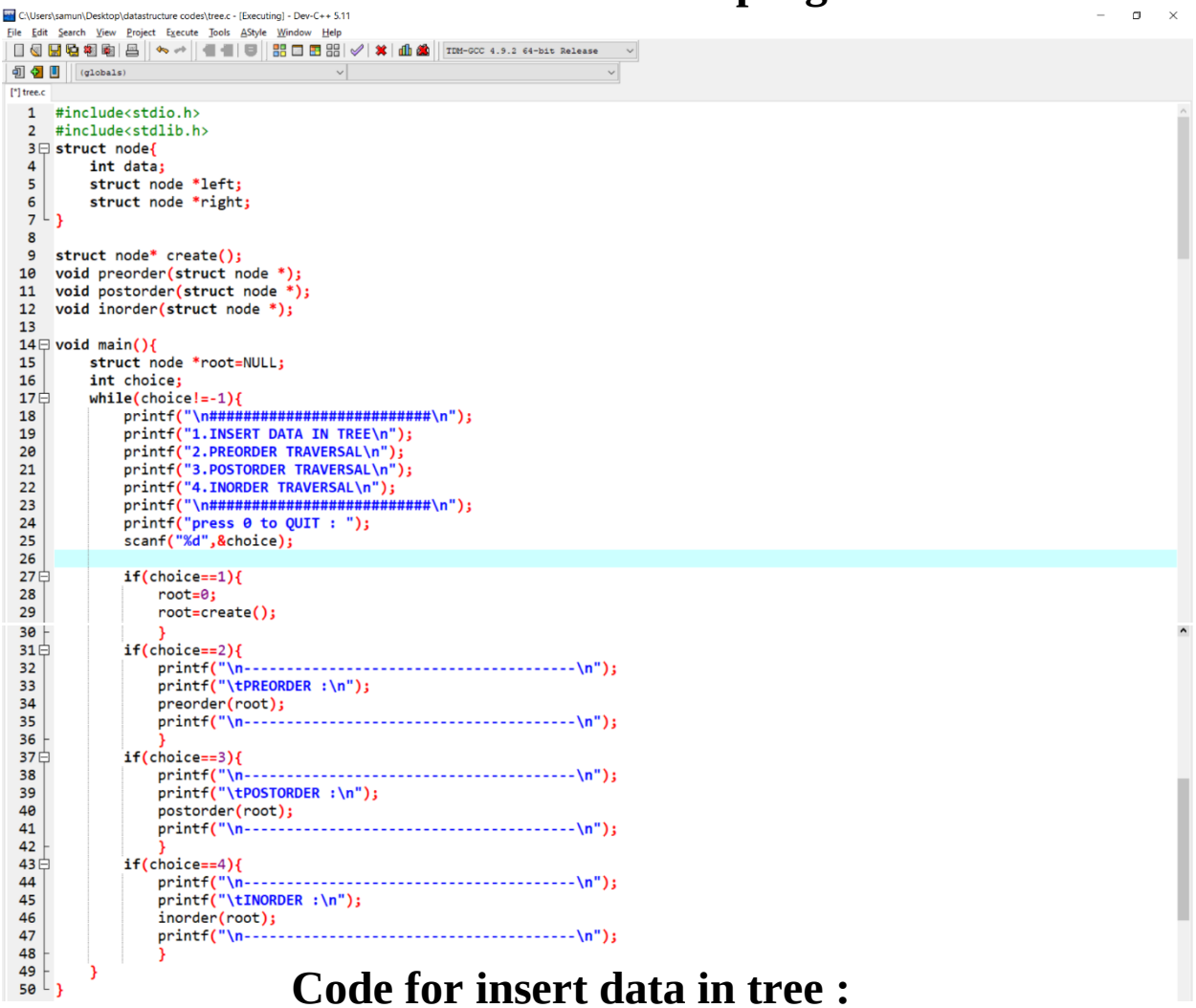
NOTE: We can implement the binary tree traversal in both recursive and non-recursive traversal algorithm.

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

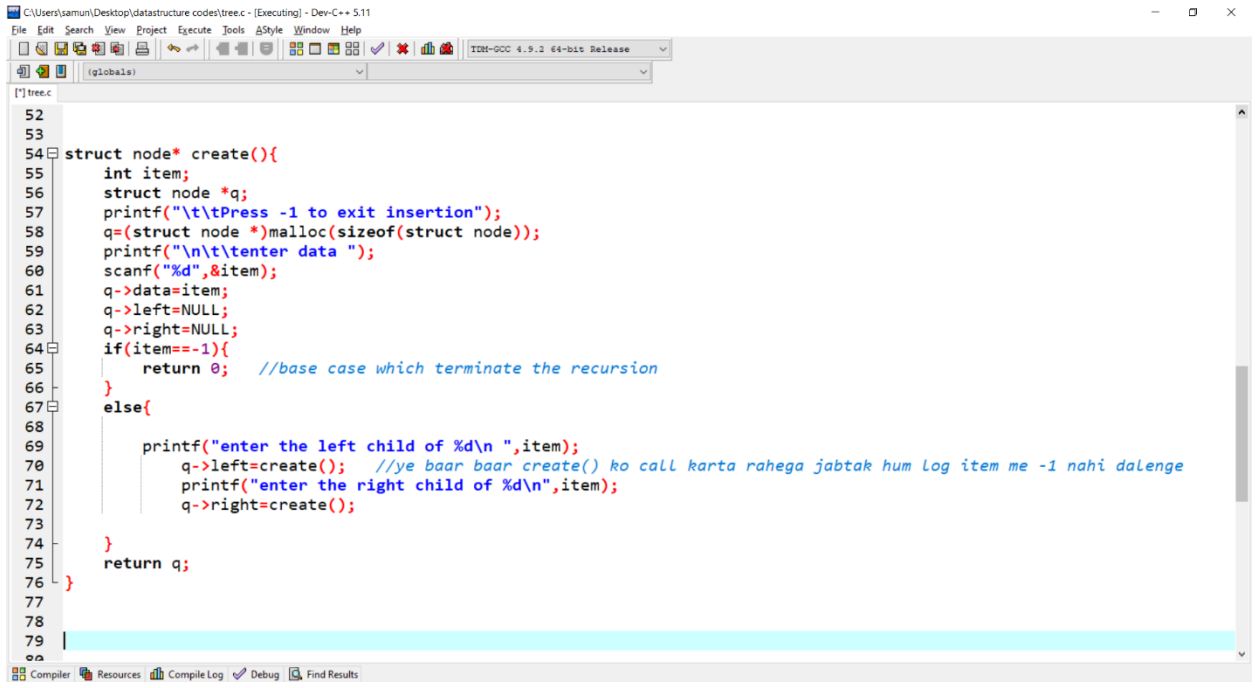
Code to Create Tree and its perform Traversal :-

Code for menu driven program :-



```c
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *left;
    struct node *right;
}

struct node* create();
void preorder(struct node *);
void postorder(struct node *);
void inorder(struct node *);

void main(){
    struct node *root=NULL;
    int choice;
    while(choice!=-1){
        printf("\n############################\n");
        printf("1.INSERT DATA IN TREE\n");
        printf("2.PREORDER TRAVERSAL\n");
        printf("3.POSTORDER TRAVERSAL\n");
        printf("4.INORDER TRAVERSAL\n");
        printf("\n############################\n");
        printf("press 0 to QUIT : ");
        scanf("%d",&choice);

        if(choice==1){
            root=0;
            root=create();
            }
        if(choice==2){
            printf("\n---------------------------------------\n");
            printf("\tPREORDER :\n");
            preorder(root);
            printf("\n---------------------------------------\n");
            }
        if(choice==3){
            printf("\n---------------------------------------\n");
            printf("\tPOSTORDER :\n");
            postorder(root);
            printf("\n---------------------------------------\n");
            }
        if(choice==4){
            printf("\n---------------------------------------\n");
            printf("\tINORDER :\n");
            inorder(root);
            printf("\n---------------------------------------\n");
            }
    }
}
```

Code for insert data in tree :

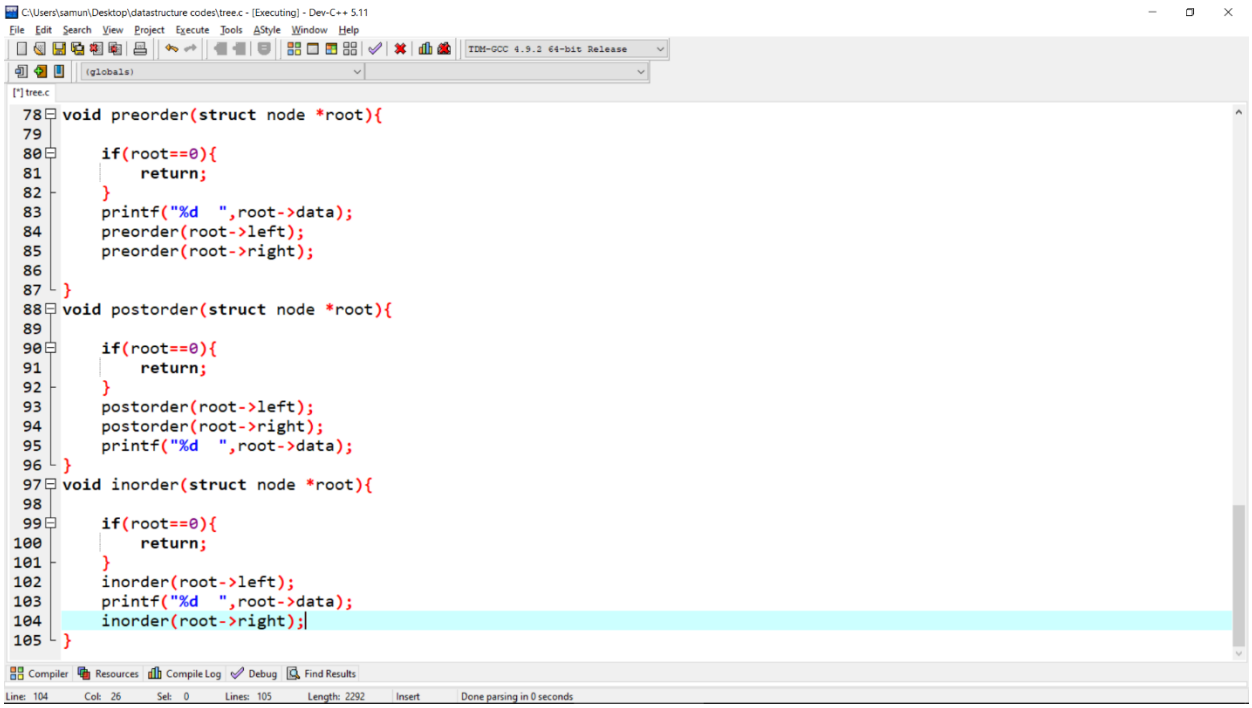


```c
struct node* create(){
    int item;
    struct node *q;
    printf("\t\tPress -1 to exit insertion");
    q=(struct node *)malloc(sizeof(struct node));
    printf("\n\t\tenter data ");
    scanf("%d",&item);
    q->data=item;
    q->left=NULL;
    q->right=NULL;
    if(item==-1){
        return 0;   //base case which terminate the recursion
    }
    else{

        printf("enter the left child of %d\n ",item);
            q->left=create();   //ye baar baar create() ko call karta rahega jabtak hum log item me -1 nahi dalenge
            printf("enter the right child of %d\n",item);
            q->right=create();

    }
    return q;
}
```

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Code for Traversal in tree :



```
78 void preorder(struct node *root){
79
80     if(root==0){
81         return;
82     }
83     printf("%d  ",root->data);
84     preorder(root->left);
85     preorder(root->right);
86
87 }
88 void postorder(struct node *root){
89
90     if(root==0){
91         return;
92     }
93     postorder(root->left);
94     postorder(root->right);
95     printf("%d  ",root->data);
96 }
97 void inorder(struct node *root){
98
99     if(root==0){
100        return;
101    }
102    inorder(root->left);
103    printf("%d  ",root->data);
104    inorder(root->right);
105 }
```

LOGIC:

1.Creating a binary tree:

Step1: first we create a structure which include 3 parts the data part that hold the data and the *left and *right pointer variable.
Step2: Then a *root address variable which hold the address of the root node, we call our create() function and this function return the root address which is stored in root so that we can then traverse from the root.
Step3: Here we are using recursion in create function ,this create function we take input form the user and recursively call for both the q->left and q->right for the sub tree until user give input -1 since it is the base condition for this recursion which stops it .
Step4: this create function each time return the address of created node which is assigned during malloc function. Hence the left and right Pointer of current node hold the address of its further sub nodes either in the left node or the right node.

1.Creating Traversal function for the tree:

Here also we have to follow the recursion

a.) PRE-ORDER Traversal:

```
void preorder (struct node *root){
    if(root==0){
        return;
        }
printf("%d  ",root->data);
preorder(root->left);
preorder(root->right);
}
```

b. POST-ORDER Traversal:

```
void postorder(struct node *root){
    if(root==0){
        return;
        }
postorder(root->left);
postorder(root->right);
printf("%d  ",root->data);
}
```

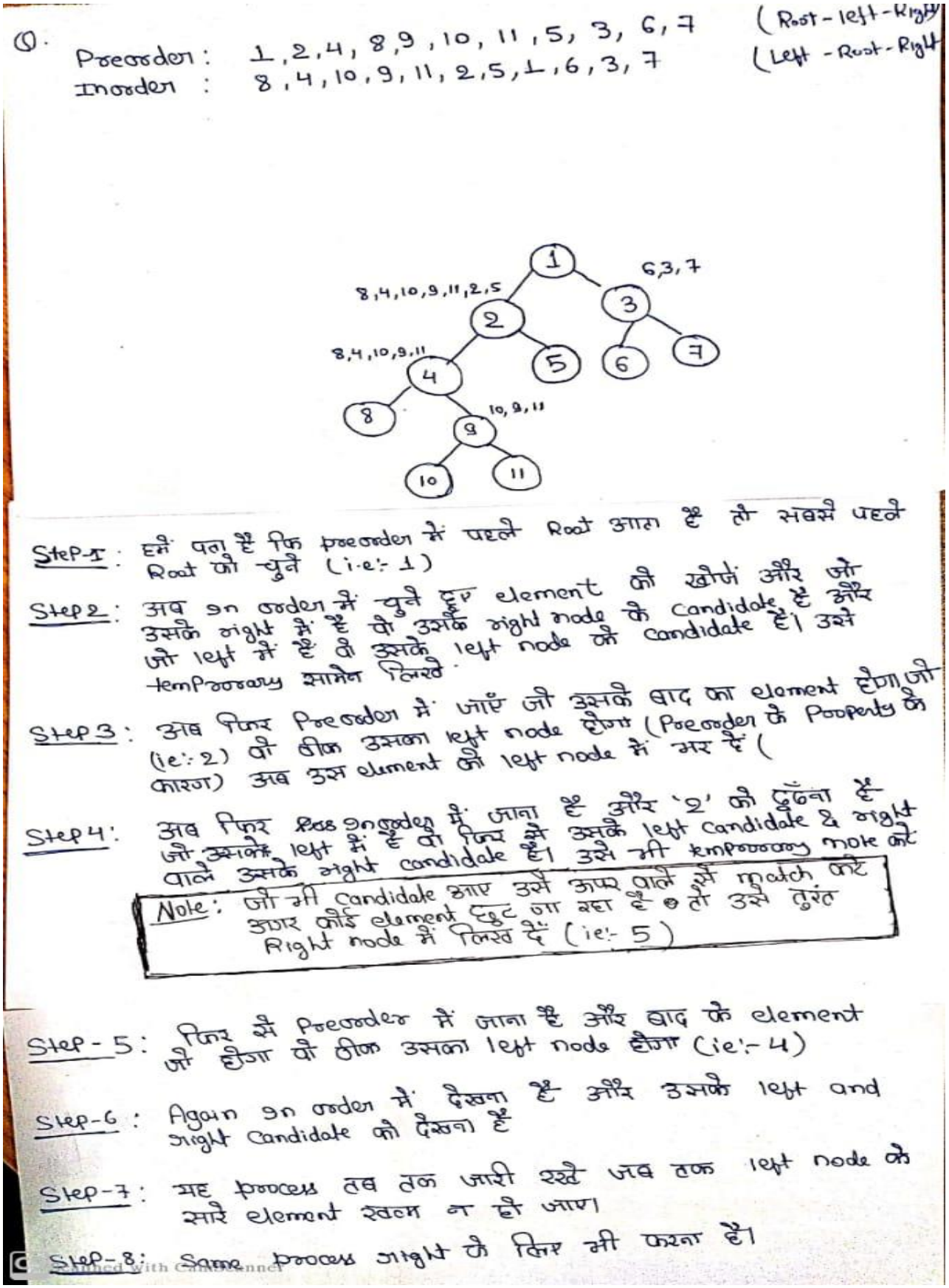
c. IN-ORDER Traversal:

```
void inorder(struct node *root){
    if(root==0){
        return;
        }
inorder(root->left);
printf("%d  ",root->data);
inorder(root->right);
}
```

Written by:
SAMUNDAR SINGH

1.2 Construction of binary tree from the traversal:-

1.) When PREORDER and INORDER traversal is given:

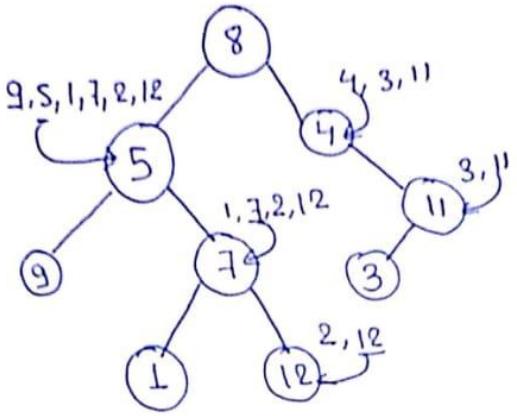


DATA STRUCTURE NOTES

2.) When POSTORDER and INORDER traversal is given:

Postorder: 9, 1, 2, 12, 7, 5, 3, 11, 4, 8 (left- Right - Root)

In order: 9, 5, 1, 7, 2, 12, 8, 4, 3, 11 (Left - Root - Right)



Step-1 :- Post order में सबसे पहले root element खोजना होगा जो हमें सबसे last में मिलेगा। (i.e:- 8)

Step-2 : Then In order में जाकर हम root element को ढूँढेंगे और उसके left & right Candidate को temp note करना है।

Step-3 : अब इन Candidate element से root (Sub-root) को खोजने के लिए फिर से Post-order में जाना है और Right से left तक देखते जाना है और इसी दौरान Candidate element से match भी करना है जो सबसे पहला match होगा वो ही next node होगा। उसको no • fill कर देंगे (ie: '5' for left node & '4' for right node).

Step-4 : Again यही process repeat करना है जब तक कि सारे node पूरी तरह भर नहीं जाए।

2.) When POSTORDER and PREORDER traversal is given:

pending

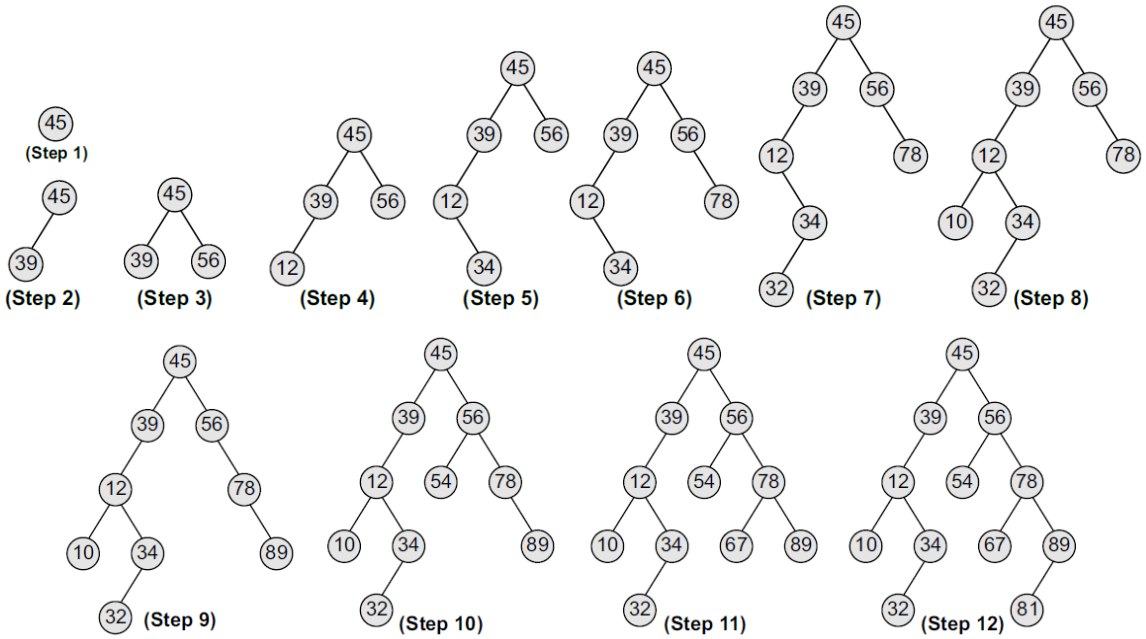
Written by:
SAMUNDAR SINGH

2.Binary Search Tree:

A binary search tree (BST) is a binary tree but the nodes are arranged in an order. In a binary search tree, all the nodes in the left sub-tree have a value less than that of the root node and all the nodes in the right sub-tree have a value either equal to or greater than the root node. In binary search tree we require very less amount of time to search an element since at every node, we get a hint regarding which sub-tree to search for desired result.

Example : Create a binary search tree using the following data elements:

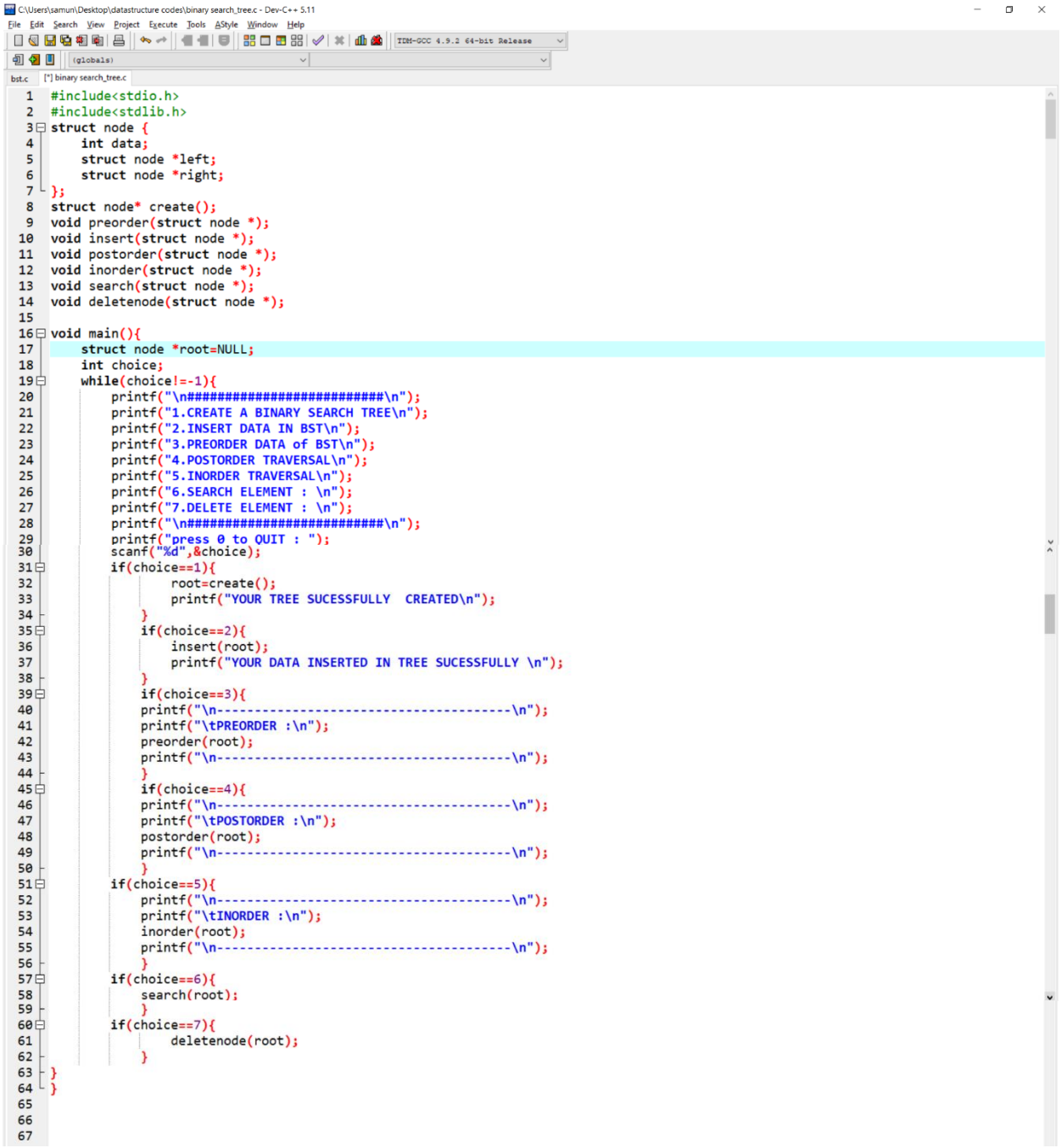
45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81



Written by:
SAMUNDAR SINGH

Code to show implementation of Binary Search Tree :-

Code for menu driven program :-

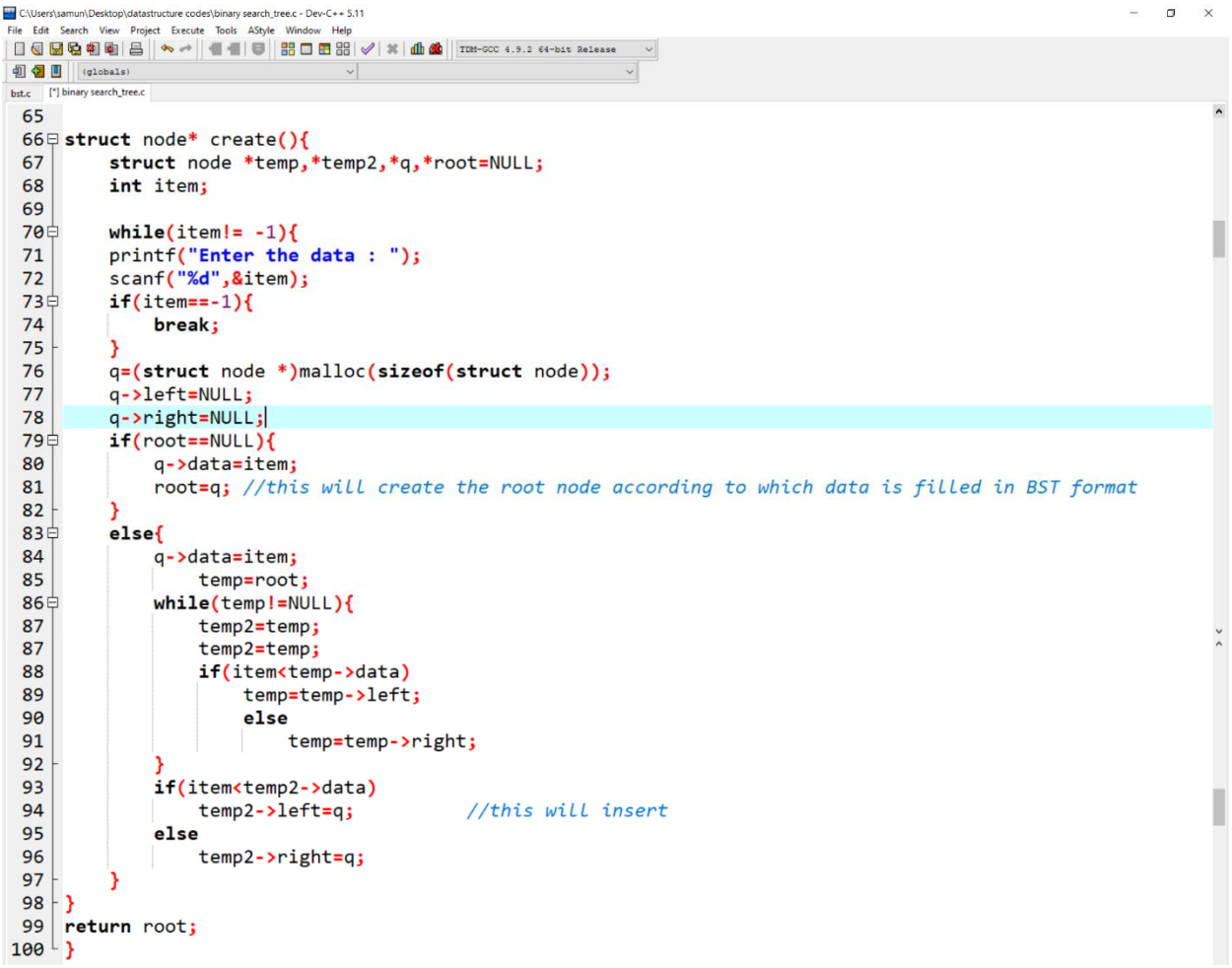


```c
#include<stdio.h>
#include<stdlib.h>
struct node {
    int data;
    struct node *left;
    struct node *right;
};
struct node* create();
void preorder(struct node *);
void insert(struct node *);
void postorder(struct node *);
void inorder(struct node *);
void search(struct node *);
void deletenode(struct node *);

void main(){
    struct node *root=NULL;
    int choice;
    while(choice!=-1){
        printf("\n############################\n");
        printf("1.CREATE A BINARY SEARCH TREE\n");
        printf("2.INSERT DATA IN BST\n");
        printf("3.PREORDER DATA of BST\n");
        printf("4.POSTORDER TRAVERSAL\n");
        printf("5.INORDER TRAVERSAL\n");
        printf("6.SEARCH ELEMENT : \n");
        printf("7.DELETE ELEMENT : \n");
        printf("\n############################\n");
        printf("press 0 to QUIT : ");
        scanf("%d",&choice);
        if(choice==1){
                root=create();
                printf("YOUR TREE SUCESSFULLY  CREATED\n");
            }
            if(choice==2){
                insert(root);
                printf("YOUR DATA INSERTED IN TREE SUCESSFULLY \n");
            }
            if(choice==3){
            printf("\n---------------------------------------\n");
            printf("\tPREORDER :\n");
            preorder(root);
            printf("\n---------------------------------------\n");
            }
            if(choice==4){
            printf("\n---------------------------------------\n");
            printf("\tPOSTORDER :\n");
            postorder(root);
            printf("\n---------------------------------------\n");
            }
        if(choice==5){
            printf("\n---------------------------------------\n");
            printf("\tINORDER :\n");
            inorder(root);
            printf("\n---------------------------------------\n");
            }
        if(choice==6){
            search(root);
            }
        if(choice==7){
                deletenode(root);
            }
}
}
```

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Code to construct a Binary search tree :-



```
65
66  struct node* create(){
67      struct node *temp,*temp2,*q,*root=NULL;
68      int item;
69
70      while(item!= -1){
71      printf("Enter the data : ");
72      scanf("%d",&item);
73      if(item==-1){
74          break;
75      }
76      q=(struct node *)malloc(sizeof(struct node));
77      q->left=NULL;
78      q->right=NULL;
79      if(root==NULL){
80          q->data=item;
81          root=q; //this will create the root node according to which data is filled in BST format
82      }
83      else{
84          q->data=item;
85              temp=root;
86          while(temp!=NULL){
87              temp2=temp;
88              temp2=temp;
89              if(item<temp->data)
90                  temp=temp->left;
91                  else
92                      temp=temp->right;
93          }
94          if(item<temp2->data)
95              temp2->left=q;          //this will insert
96          else
97              temp2->right=q;
98      }
99  }
100 return root;
    }
```

LOGIC:

Step1: First we take input from the user until user press -1 and store it in a variable item and a while loop executes until -1 encounters.

Step2: Every time the while loop executes it dynamically allocate memory to a **node (q)** and set its left and right part as NULL.

Step3: Now for data part of the node we have two condition either we are going to create a root node or its sub nodes if it is a root node then the address of the node is assigned to a pointer variable root which help to traversed the binary search tree. And if it is not the root node then we assign the data to the data part but now we have to place this node to appropriate place for this Go to step 4.

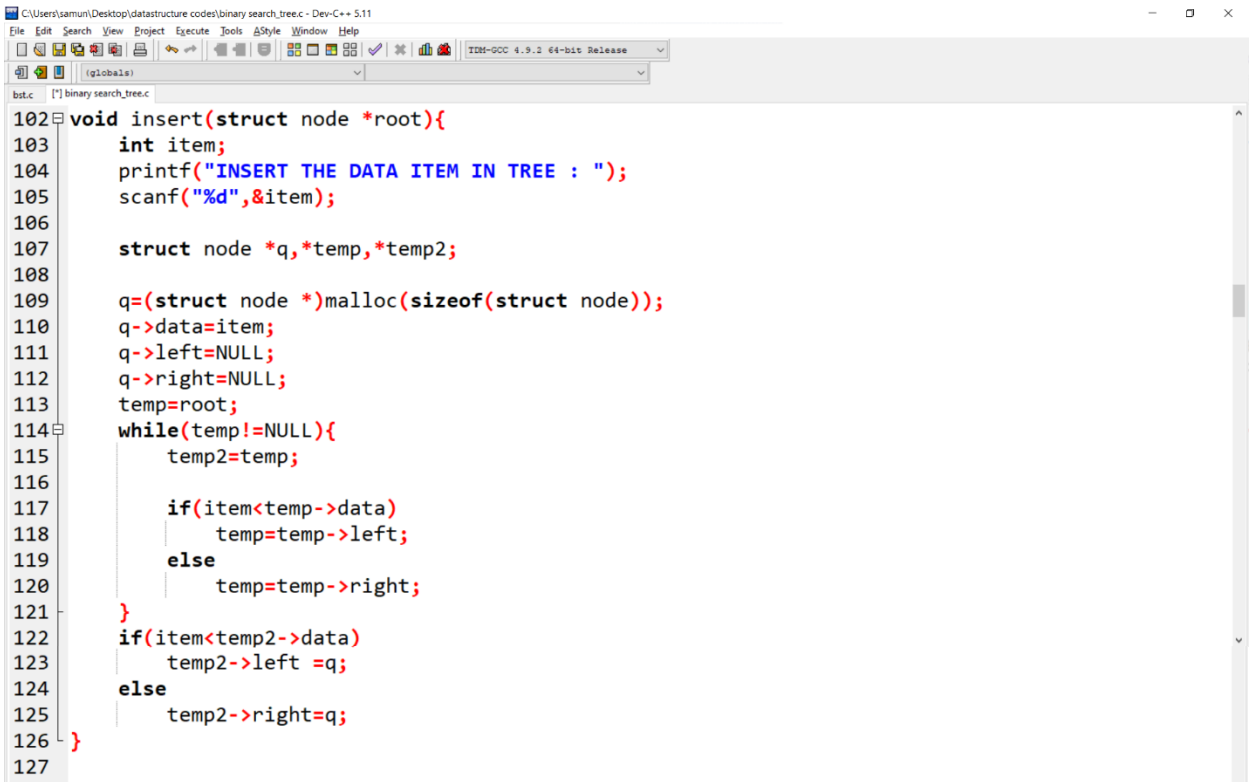
Step4: for placing the node to appropriate place we have to make a comparison according to which the data is placed either to the left side or to the right side. For this first we store the root node address to a pointer variable temp and again a while loop executed whose work is to make comparison between the user's input and the data stored in temp if the user's input data is smaller than the temp address then temp is updated to the temp's left part otherwise temp's right part until it found NULL.
(**simple word me while loop tab tak chalega jab tak ye last node tak na pahuch jaye jahan isko data insert karna hai**).

Step5: we create a another pointer variable say temp2 because the temp variable that we created earlier now hold dirty value but this temp2 has hold the address of that node from which we have to link our current node so according to the value less than or greater than this temp2 left or right address part is updated with address of newly created node .

Step6: when all the task performed the create block will return the value of the root node which we created at the beginning.

Written by:
SAMUNDAR SINGH

Code to Insert a data in Binary search tree :-



```c
void insert(struct node *root){
    int item;
    printf("INSERT THE DATA ITEM IN TREE : ");
    scanf("%d",&item);

    struct node *q,*temp,*temp2;

    q=(struct node *)malloc(sizeof(struct node));
    q->data=item;
    q->left=NULL;
    q->right=NULL;
    temp=root;
    while(temp!=NULL){
        temp2=temp;

        if(item<temp->data)
            temp=temp->left;
        else
            temp=temp->right;
    }
    if(item<temp2->data)
        temp2->left =q;
    else
        temp2->right=q;
}
```

LOGIC:

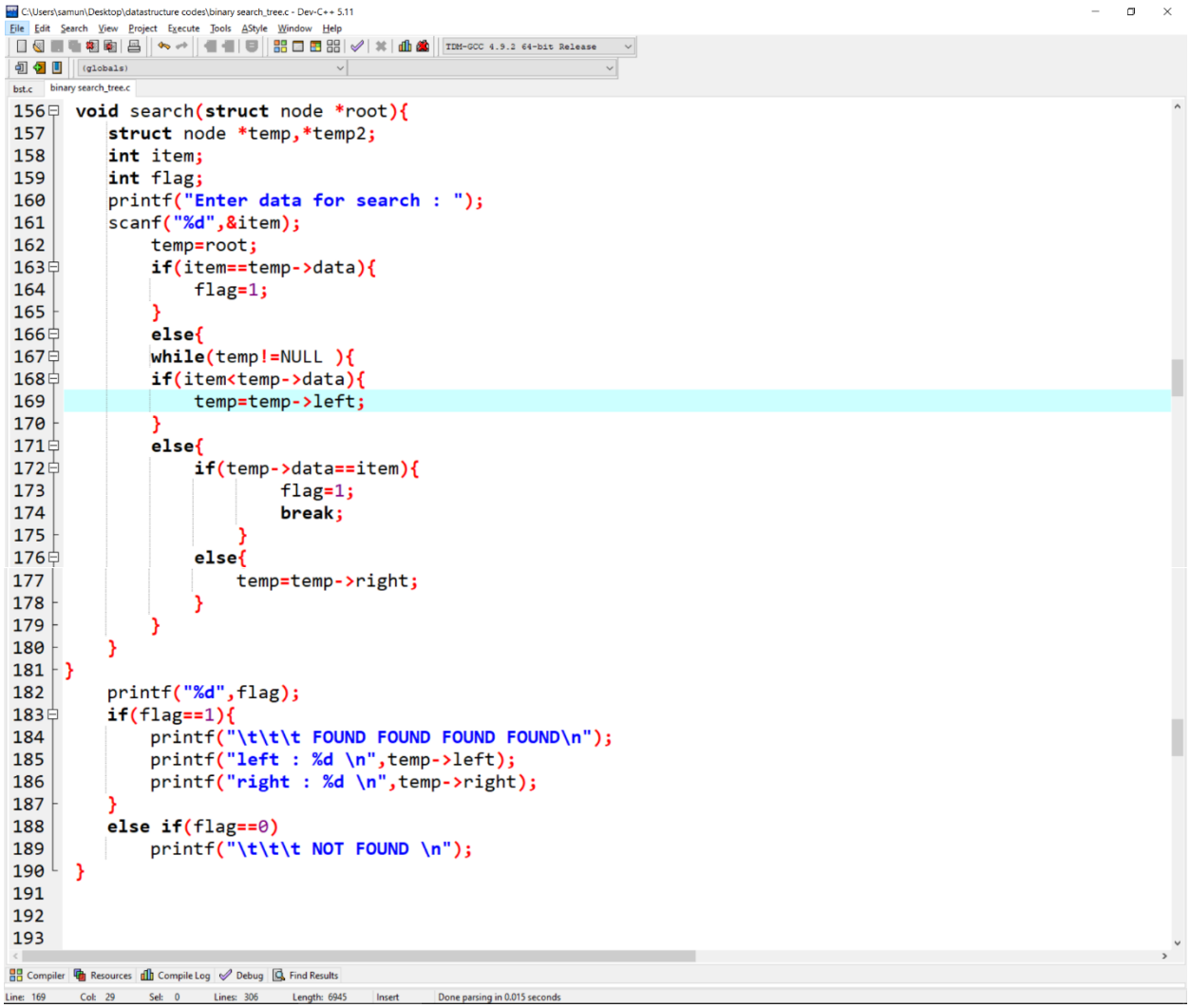
Step1: First we pass the root node address as a argument and call the function of insert

Step2: Then we traversed the BST and reached to the appropriate place according to the data value as mentioned in creation of BST and Now our created node address is updated in the traversed node either in left or right pointer variable.

Note: The insertion in BST is similar to the construction of BST

Written by:
SAMUNDAR SINGH

Code to Search a data in Binary search tree :-



```c
void search(struct node *root){
    struct node *temp,*temp2;
    int item;
    int flag;
    printf("Enter data for search : ");
    scanf("%d",&item);
        temp=root;
        if(item==temp->data){
            flag=1;
        }
        else{
        while(temp!=NULL ){
        if(item<temp->data){
            temp=temp->left;
        }
        else{
            if(temp->data==item){
                    flag=1;
                    break;
                }
            else{
                temp=temp->right;
            }
        }
    }
}
    printf("%d",flag);
    if(flag==1){
        printf("\t\t\t FOUND FOUND FOUND FOUND\n");
        printf("left : %d \n",temp->left);
        printf("right : %d \n",temp->right);
    }
    else if(flag==0)
        printf("\t\t\t NOT FOUND \n");
}
```

LOGIC:

Step1: First we pass the root node address as an argument and call the function of search.

Step2: we take input from the user and store to variable item then in an address variable temp we store the address of root .(for traversing)

Step3: Whatever we are going to compare for traversing is either greater or smaller so we have a condition if the user input the node data For that we have a separate if condition which check the root value directly.

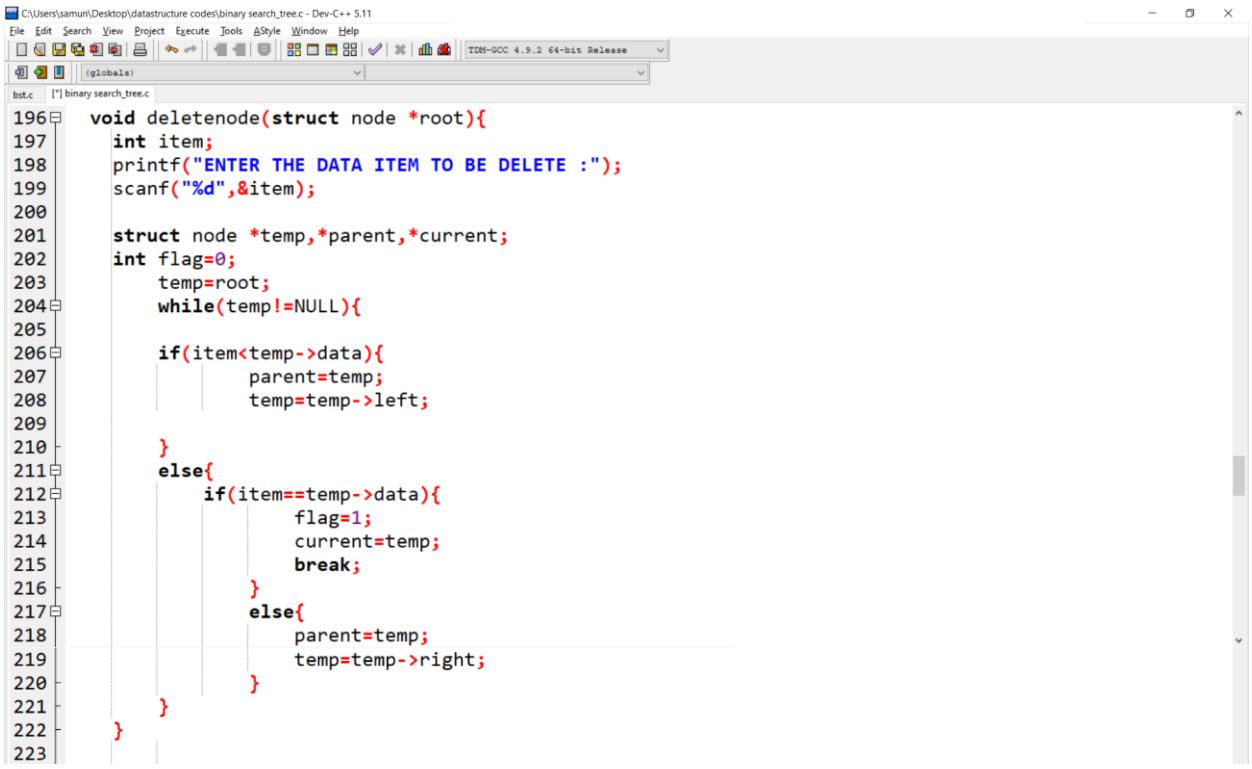
Step4: if the user's input data is not the root then we perform traversing in which if the input data is less than the current node data then the Flow shift to the left each time by updating the temp value to its temp->left value and if the value is greater then it shift the flow to the right

Step5: if the user's input match with the BST node data then simply it put flag==1 otherwise it is zero i.e flag==0.

Step6: if flag==1 than data is in the BST otherwise it is not in the BST.

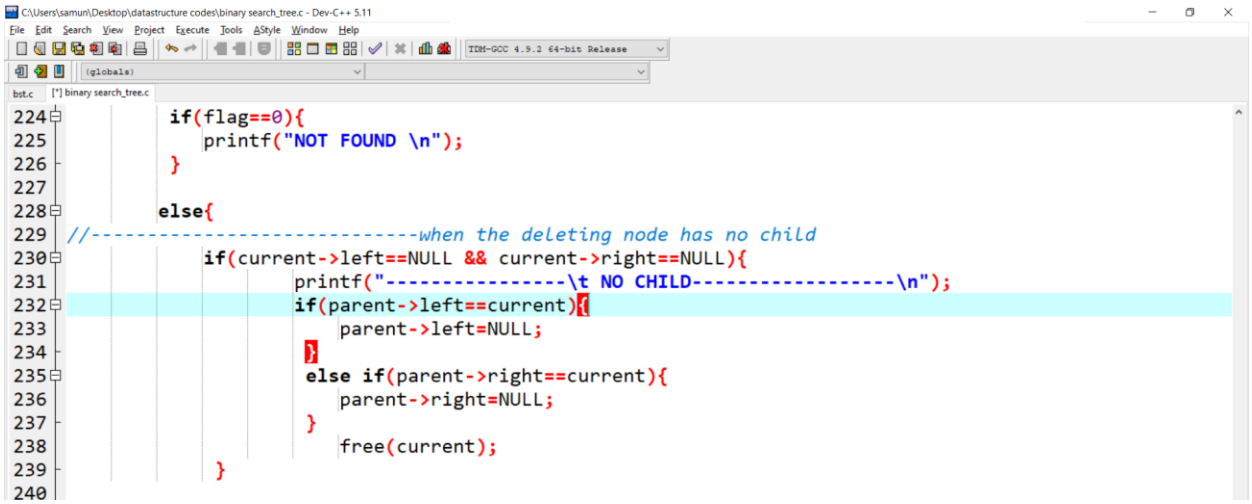
Written by:
SAMUNDAR SINGH

Code to Delete data from Binary search tree: -



```
196  void deletenode(struct node *root){
197    int item;
198    printf("ENTER THE DATA ITEM TO BE DELETE :");
199    scanf("%d",&item);
200
201    struct node *temp,*parent,*current;
202    int flag=0;
203        temp=root;
204        while(temp!=NULL){
205
206        if(item<temp->data){
207                parent=temp;
208                temp=temp->left;
209
210        }
211        else{
212            if(item==temp->data){
213                    flag=1;
214                    current=temp;
215                    break;
216                }
217                else{
218                    parent=temp;
219                    temp=temp->right;
220                }
221        }
222    }
223
```

Delete data from Binary search tree when deleted node has no child .



```
224          if(flag==0){
225             printf("NOT FOUND \n");
226          }
227
228          else{
229  //-----------------------------when the deleting node has no child
230              if(current->left==NULL && current->right==NULL){
231                      printf("----------------\t NO CHILD-----------------\n");
232                      if(parent->left==current){
233                          parent->left=NULL;
234                      }
235                      else if(parent->right==current){
236                         parent->right=NULL;
237                      }
238                          free(current);
239               }
240
```

LOGIC:

Step1: If flag ==0 means that the user's input does not match with any data item in BST.

Step2: If data get match then the node's address in which matched data is present is stored in a address variable named as **CURRENT** and and at the same time and in same code we want the parent node for this the code is written in traversing module to fetch **PARENT**

.

Step3: Now we check the Current node left or right pointer variable is Null or not If it is null them simply the parent variable from which the Current node is attached whether in left or right we have to make it NULL first and then we remove the current node using FREE function

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Delete data from Binary search tree when deleted node has one child .

```
241 //---------------------------- when the deleting node has one  child
242             else if(current->left!=NULL &&current->right == NULL){
243                     printf("----------------\t ONE CHILD-----------------\n");
244                 if(parent->right==current){
245                         parent->right=current->left;
246                     }
247                     else if(parent->left==current){
248                         parent->left=current->left;
249                     }
250                     free(current);
251             }
252
253                 else if(current->right!=NULL &&current->left == NULL){
254                     printf("----------------\t one CHILD-----------------\n");
255                     if(parent->right==current){
256                         parent->right=current->right;
257                     }
258                     else if(parent->left==current){
259                         parent->left=current->right;
260                     }
261                     free(current);
262             }
263 //-------------------------------------------------------------------------------------------------------
```

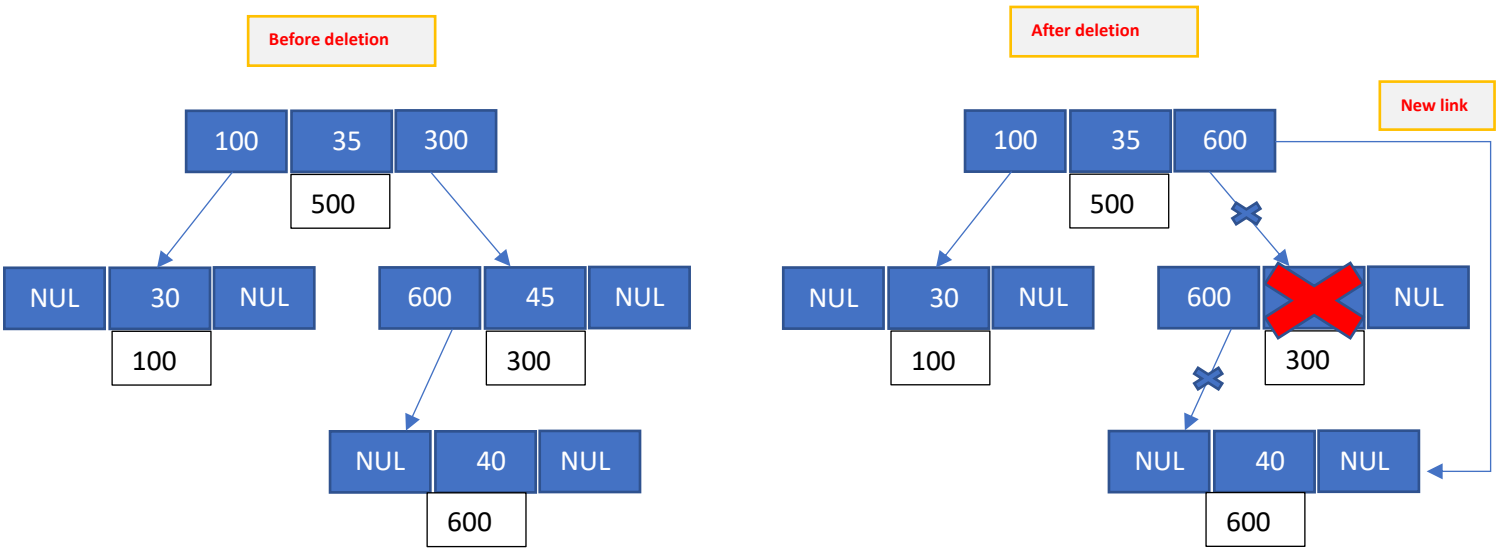
LOGIC:

Step1: If data get match then the node's address in which matched data is present is stored in a address variable named as **CURRENT** and and at the same time and in same code we want the parent node for this the code is written in traversing module to fetch **PARENT**

Step2: Now we check whether current's node left part or the right part hold the data after that we check that in parent node which side hold The address of the current node. Once we get this info then follow step 3.

Step3: Since we are deleting a node which has one child so when we delete then the child node of current node lost with it which is not ideal therefore, to avoid this loss of data we have to make link with the parent node.

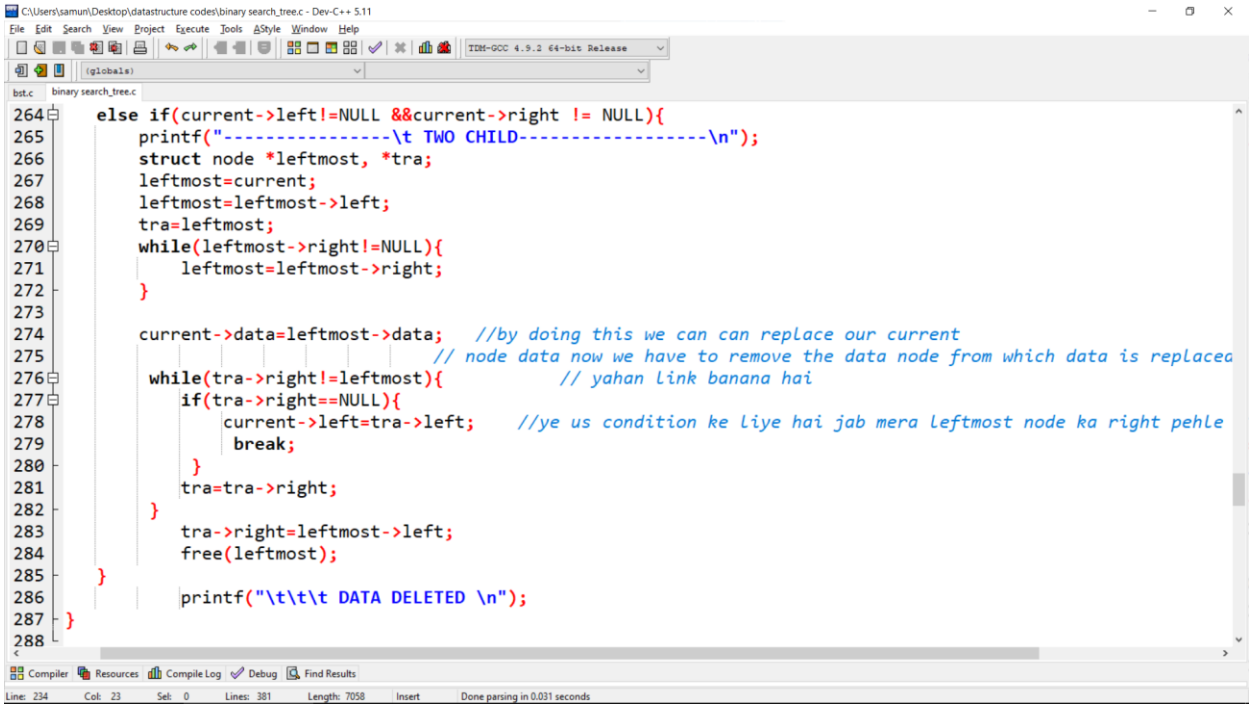
Step4: To establish a link first we have to check at which node part(either left or right) of the parent node hold the address of the current node then simply replace that node part with the current's node part which hold the child node.



Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Delete data from Binary search tree when deleted node has one two child .



```
264    else if(current->left!=NULL &&current->right != NULL){
265        printf("----------------\t TWO CHILD-----------------\n");
266        struct node *leftmost, *tra;
267        leftmost=current;
268        leftmost=leftmost->left;
269        tra=leftmost;
270        while(leftmost->right!=NULL){
271            leftmost=leftmost->right;
272        }
273
274        current->data=leftmost->data;   //by doing this we can can replace our current
275                                        // node data now we have to remove the data node from which data is replacea
276         while(tra->right!=leftmost){           // yahan link banana hai
277            if(tra->right==NULL){
278                current->left=tra->left;    //ye us condition ke liye hai jab mera leftmost node ka right pehle
279                 break;
280             }
281            tra=tra->right;
282         }
283            tra->right=leftmost->left;
284            free(leftmost);
285    }
286            printf("\t\t\t DATA DELETED \n");
287 }
288
```

LOGIC:

Step1: If data get match then the node's address in which matched data is present is stored in a address variable named as **CURRENT** and then we check whether the left or the right part of the node is NULL or not if it is NULL if it is not NULL then only this block executes

Step2: Here we struct an address variable named **leftmost** in which current node is stored then I am targeting the left potion's greatest child to replace the current node's data. And then I am going to delete that node from which data is copied.

Step3: To achieve this first we need to make the flow in the left direction for this purpose we put the current's address in the leftmost node And then a loop will execute until it encounter a NULL the task of this loop to find the greatest element and we know that the greatest element of node is always to its right part.

Step4: when it gets that node then simply, we copy the the data of that node and placed it to the current node value.

Step5: After placing the value we need to delete that node but there is a situation that the greatest node right has NULL but it's left part is not NULL, it means that the greatest node also has some data or sub tree .so if we delete the node then it data linked with it get lost.

Step6: To avoid loss of data we have to create link between the node that hold the address of the greatest node and the node that is child Nodes of the leftmost node

Step7: After proper link established then we free the leftmost node.

NOTE: Either we choose the largest number of the left sub tree which is nothing but the right most node . Which contain the NULL value as we have done above

OR

We can choose the smallest number of the right sub tre which is nothing but first left side node of the current's right side node

Written by:
SAMUNDAR SINGH

3.AVL (Adelson-Velskii and Landis) Search Tree:

An AVL tree is a self-balancing Binary Search Tree. An AVL tree has a special property called Balance Factor which is associated with every node. The balance factor of a node is calculated by subtracting the height of its right sub-tree from the height of its left sub-tree.

Balance factor = Height (left sub-tree) – Height (right sub-tree)

A binary search tree in which every node has a **balance factor of –1, 0, or 1** is said to be height balanced.

NOTE :- In binary search tree for a given set of data there are multiple BST can be formed depending upon the data arrived for insertion.so the problem is that the time to search a data element in tree depend upon the height of the BST that is max: n and min: log(n). so we can't control the height in BST but in AVL Tree we can control the height of the tree. And the searching performed in *O* (log n) which is the Average Case.

Insertion of data in an AVL tree: -

The new node inserted in the leaf node it's balance factor is equal to Zero only those nodes which lies in path of the root node and the leaf node balance factor get affected if it is not {-1,0,1} then we have to perform the rotation to restore the balance of the tree

There are four types of rotation are performed: -

1.) L-L Rotation:-

Suppose we have to insert the data 10 ,9 ,8 in a BST then the scenario is like this:

Trick : Go to that part where balance factor is not {-1,0,1} and assume that u have a compass and perform 180 rotation to get the new position. For L-L case perform right rotation (just opposite to it).

2.) R-R Rotation:

Suppose we have to insert the data 8,9,10 in a BST then the scenario is like this:

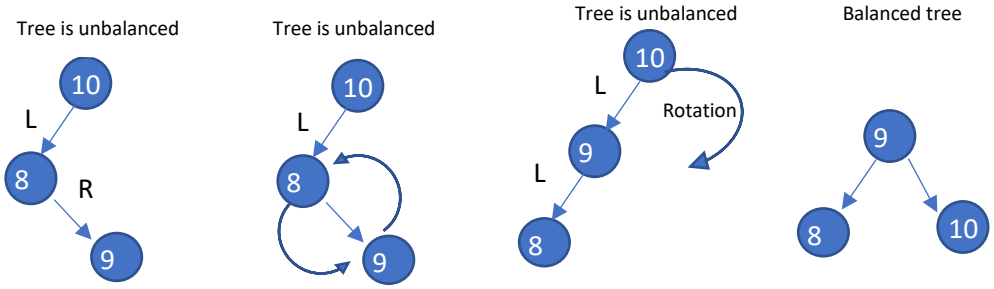
Trick : Go to that part where balance factor is not {-1,0,1} and assume that u have a compass and perform 180 rotation to get the new position. For R-R case perform Left rotation (just opposite to it).

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

3.) L-R Rotation:

Suppose we have to insert the data 10,8,9 in a BST then the scenario is like this:

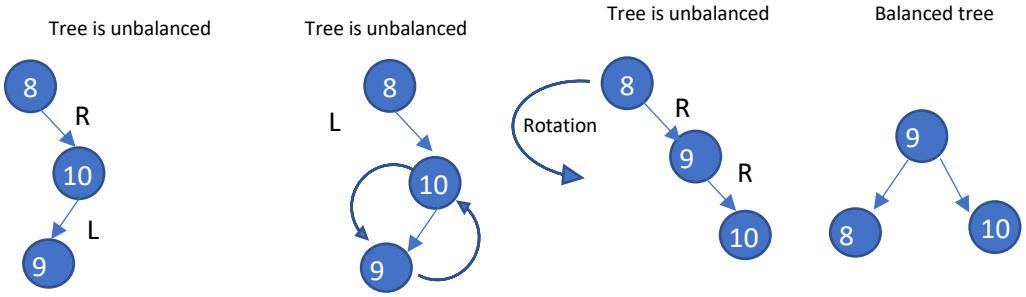


Step:1 first we have to transform the unbalanced tree into a either L-L or R-R unbalance tree by swapping numbers.
Step:1 Then we apply the L-L or R-R rotation technique.

Trick : if L-R case occur then we have to convert it into L-L unbalance tree first to learn this use this technique see the first letter of → **L-R** then we convert it into → **L-L**

4.) R-L Rotation:

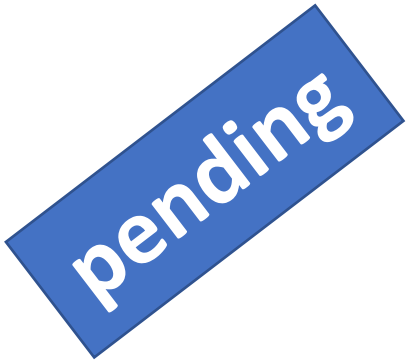
Suppose we have to insert the data 8,10,9 in a BST then the scenario is like this:



Step:1 first we have to transform the unbalanced tree into a either L-L or R-R unbalance tree by swapping numbers.
Step:1 Then we apply the L-L or R-R rotation technique.

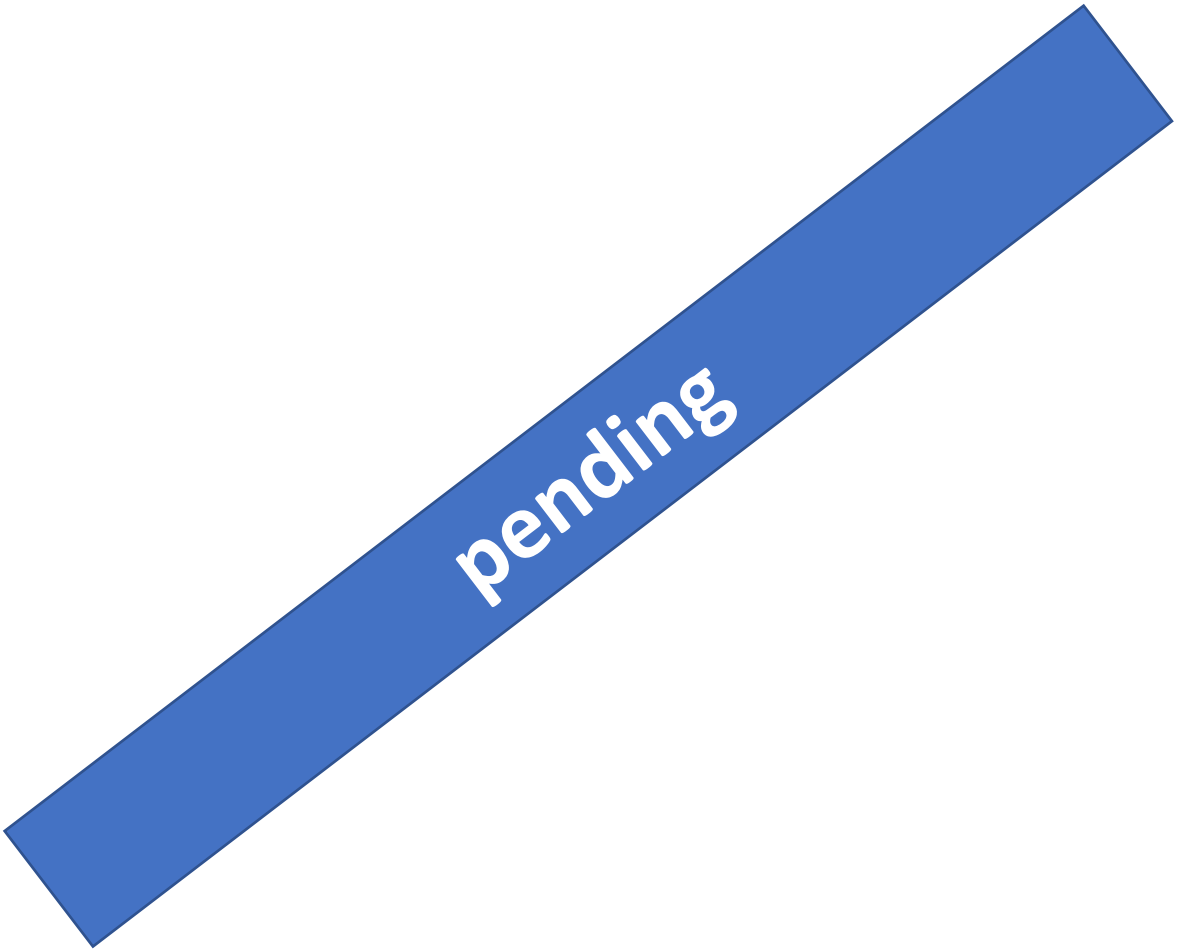
Trick : if R-L case occur then we have to convert it into R-R unbalance tree first to learn this use this technique see the first letter of → **R-L** then we convert it into → **R-R**

Deletion of data in an AVL tree: -



Written by:
SAMUNDAR SINGH

Code to show implementation of Binary Search Tree



Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

4.RED – BLACK TREE:

A Red – Black tree is a self-balancing Binary Search Tree, where insertion, deletion and searching all have the time complexity Average case that is *O* (log n) which means rather of searching all node we can search only one node from every level. In AVL tree the time complexity of the insertion and deletion had taken more time since it require multiple rotation to balance. only the searching takes time *O* (log n). That's we use red - black tree.

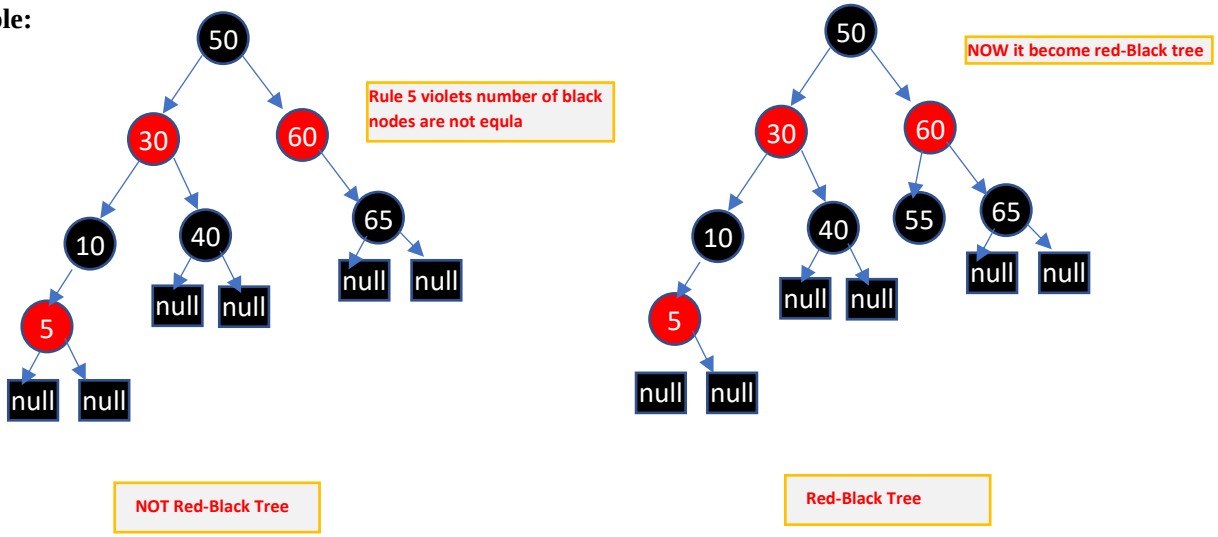
NOTE: - In Red -Black tree the insertion, deletion as well as searching all performed in time complexity having average case *O* (log n).

Properties:

1. The colour of a node is either red or black.
2. The colour of the root node is always black.
3. All leaf nodes are black.
4. Every red node has both the children coloured in black.
5. Every simple path from a given node to any of its leaf nodes has an equal number of black nodes.

NOTE: - The longest path from the root node to any leaf node is **no more than twice** as long as the shortest path from the root to any other leaf in that tree.

Example:



Insertion of data in a Red-Black tree: -

Rules:

1. If tree is empty then create a new node as root node with colour Black.
2. If tree is not empty create a new ode as leaf node with colour Red.
3. If parent of new node is black then exit.
4. If parent of new node is red then check the colour of parent's sibling of new node
   Case -I: if the colour is black or NULL then do suitable Rotation and Recolour.
   Case -II: if the colour is red then recolour and also check if parent's parent of new node is not root node then recolour it and recheck.

NOTE: It must follow the BST property and for rotation it follows the AVL tree Rotation**.**

Example: to show the insertion implementation of Red-Black tree

Written by:
SAMUNDAR SINGH

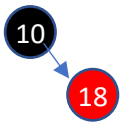
DATA STRUCTURE NOTES

Q. insert these data in the Red-Black tree: **10 , 18 , 7, 15, 16, 30, 25, 40, 60, 2**

Step: 1

Rule 1 follow

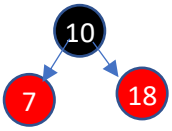
Step: 2



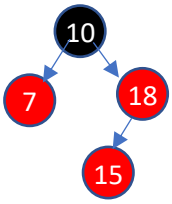
Rule 2 & 3 follow

Step: 3

Rule 2 & 3 follow



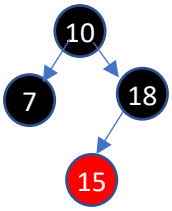
Step: 4



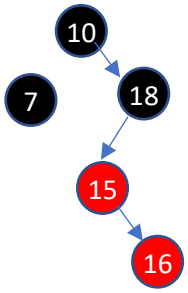
When we insert the data then, It violets the rules and have red -red combination in such situation we have to take action to remove this conflict

Step: 5

Rule 4 case: II follow



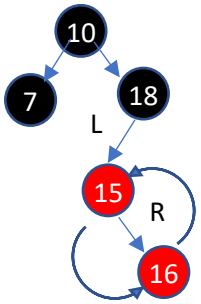
Step: 6



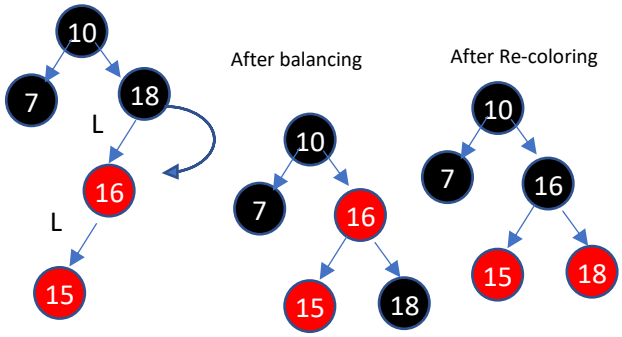
When we insert the data then, It violets the rules and have red -red combination in such situation we have to take action to remove this conflict

Step: 7

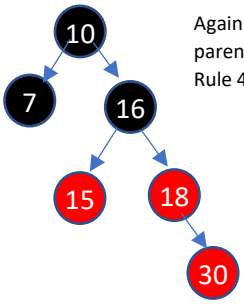
Rule 4 case: I follow



Here we perform L-R rotation in which first we convert it into L-L and then perform a right rotation due to which the median become root node after that we re-color.

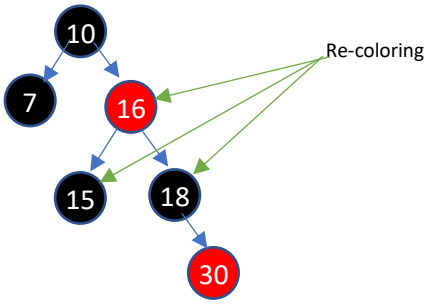


Step: 8



Again Red-Red node so parent's sibling is red so Rule 4 case: II follow

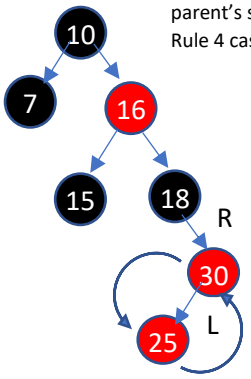
After Re-coloring



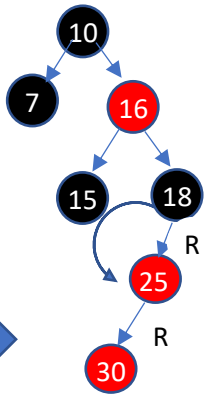
Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

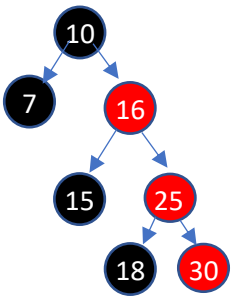
Step: 8



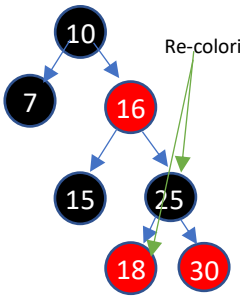
Again Red-Red node so parent's sibling is black so Rule 4 case: II follow



After balancing

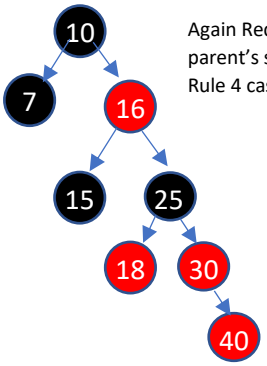


After Re-coloring



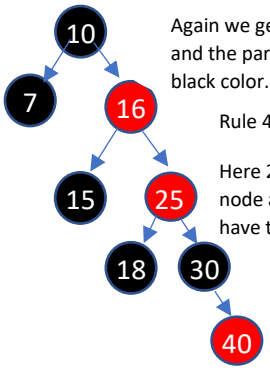
Re-coloring

Step:9



Again Red-Red node so parent's sibling is red so Rule 4 case: II follow

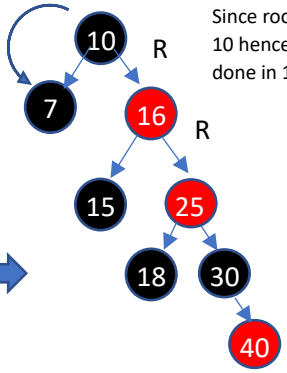
After Re-coloring



Again we get red -red conflict and the parent's sibling is of black color.

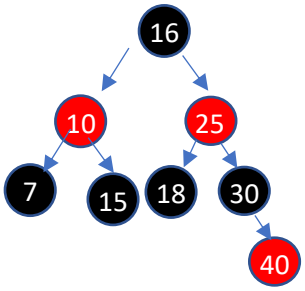
Rule 4 case: I follow

Here 25 become new node and due to 7 we have to perform

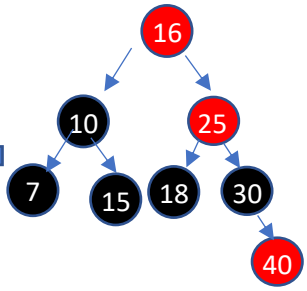


Since root node is 10 hence rotation done in 10

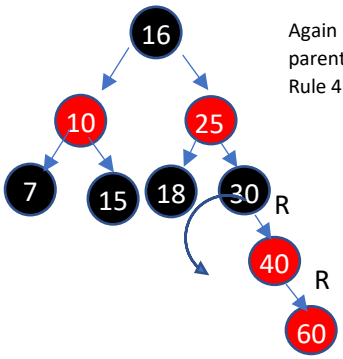
After Re-coloring



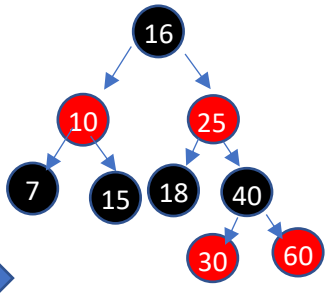
After balancing



Step:10



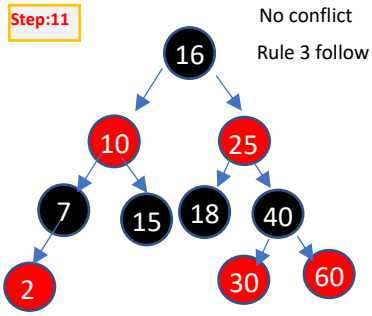
Again Red-Red node so parent's sibling is NULL so Rule 4 case: I follow



Step:11

No conflict

Rule 3 follow



Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

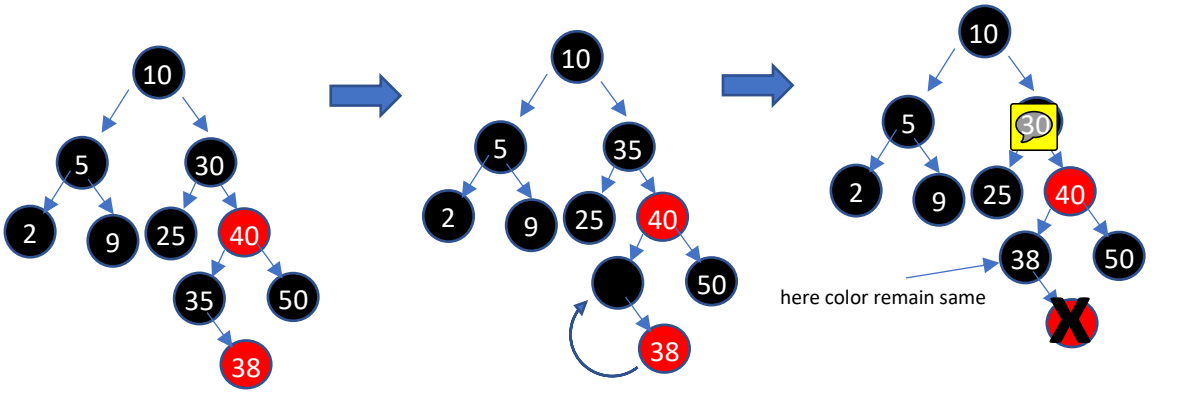
Deletion of data in a Red-Black tree: -

There are several cases to delete data from red -black tree we see each case one by one.

Rules:

1. Perform BST deletion (here take care that we do not delete any node we just replace the data from the node and delete the node from where the data has been taken.)
2. **Case: 1** If node to be deleted is red (🔴) just delete it .
   Note: while deleting process when you replace any data then do not replace the colour.

Example: **delete the node 30**



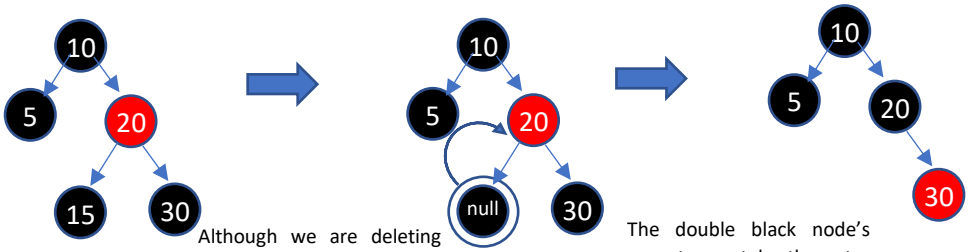
Here we have to delete 30 and according to BST deletion 35 is replaced

At last check whether all the law of red and black tree follow or Not

Case: 2 If root is double black just remove the double black.
Case: 3 If node to be deleted is Black (⚫) or the last one then just remove it , by doing so the null node take place at deleted node which create Double Black (DB) node.

- **a.)** if double black sibling is black and both children is black then remove the DB.
- **b.)** Add black to its Parent(P).
  - If the parent is red it become black.
  - If parent is black it become double black.
- **c.)** Make the sibling of double black node red.
- **d.)** If still DB Exist, apply other Cases.

Example: **delete the node 15(case when parent is red and siblings child are black)**



Although we are deleting the last element but due to red black tree property it has NULL Children which is black in color.

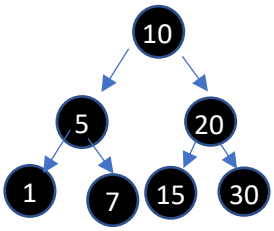
The double black node's parent now take the extra black color and the sibling which is of black color will became red

NOTE: Remember this thing via a story that whenever child has a double black problem he tell his problem to the parent so parent take the problem and now become double black and the sibling became angry so his face became red and the node also became red.

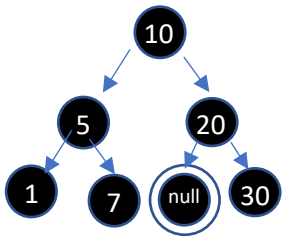
Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

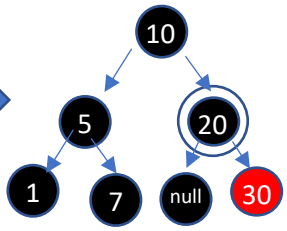
Example: **delete the node 15(case when parent is Black and sibling child are black):**



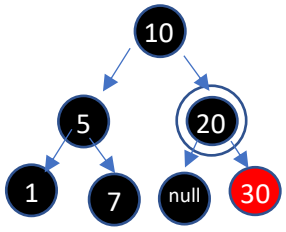
Node 15 want to deleted



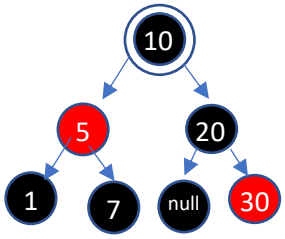
The null node takes places which create double black



Now the parent node became the double node



Still DB exists Now the root node 20 and its sibling is of black so again it go to the parent

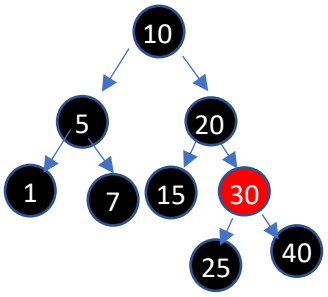


Still DB exists but it is in the root node so due to case: 2 simply remove the double black.

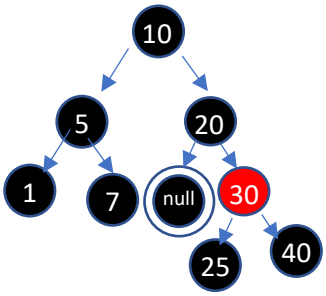
Case: 4 If double black node's sibling is red then

- **a.)** Swap colour of double black node's parent and the sibling.
- **b.)** Rotate the parent of double black's node to the double black node direction.
- **c.)** Again, take clues from case 2 and 3 and apply the necessary steps.

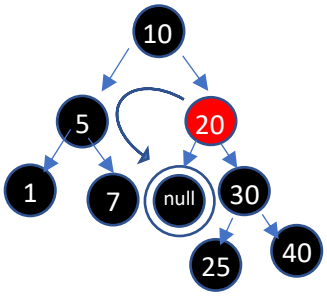
Example: **delete the node 15(if the sibling of double black node is red):**



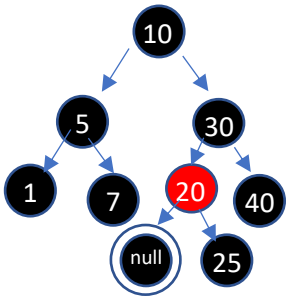
Node 15 want to deleted



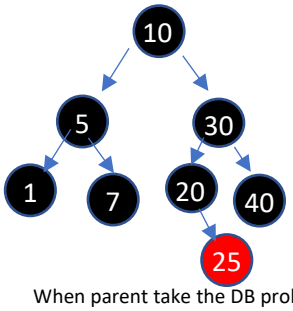
The null node takes places which create double black



Here we swap the color of the parent and the sibling and rotate the parent in the DB direction



After rotating the parent node in DB's direction it comes to situation when DB node's sibling is black. then we same rule as mentioned in above.



When parent take the DB problem of one of his children then the other child become red

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Case: 5 If double black node's sibling is black (and sibling's far child from DB is black and near child from the DB is red) then

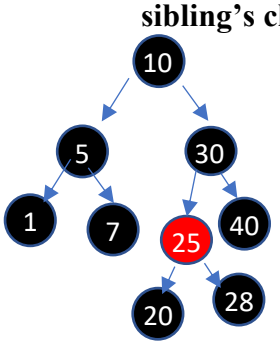
a.) Swap colour of **Db's sibling and sibling's child** who is near to the DB.
b.) Rotate sibling in opposite direction to DB.
c.) Apply case 6.

Case: 6 DB sibling is black and far child is red.

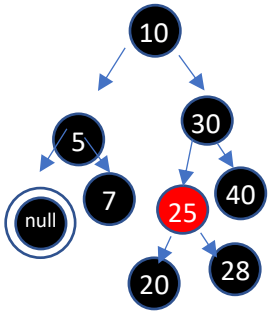
a.) Swap colour **of parent and sibling.**
b.) Rotate parent in DB's direction.
c.) Remove DB.
d.) Change colour of red child to black.

Note: only when the nearest child of the Double black node is red then it ask to swap the color of sibling with that (nearest) red node otherwise it ask to swap the color of the parent with the sibling not with the (farthest)red node.

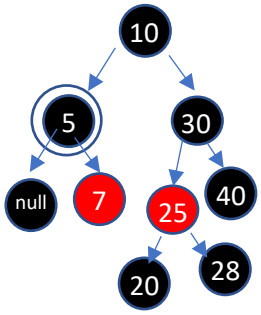
Example: **delete the node 15(if the sibling of double black node is black and any of the sibling's child is red):**



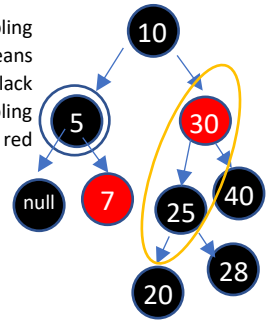
Node 1 want to deleted



Double Black problem arises

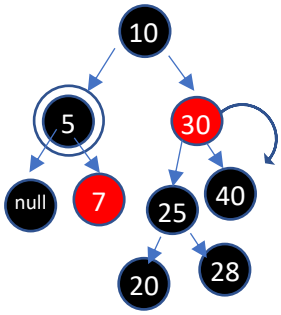


Child node give its problem of DB to his parent

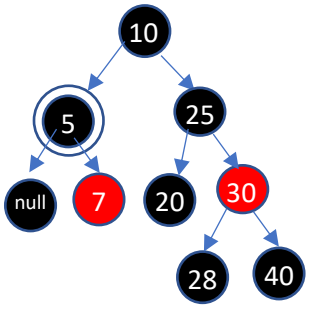


Swapping of color between the DB sibling and the child who have red color

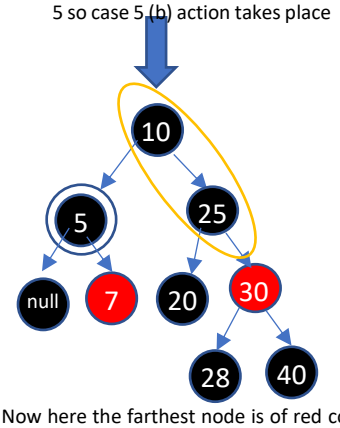
In such case the DB ask his sibling to cooldown his child means color change from red to black during this process the sibling became angry and acquire red color.



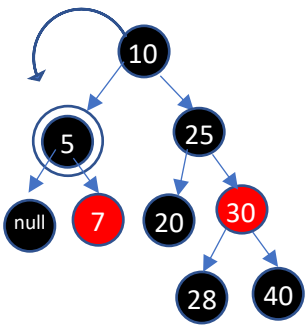
Now we have to perform rotation which depend on the distance of node having red color and the DB node before swapping of the colors.



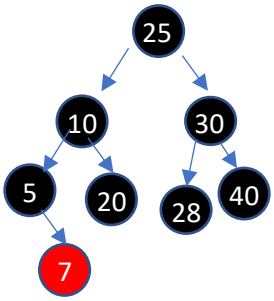
since among the nodes 25 and 40 the node 25 is nearest than the 40 to the Double black node 5 so case 5 (b) action takes place



Now here the farthest node is of red color i.e 30 so he ask from his parent to swap color between sibling and parent.



Now rotation performed Case 6 (b)

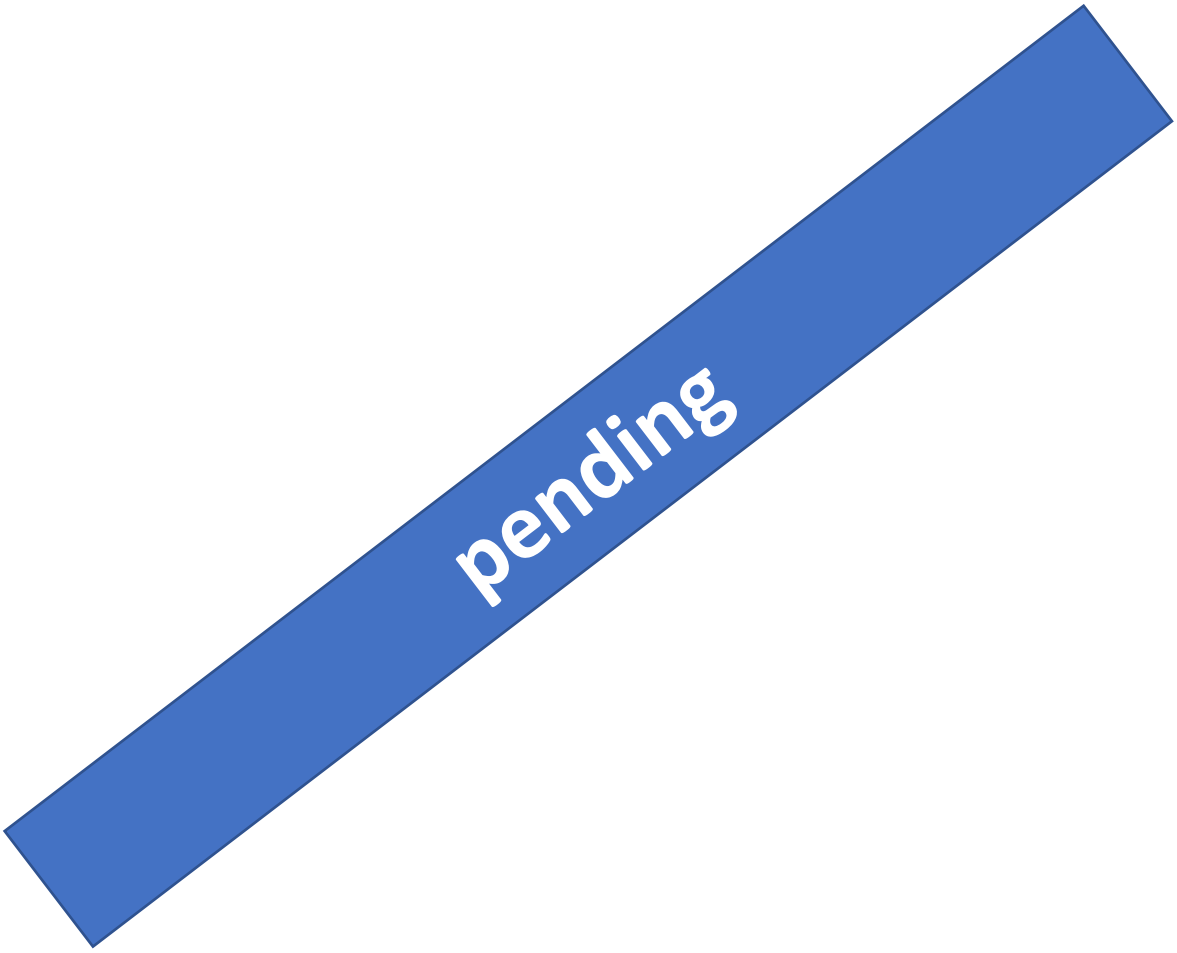


The double black is removed and red child color changed to black

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Code to show implementation of Red-Black Tree



Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Multi-Way Search tree

1.B TREES:

A B tree is a multi-way search tree which allow to store sorted data. It store more than one value to its node (unlike BST which store only one value at a node).

A B tree of order `m` can have a maximum of `m-1` keys and `m` pointers to its sub-trees.

Properties:

1. A node can have more than one key and more than 2 children.
2. It maintains sorted data.
3. All leaf node must beat same level.
4. B tree of order **m** have:

Maximum children:	**m.**
Minimum children:	at leaf '0' and at root '2'.
Total number of internal nodes:	celling function of **(m/2)** .
Every node has max no of keys:	(**m**-1) keys.
Every node has min no of keys:	at root '1' and at all other node celling function of {**(m/2)-1**}

Insertion of data in B tree: -

Rules:

1.) Data is always start inserting from the leaf node to the top.
2.) If leaf node is not full then insert the data into the leaf in an ordered manner.
3.) If leaf node is full then:
   a.) insert the new value in order into the existing set of keys.
   b.) split the node at its median into two nodes (note that the split nodes are half full).
   c.) push the median element up to its parent's node. If the parent's node is already full, then split the parent node by following the same steps

Deletion of data in B tree: -

Rules:

1.) Data can be deleted from anywhere(either leaf or parent), first find that leaf node that has to be deleted.
2.) If leaf contain more than the minimum value then simply delete it.
3.) But if leaf has not contain the minimum value then we have to borrow it :
   (a) If the left sibling has more than the minimum number of key values, push its largest key into its parent's node and pull down the intervening element from the parent node to the leaf node where the key is deleted.

   (b) Else, if the right sibling has more than the minimum number of key values, push its smallest key into its parent node and pull down the intervening element from the parent node to the leaf node where the key is deleted.

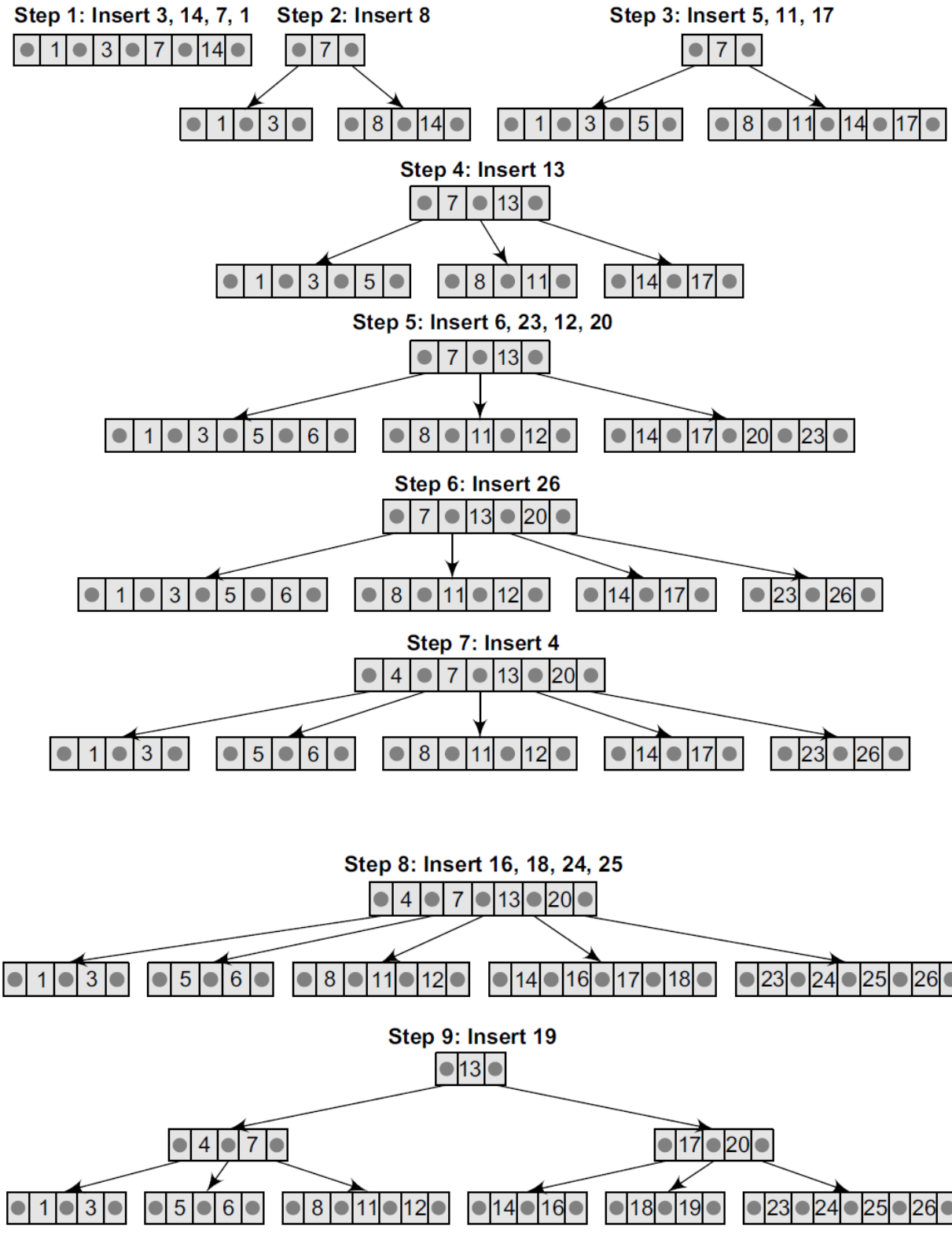
4.) If there is such a condition that while deleting the node left and right sibling has only the minimum number of element then we can't be able push by combining the two leaf nodes and the intervening element of the parent node If pulling the intervening element from the parent node leaves it with less than the minimum number of keys in the node, then propagate the process upwards, thereby reducing the height of the B tree

5.) To delete an internal node, promote the successor or predecessor of the key to be deleted to occupy the position of the deleted key. This predecessor or successor will always be in the leaf node. So the processing will be done as if a value from the leaf node has been deleted.

Written by:
SAMUNDAR SINGH

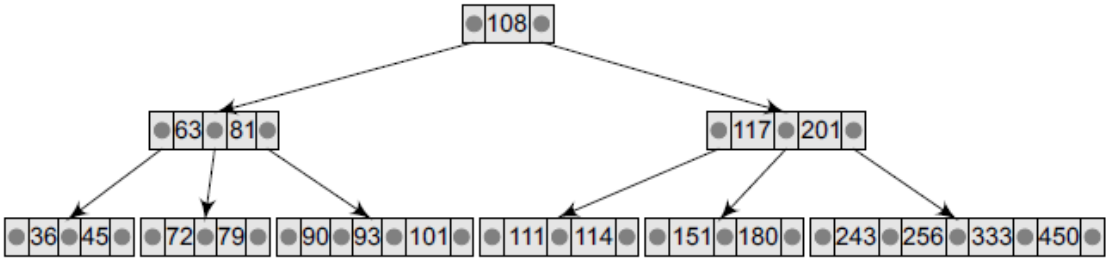
DATA STRUCTURE NOTES

Example: Create a B tree of order 5 by inserting the following elements:
3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, and 19.

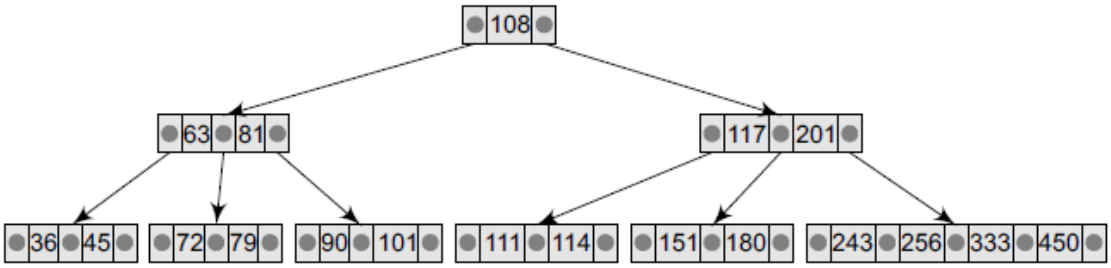


Written by:
SAMUNDAR SINGH

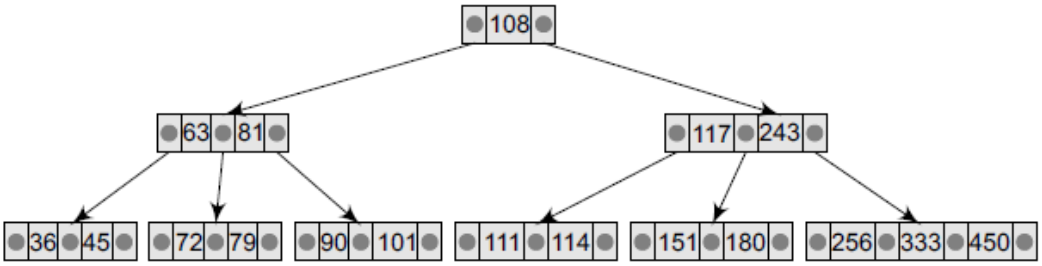
Example: Consider the following B tree of order 5 and delete values 93, 201, 180, and 72 from it



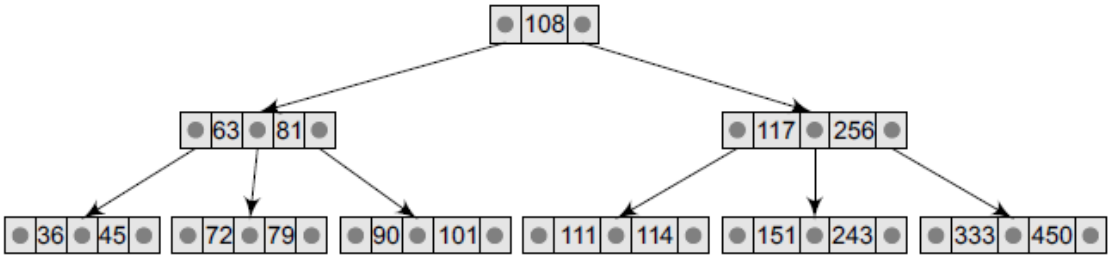
Step 1: Delete 93



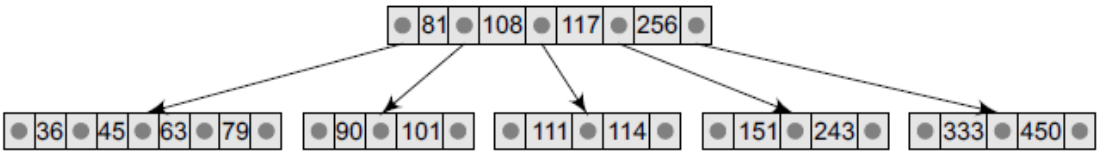
Step 2: Delete 201



Step 3: Delete 180



Step 4: Delete 72



2.B+ TREES:

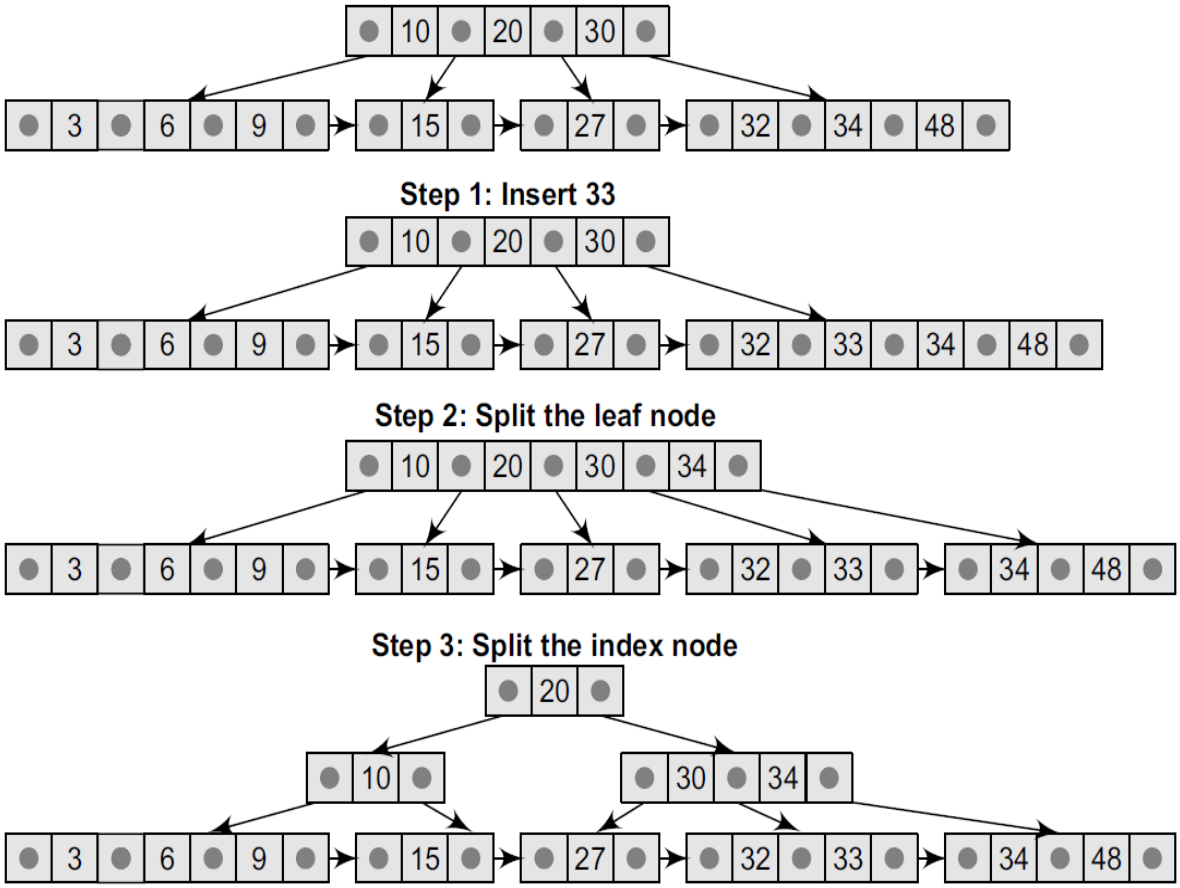
A B+ tree is a variant of B tree in which data is stored in only the leaf node not in the interior node and the leaf node is linked with each other via linked list.

Insertion of data in B+ tree: -

Rules:

1.) Insert the new data only in the leaf node.
2.) If leaf node is already full split the node and copy the middle element to the next node.
3.) If the parent (index) node is full then split the node and move the middle element to the next parent(index node).

Example: **Insertion of the 33 in the B+ tree**



Written by:
SAMUNDAR SINGH

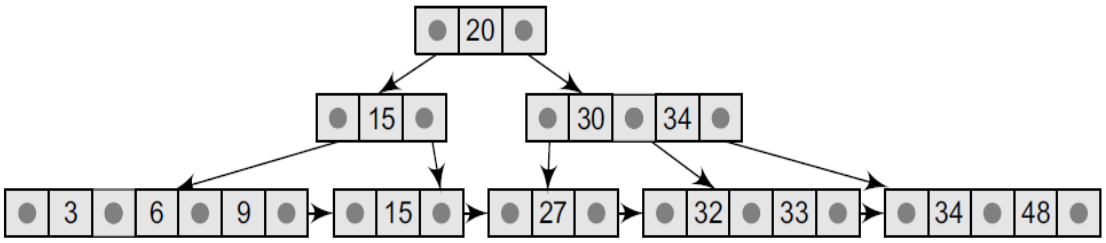
DATA STRUCTURE NOTES

Deletion of data in B+ tree: -

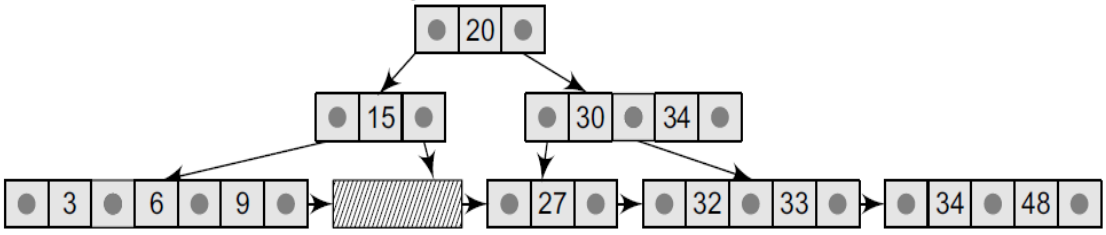
Rules:

1.) Delete the key and data from the leaves.
2.) If the leaf node underflows, merge that node with the sibling and delete the key in between them.
3.) If the index node underflows, merge that node with the sibling and move down the key in between them.

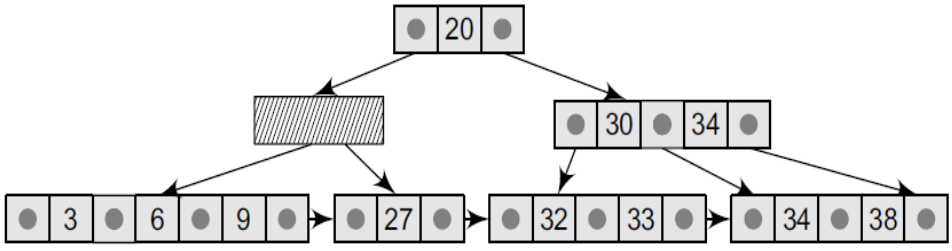
Example: **Deletion of the 15 in the B+ tree**



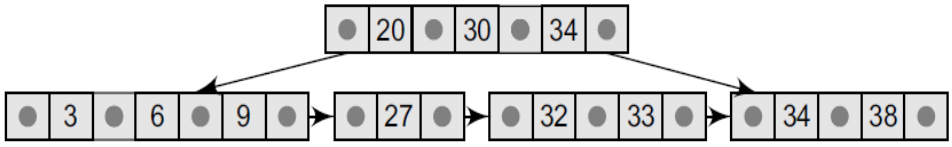
Step 1: Delete 15



Step 2: Leaf node underflows so merge with left sibling and remove key 15



Step 3: Now index node underflows, so merge with sibling and delete the node



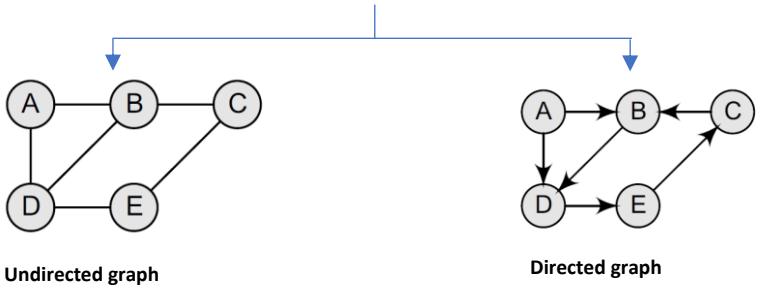
Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Graphs

Graphs are widely used to model any situation where entities or things are related to each other in pairs. A graph `G` is defined as an ordered set `(V, E)`, where `V(G)` represents the set of vertices and `E(G)` represents the edges that connect these vertices.

There are two types of graph either directed or undirected



Undirected graph

Directed graph

Representation of graph: -

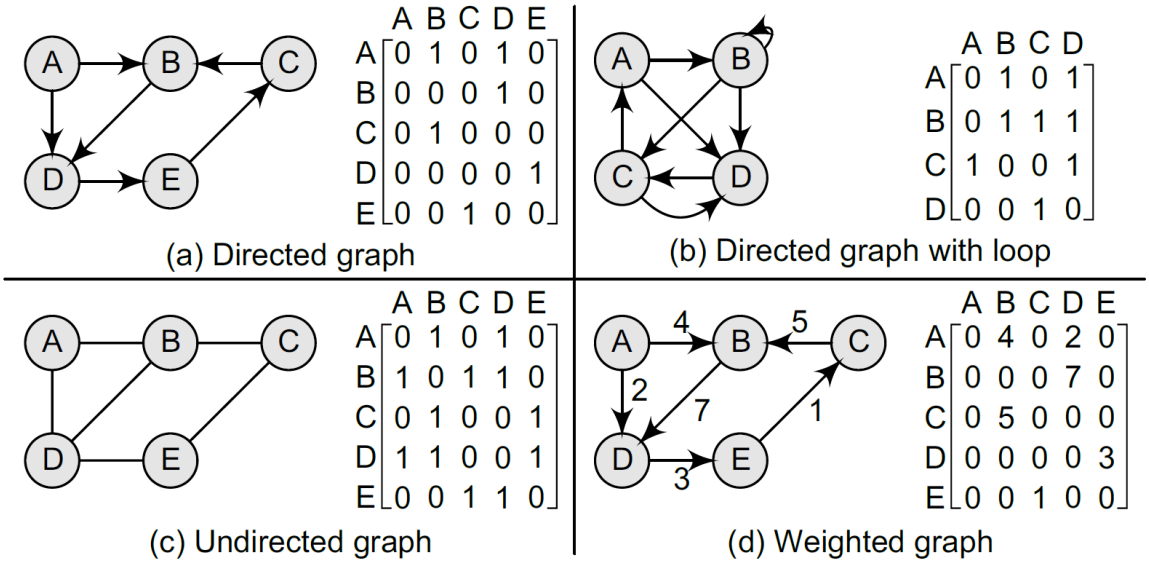
1.) Adjacency matrix representation.

2.) Adjacency list representation.

1.) Adjacency matrix representation:

Here we are using the matrix of ***n*** x ***n*** to represent the graph. It used to represent the nodes that are adjacent to one another. Two nodes are said to be adjacent when there is a edge connecting them.

Here what we do is just represent an edge with 1 and rest become zero. It means if two nodes are connected with one other then we make that connection as 1 in the ***n*** x ***n*** *matrix*. There are various cases in which a data can be represented. These are given below:



Note:

1. For a directed graph we can write 1 only for that path which is directed. (see fig (**a**) connection from A to B written only one time)
2. For undirected graph we write 1 for two times. (see fig (**c**) connection from A to B written two times A to B and B to A)
3. For loop we have to fill the diagonals of the matrix.

Written by:
SAMUNDAR SINGH

DATA STRUCTURE NOTES

Code to show implementation of Adjacency matrix:

[*] Adjacency matrix.c

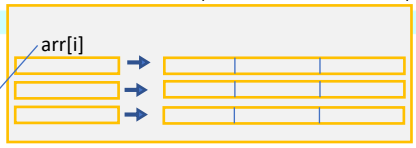
```c
#include<stdio.h>
#include<stdlib.h>
int node;

int **create();
void display(int **);

void main(){
    int choice,**ar;
    while(choice!=-1){
        printf("\n----------------------\n");
        printf("1.CREATE A ADJACENCY MATRIX.\n");
        printf("2.DISPLAY THE ADJACENCY MATRIX \n");
        printf("Enter your choice :");
        scanf("%d",&choice);
        printf("----------------------\n");
    if(choice==1){
        ar=create();
    }
    if(choice==2){
        display(ar);
    }
}
}

int  **create(){
    int data,i ,j;
    printf("ENTER THE NUMBER OF NODES :");
    scanf("%d",&data);
    node=data;

    int **arr;
    arr=(int **)malloc(sizeof(int*) * (node *node));    //this will create my array dynamically
    for(i=0;i<node;i++){
        arr[i]=(int *)malloc( node*sizeof(int));
    }

    for(i=0;i<node;i++){
        for(j=0;j<node;j++){
            arr[i][j]=0;                //this will set all the values of 2d default as 0;
        }
    }

    int u[node],v,k,neighbour,p;
    for(i=0;i<node;i++){
        printf("ENTER THE VALUES OF NODES :");
        scanf("%d",&u[i]);
    }
    printf("\n");
    for (k=0;k<node;k++){
    printf("ENTER THE NUMBER OF EDEGES of %d :",u[k]);
    scanf("%d",&v);
    for(p=0;p<v;p++){
        printf("ENTER NEIGHBOUR OF  %d :",u[k]);
        scanf("%d",&neighbour);
    for(j=0;j<node;j++){
        if(neighbour==u[j]){
            arr[k][j]=1;
        }
    }
    }
    }
    return arr;

}

void display(int **arr){
int i,j;
for(i=0;i<node;i++){
    for(j=0;j<node;j++){
        printf("%d    ",arr[i][j]);
    }
    printf("\n");
}

}
```

This global declaration is important to use the no of node in other functtion

Declare the function which return the double pointer like this

This will create the array dynamically and allocate memory for the 2d matrix

arr This will create a memory block which allocate a memory of 4*3*3



Here we create an array to store the value of the node

Here we run a loop which to check the neighbor node if it get match then the data of the 2d array is updated as 1

Here we return the double pointer for this we have to create the function whose return type is double pointer

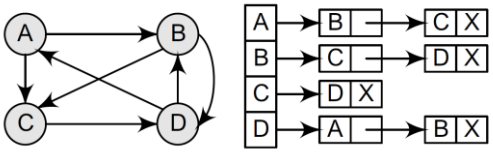
Passing the double pointer as the argument In the function

Written by:
SAMUNDAR SINGH

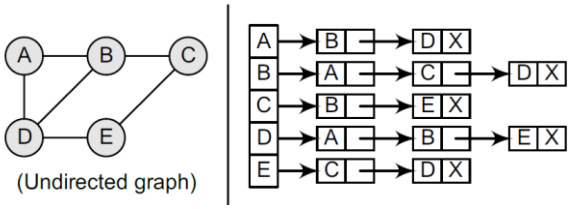
DATA STRUCTURE NOTES

1.) Adjacency list representation:

In this representation we are going to use an array and linked list to show the relation between different nodes. The data is stored in the array and a linked list hold the data of nodes connected to it.



Adjacency list of directed graph



Adjacency list of undirected graph

To be continued.....

Written by:
SAMUNDAR SINGH