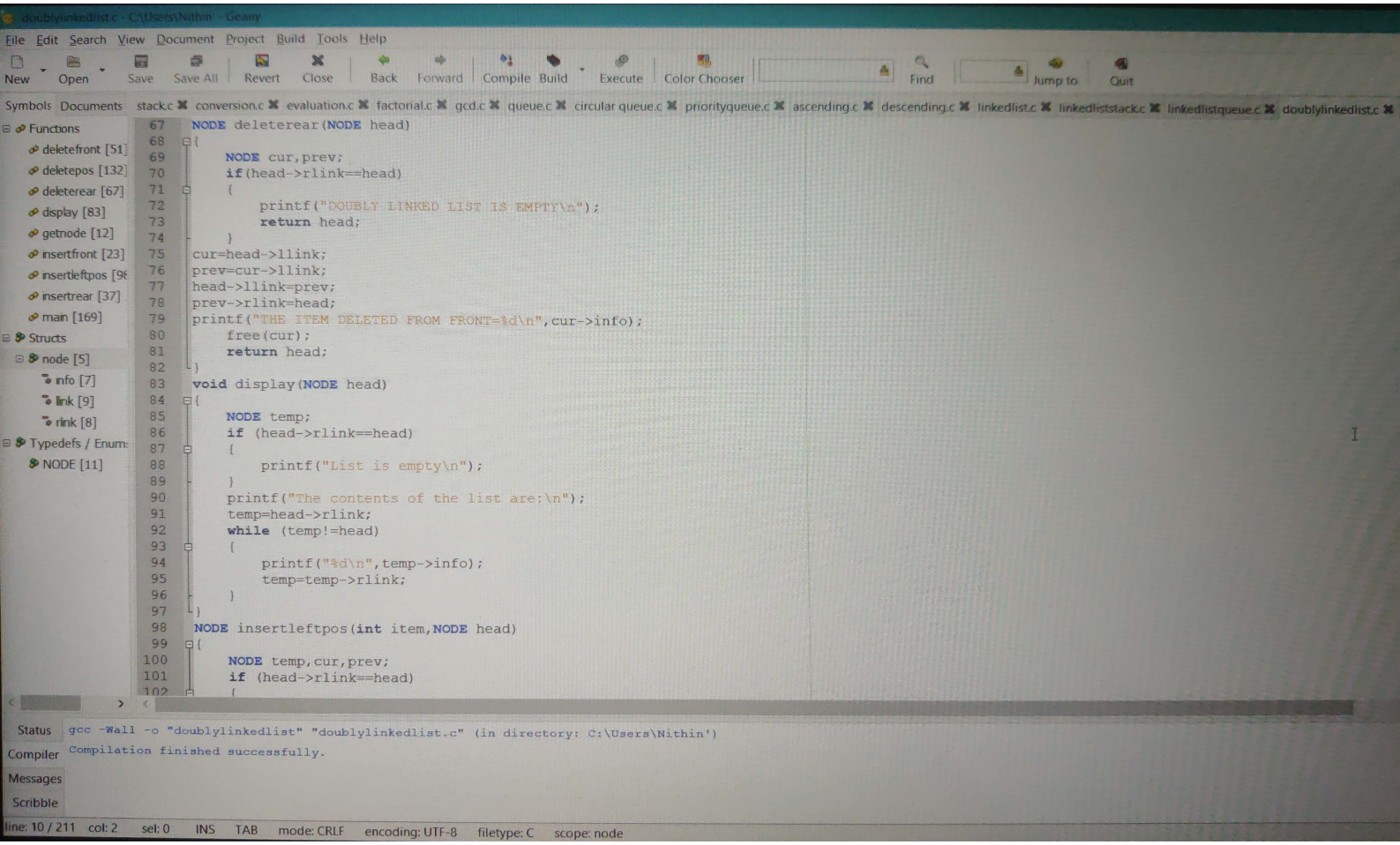
Document Project Build Tools Help

Save All | Revert | Close | Back | Forward | Compile | Build | Execute | Color Chooser | Find | Jump to

ck.c ✖ conversion.c ✖ evaluation.c ✖ factorial.c ✖ gcd.c ✖ queue.c ✖ circular queue.c ✖ priorityqueue.c ✖ ascending.c ✖ descending.c ✖ linkedlist.c ✖ linkedli

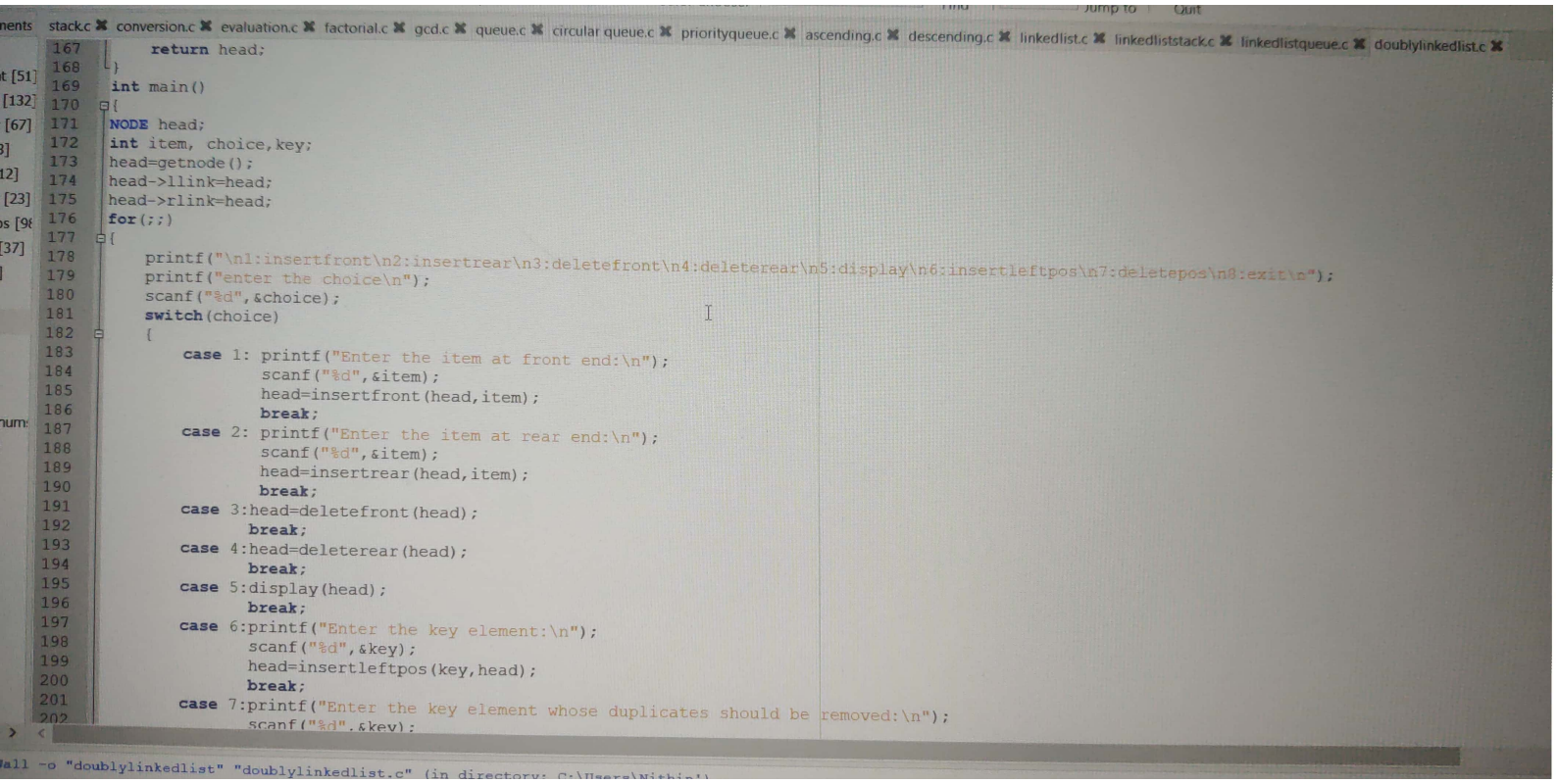
```c
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <process.h>
struct node
{
    int info;
    struct node *rlink;
    struct node *llink;
};
typedef struct node *NODE;
NODE getnode()
{
    NODE x;
    x=(NODE)malloc(sizeof(struct node));
    if (x==NULL)
    {
        printf("Memory full\n");
        exit(0);
    }
    return x;
}
NODE insertfront(NODE head,int item)
{
    NODE temp,cur;
    temp=getnode();
    temp->rlink=NULL;
    temp->llink=NULL;
    temp->info=item;
    cur=head->rlink;
    head->rlink=temp;
    temp->llink=head;
    temp->rlink=cur;
    cur->llink=temp;
    return head;
}
```

```
36  }
37  NODE insertrear(NODE head,int item)
38  {
39      NODE temp,cur;
40      temp=getnode();
41      temp->rlink=NULL;
42      temp->llink=NULL;
43      temp->info=item;
44      cur=head->llink;
45      head->llink=temp;
46      temp->rlink=head;
47      cur->rlink=temp;
48      temp->llink=cur;
49      return head;
50  }
51  NODE deletefront(NODE head)
52  {
53      NODE cur,next;
54      if(head->rlink==head)
55      {
56          printf("DOUBLY LINKED LIST IS EMPTY\n");
57          return head;
58      }
59      cur=head->rlink;
60      next=cur->rlink;
61      head->rlink=next;
62      next->llink=head;
63      printf("THE ITEM DELETED FROM FRONT=%d\n",cur->info);
64      free(cur);
65      return head;
66  }
67  NODE deleterear(NODE head)
68  {
69      NODE cur,prev;
70      if(head->rlink==head)
71      {
```


doublylinkedlist.c - C:\Users\Nithin - Geany
File Edit Search View Document Project Build Tools Help
New Open Save Save All Revert Close Back Forward Compile Build Execute Color Chooser Find Jump to Quit
Symbols Documents
stack.c conversion.c evaluation.c factorial.c gcd.c queue.c circular queue.c priorityqueue.c ascending.c descending.c linedlist.c linkedliststack.c linkedlistqueue.c doublylinkedlist.c
Functions
deletefront [51]
deletepos [132]
deleterear [67]
display [83]
getnode [12]
insertfront [23]
insertleftpos [9
insertrear [37]
main [169]
Structs
node [5]
info [7]
link [9]
rlink [8]
Typedefs / Enum
NODE [11]
67 NODE deleterear(NODE head)
68 {
69 NODE cur,prev;
70 if(head->rlink==head)
71 {
72 printf("DOUBLY LINKED LIST IS EMPTY\n");
73 return head;
74 }
75 cur=head->llink;
76 prev=cur->llink;
77 head->llink=prev;
78 prev->rlink=head;
79 printf("THE ITEM DELETED FROM FRONT=%d\n",cur->info);
80 free(cur);
81 return head;
82 }
83 void display(NODE head)
84 {
85 NODE temp;
86 if (head->rlink==head)
87 {
88 printf("List is empty\n");
89 }
90 printf("The contents of the list are:\n");
91 temp=head->rlink;
92 while (temp!=head)
93 {
94 printf("%d\n",temp->info);
95 temp=temp->rlink;
96 }
97 }
98 NODE insertleftpos(int item,NODE head)
99 {
100 NODE temp,cur,prev;
101 if (head->rlink==head)
102 {
Status
Compiler
Messages
Scribble
gcc -Wall -o "doublylinkedlist" "doublylinkedlist.c" (in directory: C:\Users\Nithin')
Compilation finished successfully.
line: 10 / 211 col: 2 sel: 0 INS TAB mode: CRLF encoding: UTF-8 filetype: C scope: node

```
    }
}
NODE insertleftpos(int item,NODE head)
{
    NODE temp,cur,prev;
    if (head->rlink==head)
    {
        printf("List is empty\n");
        return head;
    }
    cur=head->rlink;
    while (cur!=head)
    {
        if(cur->info==item)
        {
            break;
        }
        cur=cur->rlink;
    }
    if (cur==head)
    {
        printf("INVALID ITEM\n");
        return head;
    }
    prev=cur->llink;
    temp=getnode();
    temp->llink=NULL;
    temp->rlink=NULL;
    printf("Enter the item to be inserted at the left of the given item:\n");
    scanf("%d",&temp->info);
    prev->rlink=temp;
    temp->llink=prev;
    temp->rlink=cur;
    cur->llink=temp;
    return head;
}
```

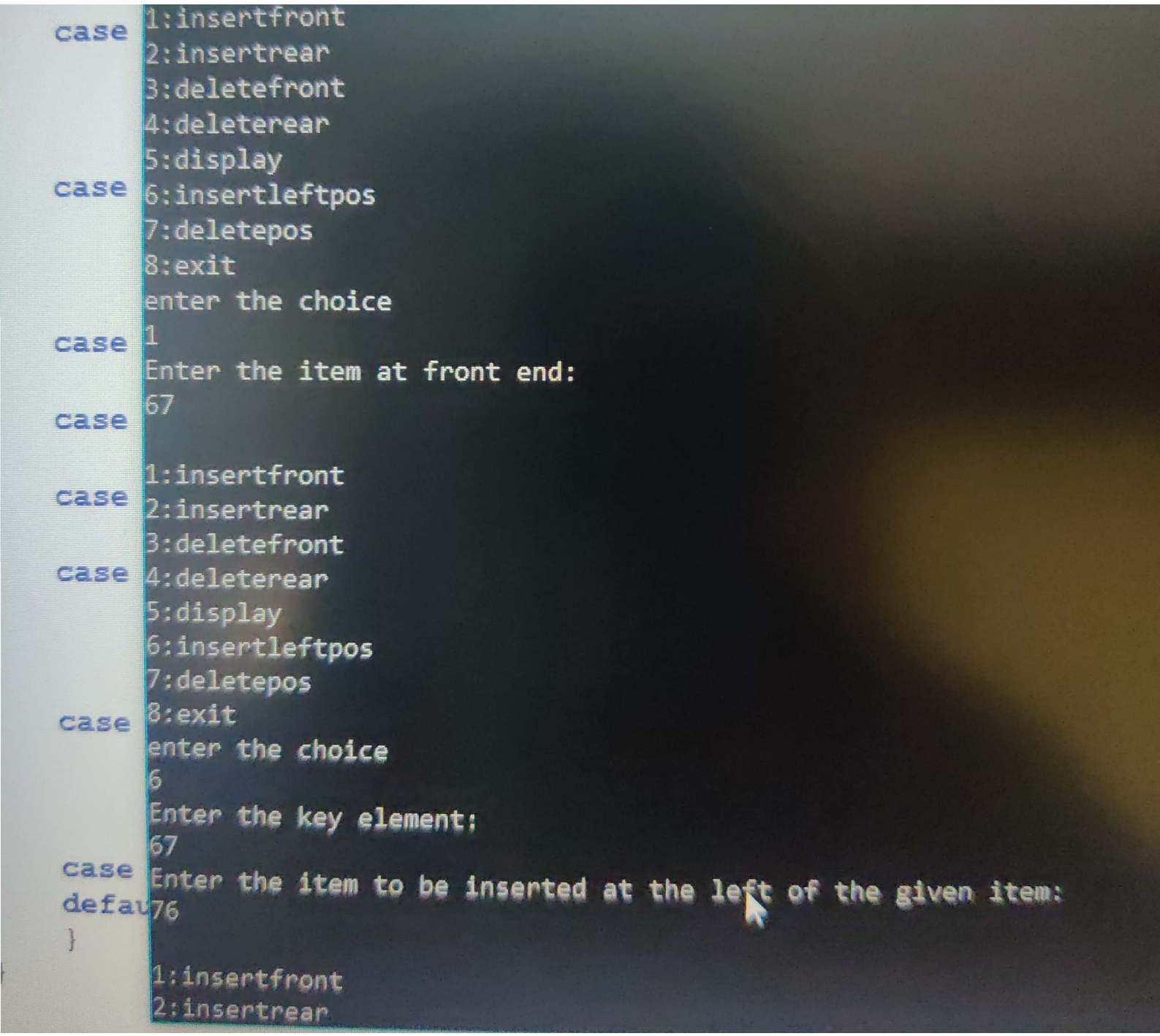
```
    return head;
}
NODE deletepos(int item,NODE head)
{
    NODE prev,cur,next;
    int count=0;
    if (head->rlink==head)
    {
        printf("List is empty\n");
        return head;
    }
    cur=head->rlink;
    while (cur!=head)
    {
        if (item!=cur->info)
        {
            cur=cur->rlink;
        }
        else
        {
                count++;
                prev=cur->llink;
                next=cur->rlink;
                prev->rlink=next;
                next->llink=prev;
                free(cur);
                cur=next;
            }
        }
    if (count==0)
    {
        printf("No such item found in the list\n");
    }
    else
    {
        printf("Removed all the duplicate elements of the given item successfully\n");
```

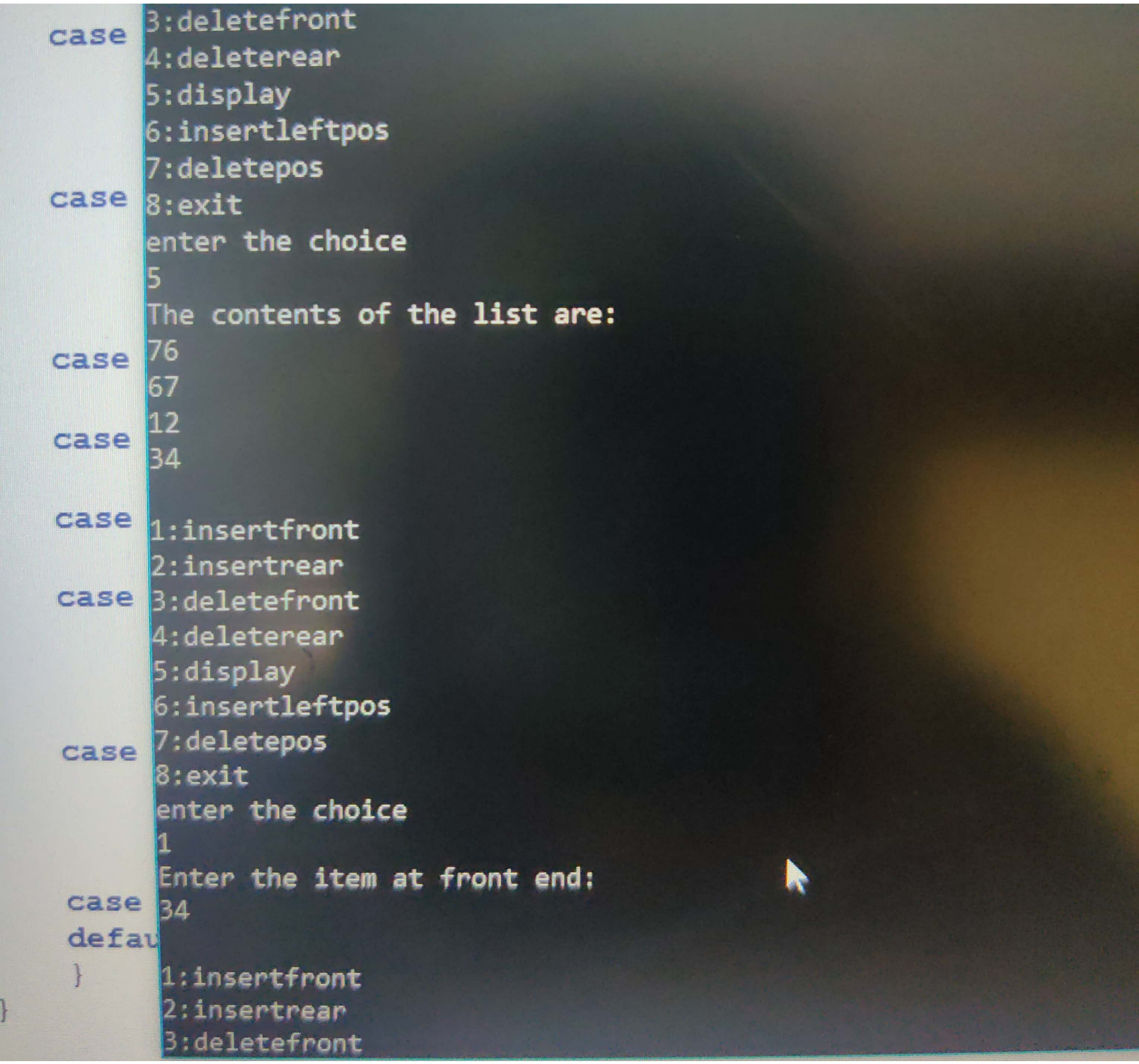

stack.c conversion.c evaluation.c factorial.c gcd.c queue.c circular queue.c priorityqueue.c ascending.c descending.c linkedlist.c linkedliststack.c linkedlistqueue.c doublylinkedlist.c
return head;
}
int main()
{
NODE head;
int item, choice,key;
head=getnode();
head->llink=head;
head->rlink=head;
for(;;)
{
printf("\n1:insertfront\n2:insertrear\n3:deletefront\n4:deleterear\n5:display\n6:insertleftpos\n7:deletepos\n8:exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("Enter the item at front end:\n");
scanf("%d",&item);
head=insertfront(head,item);
break;
case 2: printf("Enter the item at rear end:\n");
scanf("%d",&item);
head=insertrear(head,item);
break;
case 3:head=deletefront(head);
break;
case 4:head=deleterear(head);
break;
case 5:display(head);
break;
case 6:printf("Enter the key element:\n");
scanf("%d",&key);
head=insertleftpos(key,head);
break;
case 7:printf("Enter the key element whose duplicates should be removed:\n");
scanf("%d",&key);

```
1:insertfront
2:insertrear
3:deletefront
4:deleterear
5:display
6:insertleftpos
7:deletepos
8:exit
enter the choice
1
Enter the item at front end:
12

1:insertfront
2:insertrear
3:deletefront
4:deleterear
5:display
6:insertleftpos
7:deletepos
8:exit
enter the choice
2
Enter the item at rear end:
34

1:insertfront
2:insertrear
3:deletefront
```


case
case
case
case
case
defau
}
1:insertfront
2:insertrear
3:deletefront
4:deleterear
5:display
6:insertleftpos
7:deletepos
8:exit
enter the choice
1
Enter the item at front end:
67
1:insertfront
2:insertrear
3:deletefront
4:deleterear
5:display
6:insertleftpos
7:deletepos
8:exit
enter the choice
6
Enter the key element:
67
Enter the item to be inserted at the left of the given item:
76
1:insertfront
2:insertrear


case
case
case
case
case
case
defau
}
}
3:deletefront
4:deleterear
5:display
6:insertleftpos
7:deletepos
8:exit
enter the choice
5
The contents of the list are:
76
67
12
34
1:insertfront
2:insertrear
3:deletefront
4:deleterear
5:display
6:insertleftpos
7:deletepos
8:exit
enter the choice
1
Enter the item at front end:
34
1:insertfront
2:insertrear
3:deletefront

```
4:deleterear
5:display
6:insertleftpos
7:deletepos
8:exit
enter the choice
7
Enter the key element whose duplicates should be removed:
34
Removed all the duplicate elements of the given item successfully

1:insertfront
2:insertrear
3:deletefront
4:deleterear
5:display
6:insertleftpos
7:deletepos
8:exit
enter the choice
5
The contents of the list are:
76
67
12

1:insertfront
2:insertrear
3:deletefront
4:deleterear
```

```
2:insertrear
3:deletefront
4:deleterear
5:display
6:insertleftpos
7:deletepos
8:exit
enter the choice
3
THE ITEM DELETED FROM FRONT=76

1:insertfront
2:insertrear
3:deletefront
4:deleterear
5:display
6:insertleftpos
7:deletepos
8:exit
enter the choice
4
THE ITEM DELETED FROM FRONT=12

1:insertfront
2:insertrear
3:deletefront
4:deleterear
5:display
6:insertleftpos
7:deletepos
```