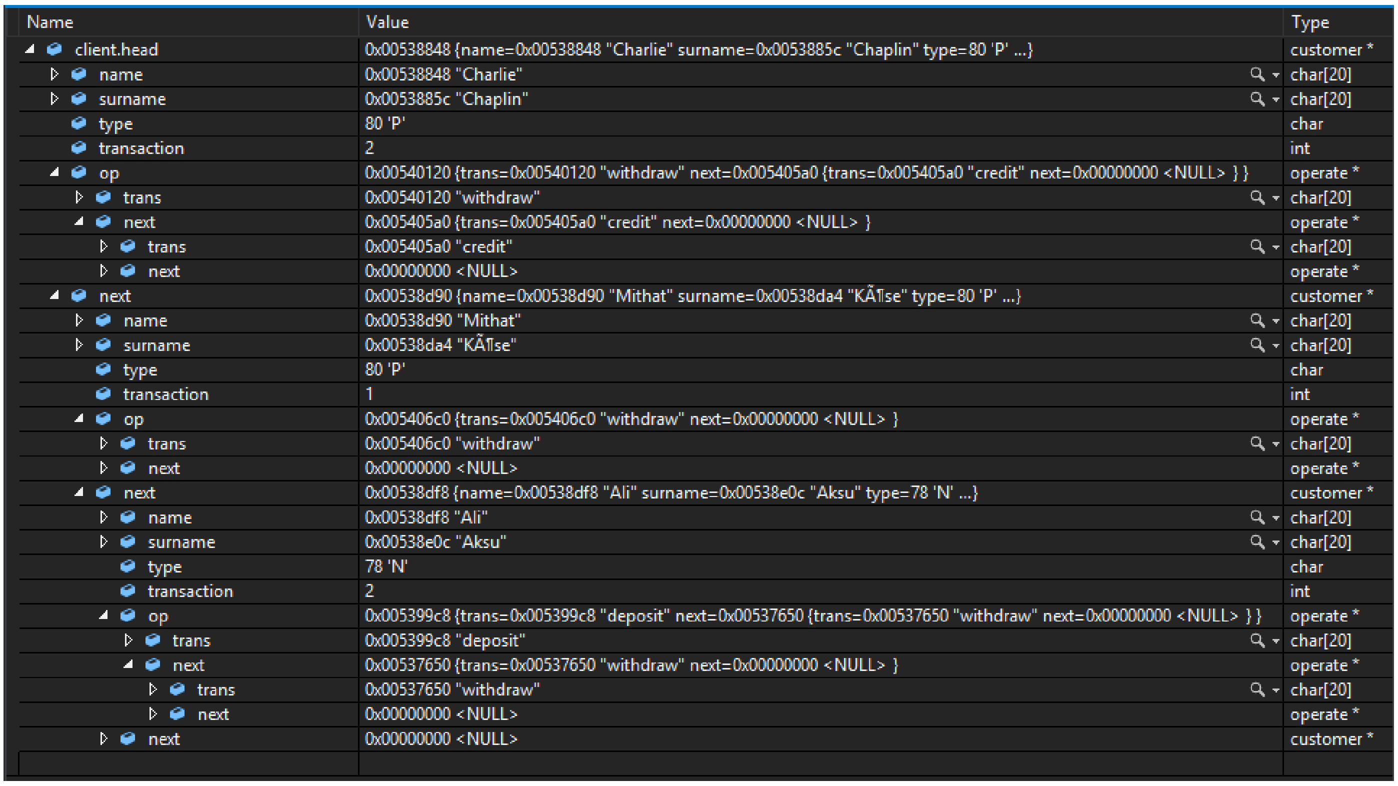
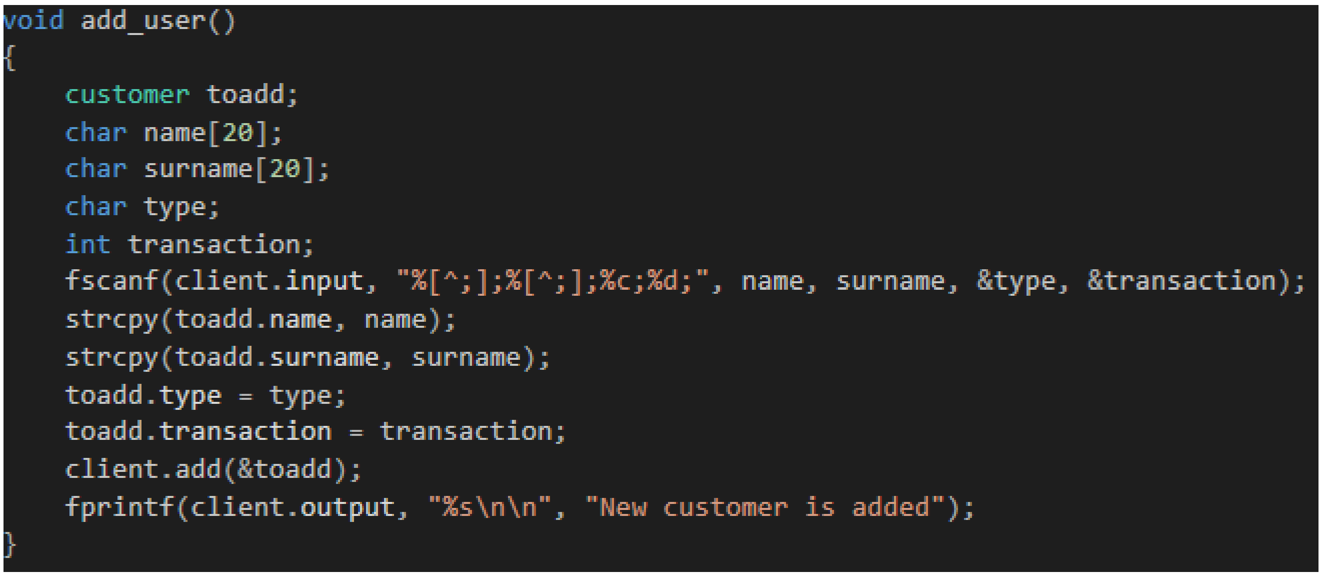
ID: 150120138 Name, Surname: Emre Özdil

1. Graphical Representation My Multi Linked List



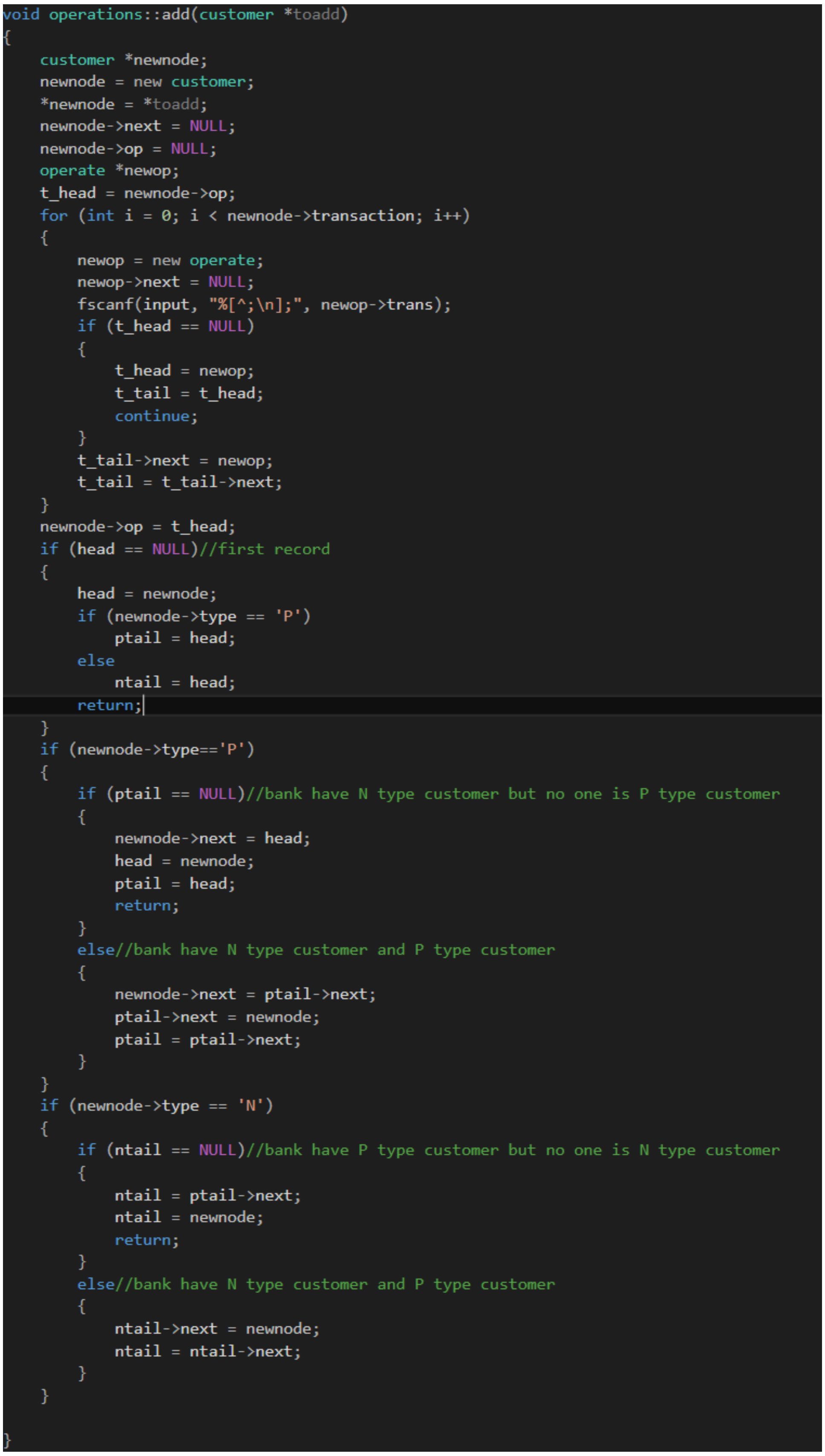
Name	Value	Type
client.head	0x00538848 {name=0x00538848 "Charlie" surname=0x0053885c "Chaplin" type=80 'P' ...}	customer *
name	0x00538848 "Charlie"	char[20]
surname	0x0053885c "Chaplin"	char[20]
type	80 'P'	char
transaction	2	int
op	0x00540120 {trans=0x00540120 "withdraw" next=0x005405a0 {trans=0x005405a0 "credit" next=0x00000000 <NULL> } }	operate *
trans	0x00540120 "withdraw"	char[20]
next	0x005405a0 {trans=0x005405a0 "credit" next=0x00000000 <NULL> }	operate *
trans	0x005405a0 "credit"	char[20]
next	0x00000000 <NULL>	operate *
next	0x00538d90 {name=0x00538d90 "Mithat" surname=0x00538da4 "KÃ¶se" type=80 'P' ...}	customer *
name	0x00538d90 "Mithat"	char[20]
surname	0x00538da4 "KÃ¶se"	char[20]
type	80 'P'	char
transaction	1	int
op	0x005406c0 {trans=0x005406c0 "withdraw" next=0x00000000 <NULL> }	operate *
trans	0x005406c0 "withdraw"	char[20]
next	0x00000000 <NULL>	operate *
next	0x00538df8 {name=0x00538df8 "Ali" surname=0x00538e0c "Aksu" type=78 'N' ...}	customer *
name	0x00538df8 "Ali"	char[20]
surname	0x00538e0c "Aksu"	char[20]
type	78 'N'	char
transaction	2	int
op	0x005399c8 {trans=0x005399c8 "deposit" next=0x00537650 {trans=0x00537650 "withdraw" next=0x00000000 <NULL> } }	operate *
trans	0x005399c8 "deposit"	char[20]
next	0x00537650 {trans=0x00537650 "withdraw" next=0x00000000 <NULL> }	operate *
trans	0x00537650 "withdraw"	char[20]
next	0x00000000 <NULL>	operate *
next	0x00000000 <NULL>	customer *

2. Adding Customer to Linked List



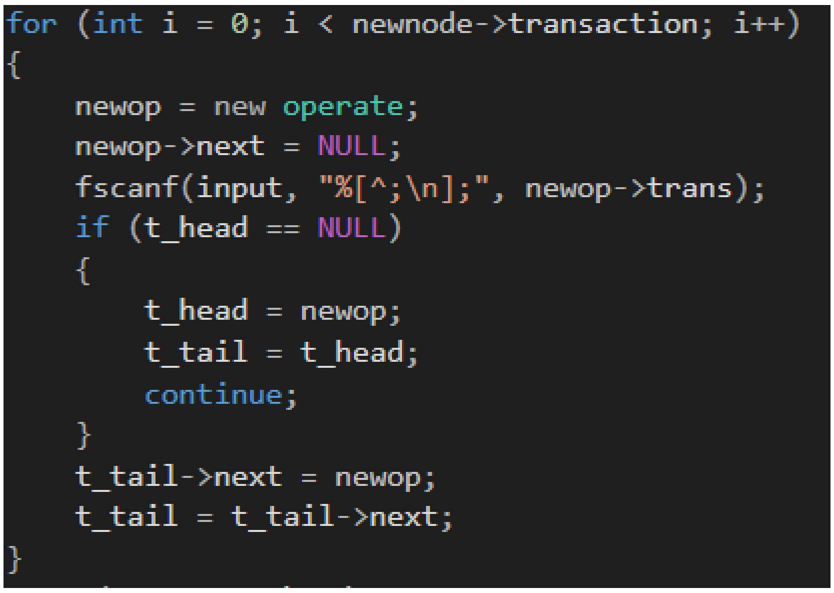
```
void add_user()
{
    customer toadd;
    char name[20];
    char surname[20];
    char type;
    int transaction;
    fscanf(client.input, "%[^;];%[^;];%c;%d;", name, surname, &type, &transaction);
    strcpy(toadd.name, name);
    strcpy(toadd.surname, surname);
    toadd.type = type;
    toadd.transaction = transaction;
    client.add(&toadd);
    fprintf(client.output, "%s\n\n", "New customer is added");
}
```

3. Inserting to Linked List



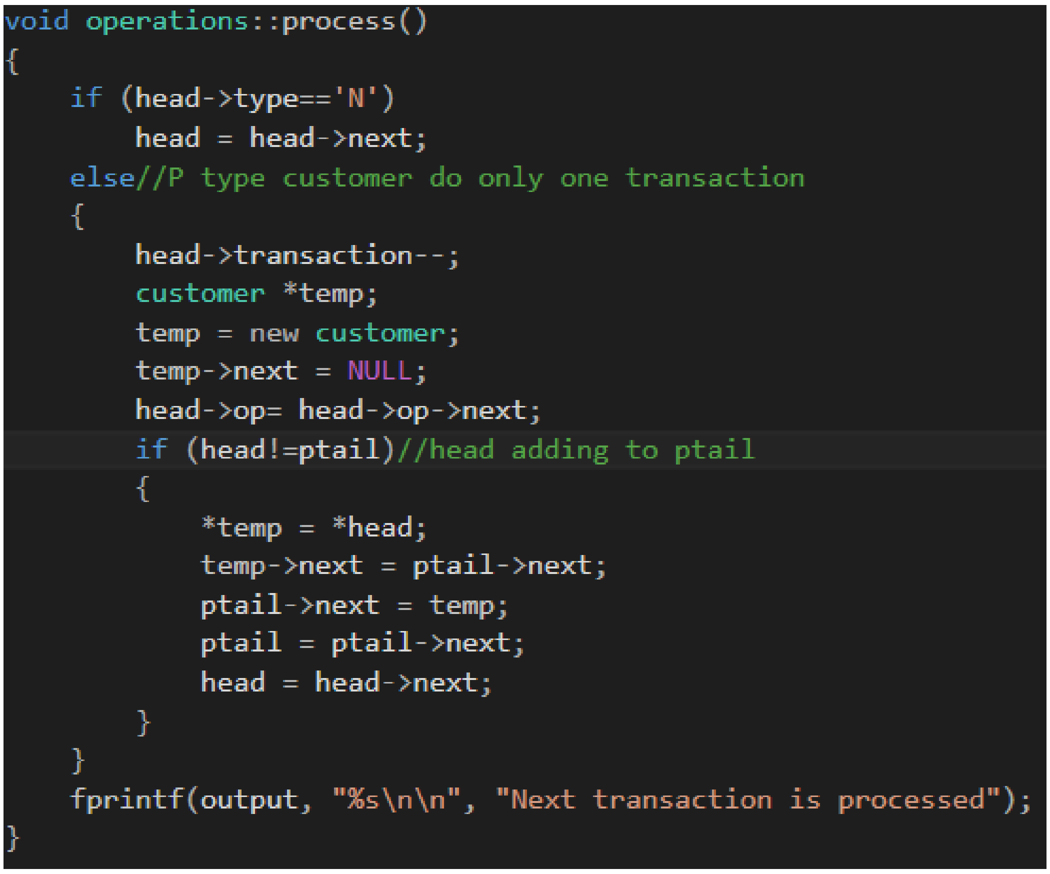
```
void operations::add(customer *toadd)
{
    customer *newnode;
    newnode = new customer;
    *newnode = *toadd;
    newnode->next = NULL;
    newnode->op = NULL;
    operate *newop;
    t_head = newnode->op;
    for (int i = 0; i < newnode->transaction; i++)
    {
        newop = new operate;
        newop->next = NULL;
        fscanf(input, "%[^;\n];", newop->trans);
        if (t_head == NULL)
        {
            t_head = newop;
            t_tail = t_head;
            continue;
        }
        t_tail->next = newop;
        t_tail = t_tail->next;
    }
    newnode->op = t_head;
    if (head == NULL)//first record
    {
        head = newnode;
        if (newnode->type == 'P')
            ptail = head;
        else
            ntail = head;
        return;
    }
    if (newnode->type=='P')
    {
        if (ptail == NULL)//bank have N type customer but no one is P type customer
        {
            newnode->next = head;
            head = newnode;
            ptail = head;
            return;
        }
        else//bank have N type customer and P type customer
        {
            newnode->next = ptail->next;
            ptail->next = newnode;
            ptail = ptail->next;
        }
    }
    if (newnode->type == 'N')
    {
        if (ntail == NULL)//bank have P type customer but no one is N type customer
        {
            ntail = ptail->next;
            ntail = newnode;
            return;
        }
        else//bank have N type customer and P type customer
        {
            ntail->next = newnode;
            ntail = ntail->next;
        }
    }

}
```

4. Adding Transaction



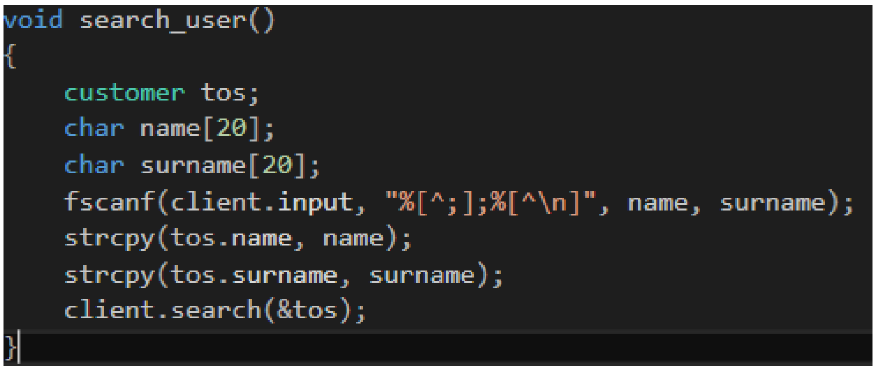
```
for (int i = 0; i < newnode->transaction; i++)
{
    newop = new operate;
    newop->next = NULL;
    fscanf(input, "%[^;\n];", newop->trans);
    if (t_head == NULL)
    {
        t_head = newop;
        t_tail = t_head;
        continue;
    }
    t_tail->next = newop;
    t_tail = t_tail->next;
}
```

5. Process



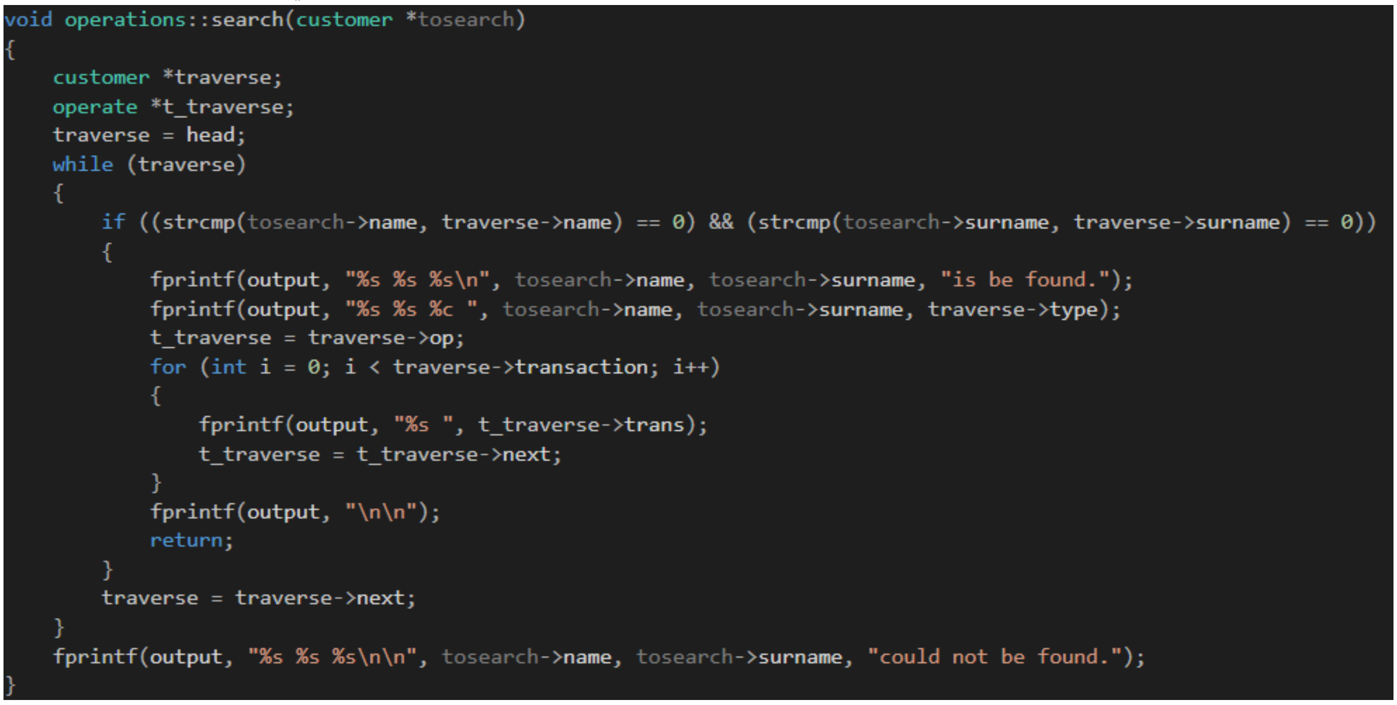
```
void operations::process()
{
    if (head->type=='N')
        head = head->next;
    else//P type customer do only one transaction
    {
        head->transaction--;
        customer *temp;
        temp = new customer;
        temp->next = NULL;
        head->op= head->op->next;
        if (head!=ptail)//head adding to ptail
        {
            *temp = *head;
            temp->next = ptail->next;
            ptail->next = temp;
            ptail = ptail->next;
            head = head->next;
        }
    }
    fprintf(output, "%s\n\n", "Next transaction is processed");
}
```

6. Search



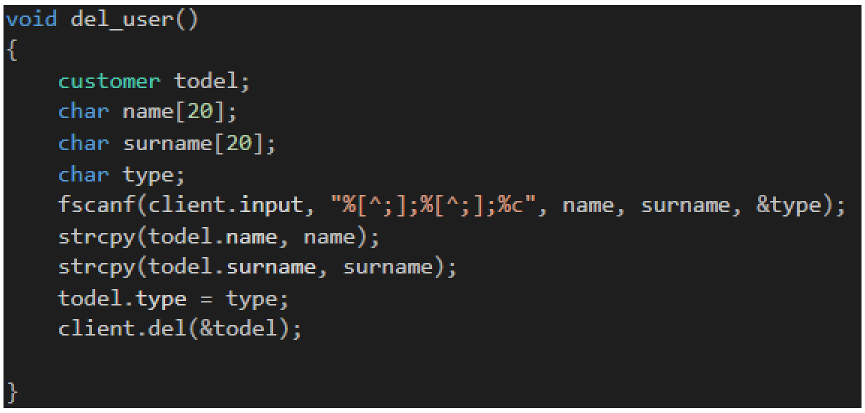
```
void search_user()
{
    customer tos;
    char name[20];
    char surname[20];
    fscanf(client.input, "%[^;];%[^\n]", name, surname);
    strcpy(tos.name, name);
    strcpy(tos.surname, surname);
    client.search(&tos);
}
```

Go to client.search()



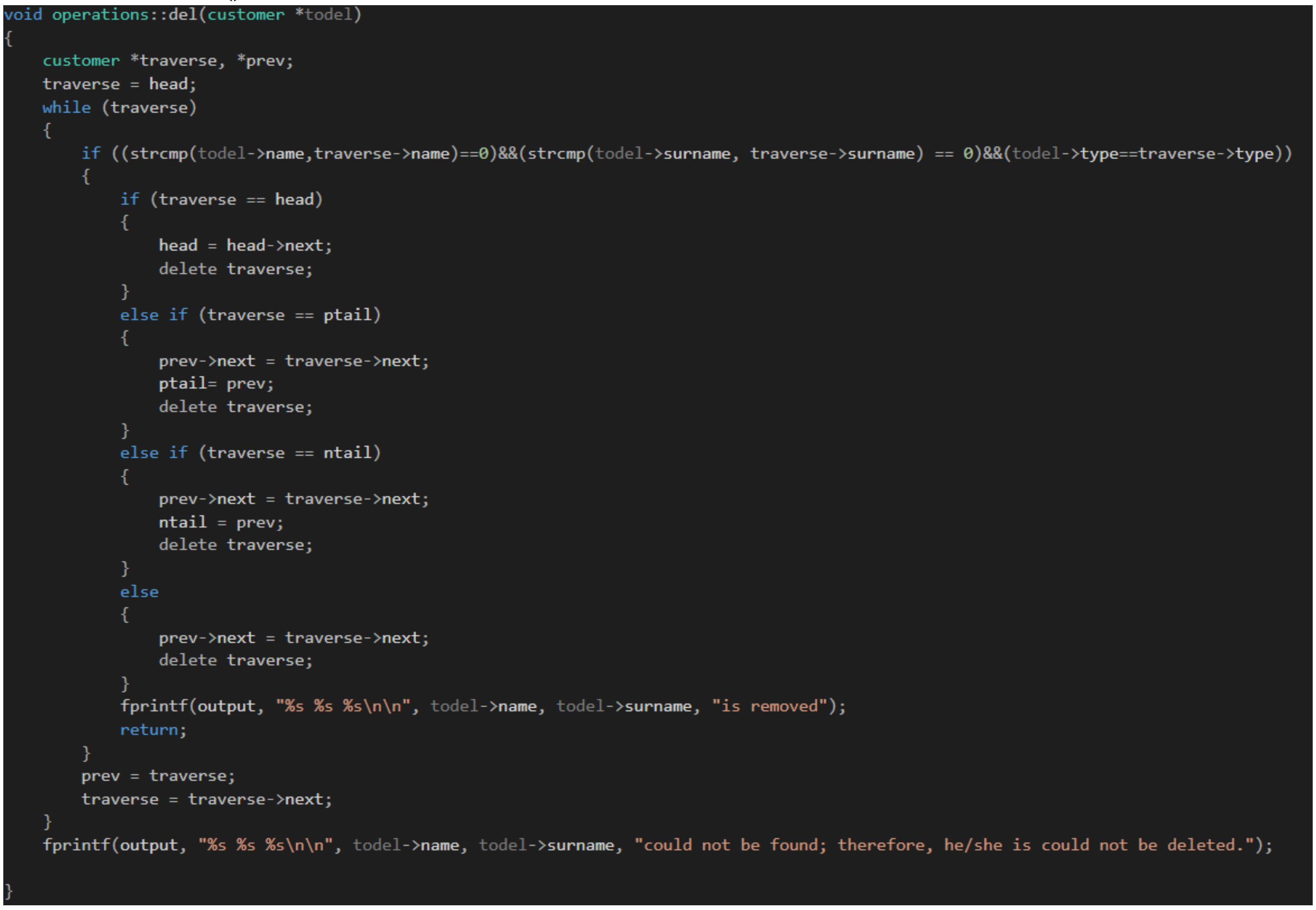
```
void operations::search(customer *tosearch)
{
    customer *traverse;
    operate *t_traverse;
    traverse = head;
    while (traverse)
    {
        if ((strcmp(tosearch->name, traverse->name) == 0) && (strcmp(tosearch->surname, traverse->surname) == 0))
        {
            fprintf(output, "%s %s %s\n", tosearch->name, tosearch->surname, "is be found.");
            fprintf(output, "%s %s %c ", tosearch->name, tosearch->surname, traverse->type);
            t_traverse = traverse->op;
            for (int i = 0; i < traverse->transaction; i++)
            {
                fprintf(output, "%s ", t_traverse->trans);
                t_traverse = t_traverse->next;
            }
            fprintf(output, "\n\n");
            return;
        }
        traverse = traverse->next;
    }
    fprintf(output, "%s %s %s\n\n", tosearch->name, tosearch->surname, "could not be found.");
}
```

7. Remove



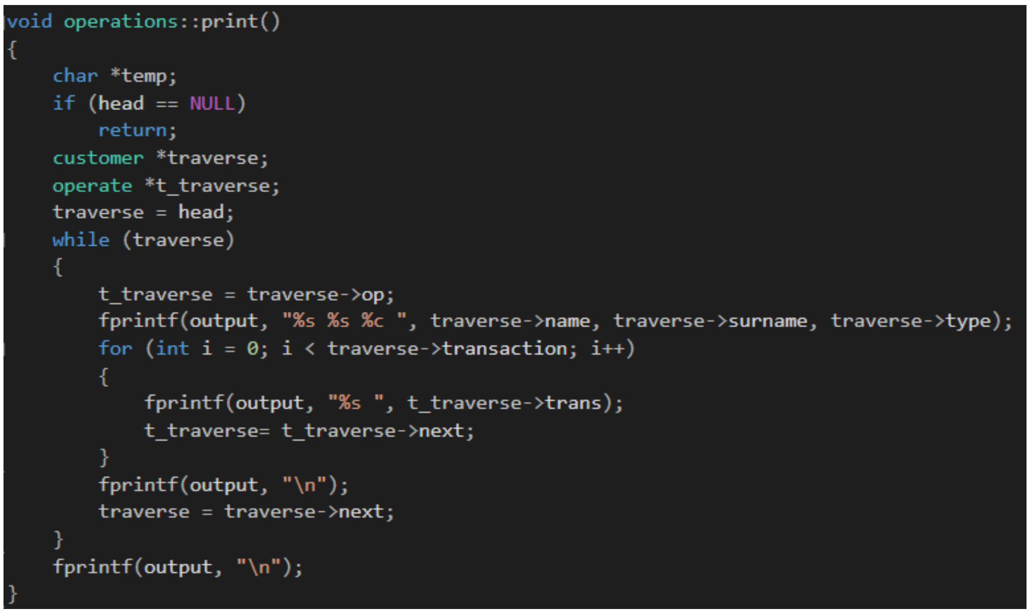
```
void del_user()
{
    customer todel;
    char name[20];
    char surname[20];
    char type;
    fscanf(client.input, "%[^;];%[^;];%c", name, surname, &type);
    strcpy(todel.name, name);
    strcpy(todel.surname, surname);
    todel.type = type;
    client.del(&todel);

}
```

Go to client.del()



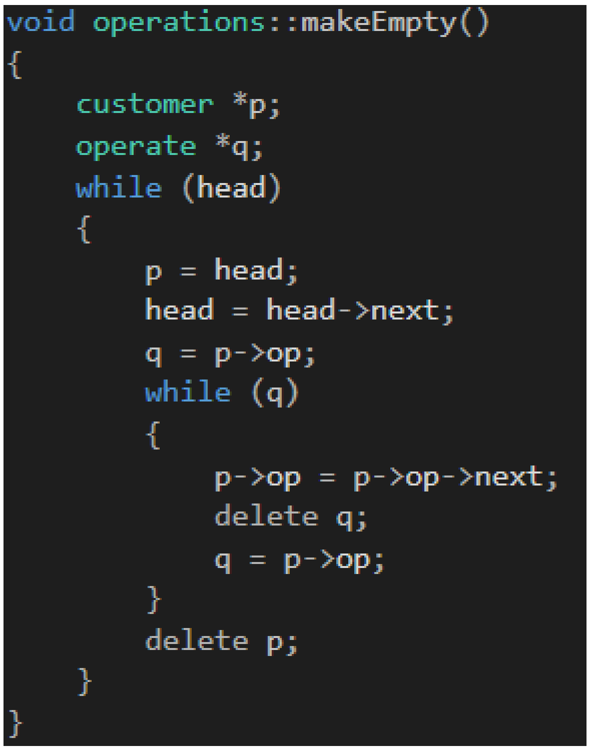
```
void operations::del(customer *todel)
{
    customer *traverse, *prev;
    traverse = head;
    while (traverse)
    {
        if ((strcmp(todel->name,traverse->name)==0)&&(strcmp(todel->surname, traverse->surname) == 0)&&(todel->type==traverse->type))
        {
            if (traverse == head)
            {
                head = head->next;
                delete traverse;
            }
            else if (traverse == ptail)
            {
                prev->next = traverse->next;
                ptail= prev;
                delete traverse;
            }
            else if (traverse == ntail)
            {
                prev->next = traverse->next;
                ntail = prev;
                delete traverse;
            }
            else
            {
                prev->next = traverse->next;
                delete traverse;
            }
            fprintf(output, "%s %s %s\n\n", todel->name, todel->surname, "is removed");
            return;
        }
        prev = traverse;
        traverse = traverse->next;
    }
    fprintf(output, "%s %s %s\n\n", todel->name, todel->surname, "could not be found; therefore, he/she is could not be deleted.");

}
```

8. Print



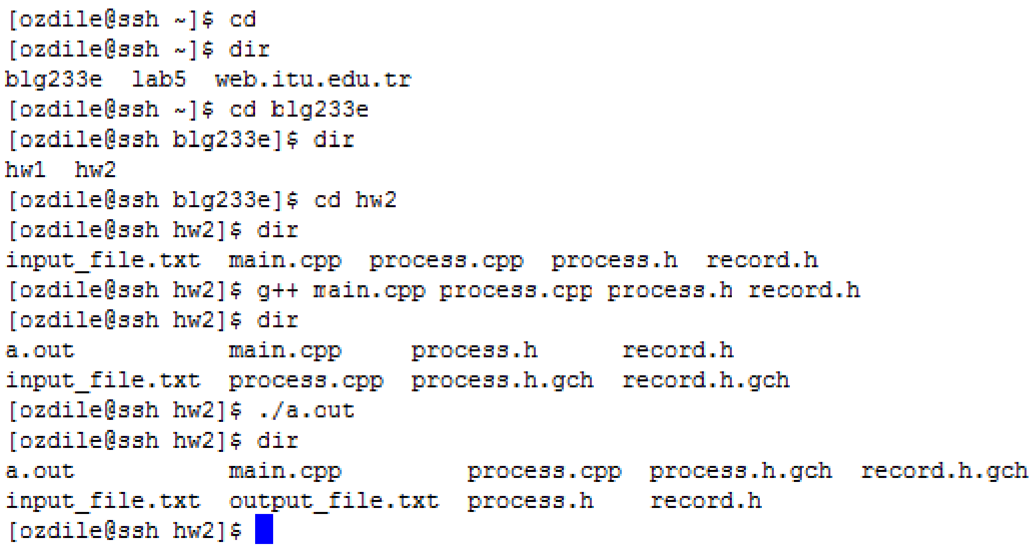
```
void operations::print()
{
    char *temp;
    if (head == NULL)
        return;
    customer *traverse;
    operate *t_traverse;
    traverse = head;
    while (traverse)
    {
        t_traverse = traverse->op;
        fprintf(output, "%s %s %c ", traverse->name, traverse->surname, traverse->type);
        for (int i = 0; i < traverse->transaction; i++)
        {
            fprintf(output, "%s ", t_traverse->trans);
            t_traverse= t_traverse->next;
        }
        fprintf(output, "\n");
        traverse = traverse->next;
    }
    fprintf(output, "\n");
}
```

9. Delete Dynamic List



```
void operations::makeEmpty()
{
    customer *p;
    operate *q;
    while (head)
    {
        p = head;
        head = head->next;
        q = p->op;
        while (q)
        {
            p->op = p->op->next;
            delete q;
            q = p->op;
        }
        delete p;
    }
}
```

10. SSH Compile and Run



```
[ozdile@ssh ~]$ cd
[ozdile@ssh ~]$ dir
blg233e  lab5  web.itu.edu.tr
[ozdile@ssh ~]$ cd blg233e
[ozdile@ssh blg233e]$ dir
hw1  hw2
[ozdile@ssh blg233e]$ cd hw2
[ozdile@ssh hw2]$ dir
input_file.txt  main.cpp  process.cpp  process.h  record.h
[ozdile@ssh hw2]$ g++ main.cpp process.cpp process.h record.h
[ozdile@ssh hw2]$ dir
a.out           main.cpp     process.h      record.h
input_file.txt  process.cpp  process.h.gch  record.h.gch
[ozdile@ssh hw2]$ ./a.out
[ozdile@ssh hw2]$ dir
a.out           main.cpp         process.cpp  process.h.gch  record.h.gch
input_file.txt  output_file.txt  process.h    record.h
[ozdile@ssh hw2]$ 
```