

Projeto Final

Cristóvão Eustáquio

Fernando Gomes

1. INTRODUÇÃO

A nossa empresa se trata de uma hamburgueria. Para dinamizar o atendimento, desenvolvemos um sistema que torna isso virtual, ou seja, o cliente pode visualizar o cardápio e fazer o pedido via PC ou celular. Os funcionários podem fazer *login* no sistema e registrar o pedido para que seja preparado. O gerente tem acesso a todas as funcionalidades já mencionadas e, além delas, pode criar, ler, atualizar e deletar usuários, cardápio e estoque.

2. IMPLEMENTAÇÃO

Dividimos o programa em módulos com as funções usadas pelo cliente, funcionário e gerente.

2.1. Arquivos

Os arquivos necessários para rodar o programa estão abaixo:

cardapio.txt	estoque.txt
1) X Egg bacon 2) Big Moc 3) Final Food 4) Atom 5) Baconese 6) Whoaper 7) X-Tudo 8) X-Bacon 9) Double Cheese	001 0010 002 1000 003 0500 004 0060 005 0037 006 0015 007 0056 008 0200 009 0500
usuarios.txt	senha_gerente.txt
aaaa 1234 ffff 1111 gggg 1010	123

Os arquivos “pedido_pendente.txt” e “pedidos_registrados.txt” são arquivos vazios e são gerados durante a execução do programa.

2.2. Módulo F_cliente:

O módulo cliente é o nível mais baixo de acesso do programa contendo 3 funções. A função “exibeCard()” imprime o arquivo inteiro, na qual esta lógica foi utilizada como base em diversas outras funções, dessa forma foi utilizado o algoritmo abaixo como base:

```

while (!feof(pcard))
{
    fflush(stdin);
    fgets(linha,200,pcard);
    printf("%s",linha);
}

```

A função “pedidos()” recebe do usuário o número correspondente ao pedido desejado, também é possível alterar o pedido antes de encaminhar, isso é feito de maneira recursiva. Como o número de pedidos é desconhecido foi alocado dinamicamente um vetor “pedido” por meio do calloc para receber o número inserido, visto que o tamanho necessário para a memória será realocada até que o usuário termine o pedido. Feito isso, o pedido é registrado no arquivo “pedido_pendente.txt”. A função “cliente()” possui um menu contendo as funções mencionadas acima.

2.3. Módulo F_funcionario:

Nesse módulo a *struct* de usuário foi utilizada para receber as informações de *login* e senha e possui 3 funções. A função “registraPedido()” abre o arquivo “pedido_pendente.txt” e registra diretamente no arquivo “pedidos_registrados.txt”. A função “exibePedidos()” é baseada na função “exibeCard()” com o intuito de imprimir na tela o arquivo “pedido_pendente.txt”.

A função “funcionario()” abre o arquivo “usuario.txt” para realizar a verificação do *login* e senha digitado (armazenado na *struct* usuario) com o *login* e senha existentes no arquivo. Posteriormente, há um menu com as funcionalidades explicadas acima.

2.4. Módulo F_gerente:

O módulo referente ao gerente, é o tipo de usuário com o maior nível hierárquico do programa, visto que toda a implementação do CRUD está contida neste. São dotados de 10 funções e 4 *structs*, na qual quase todas funções utilizam as variáveis das *structs* para guardar os dados recebidos.

```

5      typedef struct ingredientes_
6      {
7          char coding[10];
8          char qtd[5];
9      } ingredientes;
10     typedef struct prato_
11     {
12         char nprt[11];
13         ingredientes in;
14     } prato;
15     typedef struct usuario_
16     {
17         char login[11];
18         char logind[11];
19         char senha[9];
20         char senhad[9];
21     } usuario;
22     typedef struct gestor_
23     {
24         char senhacomp[100];
25         char senhadig[100];
26         char newsenha[100];
27     } gestor;

```

Nessas *structs* observa-se que a estrutura “ingredientes” está atrelada com a estrutura “prato”. A estrutura “usuário” foi utilizada para guardar as informações do arquivo referente ao *login* e senha dos usuários na qual o administrador os gerencia. Já na estrutura “gestor” é armazenada uma senha digitada e uma senha para comparar (presente no arquivo) e a nova senha (caso deseje-se alterar esta).

A função “createUsr()” recebe um novo usuário e sua respectiva senha (e a concatena com um “\n” para manter o padrão do arquivo), em seguida salva ambos no arquivo “usuario.txt”. Como representa o código abaixo:

```

28 void createUsr()
29 {
30     FILE *pc = fopen("usuarios.txt", "a+");
31     usuario usr;
32     if (pc != NULL)
33     {
34         printf("\nDigite o novo login:\n");
35         fflush(stdin);
36         gets(usr.login);
37         printf("\nDigite a senha para %s:\n", usr.login);
38         scanf("%s", usr.senha);
39         strcat(usr.senha, "\n");
40         fprintf(pc, "%s %s\n", usr.login, usr.senha);
41         system("cls");
42         printf("\nUsuario registrado!\n");
43         fclose(pc);
44     }
45 }

```

A função “exibeUsr()” foi utilizada tendo como base o algoritmo da função “exibeCard()”, sendo adaptada para o gerente. Pois foi utilizado uma *flag* “DELETADO!” caso um usuário seja deletado. Com isso essa linha será desconsiderada no momento da impressão.

Para alterar a senha do gerente foi criado uma função “altSGer()” onde é preciso inserir a senha atual, sendo feito uma comparação da senha digitada com a existente no arquivo “senha_gerente.txt” (os dados são salvos na *struct* gestor), com um limite de 3 tentativas, posteriormente é aberto o arquivo no modo *write* e sobrescrito o arquivo com a nova senha:

```

fscanf(psg, "%s", g.senhacomp);
printf("Digite sua senha:\n");
scanf("%s", g.senhadig);
fclose(psg);
while (strcmp(g.senhadig, g.senhacomp) != 0)
{
    printf("\nSenha incorreta, digite novamente:\n");
    scanf("%s", g.senhadig);
    i++;
    if (i > 2)
        break;
}
if (strcmp(g.senhadig, g.senhacomp) == 0)
{
    psg = fopen("senha_gerente.txt", "w");
    printf("\nDigite sua nova senha:\n");
    scanf("%s", g.newsenha);
    fseek(psg, 0, SEEK_SET);
    fprintf(psg, "%s", g.newsenha);
    fprintf(psg, "%s", " ");
    printf("\nSenha Alterada!\n");
    fclose(psg);
}

```

De maneira análoga, a função “altUsr()” recebe o *login* (obrigatoriamente 4 caracteres) do usuário que será alterado, e será feita uma varredura no arquivo para encontrá-lo e posicionar o cursor na linha deste. Em seguida, é registrado o novo *login* e senha (obrigatoriamente 4 caracteres) na *struct* *usuario* e posteriormente no arquivo, como mostra o código abaixo:

```
printf("\nDigite o login do usuario que deseja mudar: \n");
fflush(stdin);
gets(u.logind);
while(fgets(linha,40,pc)!=NULL)
{
    char *token = strtok(linha," ");
    if(strcmp(u.logind,token)== 0)
    {
        printf("\nDigite o novo login (obrigatoriamente 4 digitos!):\n");
        fflush(stdin);
        gets(u.login);
        printf("\nDigite a nova senha (obrigatoriamente 4 digitos!):\n");
        fflush(stdin);
        gets(u.senhad);
        fseek(pc,-11,SEEK_CUR);
        fprintf(pc,"%s %s\n",u.login,u.senhad);
        system("cls");
        printf("\nUsuario alterado com sucesso!\n");
        fclose(pc);
        break;
    }
}
```

A função “delUsr()” faz a mesma varredura de busca no arquivo “usuário.txt” para encontrar o *login* a ser deletado de acordo com o digitado armazenado na *struct* usuário. Assim, é substituído a linha do *login* por uma flag “DELETADO!”. De acordo com o código abaixo:

```
fseek(pc,-11,SEEK_CUR);
fprintf(pc,"DELETADO!\n");
system("cls");
printf("\nUsuario deletado!\n");
fclose(pc);
```

As funções de criar, alterar e deletar serão derivadas das exemplificadas acima, alterando somente os arquivos e as variáveis das *structs*. A função “creatIng()” recebe um novo código e uma nova quantidade de um ingrediente, armazena na *struct* *prato* aninhada com a *struct* *ingredientes*, em seguida a concatena com zeros (para manter o padrão do arquivo) e, finalmente, a registra no arquivo “estoque.txt”. A função “exibeEst()” possui algoritmo já citado anteriormente e foi adaptado para imprimir o arquivo “estoque.txt”.

A função “altQtIng()” é baseado nas funções que alteram determinado dado, porém para essa função foi utilizado o arquivo “estoque.txt”. Para isso salva-se em uma *struct* o código do ingrediente que deseja-se atualizar a quantidade, dessa forma posiciona-se o cursor na quantidade e a altera. Para criar um prato utilizamos a função “createCard()”, na qual abrimos o arquivo “cardapio.txt”, no modo anexo e digitamos o novo nome do prato que será armazenado na *struct* *prato*. Dessa forma, é registrado no arquivo o novo prato.

A função gerente abre o arquivo de “senha_gerente.txt” para conferir a permissão por meio da comparação da senha existente com a senha digitada –

salva no *struct* gestor. Feito isso, foi feito um switch com um menu de todas as 8 funcionalidades agregadas ao gerente (funções citadas acima).

2.5. Main

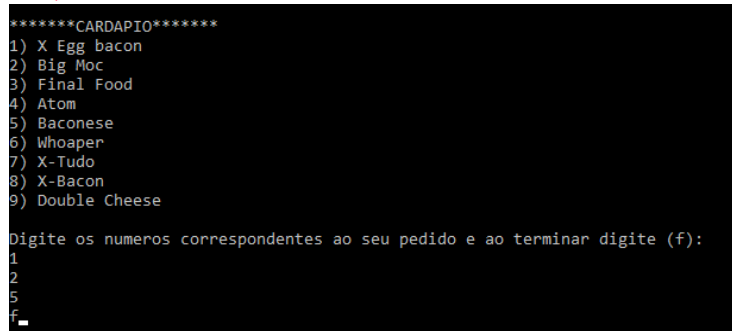
O módulo principal possui um menu inicial, na qual as opções são os tipos de usuários.

3. TESTE PARA VALIDAÇÃO

3.1. Verificar se o pedido digitado está sendo escrito no arquivo “pedido_pendente.txt”

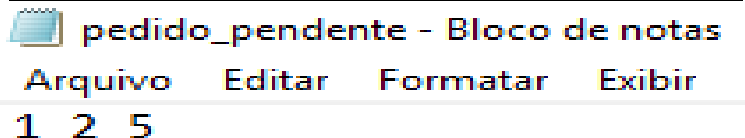
Entre como cliente e faça o pedido, digitando os números correspondentes ao cardápio e ao final digite ‘f’. Verifique que os números digitados estão no salvos no arquivo.

```
while(fgets(linha,30,pcard)!=NULL)
{
    l++;
}
printf("\n\nDigite os numeros correspondentes ao seu pedido e ao terminar digite (f):\n");
while(scanf("%d",&n) != 0)
{
    if(n<1 || n>1)
    {
        printf("\nPedido invalido %d nao existe no cardapio\n",n);
        printf("\nDigite o numero do pedido correspondente ao cardapio: \n");
        continue;
    }
    else
    {
        pedido = (int*)realloc(pedido,sizeof(int)*i);
        pedido[i-1]= n;
        fprintf(pped,"%d ",pedido[i-1]);
        i++;
    }
}
```



```
*****CARDAPIO*****
1) X Egg bacon
2) Big Moc
3) Final Food
4) Atom
5) Baconese
6) Whoaper
7) X-Tudo
8) X-Bacon
9) Double Cheese

Digite os numeros correspondentes ao seu pedido e ao terminar digite (f):
1
2
5
f
```



```
pedido_pendente - Bloco de notas
Arquivo  Editar  Formatar  Exibir
1 2 5
```

3.2. Testar o acesso por senha

Entre como gerente, verifique a senha no arquivo “senha_gerente.txt” e digite-a quando for pedido. Digitando a senha correta o sistema liberará o acesso ao menu do gerente, caso contrário, será solicitado que digite novamente a senha.

```

FILE *psg=fopen("senha_gerente.txt","r");
if(psg!=NULL)
{
    fscanf(psg,"%s", h.senhacomp);
    printf("Digite sua senha:\n");
    scanf("%s",h.senhadig);
    while(strcmp(h.senhadig,h.senhacomp) != 0)
    {
        printf("\nSenha incorreta, digite novamente:\n");
        scanf("%s",h.senhadig);
        i++;
        if(i>2)
            break;
    }
    while(strcmp(h.senhadig,h.senhacomp) == 0)
    {
        printf("\nDigite (1) para criar usuario;\nDigite (2)

```

Senha correta	Senha incorreta
<pre> Digite sua senha: 123 Digite (1) para criar usuario; Digite (2) para alterar usuario; Digite (3) para adicionar ingrediente no estoque; Digite (4) para alterar senha gerente; Digite (5) para deletar usuario; Digite (6) para exibir pedido; Digite (7) para alterar quantidade do estoque; Digite (8) para criar prato; Digite (9) para sair. Entrada: </pre>	<pre> Digite sua senha: 5678 Senha incorreta, digite novamente: </pre>

Verifique também essa funcionalidade para o funcionário, tendo em vista que também é necessário digitar um *login* válido que pode ser consultado no arquivo “usuarios.txt”.

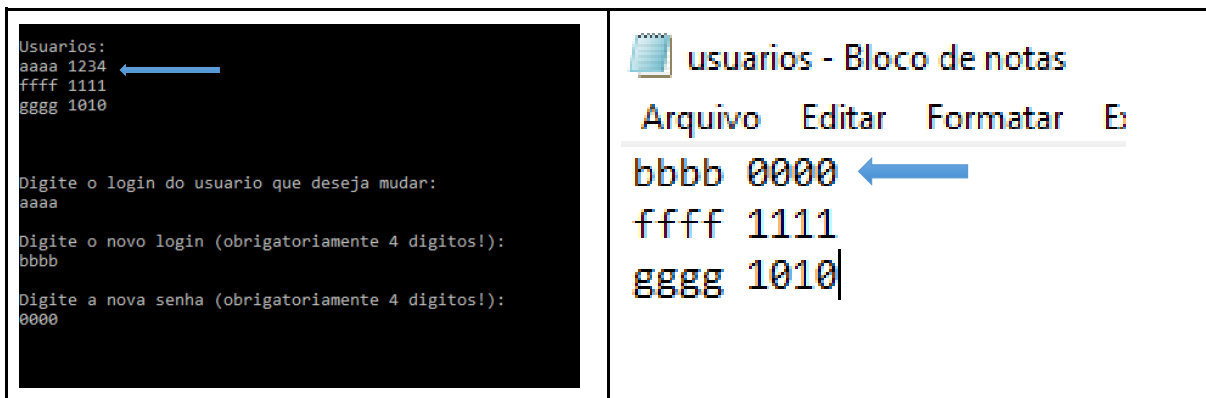
3.3. Testar a alteração de usuário

Entre como gerente e digite a opção para alterar usuário. Digitando um dos *logins* de usuários que foi listado na tela, aparecerá a opção de digitar o novo *login* e senha e será alterado no arquivo.

```

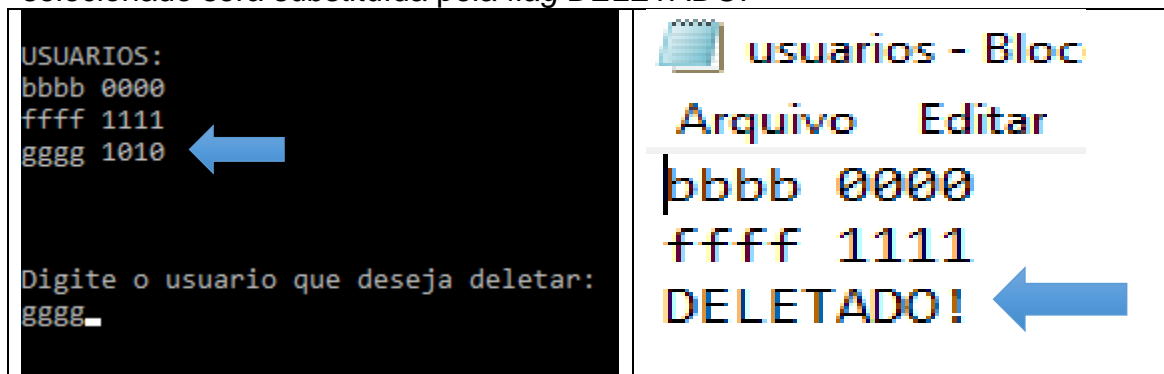
fflush(stdin);
gets(u.logind);
while(fgets(linha,40,pc)!=NULL)
{
    char *token = strtok(linha," ");
    if(strcmp(u.logind,token)== 0)
    {
        printf("\nDigite o novo login (obrigatoriamente 4 digitos!):\n");
        fflush(stdin);
        gets(u.login);
        printf("\nDigite a nova senha (obrigatoriamente 4 digitos!):\n");
        fflush(stdin);
        gets(u.senhad);
        fseek(pc,-11,SEEK_CUR);
        fprintf(pc,"%s %s\n",u.login,u.senhad);
        system("cls");
        printf("Usuario alterado com sucesso!\n");
        fclose(pc);
        break;
    }
}

```



3.4. Testar se o usuário foi deletado

Entre como gerente e digite a opção de deletar usuário. Será solicitado o *login* do usuário que deseja deletar e a linha do arquivo que está o *login* selecionado será substituída pela *flag* DELETADO!



Verifique também que a *flag* não é exibida quando a lista de usuários é impressa na tela.

4. CONCLUSÃO

O programa apresentado executa as funções definidas de maneira fluida e correta, entretanto não foi possível desenvolver funções mais elaboradas, devido à complexidade enfrentada. Ademais, a divisão de módulos por tipo de usuário, os comentários e as funções separadas foram essenciais para garantir a organização e clareza do programa, além de auxiliar na identificação de erros, apesar de que poderia haver mais funcionalidades, como por exemplo a atualização automática do estoque. Um dos principais empecilhos encontrados no desenvolvimento do trabalho foi a busca de determinados dados em arquivo e verificações de permissão (ex: *login* e senha). Para contornar o problema, foi definido tamanhos fixos de caracteres para realizar a busca nos arquivos e conseguir sobrescrevê-los.

5. REFERÊNCIAS BIBLIOGRÁFICAS

5.1. Sites

- Stack Overflow – disponível em: <https://pt.stackoverflow.com/>;
- Cplusplus – disponível em: <http://www.cplusplus.com/>.

5.2. Aulas

Foi consultado as aulas da Professora Camila Laranjeira.