***Variavéis globais:***

-**restaurante[17][41]**: Variável que recebe todos os caracteres da planta do restaurante.

-**NumMesa**: Recebe o numero da mesa escolida para colocar os clientes caso a mesa tenha sido escolhida pelo sistema(autumaticamente).

-**Cont\_Salao, Cont\_Varanda**: Variaveis para contar quantas mesas há na área do salão e da varanda, respectivamente. Como precisava diminuir o contador na hora de esvaziar a mesa, dentro da função, ambas as variáveis foram declaradas como globais para conseguir ser manipuladas.

-**Mesa\_Salao[13], Mesa\_Varanda[5]:** Ambas as variáveis com uma unidade a mais, para poder receber o índice 1 na ultima posição, para executar a função checkmesaok, que será explicada melhor mais para frente.

- **tempo\_estadia[16]**: Variável feita para receber o tempo que cada cliente permanece em cada mesa, necessida ser global para ser acessada de qualquer função.

***Funções :***

**Sleep(int ms)** : Função que freeza o programa por um tempo , em microsegundos, determinado pelo programador.

**Clear()**: Função que limpa a tela.

**Ler\_arquivo()**: Função que abre o arquivo txt da matriz, e executa um laço para ler linha por linha da matriz.

**Mostrar\_arquivo()**: Função que printa os caracteres que foram salvos na variável global restaurante[17][41].

**Mostrar\_arquivoAnimacao():** Printa a matriz sem a mensagem de local ocupado (Caso tenha).

**EscolherMSalao(Char Continuar2, int Client)**: Função percorre todas as mesas do salão "Mesa\_Salao[13]" (12 mesas, a última é exclusiva da função checkarmesaok) e, dependendo da escolha do cliente (Continuar2), referente a pergunta "Deseja escolher a mesa?", o sistema encaminha o(s) cliente(s) para a primeira mesa vazia, ou entao vai direto para a mes escolhida, que com certeza é valida pois ja passaram pela função checkmesaok.

**EscolherMVaranda(char Continuar2, int Client)**: análogo para mesa da varanda.

**Animacao\_desfilar():** Esta parte da animação foi feita com o boneco andando para frente, simplesmente substituindo um caracter ' ' por 'O' e limpando o anterior voltando para ' ', printando na tela e limpando depois, criando a ilusão do boneco estar andando.

**Animacao\_mesa\_varanda()**: O boneco se descola para baixo até para em frente ao corredor da varanda. Realizado com o mesmo esquema da função de cima.

**Animacao\_mesa\_salao (int mesa)**: Esta animação depende para qual mesa o boneco irá, caso seja para as 4 últimas mesas, o contador da linha incremente 1 para o boneco começar na linha de baixo, depois executa um laço com 2 passos, pois só falta o boneco se movimentar mais 2 vezes para chegar na frente do corredor, utilizando a mesma mecânica da animação de cima. Caso o boneco vá para mesas menores que 9, o contador é decrementado em 1 unidade para começar na posição de cima, pois vai subir. Executando um laço j vezes que foi estabelecido como 6, fazendo com que suba até o primeiro corredor, porém o valor de j recebe 2 caso vá para as mesas maiores que 3 necessitando de subir apenas mais 2 vezes, e novamente com a mesma mecânica da animação anterior.

**Animacao\_mesa\_varanda\_corredor(int mesa)**: Nesa função o boneco já começa na entrada da varanda e caminha até o final, que dão o total de 13 passos, porém determinei uma fórmula matematica "if( i == 12 - (mesa \* 3) )" pois cada ponta da mesa se distancia em 3 unidades, e o valor de i atigir a condição, o laço é interrompido pois o boneco estará na ponta da mesa. A animação novamente executada com a mesma mecânica.

**Animacao\_mesa\_salao\_corredor(int mesa)**: Mesma mecanida da animação void animacao\_mesa\_salao e void animacao\_mesa\_varanda\_corredor, pois inicializo a posição da linha do boneco de acordo com sua mesa, e o único diferencial é que dependendo da mesa, se for entre 9 e 12, eu decremento 8 unidades para a mesa valer entre 1 e 4 ou caso a mesa for entre 5 e 8 decremento 4 unidades para também valer um número entre 1 e 4 e poder executar a condição de parada "if( i == 12 - (mesa \* 3) )".

**CheckLotado(int Cont\_Salao, int Cont\_Varanda )**: checka se algum lugar está lotado de acordo com as variáveis globais Cont\_Salao, Cont\_Varanda.

**ValidarTipoMesa(int checkM, char local)**: Checka inicialmente se, de a cordo com o retorno da função anterior, se o local escolhido está ocupado ou nao.

**ZerarMesas(int Mesa\_Salao[],int Mesa\_Varanda[])**: Inicializa todas as mesas como 0 para nao pegar lixo de memória, inicializa tempo\_estadia[i] com um tempo máximo para nunca o tempo ser maior que o atual, para nunca descupar caso nao tenha gente e, para a função checkarmesaok, inicializei as ultimas posições dos vetores globais Mesa\_Varanda[4] = Mesa\_Salao[12] = 1, no qual explicarei na função.

**CompContinuar(char Continuar):** incia um loop que só acaba quando o usuário digitar um caracter que seja S , s , N ou N, e retorna ele maísculo para facilitar nos if's.

**CompLocal(char Local)**: Mesmo esquema da anterior só que para S, s , V, v.

**CompClient(int Client)**: Entra em loop até o usuario digitar um numero entre 1 e 4.

**CompMesa(int NumMesa , char Local)**: Entra em loop até o usário digitar uma mesa que seja entre 1 a 12 (Caso seja salão) e 1 a 4 (Varanda).

**Menu()**: Printa na tela a mensagem do menu.

**Chamadamesa()**: Printa na tela informando a quantidade máxima de pessoas por mesa.

**Chamadalocal()**: Printa na tela informando os locais que o cliente pode escolher , apenas digitar (S/V), S para Salão e V para varanda.

**ColocarmesaS(int mesa, int pessoas)**: Substitui a mesa pelas pessoas, seto tudo como \* , - , - , - , pois no mínimo vai ter uma pessoa que irá sentar, e de acordo com o numero de pessoas, troco os demais “-“ por “\*”. A fórmula matemática para alocar os caracteres foi obtida a partir de uma análise prévia de todas as mesa e possibildades de clientes e, observei que a cada corredor que avançava, a linha se somava 4, e foi daí que tirei a fórmula da linha "i + (4 \* (aux / 4))", no qual se soma 4 a cada vez que que avança de corredor, (divisão de 4 , 5 , 6 , 7 por 4 da 1 [Segundo corredor], e por 8 , 9 , 10 , 11 da 2 [Terceiro Corredor]). Já para a coluna, se soma 3 a vada vez que anda de mesa, executando a mecânica igual da "void animacao\_mesa\_salao\_corredor(int mesa)", no qual se a mesa for entre 8 e 11, eu decremento 8 unidades para a mesa valer entre 0 e 3 ou caso a mesa for entre 4 e 9 decremento 4 unidades para também valer um número entre 0 e 3 e poder executar a condição "17 + (mesa \* 3)".

**ColocarmesaV(int mesa, int pessoas)**: Mesma mecânica da anterior porém só incrementava a coluna pois só tinha um corredor.

**Salaocheio()**: Printa na tela cas o salão esteja cheio.

**Varandacheia()**: Análogo para varanda.

**MenuEscolhaMesa\_Varanda()**: Enumera as mesas para facilitar a escolha do cliente em qual mesa da varanda sentar.

**MenuEscolhaMesa\_Salao()**: Análogo para o salão.

**LimparMenuEscola()**: Após enumerar as mesas para a escolha do salão, esta função volta a matriz como era originalmente preenchendo os corredores com ' '.

**Chamadaopcaomesa()**: Printa na tela perguntando se o cliente deseja escolher mesa.

**Checkmesaok(int NumMesa, char Local()**: Parte mais abstrata, nesta função checa se a mesa escolhida está dentro do intervalo de mesas existentes e se a mesa escolhida está ocupada ou nao. A checagem começa se a mesa escolhida está vazia ou nao, se nao, a função primeiramente printa que a mesa está ocupada, (Obs: o primeiro numero que entrará na função para ver se a mesa está vazia ou nao é orbigatoriamente uma mesa existente pois ja veio corrigida da função "CompMesa", portanto nao pega lixo de memória e nao precisa printar inicialmente que a mesa é inexistente). Após printar que a mesa está ocupada, scaneia uma nova mesa, caso seja inexistente entra no segundo if, setando a flag para 0 para nao printar que está ocupada novamente (Pois nao existe) e troco a mesa para 13, pois no final de tudo será decrementada, se tornando a mesa 12, declarada lá no começo do programa como 1 e por ser uma mesa que acessa uma posição de memória válida, nunca pegará lixo de memória e também não sairá do while pois 100% de certeza esta mesa nao tem o numero 0 pois setamos como 1. A função só acabará com o while quand o cliente escolher uma mesa existente e que esteja vazia (Sempre que entrar nessa função é porque há pelo menos 1 mesa disponível, pois caso contrário a função "ValidarTipoMesa" nao deixa que o processo continue).

**Desocupar(int NumMesa, char Local)**: Esta função realiza 2 operações, primenramente ela seta o tempo de permanencia da ultima mesa ocupada, e depois anda por todas as mesas e checka se o tempo atual é maior que o tempo setado para a mesa permanecer ocupada, caso for verdade, printa na tela a mensagem de desocupação e depois executa a desocupação, que acontece com a mesma mecânica da formula matemática da função "colocarmesaS".