

ESTRUTURA DE DADOS

A ESTRUTURA DE DADOS – FILA

Olá!

Nesta aula, você irá:

1. Conceituar a estrutura de dados Fila simples e Fila Circular;
2. Representar a estrutura de dados Fila por contiguidade (Fila simples);
3. Compreender e implementar as operações com Fila simples;
4. Representar a estrutura de dados Fila por contiguidade (Fila circular);
5. Compreender e implementar as operações com Fila circular.

1 Tema: Organizando a Fila para agilizar

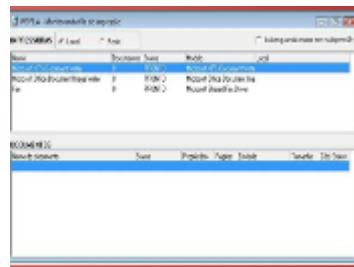
À medida que vamos conhecendo novas estruturas de dados, passamos a contar com maiores recursos para elaborarmos nossos algoritmos mais próximos do ideal. Sei que no início pode parecer que estas estruturas de dados, que estamos estudando, não são necessárias, mas talvez porque nunca tenhamos construído um grande sistema ou então, nossos sistemas estão restritos a determinadas áreas onde o "velho triângulo: entrar-armazenar-consultar" é a base do sistema que pode até prescindir destas estruturas. Então, vou tentar, de forma bem primária, mostrar que as estruturas de dados estão presentes em várias situações que você já vivenciou e que agora está formalizando os conceitos.

Siga o raciocínio a seguir:

- 1) Imagine uma turma em uma sala, fazendo prova final no computador.



2) Todas as provas deverão ser impressas e todos os micros estão compartilhando uma impressora que se encontra na sala. Somente quando o aluno finaliza a prova, ela é corrigida e enviada para impressão, entrando em uma -> FILA.



3) A nota é lançada automaticamente para que o professor possa visualizá-la, conferi-la e clicar em um botão para confirmar, sendo então chamada a função verificaSituacao que retorna a situação final do aluno. Quando a função é chamada, o endereço de retorno é colocado em uma -> PILHA.



4) Quando o professor repete o procedimento do item três para todos os alunos, solicita que seja gerado um relatório que constará o nome dos aprovados, caracterizando uma -> LISTA



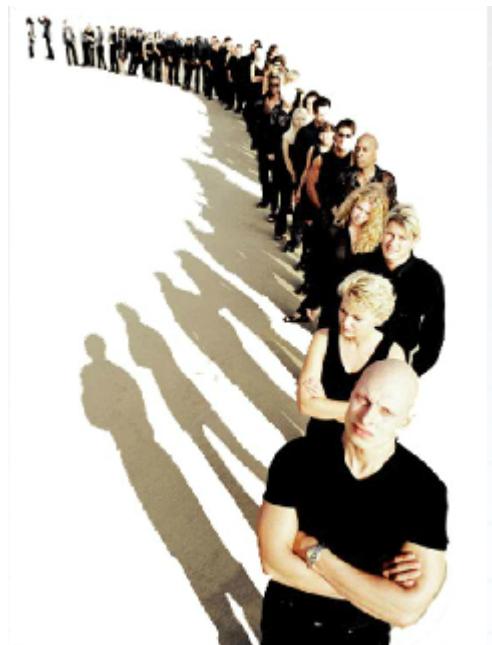
"Uma fila é um tipo especial de Lista Linear em que as inserções são realizadas num extremo, ficando as remoções restritas ao outro. O extremo onde os elementos são inseridos é denominado final da fila, e aquele de onde são removidos é denominado começo da fila." (PEREIRA, Silvio. L., 2004, p. 57)

A Fila, assim como a Pilha, também é um caso particular de Lista Linear e se diferencia dela pela forma de acesso porque a inserção sempre acontece em uma extremidade e a remoção, em outra.

O conceito da *Fila* na programação tem a mesma filosofia que a *fila* do mundo real onde o primeiro que chega é o primeiro a ser atendido.

Lembre-se da primeira aula quando Sr. Augusto chegou cedo ao aeroporto e, como foi o primeiro da *Fila*, foi o primeiro a sair.

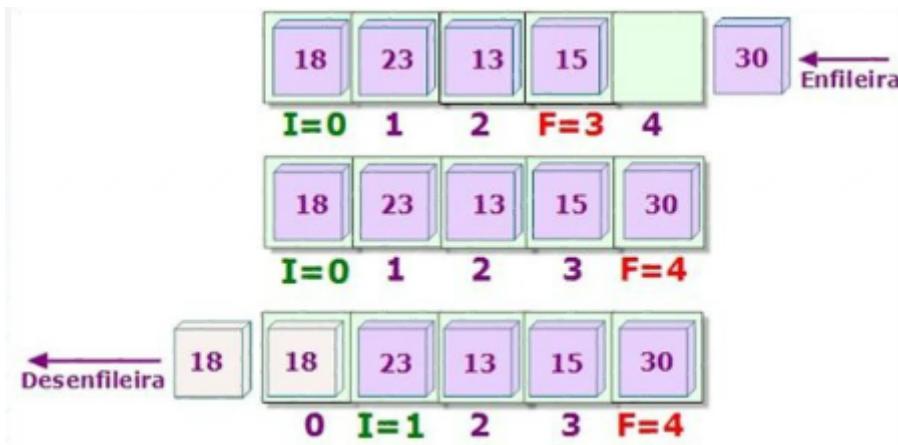
Este método é conhecido como First In, First Out(FIFO).



2 Fila simples

A fila, em inglês queue, possui duas funções básicas: **enqueue(enfileira)** de **dequeue(desenfileira)**.

Observe a figura abaixo.



A operação enqueue só poderá ser realizada se a fila não estiver cheia assim como a operação dequeue, se não estiver vazia. Percebe-se então que, pelo menos, mais duas operações serão necessárias.

Atenção

Overflow na fila – Quando se deseja inserir um elemento, mas a fila está cheia. Trecho de proteção evita este problema.

Underflow na fila - Quando se deseja remover um elemento, mas a fila está vazia. Trecho de proteção evita este problema.

As filas possuem uma operação semelhante a de uma pilha. A seguir, apresentaremos os nomes pelos quais elas são reconhecidas, porém, você pode usar outra nomenclatura para as funções que criar em seus programas:

Inicializa() ou Init() -> Inicializa os indicadores de início e fim da Fila

Enfileira() ou Enqueue() -> Insere elemento no final da Fila

Desenfileira() ou Dequeue() -> Remove o elemento do início da Fila

elemPrimeiro () ou firstElem() -> Retorna o primeiro elemento sem retirá-lo da Fila

verificaFilaCheia() ou isFull() -> Verifica se Fila está cheia

verificaFilaVazia() ou isEmpty() -> Verifica se Fila está vazia

3 Implementação

A implementação de filas pode ser feita por arrays (matrizes) ou ponteiros, mas, nesta aula, porque ainda não tivemos a oportunidade de estudarmos ponteiros, implementaremos com matrizes e structs.

A vantagem de começarmos usando matriz é sua fácil implementação, mas temos que ter consciência das desvantagens, uma vez que, na alocação estática, precisaremos definir um espaço suficientemente grande para armazenar um grande volume de dados. Além disso, indicadores de início e fim da fila.

Os indicadores serão variáveis que sinalizarão o estado da Fila em relação à possibilidade, ou não, da inserção de novos elementos.

O indicador de fim de Fila será incrementado toda vez que um elemento é inserido na Fila, enquanto que o indicador de início de Fila será incrementado quando um elemento for removido, sendo este um grande problema porque, após sucessivas remoções, teremos uma Fila com vários nodos disponíveis no início, mas que não poderão ser ocupados.

Sei que você poderia pensar que poderia se fazer um deslocamento para esquerda de todos os elementos, mas já pensou como seda isso se a Fila fosse enorme?



Este problema será resolvido na aula de hoje quando conseguirmos visualizar o array de forma circular, mas vamos dar um passo de cada vez porque precisamos primeiro entender as operações básicas da Fila.

Nós vamos construir trechos para uma Fila que poderá armazenar até 5 números inteiros.

Estarei apresentando, quando existir, o trecho que chama a função e a função.

Antes de apresentar as operações básicas, gostaria de mostrar a variável estrutura que foi criada para trabalhar com a Fila.

Como no estudo da estrutura Pilha usei uma matriz e uma variável simples, achei que seria mais produtivo implementar com variável estrutura que apresento abaixo.

```
struct queue
{
    float f[TAM];
    int inicio,fim;
};
```

4 Inicializar

A inicialização da Fila irá se resumir a comandos de atribuição, preparando-a para receber os elementos que serão inseridos. Na Fila, os dois extremos são importantes e, por esta razão, estaremos trabalhando com duas variáveis. Sendo assim, para inicializar uma Fila, precisaremos atribuir valores a essas duas variáveis. Não se preocupe porque não inicializarei da mesma forma nos exemplos para que você não ache que só existe uma maneira. Assim como na Pilha, inicializar não significa zerar todos os elementos da Fila.

Neste exemplo, fila.fim foi inicializada com -1 porque é comum você encontrar assim com a justificativa que em um primeiro momento fila.inicio seria igual a fila.fim, mas se você fizer os teste de fila vazia e fila cheia de forma correta, não tem problema começar por 0 as duas varáveis.

```
//Inicializa a fila
fila.inicio = 0;
fila.fim = -1;
```

5 Enfileirar (Enqueue)

Esta é uma operação básica da *Fila*. Para que possa ser enfileirado um elemento, é preciso testar se existe lugar na *Fila*.

Verificar se a *Fila* está cheia poderia ser feito em separado, mas, assim como fiz na Pilha, resolvi fazer neste trecho.

```
enfileira(fila);
```

```
void enfileira(queue &fil)
{
    float valor;
    if (fil.fim == TAM - 1) // testando se a fila cheia está cheia
        cout<<"\nATENCAO. Fila Cheia\n";
    else
    {
        cout<<"Digite o valor a ser enfileirado: ";
        cin>>valor;
        fil.fim++;           //atualiza o final da fila
        fil.f[fil.fim] = valor; //guarda o valor no final da fila
    }
}
```

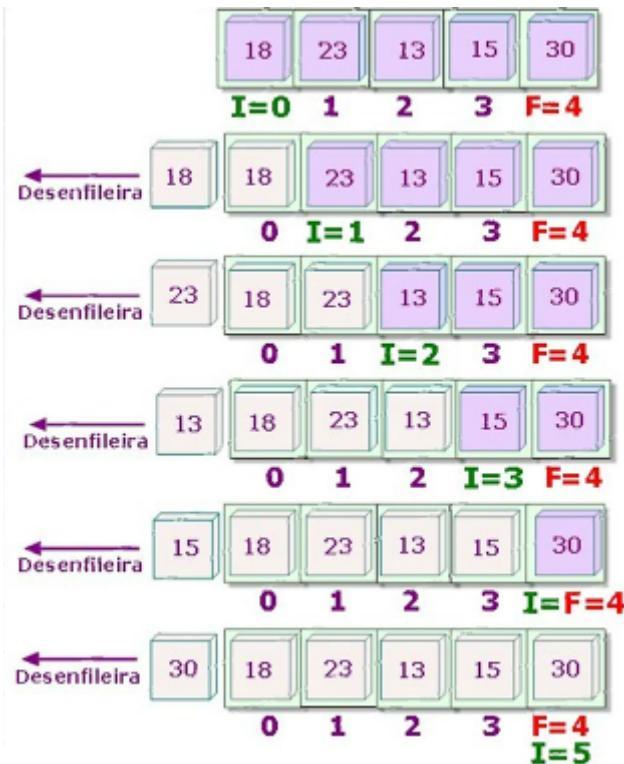
Atenção

A função recebe, por referência, o endereço da variável estrutura e como todas as variáveis necessárias, inclusive o vetor, são membros das estruturas, tudo é passado de uma só vez, aumentando a legibilidade.

Começa testando se a Fila está cheia e, se não estiver, o valor é solicitado, *fil.fim* é incrementado e o valor armazenado na Fila.

É uma função de fácil implementação.

Acompanhe agora, na figura ao lado, supondo que você digitou cinco números para serem enfileirados.



6 Desenfileirar (Dequeue)

Esta é outra operação básica da Fila. Para que possa ser desenfileirado um elemento, é preciso testar se a *Fila* não está vazia.

desenfileira(fila);

```
void desenfileira(queue &fil)
{
    if (fil.inicio > fil.fim) // testando se a fila está vazia
        cout<<"\nATENCAO. Fila Vazia\n";
    else
    {
        cout<<"\nValor Removido: "<<fil.f[fil.inicio]; // exibe o valor "removido"
        fil.inicio++; //atualiza o inicio da fila
    }
}
```

A função recebe, por referência, o endereço da variável estrutura cujo um dos membros contem a posição do elemento a ser “desenfileirado”.

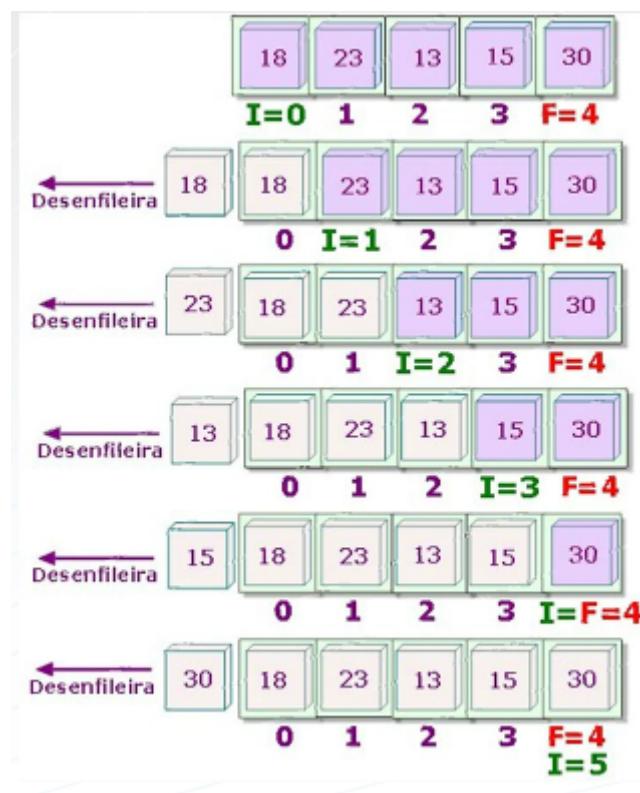
Atenção

Observe que depois de exibir o conteúdo do elemento que era o primeiro da Fila, a variável *fil.inicio* é incrementada.

Em nenhum momento foi removido o valor que estava armazenado na posição. Só o início que se deslocou para a direita.

Assim como a função enfileirar, a função de desenfileirar é de fácil implementação e bem parecida.

Acompanhe a agora, na figura abaixo, supondo que você escolheu desempilhar cinco números.



7 Primeiro elemento (firstElem)

Esta é uma função que possibilita que seja visualizado o primeiro elemento da Fila sem “removê-lo”. Na verdade, você não precisaria criar esta função tal sua simplicidade. Poderia simplesmente colocar no case. Só criei para modularizar todos os casos.

Como todas as seis operações já foram apresentadas, vou acrescentar mais uma. Nada importante, mas que poderá ser útil em algum momento. Assim como a função anterior, você não precisaria criá-la.

elemPrimeiro(fila);

```
void elemPrimeiro(queue &fil)
{
    if(fil.inicio > fil.fim)
        cout<<"\nATENCAO. Fila Vazia\n";
    else
        cout<<"\nElemento do Inicio da Fila: "<<fil.f[fil.inicio];
}
```

situacaoFila(fila);

```
void situacaoFila(queue &fil)
{
    if(fil.inicio > fil.fim)
        cout<<"\nATENCAO. Fila Vazia\n";
    else
    {
        cout<<"\nTotal de elementos na fila: "<<fil.fim - fil.inicio + 1<<"\n";
        cout<<"\n\nEspaco disponivel na fila: "<<TAM-(fil.fim+1)<<"\n";
        cout<<"\n\nPosicao do primeiro elemento da fila: "<<fil.inicio<<"\n";
        cout<<"\n\nPosicao do ultimo elemento da fila: "<<fil.fim<<"\n";
        cout<<"\nPARA FINS DIDATICOS, EXIBINDO A FILA\n";
        cout<<"\nValor(Posicao no Vetor)\n";
        for(int x=fil.inicio; x<=fil.fim; x++)
            cout<<"\n"<<fil.f[x]<<"\n";
    }
}
```

Atenção

Esta função testa se a fila está vazia, se não estiver, exibirá vários dados sobre ela a fim de que você possa acompanhar melhor a execução do programa. Preste bem atenção às funções apresentadas e você verá que elas são muito diferentes das que operam a pilha, afinal, ambas as estruturas são casos particulares da Lista Linear.

Depois de termos visto as chamadas das funções e as funções, vamos analisar o código fonte e acompanhar, rapidamente a execução do programa.

TELA PRINCIPAL

```
C:\Documents and Settings\ANITA LUIZA\Meus... □ X  
FILA< FIFO - First In - First Out >  
  
1- Inserir um valor na fila  
2- Remover um valor da fila  
3- Mostrar o elemento do inicio da fila  
4- Mostrar situacao da fila  
5- Sai  
Opcao:
```



```

1 #include <iostream>
2 #include <cstdlib>
3 #define TAM 5
4 using namespace std;
5 //variavel global
6 struct queue
7 {
8     float f[TAM];
9     int inicio,fim;
10 };
11 void enfileira(queue &fil);
12 void desenfileira(queue &fil);
13 void elemPrimeiro(queue &fil);
14 void situacaoFila(queue &fil);
15 int main()
16 {
17     char resp[10]; int op;
18     queue fila;
19 //Inicializa a fila
20     fila.inicio = 0;
21     fila.fim = -1;
22     do
23     { system("cls");
24         system("color 2f");
25         cout<<"\nFILA( FIFO - First In - First Out )\n\n";
26         cout<<"\n1- Inserir um valor na fila";
27         cout<<"\n2- Remover um valor da fila";
28         cout<<"\n3- Mostrar o elemento do inicio da fila";
29         cout<<"\n4- Mostrar situacao da fila";
30         cout<<"\n5- Sai";
31         cout<<"\nOpcão: ";
32         cin>>resp;op=atoi(resp);
33         system("cls");
34         switch(op)
35         {
36             case 1: enfileira(fila);
37                     break;
38
39             case 2: desenfileira(fila);
40                     break;
41
42             case 3: elemPrimeiro(fila);
43                     break;
44
45             case 4: situacaoFila(fila);
46                     break;
47
48             case 5: cout<<"\nPrograma basico da FILA\n";
49                     break;
50
51             default: cout<<"\nOPCAO INVALIDA\n";
52         }
53         cout<<"\n\n";system("pause");
54     }while(op!=5);
55 }
56
57 void enfileira(queue &fil)
58 {
59     float valor;
60     if (fil.fim == TAM - 1) // testando se a fila cheia está cheia
61         cout<<"\nATENCAO. Fila Cheia\n";
62     else
63     {
64         cout<<"Digite o valor a ser enfileirado: ";
65         cin>>valor;
66         fil.fim++;           //atualiza o final da fila
67         fil.fil.fim = valor; //guarda o valor no final da fila

```

```

68 }
69 }
70
71 void desenfileira(queue &fil)
72 {
73     if (fil.inicio > fil.fim) // testando se a fila está vazia
74         cout<<"\nATENCAO. Fila Vazia\n";
75     else
76     {
77         cout<<"\nValor Removido: "<<fil.f[fil.inicio]; // exibe o valor "removido"
78         fil.inicio++; //atualiza o inicio da fila
79     }
80 }
81
82 void elemPrimeiro(queue &fil)
83 {
84     if(fil.inicio > fil.fim)
85         cout<<"\nATENCAO. Fila Vazia\n";
86     else
87         cout<<"\nElemento do Inicio da Fila: "<<fil.f[fil.inicio];
88 }
89
90 void situacaoFila(queue &fil)
91 {
92     if(fil.inicio > fil.fim)
93         cout<<"\nATENCAO. Fila Vazia\n";
94     else
95     {
96         cout<<"\nTotal de elementos na fila: "<<fil.fim - fil.inicio+1<<"\n";
97         cout<<"\n\nEspaco disponivel na fila: "<<TAM-(fil.fim+1)<<"\n";
98         cout<<"\n\nPosicao do primeiro elemento da fila: "<<fil.inicio<<"\n";
99         cout<<"\n\nPosicao do ultimo elemento da fila: "<<fil.fim<<"\n";
100        cout<<"\nPARA FINS DIDATICOS, EXIBINDO A FILA\n";
101        cout<<"\nValor\tPosicao no Vetor\n";
102        for(int x=fil.inicio; x<=fil.fim; x++)
103            cout<<"\n"<<fil.f[x]<<"\t"<<x;
104    }
105 }

```

```

C:\Documents and Settings\ANITA LUIZA\Meus... Digite o valor a ser enfileirado: 23
Pressione qualquer tecla para continuar. . .

C:\Documents and Settings\ANITA LUIZA\Meus... Elemento do Inicio da Fila: 23
Pressione qualquer tecla para continuar. . .

C:\Documents and Settings\ANITA LUIZA\Meus... Valor Removido: 23
Pressione qualquer tecla para continuar. . .

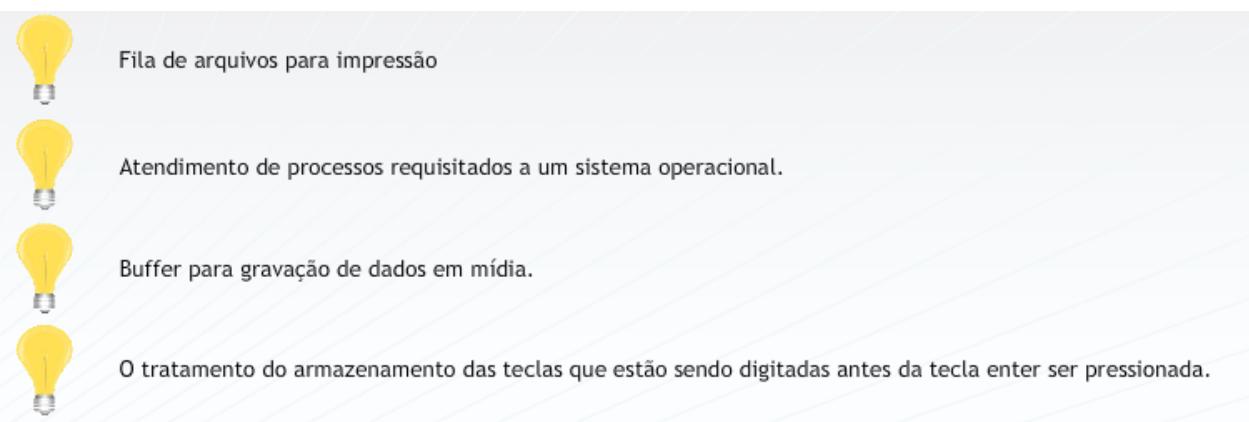
C:\Documents and Settings\ANITA LUIZA\Meus... Total de elementos na fila: 3
Espaco disponivel na fila: 1
Posicao do primeiro elemento da fila: 1
Posicao do ultimo elemento da fila: 3
PARA FINS DIDATICOS, EXIBINDO A FILA
Valor Posicao no Vetor
13 1
62 2
15 3
Pressione qualquer tecla para continuar. . .

```

8 Aplicações com Filas

As Filas têm muitas aplicações e acho que todos concordam que o uso mais comum seja em Sistemas Operacionais.

Alguns exemplos:



Nesta aula, procurei selecionar exemplos interessantes para provar a importância desta estrutura e simplifiquei as soluções para facilitar o entendimento de cada função. Entretanto, anexei os originais para aqueles que têm mais experiência.

9 Uma aplicação - o programa da Agenda

O programa que será apresentado a seguir é uma adaptação do código fonte do livro C Completo e Total de Herbert Schildt, cujo site está indicado no Aprenda Mais desta aula para que você faça o download dos códigos. É um livro clássico e, nesta aula, selecionei dois programas dele.

Evidentemente, algumas modificações foram necessárias porque os códigos estavam escritos na linguagem C e usavam matriz de ponteiros assunto que será estudado na próxima aula.

Para este exemplo, mais uma vez, considerei uma matriz de 5 elementos para que você possa entender o funcionamento da aplicação.

1) A variável estrutura

```
struct queue
{
    char f[5] [255];
    int lpos, rpos;
};
```

Como desejei, nesta aula, trabalhar com variável estrutura, me afastei um pouco do código.

As variáveis lpos e rpos, membros da variável estrutura serão explicadas logo a seguir.

2) Inicialização

```
//inicializa matriz
fila.lpos = 0; // posicao de armazenamento livre
fila.rpos = 0; // posicao do item a ser recuperado
```

3) Inserção de eventos na agenda

```
entra(fila);
```

```
void entra(queue &fil)
{
    char s[255];
    if(fil.lpos==MAX)
    {
        cout<<"\nAtenção. Agenda Cheia\n";
        return;
    }
    cout<<"\nEnter com o "<< fil.lpos+1<<"º evento\n";
    cin.getline(s,255);
    strcpy(fil.f[fil.lpos],s);
    fil.lpos++;
}
```

A função começa testando se a *Fila* está cheia e, caso não esteja, solicita a entrada do evento e copia o evento na Fila, incrementando *fil.lpos* que sinaliza a próxima posição livre.

Não entendi esse return na função void. Será que poderia usar o else e não usar return?

Respondendo: A estrutura do if tem várias etapas na execução e quanto mais rápida for executada, melhor.

Usando o if simples, em caso de verdade, o retorno é rápido e o fluxo do programa é mais natural. Essa era uma preocupação dos programadores antigamente, mas que ressurge. Sempre vale a pena ter um programa eficiente sendo executado em menor tempo.

4) Remoção de eventos da agenda

Sabe-se que o único evento que pode ser “removido” é o que se encontra no início da Fila.

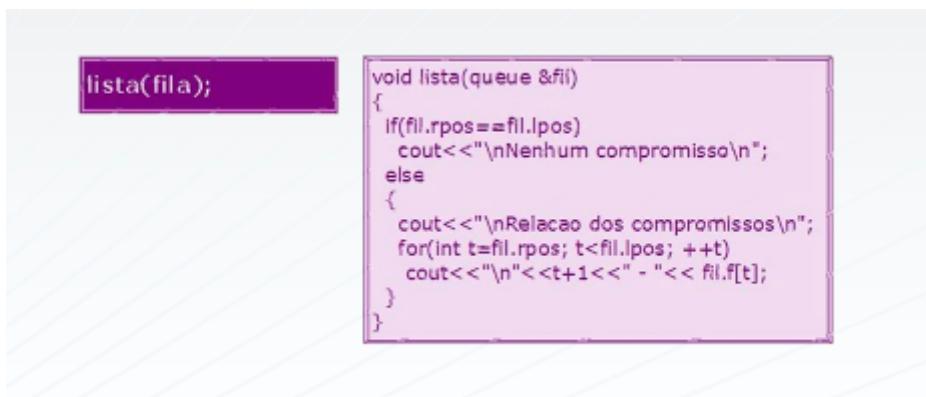
Coloco entre aspas porque não é, de fato, removido como você pode observar na figura que desenfileirou, pois, deixei mais claro os blocos, para enfatizar isso.

```
deleta(fila)
```

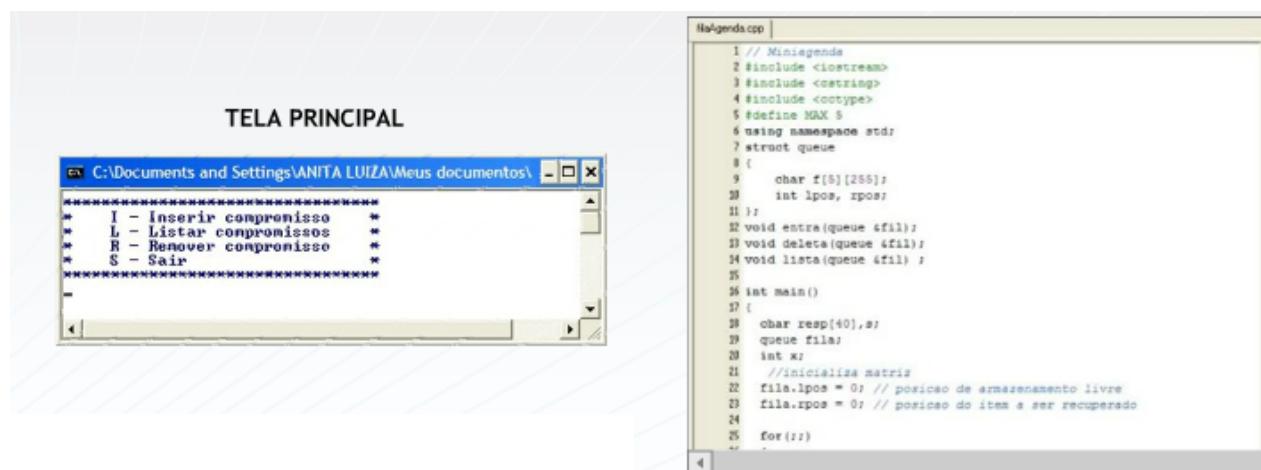
```
void deleta(queue &fil)
{
    if(fil.rpos==fil.lpos)
        cout<<"\nNenhum compromisso\n";
    else
    {
        cout<<"\n"<<fil.f[fil.rpos]<<", cumprido";
        fil.rpos++;
    }
}
```

5) Exibindo os eventos da agenda

Para que você pudesse acompanhar melhor a execução do programa, criei essa função, mas só estou fazendo isso porque a implementação é com vetor e espero lhe ajudar a compreender melhor esta estrutura que, como já disse, tem muitas aplicações.



Depois de termos visto as chamadas das funções e as funções, vamos analisar o código fonte e acompanhar, rapidamente a execução do programa.



10 Acompanhando a execução do programa



Considerações sobre o programa

Este programa da Agenda de Compromissos é uma boa aplicação para Fila, principalmente, para quem está começando.

Ele será adaptado, nesta aula para Fila circular onde será resolvido o problema dos espaços ociosos à medida que os elementos vão sendo “removidos”.

Depois de termos visto as chamadas das funções e as funções, vamos analisar o código fonte e acompanhar, rapidamente a execução do programa.

11 Fila Circular

Para resolvermos o problema do espaço ocioso, o ideal seria que a última posição antecedesse à primeira e, se pegássemos um pedaço de barbante e uníssemos suas pontas sobre uma mesa, perceberíamos que ela se fecharia em uma curva. Pronto. Encontramos a solução: uma **Fila Circular**.

(http://estaciocdocente.websaula.com.br/cursos/gon119/docs/aula07_t16-FILA%20CIRCULAR.pdf).

Atenção

As operações básicas da Fila Circular são as mesmas da Fila Simples, exceto pela implementação do código, porque um vetor circular é algo que acontece no imaginário da programação, não existindo fisicamente na Memória Principal, pois sua organização é linear. Você poderá observar no código que a solução encontrada se baseou em uma variável, que é incrementada cada vez que um elemento é inserido na Fila e decrementada cada vez que um elemento é removido. Desta forma, enquanto esta variável não estiver com o valor máximo, elementos poderão ser inseridos.

1) A variável struct

```
struct queue
{
    float circular[MAX];
    int total, inicio, final;
};
```

2) Inicializar

```
//inicializa matriz
fila.inicio = 0; // posicao do item a ser recuperado
fila.total = 0; //total de elemento inseridos
fila.final=0; // posicao do fim da fila
```

3) Enfileirar (Enqueue)

```
entra(fila);
```

```
void entra(queue &fil)
{
    float valor;
    if(fil.total==MAX)
        cout<<"\nAtenção. Fila Cheia\n";
    else
    {
        cout<<"\nDigite valor: ";
        cin>>valor;
        fil.circular[fil.final]=valor;
        fil.final++;
        if(fil.final==MAX)fil.final=0;
        fil.total++;
    }
}
```

Atenção

A função começa testando a variável fil.total para verificar se seu valor é igual ao máximo.

Caso não seja, o valor é solicitado e inserido na Fila. Depois, a variável fil.final é incrementada e testada. Se atingir o valor final, será reinicializada. Lembre-se de que ela começou com zero.

A variável fil.total é incrementada e, na próxima vez que se tentar inserir um elemento, a mensagem Atencao.

Fila Cheia, irá aparecer até que um elemento seja removido.

4)Desenfileirar (Dequeue)

```
deleta(fila)
```

```
void deleta(queue &fil)
{
    if(fil.qtdadI==0)
    { cout<<"\nNenhum compromisso\n";return; }
    fil.qtdadI--;
    cout<<"\n"<<fil.f[fil.rpos]<<", cumprido";
    fil.rpos++;
    if(fil.rpos==MAX){fil.rpos=0;}//reinicialização
}
```

Esta função também começa testando a variável fil.total para verificar se a Fila está vazia.

A variável fil.inicio é incrementada porque um elemento foi removido e fil.total, decrementada.

Já falei nesta aula, que o trecho de remoção, não estava removendo valor da Fila, mas neste exemplo, resolvi, toda vez que fosse removido um elemento, atribuir o valor -999 para que pudesse, no próximo trecho, listar os elementos da Lista Circular para aqueles que dizem: "Só acredito vendo!"

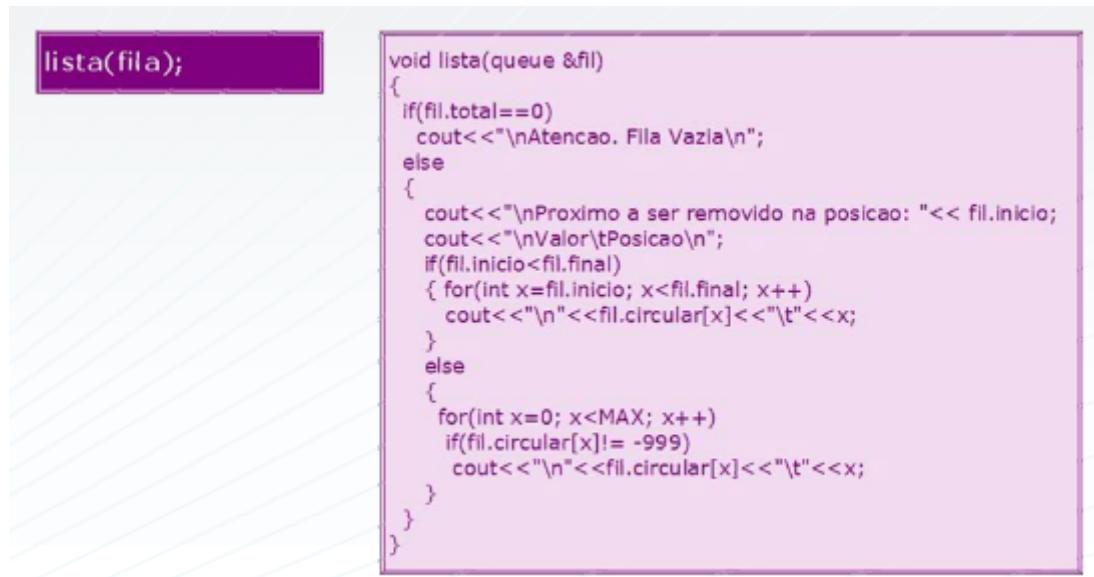
Não é para fazer disso um hábito, mas nunca se sabe se um dia você pode precisar usar algo parecido, não é?

Meu objetivo foi didático.

5)Função lista

Somente a primeira linha de código do else seria necessária para esta função. Todo o conjunto de comando foi acrescido para fins didáticos, visto que fica muito difícil visualizar, num primeiro momento, o comportamento da Fila Circular.

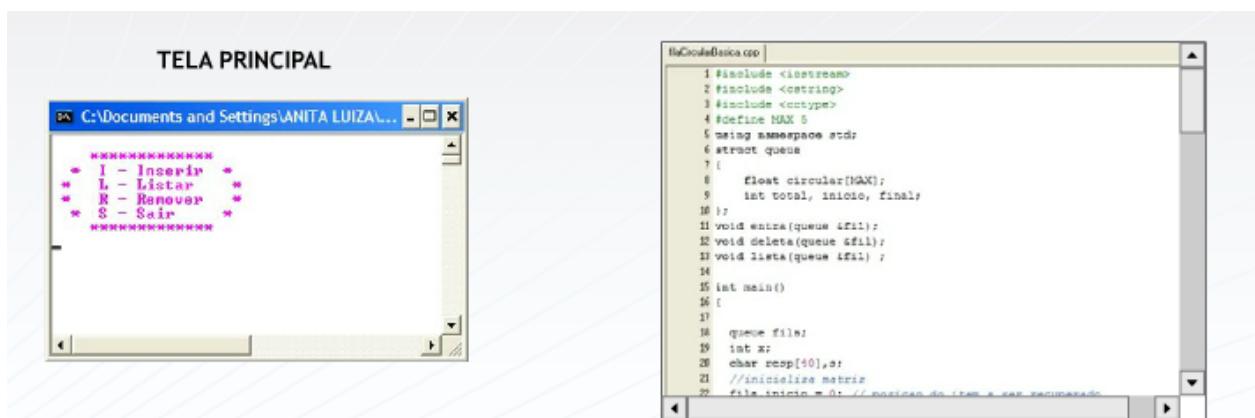
Foi por esta razão que no trecho de remoção que atribui -999 à posição quando era “removido” um elemento.



```
lista(fila);
```

```
void lista(queue &fil)
{
    if(fil.total==0)
        cout<<"\nAtenção. Fila Vazia\n";
    else
    {
        cout<<"\nProximo a ser removido na posicao: "<< fil.inicio;
        cout<<"\nValor\tPosicao\n";
        if(fil.inicio<fil.final)
        { for(int x=fil.inicio; x<fil.final; x++)
            cout<<"\n"<<fil.circular[x]<<"\t"<<x;
        }
        else
        {
            for(int x=0; x<MAX; x++)
                if(fil.circular[x]!= -999)
                    cout<<"\n"<<fil.circular[x]<<"\t"<<x;
        }
    }
}
```

Depois de termos visto as chamadas das funções e as funções, vamos analisar o código fonte e acompanhar, rapidamente a execução do programa.



TELA PRINCIPAL

```
C:\Documents and Settings\ANITA LUIZA... -> x
```

```
*****
* I - Inserir  *
* L - Listar   *
* R - Remover  *
* S - Sair     *
*****
```

```
BaCiculaBasica.cpp
```

```
1 #include <iostream>
2 #include <cstring>
3 #include <ctype.h>
4 #define MAX 5
5 using namespace std;
6 struct queue
7 {
8     float circular[MAX];
9     int total, inicio, final;
10 };
11 void entra(queue &fil);
12 void deleta(queue &fil);
13 void lista(queue &fil);
14
15 int main()
16 {
17
18     queue fila;
19     int x;
20     char resp[50];
21     //inicializa matriz
22     fila.inicio = 0; //posicao do item a ser removido
```

Acompanhando o programa



Uma aplicação - o programa da agenda com programa circular

1) A variável estrutura

```
struct queue
{
    char f[5][255];
    int lpos, rpos, qtdadl;
};
```

2) Inicialização

```
//inicializa matriz
fila.lpos = 0; // posicao de armazenamento livre
fila.rpos = 0; // posicao do item a ser recuperado
fila.qtdadl=0; //total de elementos inseridos
```

3) Inserção de eventos na agenda

entra(fila);

```
void entra(queue &fil)
{
    char s[255];
    if( MAX == fil.qtdadI)
    {
        cout<<"\nAtencao. Agenda Cheia\n";
        return;
    }
    if(fil.lpos==MAX ){ fil.lpos=0;}
    cout<<"\nEnter com o proximo evento\n";
    cin.getline(s,255);
    if(strcmp(s, "\0") == 0) return; // nenhuma entrada
    strcpy(fil.f[fil.lpos],s);
    fil.lpos++;
    fil.qtdadI++;
}
```

4) Remoção da agenda

deleta(fila)

```
void deleta(queue &fil)
{
    if(fil.qtdadI==0)
    { cout<<"\nNenhum compromisso\n";return; }
    fil.qtdadI--;
    cout<<"\n" << fil.f[fil.rpos] << ", cumprido";
    fil.rpos++;
    if(fil.rpos==MAX){fil.rpos=0;}//reinicializaçao
}
```

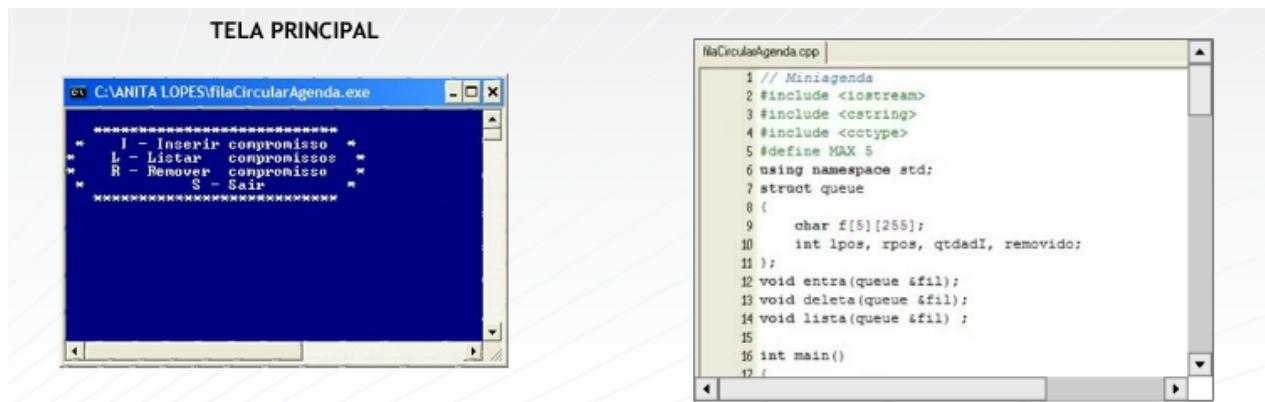
5) Exibindo os eventos na agenda

Esta função só lista o próximo evento na agenda.

lista(fila);

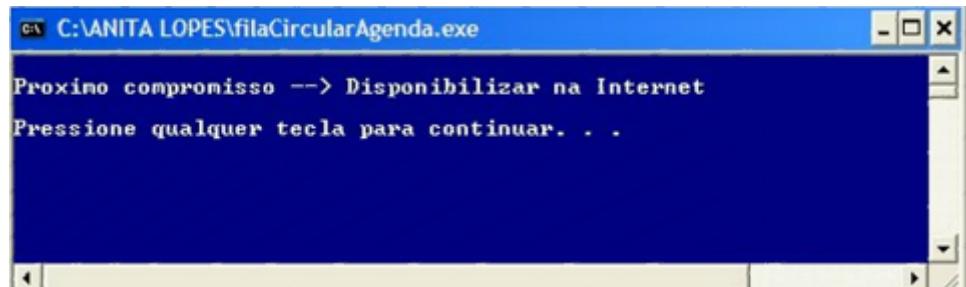
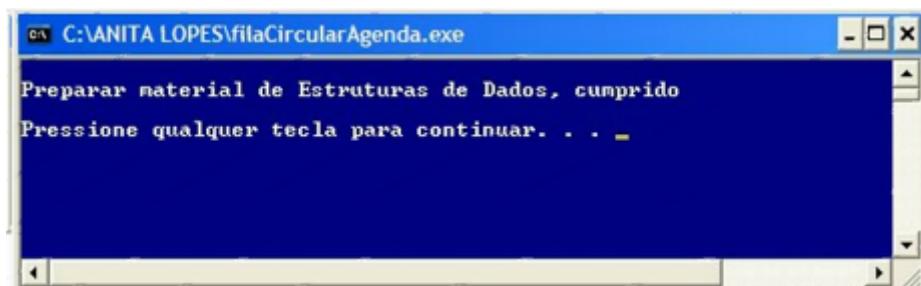
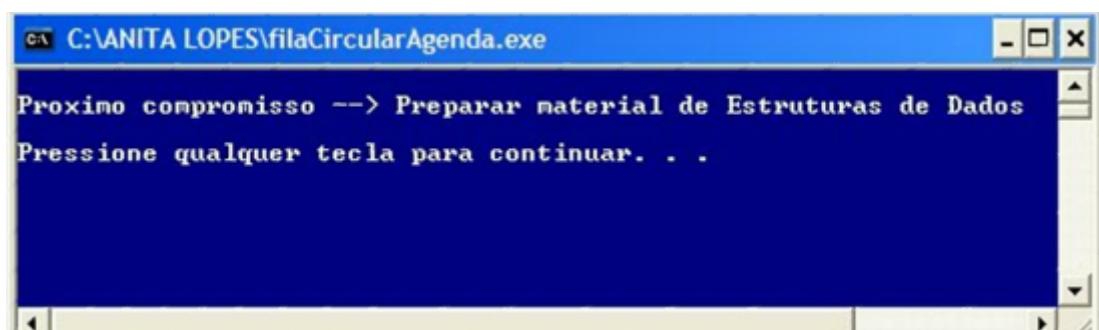
```
void lista(queue &fil)
{
    if(fil.qtdadI==0)
    { cout<<"\nNenhum compromisso\n";return; }
    cout<<"\nProximo compromisso -->" << fil.f[fil.rpos];
}
```

Depois de termos visto as chamadas das funções e as funções, vamos analisar o código fonte e acompanhar, rapidamente a execução do programa.



Acompanhando a execução do programa





Espero que você tenha entendido esta aula porque a Fila é uma estrutura importante. Bem, todas são.

Foi uma aula muito ilustrada porque estou fazendo de tudo para que você se apaixone por Estruturas de Dados.

Sei que não é fácil, mas se você se dedicar bastante, vai conseguir.

Saiba que estamos aqui para lhe ajudar no que for possível e preciso que você se dedique um pouquinho mais porque essas três últimas aulas têm um conteúdo mais complicado, mas tenha certeza que vou tentar, descomplicar.

Até a próxima aula.

Saiba mais



Para saber mais sobre os tópicos estudados nesta aula, pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto. Se ainda tiver alguma dúvida, fale com seu professor on-line utilizando os recursos disponíveis no ambiente de aprendizagem.

Realize os exercícios da Lista_6 e se auto-avalie.

Sugestões de sites:

http://www.ime.unicamp.br/~clovis/simao/fila_p.html

REDES DE COMPUTADORES - FILAS DE PACOTES E PRIORIZAÇÃO

http://www.mhprofessional.com/getpage.php?c=computing_downloads.php&cat=112#A-C

O que vem na próxima aula

Na próxima aula, você estudará os seguintes assuntos:

- Conceituar ponteiro;
- Conceituar os operadores & e *;
- Manipular ponteiro com os operadores & e *;
- Compreender o uso do operador seta;
- Compreender o uso de ponteiro no estudo de Listas Lineares;
- Conceituar Listas Lineares encadeadas;
- Conceituar alocação dinâmica de memória;
- Conceituar listas lineares simplesmente encadeadas;
- Representar listas lineares simplesmente encadeadas;
- Implementar operações com listas lineares simplesmente encadeadas, realizando aplicações.

CONCLUSÃO

Nesta aula, você:

- Compreendeu a importância da Estrutura de Dados Fila simples e circular;
- Compreendeu e usou várias operações que podem ser feitas com a Fila simples e circular;
- Analisou aplicações da Fila e acompanhou a execução de programas.