UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Bauru

Faculdade de Ciências

Disciplina: Estruturas de Dados I – **ED1**
Professora: Simone das Graças Domingues Prado
e-mail: simone.prado@unesp.br

Apostila 02 – Conceito e implementação de Filas

Objetivos:

⇒ Trabalhar com filas estáticas
⇒ Trabalhar com filas dinâmicas
⇒ Conhecer e implementar essas filas

Conteúdo:

1. Introdução
2. Filas estáticas
3. Filas dinâmicas
4. Fila com elementos de tipos diferentes
5. Exercícios

1. Introdução

As filas são estruturas de dados que se comportam como as filas que conhecemos. Nada mais é do que um vetor ou uma lista encadeada que possui restrições de acesso. A restrição é: o primeiro que entra é o primeiro que sai (PEPS, ou FIFO - First Input First Output).

2. Filas Estáticas

Para a implementação de filas estáticas existem várias formas de construção. Aqui serão abordadas algumas dessas formas e outras serão vistas em forma de exercício.

Independente de qual seja a construção da fila, as operações básicas são:

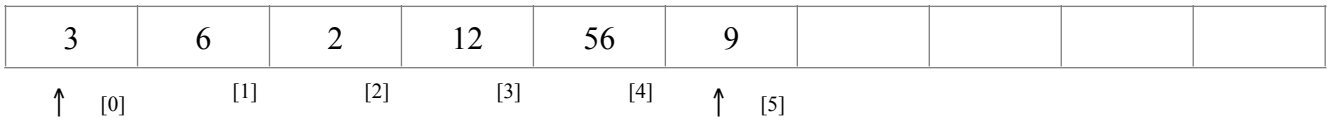
- verificação de fila vazia e fila cheia
- inserção de um elemento na fila, sem ultrapassar o tamanho máximo permitido (enfileira)
- remoção de um elemento da fila, se a fila tiver elementos (desenfileira)

Quando se pensa em uma fila tem-se que sinalizar o início e o final da mesma para saber onde se deve inserir e de onde se deve remover um elemento. Além disso, deve-se escolher o método para a inserção e remoção do elemento.

DEFINIÇÃO 01: uma definição de fila bem simples poderia ser:

```
#define   max  10
typedef   int  def_fila[max];
def_fila  Fila;
int  Inicio, Final;
```

Fila:



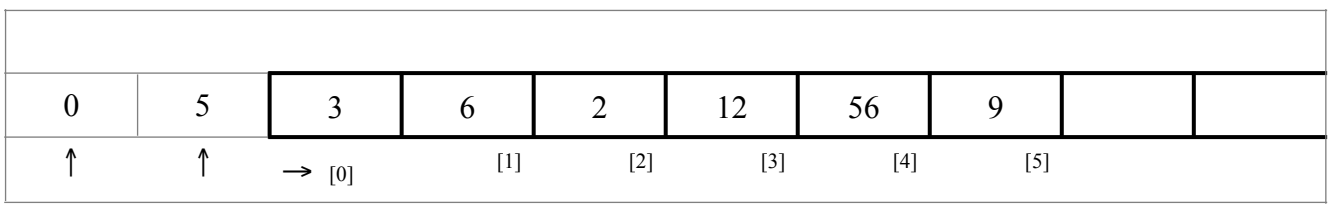
Inicio Final

Início = 0 Final = 5

DEFINIÇÃO 02: uma definição de fila um pouco mais sofisticada poderia ser:

```
#define max  10
typedef struct def_fila{
        int  elemento [max];
        int inicio, final ;}
def_fila Fila;
```

Fila:



inicio final elementos

Os métodos

Considerando a primeira definição de fila mostrada acima, os métodos para a inserção e remoção podem ser pensados como sendo:

MÉTODO 01

Supondo que se tenha a uma fila de números, e essa fila obedeça à primeira definição dada acima, uma representação gráfica poderia ser:

3	6	2	12	56	9				
↑ [0]	[1]	[2]	[3]	[4]	↑ [5]				

Inicio Final

Assim, *Inicio* = 0 e *Final* = 5.

A cada inserção na fila, o elemento é acrescentado no final da fila e por conseqüência o valor da variável *Final* é alterado. A cada remoção na fila, o elemento é retirado do início da fila e a variável *Inicio* é alterada. Por exemplo,

Após uma inserção (elemento = 7), tem-se:

3	6	2	12	56	9	7			
↑ [0]	[1]	[2]	[3]	[4]	[5]	↑ [6]			

Inicio Final

Assim, *Inicio* = 0 e *Final* = 6.

Após um remoção, tem-se:

	6	2	12	56	9	7			
[0]	↑ [1]	[2]	[3]	[4]	[5]	↑ [6]			

Inicio Final

Assim, *Inicio* = 1 e *Final* = 6.

Percebe-se que, após sucessivas inserções e remoções, a variável *Final* atingirá o tamanho máximo do vetor. Isso acarreta a impossibilidade de inserir novos elementos mesmo tendo lugar sobrando no início do vetor. Por outro lado, esse método é muito simples de implementar.

MÉTODO 02

Considere a mesma fila da Definição 01. Agora a remoção da fila implicará no deslocamento de todos os elementos para o inicio da mesma. Veja o exemplo abaixo:

Após uma inserção (elemento = 5):

3	6	2	12	56	9	7	5		
↑ [0]	[1]	[2]	[3]	[4]	[5]	[6]	↑ [7]		

Inicio Final

Assim, *Inicio* = 0 e *Final* = 7.

Após uma remoção:

6	2	12	56	9	7	5			
↑ [0]	[1]	[2]	[3]	[4]	[5]	↑ [6]			

Inicio Final

Assim, *Inicio* = 0 e *Final* = 6.

Percebe-se que o valor de *Inicio* sempre será igual a 0, então não é necessário usá-lo. Aqui se usa todo o vetor, mas tem o problema de deslocamento de todos os elementos a cada remoção, tornando o trabalho mais lento.

MÉTODO 03

Considere a mesma fila da Definição 01. No Método 01, a inserção trazia problema porque poderia não conseguir inserir elementos mesmo tendo lugares para isso. No método 02 isso foi resolvido usando deslocamento a cada remoção, só que isso é caro computacionalmente.

No Método 03, o deslocamento será feito só se chegar na situação de não ser possível inserir um elemento por já estar no final do vetor. Assim, é adiado o deslocamento, e quando o fizer será para movimentar todos de uma vez para o início da fila.

Veja o exemplo abaixo:

Após uma inserção (elemento = 1):

3	6	2	12	56	9	7	5	1	
↑ [0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	↑ [8]	

Inicio Final

Assim, *Inicio* = 0 e *Final* = 8.

Após uma remoção:

	6	2	12	56	9	7	5	1	
	↑ [1]	[2]	[3]	[4]	[5]	[6]	[7]	↑ [8]	
	Inicio							Final	

Assim, *Inicio* = 1 e *Final* = 8.

Após outra remoção:

		2	12	56	9	7	5	1	
		↑ [2]	[3]	[4]	[5]	[6]	[7]	↑ [8]	
		Inicio						Final	

Assim, *Inicio* = 2 e *Final* = 8.

Após uma inserção (elemento = 45):

		2	12	56	9	7	5	1	45
		↑ [2]	[3]	[4]	[5]	[6]	[7]	[8]	↑ [9]
		Inicio							Final

Assim, *Inicio* = 2 e *Final* = 9

Após outra inserção (elemento = 10):

2	12	56	9	7	5	1	45	10	
↑ [0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	↑ [8]	
Inicio								Final	

Assim, *Inicio* = 0 e *Final* = 8

Assim são aproveitados todos os espaços e são feitos menos deslocamentos durante o processo.

MÉTODO 04

Outra forma de resolver é usar a idéia de um "vetor circular", ou seja, toda vez que tentar inserir e se estiver na última posição da fila, recomeça a inserção no inicio da fila (se houver posição vazia, claro!).

Veja exemplo abaixo:

Após uma inserção (elemento = 1):

3	6	2	12	56	9	7	5	1	
↑ [0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	↑ [8]	

Inicio Final

Assim, *Inicio* = 0 e *Final* = 8.

Após uma remoção:

	6	2	12	56	9	7	5	1	
	↑ [1]	[2]	[3]	[4]	[5]	[6]	[7]	↑ [8]	

Inicio Final

Assim, *Inicio* = 1 e *Final* = 8.

Após outra remoção:

		2	12	56	9	7	5	1	
		↑ [2]	[3]	[4]	[5]	[6]	[7]	↑ [8]	

Inicio Final

Assim, *Inicio* = 2 e *Final* = 8.

Após uma inserção (elemento = 45):

		2	12	56	9	7	5	1	45
		↑ [2]	[3]	[4]	[5]	[6]	[7]	[8]	↑ [9]

Inicio Final

Assim, *Inicio* = 2 e *Final* = 9

Após outra inserção (elemento = 10):

10		2	12	56	9	7	5	1	45
↑ [0]		↑ [2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Final Inicio

Assim, *Inicio* = 2 e *Final* = 0

3. Filas Dinâmicas

As filas dinâmicas são aquelas em que serão usadas estruturas dinâmicas de armazenamento: os ponteiros.

Fila usando uma Lista Linear Simplesmente Encadeada

```
struct no {
            int dado;
            struct no* prox;};
typedef struct no* def_fila;
```

Serão declarados dois ponteiros para a FILA: um que indicará o inicio da FILA (Inicio) e outro que indicará o final da FILA (Final):

```
def_fila Inicio=NULL, Final=NULL;
```

Rotinas:

1. Inserção de um novo elemento será no final da FILA:

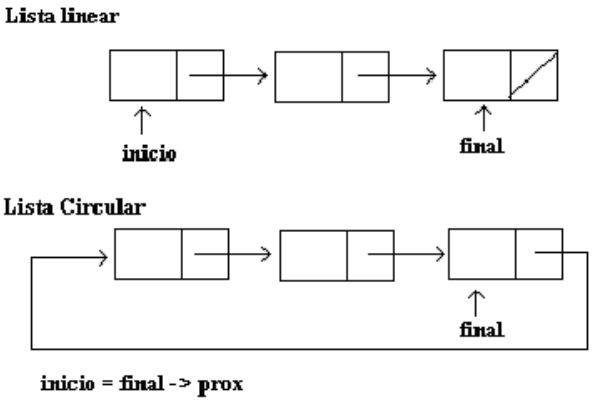
```
void enfileira(def_fila* Inicio, def_fila* Final, int numero){
     def_fila q = (def_fila)malloc(sizeof(struct no));
     q->dado=numero;
     q->prox=NULL;
     if (*Final != NULL) (*Final)->prox=q;
     else *Inicio = q;
     *Final=q;
}
```

2. Remoção de um elemento será do inicio da FILA:

```
int desenfileira (def_fila* Inicio,def_fila* Final, int* numero){
    def_fila q;
    if (*Inicio == NULL) return 0;
    q = *Inicio;
    *numero = q->dado;
    *Inicio = (*Inicio)->prox;
    if (*Inicio == NULL) *Final=NULL;
    free(q);
    return 1;
}
```

Fila usando uma Lista Circular Simplesmente Encadeada

Ao invés de usar uma lista linear pode-se usar uma lista circular para implementar uma fila. Com isso não será preciso dois ponteiros: um para apontar para o início e outro para o fim da fila. Basta ter um ponteiro: o que aponta para o final da fila. O início será o próximo nó apontado pelo ponteiro final. Veja o desenho:



Definição da fila:

```
typedef struct no {
                  int dado;
                  struct no* prox;
                  }* def_fila;
```

Será declarado um único ponteiro para a FILA: ele indicará o final da FILA (Final). Assim, para saber o início da fila, basta encontrar o proximo elemento da variável Final;

```
def_fila Final=NULL;
```

Rotinas:

1. Inserção de um novo elemento da fila será feito no final da FILA

```
void enfileira (def_fila* Final, int numero){
     def_fila q = (def_fila)malloc(sizeof(struct no));
     q->dado = numero;
     if (vazia(*Final)) (*Final) = q;
     else q->prox = (*Final)->prox;
     (*Final)->prox = q;
     *Final=q;
}
```

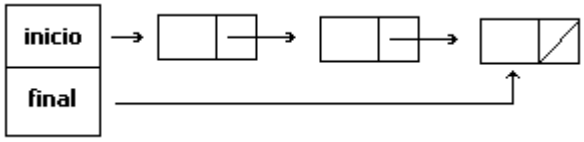
2. Remoção de um elemento da fila será feito do inicio da FILA

```
int desenfileira (def_fila* Final, int* numero){
    def_fila q;
    if (vazia(*Final)) return 0;
    q = (*Final)->prox;
    *numero = q->dado;
    if (q == *Final) *Final=NULL;
    else (*Final)->prox = q->prox;
    free(q);
    return 1;
}
```

Fila usando uma Lista Linear Simplesmente Encadeada com Descritor

No descritor colocam-se os ponteiros para o início da fila e um para o final da fila. Veja como ficaria.

```
typedef struct no {
                  int dado;
                  struct no* prox;} No;
typedef struct Descritor {
                  No* inicio;
                  No* final;} def_fila;
```



Rotinas

1. Inserção de um novo elemento da fila.

```
void enfileira (def_fila* Fila, int numero){
      No* q = (No*)malloc(sizeof(struct no));
     q->dado=numero;
     q->prox=NULL;
     if (vazia(*Fila)) Fila->inicio = q;
     else Fila->final->prox = q;
     Fila->final=q;
}
```

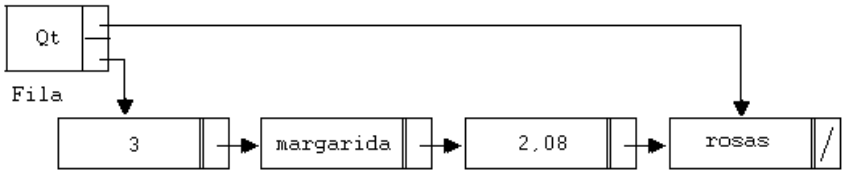
2. Remoção de um elemento da fila

```
int desenfileira (def_fila* Fila, int* numero){
    No* q;
    if (vazia(*Fila)) return 0;
    q = (*Fila).inicio;
    *numero = q->dado;
    if ((*Fila).inicio == (*Fila).final)
            (*Fila).final=NULL;
    (*Fila).inicio = q->prox;
     free(q);
     return 1;
}
```

4. Fila com elementos de tipos diferentes

Uso do union

Normalmente são gerados os nós da fila iguais (do mesmo tipo). Assim as filas ficam com elementos homogêneos. Em algumas aplicações, pode ser necessário que a fila armazene elementos de tipos diferentes, como na figura abaixo:



Para essa implementação pode-se usar o tipo estruturado union da linguagem C para criar o nó da fila. Quando se cria uma união cria-se uma localização de memória que será compartilhada por diferentes variáveis, que podem ser de tipos diferentes. O espaço reservado será do tamanho do maior de seus campos. As uniões são usadas quando se querem armazenar valores heterogêneos num mesmo espaço de memória. A definição de uma união é parecida com a de uma estrutura. Só que na união armazena-se um único elemento por vez.

Veja uma união que pode armazenar um inteiro, uma palavra ou um número real:

```
typedef union {
                int int_info;
                char string_info[20];
                float float_info;
               } no_info;
```

Para criar uma fila com esse tipo de elemento, cria-se o tipo:

```
typedef struct tipo_no {
                  int tipo;
                  no_info info;
                  struct tipo_no* prox;} No;

typedef struct {  No* inicio;
                  No* final;} def_fila;
```

Como só pode acessar um dado da união por vez, é necessário especificar com qual tipo se trabalhando, por isso aparece o tipo na definição do nó. O tipo poderá assumir um dos valores:

```
#define IT 1  /* inteiro */
#define ST 2  /* string */
#define FL 3  /* float */
```

Sempre que for ler ou escrever dados no nó tem de fazer isso verificando o tipo e fazendo as atribuições corretamente.

Uso do void*

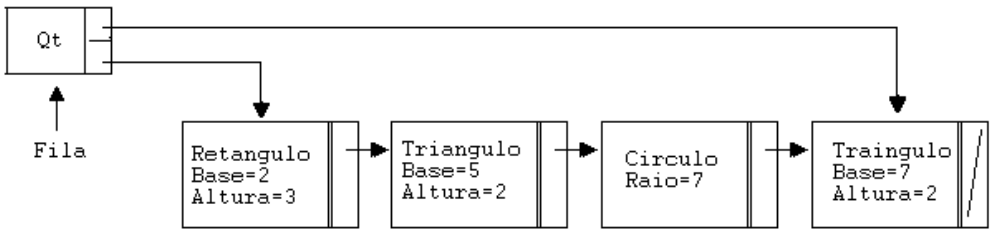
Outra maneira de resolver o problema elaborado na seção anterior é criar um ponteiro genérico - void*. Veja a definição:

```
typedef struct lista {
                  int tipo;
                  void *info;
                  struct lista* prox;} lista_generica;

typedef struct descritor{
                  lista_generica* inicio;
                  lista_generica* final;} *def_fila;
```

Como se construiu um ponteiro genérico - void* não se definiu o que acessar através deste ponteiro. Por essa razão o nó dessa fila deve guardar explicitamente um identificador do tipo do objeto que está armazenado (*int tipo*). Consultando essa variável é possível converter o ponteiro genérico para um ponteiro específico e aí sim, criar e acessar o objeto.

Suponha uma aplicação: armazenar numa fila dados de figuras geométricas. Cada figura geométrica tem suas particulares de dimensionamento. Suponha que se queiram guardar informações dessas figuras para que se possa calcular a área delas e por final calcular a maior área dentre as figuras armazenadas. A nossa fila seria:



Para criar o nó é necessário fazê-lo para todos os tipos de figuras em que está trabalhando. Sejam as figuras: Retângulo, Triângulo e o Círculo:

```
#define RET 1
#define TRI 2
#define CIR 3
typedef struct retangulo{
                  float base;
                  float altura;
                  } Retangulo;

typedef struct triangulo{
                  float base;
                  float altura;
                  } Triangulo;
typedef struct circulo{
                  float raio;
                  } Circulo;
```

A criação do nó do tipo retângulo seria:

```
lista_generica* cria_retangulo(float base, float altura){
   Retangulo* R;
   lista_generica* p;

   R = (Retangulo*)malloc(sizeof(Retangulo));
   R->base = base;
   R->altura = altura;

   p = (lista_generica*)malloc(sizeof(lista_generica));
   p->tipo = RET;
   p->info = R;
   p->prox = NULL;
   return p;
}
```

Essa criação do nó pode ser feita no momento da leitura dos dados e depois se chama a rotina enfileira já passando esse ponteiro. Veja como fica a rotina enfileira:

```
void enfileira (def_fila* Fila, lista_generica* dado)
{
      if (vazia(*Fila)) (*Fila)->inicio = dado;
      else (*Fila)->final->prox = dado;
      (*Fila)->final = dado;
}
```

Na rotina de remoção de um elemento da fila, o nó seria devolvido.

5.Exercícios

1) Demonstre como implementar um **fila estática** de inteiros em C, usando um vetor fila[100], onde fila[0] é usado para indicar o início da fila, fila[1] para indicar o final da fila e as outras posições (fila[2] a fila[99]) guardam os elementos da fila. Demonstre como inicializar esse vetor de modo a representar a fila vazia. Escreva as rotinas desenfileira (.),enfileira(.), vazia(.) e cheia(.) para tal implementação.

2) Em um Banco se deseja saber qual o tempo médio que uma pessoa fica na fila num dia de grande movimento (por exemplo: dia de pagamento). Para isso são distribuídos 100 bilhetes. Ao entrar na fila única dos caixas, a pessoa recebe um bilhete com a hora de entrada marcada. Ao ser atendido pelo caixa, nesse mesmo bilhete é marcada a hora do atendimento. Ao acabarem de recolher todos os 100 bilhetes, eles são computados e calcula-se o tempo médio de espera de uma pessoa na fila do caixa nesse período de amostragem. Fazer um programa que simule esse ambiente. Para isso construa a rotina de entrada do cliente na fila, a rotina de saída do cliente da fila e a rotina de cálculo do tempo médio. Use **Fila Estática** na implementação.

3) Construa um programa que administre as **filas de reservas de filmes** em uma videolocadora, sendo que para cada filme existem sete filas (uma para cada dia da semana) e é o usuário que determina qual é o dia da semana de sua preferência para alugar o filme. O programa deve permitir inclusões nas filas em qualquer ocasião e remoções das mesmas apenas nos respectivos dias - quando o cliente é comunicado por telefone da disponibilidade da fita e quando é confirmada sua locação (caso não seja locado, o próximo da fila será procurado). Use **Fila Estática** na implementação.

4) O Estacionamento Central contém uma única alameda que guarda até 10 carros. Os carros entram pela extremidade sul do estacionamento e saem pela extremidade norte. Se chegar um cliente para retirar um carro que não esteja estacionado na posição do extremo norte, todos os carros ao norte serão deslocados para fora, o carro sairá do estacionamento e os outros carros voltarão à mesma ordem em que se encontravam inicialmente. Sempre que um carro deixa o estacionamento, todos os carros ao sul são deslocados para frente, de modo que, o tempo inteiro, todos os espaços vazios estão na parte sul do estacionamento. Escreva um programa que leia um grupo de linhas de entrada. Cada entrada contém um C (de chegada) e um P (de partida), além da placa do carro. O programa deve imprimir uma mensagem cada vez que um carro chegar ou partir. Quando o carro chegar, a mensagem deverá especificar se existe ou não vaga para o carro. Se não existir vaga o carro esperará pela vaga ou até que uma linha de partida seja lida para o carro. Quando houver espaço disponível, outra mensagem deverá ser impressa. Quando o carro partir, a mensagem deverá incluir i número de vezes que o carro foi deslocado dentro do estacionamento, incluindo a própria partida, mas não a chegada. Esse número será 0 se o carro for embora a partir da linha de espera. Presume-se que os carros chegarão e partirão na ordem especificada pela entrada. Use **Fila Estática** na implementação

5) Alterar o programa acima para que o carro possa sair quando for preciso, respeitando a idéia inicial de fila dentro do estacionamento. Use **Fila Estática** na implementação.

6) Refazer o exercício 03 (Fila de reserva de filmes) usando **Fila Dinâmica**, já que pode ter dia sem reserva de filme.

7) Escreva um programa para que lida uma lista de números inteiros e armazenados numa estrutura de dados do tipo fila, sejam construídas outras duas filas onde uma contém os números pares digitados e a outra os números ímpares.

8) Escreva um programa para que lida uma lista de números do tipo float e armazenados numa estrutura de dados do tipo fila, sejam construídas outras duas filas onde uma contém a parte inteira do número e a na outra, a parte decimal do número.

9) Dada uma fila de inteiros, escreva um programa que exclua todos os números negativos sem alterar a posição dos outros elementos da fila. Utilize o protótipo: void remove_negativo(Fila *F);

10) Escreva um programa para que lida uma cadeia de caracteres e armazenada numa estrutura de dados do tipo fila, sejam construídas outras três filas onde uma contém as vogais da cadeia de caracteres digitada, a outra as consoantes e na última, os caracteres que não encaixam na definição de vogal e consoante.

11) Escreva um programa que recebe uma cadeia de caracteres e a armazena em uma FILA (de caracteres). Depois essa FILA é esvaziada e seus elementos são colocados em 2 PILHAS: uma que contém os caracteres válidos (vogais e consoantes) e outra que conterá os outros caracteres (números, espaços, símbolos etc).

12) Escreva um programa para simular um sistema de computadores multiusuários, como segue: cada usuário tem um ID exclusivo e deseja executar uma operação no computador. Entretanto, somente uma transação pode ser processada por vez. Cada linha de entrada representa um único usuário e contem o ID do usuário seguido de uma hora de inicio e a duração (em milisegundos) de sua transação. O programa deve simular o sistema e imprimir uma mensagem contendo o ID do usuário e a hora em que a transação começou e terminou. No final da simulação, ele deve imprimir o tempo médio geral de espera para uma transação. (O tempo de espera de uma transação é o intervalo de tempo entre a hora em que a transação foi solicitada e a hora em que ela foi iniciada).

13) Simule uma agência de banco que possua três caixas atendendo uma fila única de clientes, de forma a determinar o tempo médio de espera do cliente na fila. A medida que cada qualquer um dos caixas fica livre, o primeiro cliente da fila o utiliza. Quando o cliente entra na fila, o horário é anotado. Quando ele sai, verifica-se quanto tempo ele aguardou. O tempo que o cliente vai demorar no caixa vai depender da transação a ser realizada. Na simulação essa transação deverá ser aleatória e escolhida na tabela abaixo. Use um cronometro para simular o tempo. Quando terminar o expediente (a ser definido pelo usuário e controlado pelo cronometro) o processo de atendimento é imediatamente interrompido. Além de mostrar o tempo médio de espera do cliente na fila, mostre quantas vezes cada transação foi feita, quantos clientes foram atendidos, quantos clientes não foram atendidos (os que estavam na fila na hora que terminou o expediente).

Transação	Código	Tempo (seg)
Saldo	0	10
Saque	1	20
Aplicação	2	30
Extrato Semanal	3	40
Extrato Mensal	4	50
Pagamento de conta em dinheiro	5	60
Pagamento de conta em cheque	6	70
Pagamento de conta com saque	7	80

14) Modifique o exercício anterior para que os clientes que estavam na fila no momento do final do expediente possam ser atendidos. Lembre-se que não se deve permitir que ninguém mais entre na fila.

15) Fazer um programa que gerencie o uso de 3 impressoras que estão ligadas a um microcomputador. Sabe-se que o microcomputador fica enviando partes dos documentos para impressora e não o documento inteiro (só se ele for pequeno). Por exemplo, se queremos imprimir 5 arquivos (Arq1, Arq2, Arq3, Arq4, Arq5). O micro pega parte do Arq1 e envia para a impressora-01, parte do Arq2 para impressora-02 e parte do Arq3 para a impressora-03. Nesse momento o micro volta à impressora-01 e envia uma segunda parte do Arq1. Faz isso com os outros arquivos até que eles tenham sido inteiramente impressos. Na medida em que um arquivo termine de ser impresso se pega o próximo arquivo da fila e inicia-se a sua impressão. Deve-se tomar cuidado para que um arquivo não comece a ser impresso numa impressora e termine em outra. Considere que o micro mande 1kbyte de texto por vez para a impressora e cada impressora gaste 3 segundos para imprimir 1kbyte de texto. No final mostre quantos arquivos foram impressos, quanto tempo (em segundos) cada uma das impressoras funcionou, quanto (em Kbytes) cada uma imprimiu, e qual o tempo médio de impressão dos arquivos da fila.

16) Na seção 4 foi visto como resolver o problema de se ter elementos de tipos diferentes numa fila. Inverta a resolução dos exemplos, ou seja, resolva o exercício de uma fila que contem palavras, inteiros e float usando ponteiro genérico(void*) e resolva a fila de figuras geométricas usando union. Use nas duas implementações a estrutura de Lista que achar conveniente.

17) Considere a existência de um tipo abstrato Fila de números de ponto flutuante implemente uma função que receba três filas Fila1, Fila2, Fila3 e transfira alternadamente os elementos de Fila1 e Fila2 para Fila3 que vai conter todos os valores que estavam originalmente em Fila1 e Fila2.

18) Como você implementaria uma Fila de Pilhas? E uma Pilha de Filas? Escreva rotinas para implementar as operações corretas para essa estrutura de dados.

19) Existem partes de sistemas operacionais que cuidam da ordem em que os programas devem ser executados. Por exemplo, em um sistema de computação de tempo compartilhado (do inglês time-shared) existe a necessidade de manter um conjunto de processos em uma fila, esperando para serem executados. Escreva um programa que seja capaz de ler uma série de solicitações para: (i) incluir novos processos na fila de processos; (ii) retirar da fila o processo com o maior tempo de espera; e (iii) imprimir o conteúdo da lista de processos em determinado momento. Assuma que cada processo é representado por um número identificador do processo.

20) Refaça o exercício acima considerando que diferentes processos podem conter como identificação informações do tipo:
    - O número identificador do processo como tipo int
    - O número identificador do processo como tipo float
    - O nome do processo com até 15 caracteres
    - O tipo de documento como sendo DOCX, XLSX, PPTX etc, ou seja, somente o tipo com 4 caracteres.

    Para resolver esse exercício use a estrutura Union ou ponteiro do tipo Void, como visto em aula.

21) Uma estrada simples é cortada por um rio. Para a travessia do rio existe uma balsa que comporta três filas de veículos: uma fila de caminhões (max 02), uma fila de carros (max 06) e uma fila de ônibus (max 03). Se precisar, pode-se trocar um caminhão por 3 carros ou um ônibus por 2 carros. Do outro lado do rio, desce um veículo por vez. A escolha da ordem de saída da balsa depende do responsável naquele momento.
    (a) Defina a estrutura da estrada (01 fila de veículos) e da balsa (03 filas de veículos);
    (b) Escreva as rotinas de inserção, remoção e impressão dos veículos da estrada.
    (c) Escreva uma rotina que pegue os veículos da estrada e os coloque nas filas certas da balsa.
    (d) Escreva uma rotina que retire os veículos da balsa e os "coloque" na estrada do outro lado. O critério para retirada dos veículos da balsa fica a cargo do programador.

22) **PROBLEMA:** Considere a praça de pedágio da Centrovias Km 199,3 do município de Jaú. Sabe-se que essa praça de pedágio possui 11 pistas de arrecadação e o pedágio é cobrado nas duas direções. As categorias e preço por cada categoria pode-se ver no site abaixo. Imagine que, num momento 't' seja feita uma foto aérea da praça de pedágio. A foto mostra quantos veículos (e de que tipo) existem em cada pista, que tem 5 pistas funcionando no sentido Jaú-Bauru e 6 no sentido Bauru-Jau. A partir dessa foto é feito o cálculo de arrecadação.
    SITE: **www.centrovias.com.br/?link=sua_rodovia.pedagios&target=centrovias**
    IMPLEMENTAÇÃO: Implemente o que se pede seguindo as diretrizes:
    (a) Defina a estrutura da praça do pedágio sabendo que cada pista deve ser uma lista de carros (basta armazenar a classe a qual o carro pertence);
    (b) Construa as rotinas:
        ➢ Escreva uma rotina que faça a remoção do carro de uma determinada pista do pedágio e devolva o valor pago pelo carro.
        ➢ Escreva uma rotina que faça o cálculo do pedágio arrecadado naquele momento "t" da foto. Ao final dessa rotina, a estrutura praça do pedágio deverá estar vazia. Use a rotina anterior para a retirada do carro da pista.

23) Uma *fila dupla* (= *deque*, pronuncia-se *deck*) permite inserção e remoção em qualquer das duas extremidades da fila. Implemente uma fila dupla e codifique as funções de manipulação dessa estrutura.

24) Considere uma pilha P e uma fila F. Utilizando apenas as operações: Enfileira, Desenfileira, Empilha, Desempilha, escreva uma função que inverta a ordem dos elementos da fila.

25) No final da Av Duque de Caxias (já na R Ver Gomes dos Santos) em Bauru (Figura 1), tem uma rotatória que permite que o condutor do veículo escolha 05 caminhos diferentes. Num certo dia, a polícia resolveu bloquear a entrada 01 para caminhões (tipo=2), a entrada 02 para motos (tipo=3), a entrada 03 para bicicletas(tipo=4), a entrada 04 para as Vans(tipo=5) e a entrada 05 para carros (tipo=1). Claro que virou a maior confusão nesta rotatória....

Figura 01. Entroncamento.

a) Declare a estrutura de dados DINÂMICA a ser usada para representar a Av Duque de Caxias (antes da rotatória), sabendo que a capacidade da via é variável..Considere que deva ser guardada a informação do tipo do carro (1-carro, 2-caminhão, 3-moto, 4-bicicletas, 5-vans) e qual caminho o condutor gostaria de fazer a partir da rotatória (1 a 5)
b) Declare a estrutura que representará os caminhos a partir da rotatória (pode ser dinâmica ou estática).
c) Crie as rotinas de (1) entrada de um veículo fila de carros e (2) a saída do veículo da fila de carros já estão prontas.
d) Construa uma rotina que simule o comportamento do policial na rotatória (num determinado Tempo T – ou seja, dados os carros que estão na avenida aguardando na fila para entrada na rotatória). Dependendo do veículo que chegou na rotatória e para onde o veículo quer ir, o policial permite ou não. Se o policial não permitir, ele o encaminha para o próximo caminho possível.

26) Implemente a Ordenação por Distribuição (Szwarcfiter & Markenson(1994,p.41)). O método é o seguinte:
- Seja uma lista composta de n chaves, cada qual representada por um número inteiro numa base B > 1. O problema consiste em ordenar essa lista.
- O algoritmo usa B filas, denotadas por F_i, 0 <= i <= B-1. Seja D o comprimento máximo da representação das chaves na base B. O algoritmo efetua D iterações, em cada uma das quais a tabela é percorrida.

- A primeira iteração destaca, em cada nó, o dígito menos significativo da representação B-ária de cada chave. Se este for igual a k, a chave correspondente será inserida na fila F_k.
- Ao terminar o percurso da tabela, esta se encontra distribuída pelas filas, que devem então ser concatenadas em sequência, isto é, F_0, depois F_1, F_2, etc.
- Para essa tabela, já disposta numa ordem diferente da original, o processo deve ser repetido levando-se em consideração o segundo dígito da representação, e assim sucessivamente até que tenham sido feitas tantas distribuições quantos são os dígitos na chave de ordenação.

Veja um exemplo dessa ordenação para B = 10 (ou seja, para números decimais) e D = 2 (ou seja, os números têm no máximo 2 dígitos).

Seja o conjunto de números: {19, 13, 05, 27, 01, 26, 31, 16, 02, 09, 11, 21, 60, 07}
Vetor inicial: 19, 13, 05, 27, 01, 26, 31, 16, 02, 09, 11, 21, 60, 07
Interação 01: 1ª distribuição (unidades simples)

Fila0: 60
Fila1: 01, 31, 11, 21
Fila2: 02
Fila3: 13
Fila4:
Fila5: 05
Fila6: 26, 16
Fila7: 27, 07
Fila8:
Fila9: 19, 09

Então se obtém o Vetor: 60, 01, 31, 11, 21, 02, 13, 05, 26, 16, 07, 27, 19, 09
Vetor: 60, 01, 31, 11, 21, 02, 13, 05, 26, 16, 07, 27, 19, 09
Interação 02: 2ª distribuição (dezenas simples)

Fila0: 01, 02, 05, 07, 09
Fila1: 11, 13, 16, 19
Fila2: 21, 26, 27
Fila3: 31
Fila4:
Fila5:
Fila6: 60
Fila7:
Fila8:
Fila9:

Vetor Final: 01, 02, 05, 07, 09, 11, 13, 16, 19, 21, 26, 27, 31, 60

27) Implemente uma fila de atendimento (FA) de uma loja que separa a FA em outras duas para que as pessoas mais velhas sejam atendidas primeiro. Assim, ao entrar na FA, deve-se informar a idade. Aí o sistema de gerenciamento da FA, num primeiro processamento (ou após 5 atendimentos), verifica a FA e separa-a em duas "filas": uma que obedece a lei FIFO (Fila) e outra é ordenada por idade (idade > 60 anos)(Preferencial). A Fila terá a sua disposição 3 caixas. Na medida em que se libera um desses três caixas, a pessoa é atendida. A Preferencial tem a sua disposição 2 caixas. Na medida em que está liberado um caixa, uma pessoa da Preferencial é chamada. Mas se uma das "filas" estiver vazia, todos os cinco caixas atendem a "fila" existente.

28) Refaça o exercício 33 para que a "fila" preferencial se torne uma FILA de acordo com o conceito estudado em aula, ou seja, o primeiro a entrar é o primeiro a sair, e não ordenado pela idade do cliente.