Estruturas

Fila estática

Entenda o mecanismo de implementação de filas com a estrutura de um vetor

---

Declarando a fila

A nossa fila estática será composta de duas estruturas: a primeira, `REGISTRO` compreende o conjunto de informações que desejamos armazenar em nossa fila. Ela poderia ser substituída por uma única variável. No entanto, pensando nas aplicações do mundo real, faz muito mais sentido pensar no `REGISTRO` como uma estrutura, que pode conter outras variáveis.

```
typedef struct{
    int chave;
    char nome[50];
    int idade;
} REGISTRO;
```

Como já declaramos a estrutura `REGISTRO`, podemos agora criar uma estrutura chamada `FILA`, que representará a fila de registros. Por estarmos lidando com uma fila estática, a estrutura `FILA` é composta por: um vetor de `REGISTRO`, além de duas variáveis de controle: `inicio` e `qtde`.

```
typedef struct{
    REGISTRO itens[TAM];
    int inicio;
    int qtde;
} FILA;
```

Na fila, as duas variáveis (**inicio** e **qtde**) assumem papéis importantes para auxiliar na implementação.

Logo, o código completo ficaria:

```
#define TAM 5

typedef struct{
    int chave;
} REGISTRO;

typedef struct{
    REGISTRO itens[TAM];
    int inicio;
    int qtde;
} FILA;
```

Inicializando a fila estática

O processo de inicializar a fila estática também é bastante simples: precisamos apenas inicializar os valores das variáveis `qtde` e `inicio` com `0`.

```
void inicializarFila(FILA* f){
    f->inicio = 0;
    f->qtde = 0;
}
```

Inserindo elementos na fila (enfileirando)

Uma abordagem mais óbvia

A operação de inserir elementos numa estrutura do tipo fila é chamada de enfileiramento. Aqui temos uma estrutura bastante peculiar, que a diferencia da pilha. Vamos observar os motivos para isso.

Numa fila, a remoção de elementos acontece sempre no início. Em uma fila estática, isso gera um pequeno problema: o primeiro elemento a ser removido é aquele que está na primeira posição do vetor.

No entanto, desde **Algoritmos e Estruturas de Dados I**, costumamos percorrer o vetor a partir da posição `0`. Retirar o primeiro elemento demandaria movimentar os elementos restantes para uma casa anterior, passando o elemento `n` para a posição `n-1`, assim por diante. Esse procedimento

tem um custo computacional que não pode ser ignorado - e que pode ser repetido diversas vezes, sempre que um elemento for removido da fila.

Uma abordagem mais eficiente

A solução para isso é bastante interessante: uma forma mais inteligente de usar a estrutura do vetor. A lógica é flexibilizar, de forma virtual, as posições de início e de fim, não necessariamente se limitando à abordagem original do vetor. Veja a fundamentação para essa ideia:

- A fila tem um contador de registros armazenados, representado pela variável `qtde`.
- Naturalmente, `qtde` não pode ser superior ao tamanho do vetor de itens da fila (no nosso caso, o tamanho é definido por `TAM`). Em outras palavras: a fila sempre guardará, no máximo, a quantidade de itens que o seu vetor pode armazenar.

O dilema que pode surgir aqui é: após três remoções da fila, `f->inicio` apontará para uma posição do meio do vetor. A posições que restam, então, não são suficientes para armazenar toda a quantidade de itens que a fila suporta. Eis o segredo: fazer com que as posições que já foram ocupadas e foram excluídas sejam reutilizadas. São as posições que, no índice do vetor, antecedem `f->inicio`.

Veja: se somarmos o f->inicio (posição que representa o início da fila) e `f->qtde`, que é a quantidade de itens que continuam na fila, podemos chegar a duas situações: 1. um valor menor que o índice máximo que o vetor suporta; 2. uma posição inexistente no vetor, pois está acima do índice máximo do vetor. Neste caso, o segredo é encontrar o resto de `(f->inicio + f->qtde) % TAM`. Assim, identificamos a primeira posição à esquerda do vetor que ainda está disponível.

Veja que interessante é a implementação desta função de `inserir`:

```
bool inserir(FILA *f, REGISTRO r){
    if(f->qtde < TAM){
        int pos = (f->inicio + f->qtde) % TAM;
        f->itens[pos] = r;
        f->qtde++;
        return true;
    }
    return false;
}
```

Removendo os elementos

Agora que você já entendeu toda a lógica de funcionamento da fila estática, o processo de remover é bem simples: retiramos o elemento do início da fila. Depois, identificamos quem é o próximo da fila.

No uso tradicional do vetor, isso funcionaria apenas incrementando o valor de início. No entanto, aqui temos um caso especial e mais eficiente, lembra?

O próximo elemento é definido pelo resto do incremento pelo tamanho. Veja só:

```
bool remover(FILA *f, REGISTRO *r){
    if(f->qtde > 0){
        *r = f->itens[f->inicio];
        f->inicio = (f->inicio + 1) % TAM;
        f->qtde--;
        return true;
    }
    return false;
}
```

Note que, na função acima, `remover` recebe um ponteiro de `REGISTRO` como parâmetro. Isso é importante para que a função que a chamou consiga ter acesso aos dados do registro removido da fila.

Imprimindo a fila

Por conta desse mecanismo que usamos para armazenar a fila em um vetor, o processo de imprimir também é um pouco peculiar. Veja a lógica envolvida:

1. Criamos uma variável `pos`, que começa no início da fila;
2. Com um laço `for`, para cada um dos elementos já armazenados, fazemos o seguinte:
    1. Imprimimos o valor da chave do registro armazenado na posição `pos` do vetor de itens da fila;
    2. Atualizamos o valor de `pos`, seguindo a mesma lógica que usamos na remoção:
        - O novo valor de `pos` será definido pelo resto de seu incremento pela capacidade do vetor;
        - Isso permite que a impressão considere as primeiras posições do vetor, caso `f->inicio` já não esteja na posição `0`.

```
void imprimir(FILA *f){
    int pos = f->inicio;
    for(int i = 0; i < f->qtde; i++){
        printf("%d\n", f->itens[pos].chave);
        pos = (pos + 1) % TAM;
    }
}
```

Usando a fila estática

O uso de uma fila estática na função principal é bastante simples. Um ponto importante é que, após a declaração da variável, é necessário chamar a função de inicialização da fila. Veja:

```
int main(void){
    FILA f;
    inicializarFila(&f);

    return 0;
}
```

Uma vez que a fila está inicializada, podemos começar a inserção de registros. Uma abordagem tradicional pode ser:

```
REGISTRO r1 = {25};
inserir(&f, r1);
```

Essa abordagem é completamente válida e correta, mas há um caminho mais curto:

```
inserir(&f, (REGISTRO){25});
```

Confirmação da criação

Observe que, nos exemplos acima, estamos chamando `inserir` como se fosse um procedimento, ou seja, sem considerar o seu retorno. O mais adequado aqui é considerá-la como função que retorna um booleano: `true` caso a inserção tenha sido concretizada e `false` caso contrário.

Veja como ficaria:

```
bool r = inserir(&f, (REGISTRO){25});
if(r) { printf("Deu certo!\n"); }
else { printf("Não inserido!\n"); }
```

Ou, com operador ternário, mas de forma menos legível:

```
inserir(&f, (REGISTRO){25}) ? printf("Inserido.\n") : printf("Erro ao inserir.\n");
```

Agora, para remover um registro, fazemos o seguinte:

```
REGISTRO r;
if(remover(&f, &r)){
    printf("%d removido com sucesso\n", r.chave);
}
```

Exemplo completo

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>

#define TAM 5

typedef struct{
    int chave;
} REGISTRO;

typedef struct{
    REGISTRO itens[TAM];
    int qtde;
    int inicio;
} FILA;

void inicializarFila(FILA* f){
    f->inicio = 0;
    f->qtde = 0;
}

bool inserir(FILA *f, REGISTRO r){
```

```
    if(f->qtde < TAM){
        int pos = (f->inicio + f->qtde) % TAM;
        f->itens[pos] = r;
        f->qtde++;
        return true;
    }
    return false;
}

bool remover(FILA *f, REGISTRO *r){
    if(f->qtde > 0){
        *r = f->itens[f->inicio];
        f->inicio = (f->inicio + 1) % TAM;
        f->qtde--;
        return true;
    }
    return false;
}

void imprimir(FILA *f){
    int pos = f->inicio;
    for(int i = 0; i < f->qtde; i++){
        printf("%d\n", f->itens[pos].chave);
        pos = (pos + 1) % TAM;
    }
}

int main(void){
    FILA f;
    inicializarFila(&f);
    inserir(&f, (REGISTRO){10});
    inserir(&f, (REGISTRO){20});
    inserir(&f, (REGISTRO){30});

    REGISTRO r;
    if(remover(&f, &r)){
        printf("%d removido com sucesso\n", r.chave);
    }

    inserir(&f, (REGISTRO){40});
    inserir(&f, (REGISTRO){50});
    inserir(&f, (REGISTRO){60});
    inserir(&f, (REGISTRO){70}); // a fila já está cheia

    imprimir(&f);

    return 0;
```

```
}
```

Estruturas
← Pilha dinâmica

Estruturas
Fila dinâmica →