Conteúdo da Etapa 1

Conceitos de pilha e fila

Tanto pilhas quanto filas são tipos distintas de listas, cada uma seguindo um princípio de inserção de dados diferentes:

- **Uma pilha** é uma lista de itens que seguem o **princípio LIFO** (*Last-in, first-out*), ou seja, **o PRIMEIRO elemento a ser retirado é o ÚLTIMO que foi inserido na lista** (o inverso também é válido, ou seja, **o ÚLTIMO elemento retirado é o PRIMEIRO inserido**)! Está presente em vários contextos da vida real: imagine que estamos colocando um prato em cima do outro, formando uma **pilha de pratos**. Para evitar o risco dos pratos caírem e serem quebrados, devemos ir retirando o prato que está no topo, exatamente como numa dinâmica de pilha.

Arriscaria retirar os pratos de baixo deste conjunto?? Usando o princípio de pilha, evitamos maiores acidentes!

- Já **as filas** seguem o **princípio FIFO** (*First-in, first-out*), ou seja, **o PRIMEIRO elemento a ser retirado é o PRIMEIRO que foi inserido** (novamente, vale o inverso, **o ÚLTIMO elemento retirado é o ÚLTIMO inserido**). Seu nome é em referência às filas que vemos em diversas situações do cotidiano, como **uma fila de mercado** (a primeira pessoa que entrou na fila será a primeira a ser atendida, assim sendo a primeira a sair da fila).

Às vezes enfrentar filas pode ser entediante... Mas sua dinâmica respeita a ordem de chegada! Quem veio primeiro se sai melhor!

Em termos de algoritmos, **há dois tipos fundamentais de implementação** tanto de pilhas como de filas, relacionados diretamente à ESTRUTURA de armazenamento de dados. São elas: **ESTÁTICA** e **DINÂMICA**. Confira abaixo cada uma delas e os métodos implementados:

Pilha e fila dinâmica

Ambas são EXTREMAMENTE semelhantes, diferindo apenas como os dados são introduzidos na lista (como discutido anteriormente). Aqui, implementamos como um *struct* (**No** para *pilha* e **NoF** para *fila*) com dois atributos, funcionando como um **nó** de uma lista geral:

- **dado**: Dado do respectivo nó
- **prox**: Ponteiro do nó seguinte ao atual na lista

Portanto, **pilhas e filas dinâmicas TEORICAMENTE podem ter tamanho INFINITO** (na prática, o maior tamanho possível é **o espaço de armazenamento disponível na memória**). Com isso, confira abaixo as funções disponíveis nos arquivos ***hpp*** para manipulação de *pilhas* e *filas* dinâmicas!

Funções (parâmetros)	O que faz cada função?	O que retorna?

adicionaP(pilha, dado) **adicionaF(fila, dado)**	**Adiciona um dado** no FINAL de uma pilha ou fila.	Ponteiro do **primeiro nó (para fila)** ou **último nó (para pilha)**
removeP(pilha)	Remove o **último elemento de uma pilha**, seguindo a dinâmica do **LIFO**.	Ponteiro do **PENÚLTIMO nó da pilha** (*ou nulo se sobrar nenhum item*)
removeF(fila)	Remove o **primeiro elemento de uma fila**, seguindo a dinâmica do **FIFO**.	Ponteiro do **SEGUNDO nó da fila** (*ou nulo se sobrar nenhum item*)
destroiP(pilha) **destroiF(fila)**	**Deleta TODOS os nós** de uma pilha ou fila.	**Ponteiro NULO** (pilha ou fila NÃO tem itens)
quantosItens(pilha) **quantosItens(fila)**	**Faz a contagem de nós** da lista a partir do nó de entrada (último nó pra pilha, primeiro pra fila)	**Int** correspondente à quantidade
buscaP(pilha, dado) **buscaF(fila, dado)**	Identifica **se existe um determinado dado** na pilha ou fila.	**Booleano** indicando se dado foi encontrado (*true*) ou não (*false*)
mostraP(pilha, nome) **mostraF(fila, nome)**	**Mostra a lista** de dados armazenados em forma de tabela: com o **título em cima**, uma coluna com os **endereços dos nós** e a outra com os dados em si.	Nada (função *void*)

Observação IMPORANTÍSSIMA: Usamos um **ponteiro SIMPLES iniciado como *nullptr*** para criar uma nova pilha ou fila **dentro da função main**. Se esta pilha ou fila dinâmica for passada por PARÂMETRO em uma função FORA do main, ela deve ser passada por um ponteiro DUPLO (ponteiro de ponteiro), permitindo sua **modificação no MAIN!!!** Confira os exemplos:

```
10      No<int>* P = nullptr; // Inicia a pilha como PONTEIRO NULO
11
12      // Adicionando números na fila (enfileirando...)
13      P = adicionaP(P, 10);
14      P = adicionaP(P, 20);
15      P = adicionaP(P, 30);
16      P = adicionaP(P, 40);
17      P = adicionaP(P, 50);
18      P = adicionaP(P, 60);
19      P = adicionaP(P, 70);
20      mostraP(P, "de 10 a 70 (do FIM ao INÍCIO)");
```

Nesta imagem, criamos uma pilha dinâmica e **adicionamos valores DIRETAMENTE na função main!**

```
11  void adicionarValores(No<int>** P) { // Ponteiro duplo: **P
12
13      *P = adicionaP(*P, 10); // Use *P!!!
14      *P = adicionaP(*P, 20); // Use *P!!!
15      *P = adicionaP(*P, 30); // Use *P!!!
16      *P = adicionaP(*P, 40); // Use *P!!!
17      *P = adicionaP(*P, 50); // Use *P!!!
18      *P = adicionaP(*P, 60); // Use *P!!!
19      *P = adicionaP(*P, 70); // Use *P!!!
20      cout << ">> Adicionado VÁRIOS valores à pilha... Confira!!\n\n";
21  }
```

Aqui, adicionamos os mesmos valores à pilha, mas dentro de uma **função FORA da main**. Portanto, **a pilha é passada em um ponteiro DUPLO!**

Pilha estática

Pode ser estabelecido como um *struct* com quatro atributos básicos.

1. **dados**: É um ponteiro que abriga um **vetor de tamanho FIXO de dados** a serem comportados.
2. **tam**: É o **tamanho do vetor que comporta os dados**. É definido quando inicializamos a pilha!
3. **base**: Consiste no índice do **primeiro elemento** menos um. No contexto de pilha, ela é uma CONSTANTE **(base = -1)**
4. **topo**: Consiste no índice do **último elemento** da pilha (se a pilha estiver VAZIA, **topo = -1**, mas se a pilha estiver LOTADA, **topo = tam - 1**)

Confira um exemplo de remoção de itens em uma pilha de tamanho 6, com 5 itens inicialmente armazenados:

-1	0	1	2	3	4	5
	10	**20**	**30**	**40**	**50**	?
base					topo	

- O último elemento da pilha é removido, logo **o 50 será removido e topo será 3!**

-1	0	1	2	3	4	5
	10	**20**	**30**	**40**	?	?
base				topo		

- O último elemento da pilha é removido, logo **o 40 será removido e topo será 2!**

-1	0	1	2	3	4	5
	10	**20**	**30**	?	?	?
base			topo			

Nota-se como o tamanho SEMPRE é fixo, mas o que muda conforme a remoção dos dados (e a inserção também) é **o topo da lista** (diminui 1 a cada remoção, e aumenta 1 a cada inserção)**!**

Fila circular estática

Pode ser estabelecido como um *struct* com cinco atributos básicos.

1. **dados**: Um ponteiro que abriga um **vetor de tamanho FIXO de dados** a serem comportados.
2. **tam**: **Tamanho do vetor que comporta os dados**. É definido quando inicializamos a fila!

3. **total**: **Quantidade de dados que estão sendo armazenados**. Quando total = 0, a fila está VAZIA. Se total = tam, a fila está CHEIA!
4. **inicio**: Consiste no **índice do PRIMEIRO elemento** da fila.
5. **fim**: Consiste no **índice do PRÓXIMO elemento** que será adicionado na fila (NÃO corresponde ao do ÚLTIMO elemento!!!)

O índice *inicio* aumenta conforme a remoção de elementos, enquanto o índice *fim* aumenta conforme a adição de elementos. Quando definimos que **a fila é circular**, estabelecemos que **tanto o *fim* quanto o *inicio* poderão retornar ao valor zero** (se tanto o *inicio* quanto o *fim* tiverem valor **tam - 1**, o próximo valor será 0 novamente), **como se os atributos estivessem "dando a volta" na fila**, sempre possibilitando a inserção de mais e mais elementos (desde que a fila não esteja lotada). Confira abaixo um exemplo prático:

0	1	2	3	4
?	**10**	**20**	**30**	?
	inicio			**fim**

A fila acima tem tamanho (*tam*) 5, *total* 3 e índice *fim* 4. Como **fim = tam - 1**, se adicionarmos um novo valor para a fila, ***fim* assumirá valor 0!** Então vamos adicionar aqui o número 40...

0	1	2	3	4
	10	**20**	**30**	**40**
fim	**inicio**			

Adicionando mais um número na fila, ela ficará lotada e **os atributos *inicio* e *fim* assumirão o mesmo valor/índice!!!**

0	1	2	3	4
50	**10**	**20**	**30**	**40**
	inicio/fim			

Observação IMPORTANTE: Se for tentado a adição de novo elemento tanto em pilha quanto em fila estática CHEIA, **este método NÃO IMPEDIRÁ a execução de métodos seguintes à este!!!**

Agora, se quisermos retirar TODOS os elementos da nossa fila, **o valor *inicio* TAMBÉM "dará a volta" em torno da fila**. Confira o passo-a-passo:

- Removido **10** com sucesso!				
0	1	2	3	4
50	?	**20**	**30**	**40**
	fim	inicio		

- Removido **20** com sucesso!				
0	1	2	3	4
50	?	?	**30**	**40**
	fim		inicio	

- Removido **30** com sucesso!				
0	1	2	3	4
50	?	?	?	**40**
	fim			inicio

- Removido **40** com sucesso!				
0	1	2	3	4
50	?	?	?	?
inicio	fim			

0	1	2	3	4
?	?	?	?	?
	inicio/fim			

Deletado TODOS os elementos da fila, NOVAMENTE **os valores *inicio* e *fim* se coincidem** no MESMO índice. Então, **VALE RESSALTAR que**:

Se ***inicio* = *fim***, mas ***total* = 0**, a **fila está VAZIA!**
Se ***inicio* = *fim***, mas ***total* = *tam***, a **fila está CHEIA!**

Métodos para pilha e fila estática

Funções (parâmetros)	O que faz cada função?	O que retorna?
inicializaP(pilha, tam) **inicializaF(fila, tam)**	**Cria um novo vetor de dados**, com tamanho passado por parâmetro.	Nada (função *void*)
verificaInicializacaoP(pilha) **verificaInicializacaoF(fila)**	O próprio nome já diz...	**Booleano** indicando se foi inicializado (*true*) ou não (*false*)
empilhaP(pilha, dado) **enfileiraF(fila, dado)**	**Adiciona um dado** no FINAL de uma pilha ou fila. Se **a lista JÁ está lotada**, dado simplesmente **NÃO será inserido!!**	Nada (função *void*)
desempilhaP(pilha)	"Remove" o **último elemento de uma pilha (LIFO)**.	**Dado removido.**
desenfileiraF(fila)	"Remove" o **primeiro elemento de uma fila (FIFO)**.	**Dado removido.**
esvaziaP(pilha) **destroiF(fila)**	Parâmetros são alterados para indicar que **"todos os dados foram deletados"** (na prática, NÃO são realmente deletados!)	Nada (função *void*)
vaziaP(pilha) **vaziaF(fila)**	Verifica **se NÃO há dados sendo armazenados** ou não na pilha ou fila.	**Booleano** indicando se lista está vazia (true) ou não (false)
cheiaP(pilha) **cheiaF(fila)**	Verifica **se os dados estão ocupando TODO o vetor** de dados da pilha ou fila.	**Booleano** indicando se lista está cheia (true) ou não (false)
buscaP(pilha, dado) **buscaF(fila, dado)**	Identifica **se existe um determinado dado** na pilha ou fila.	**Booleano** indicando se dado foi encontrado (*true*) ou não (*false*)
mostraP(pilha, nome) **mostraF(fila, nome)**	**Mostra a lista** de dados armazenados.	Nada (função *void*)

Confira COM MUITA ATENÇÃO os arquivos de exemplos tanto das listas ESTÁTICAS quanto das DINÂMICAS (com ênfase principalmente nos **exemplos das dinâmicas COM funções fora da main ou SEM elas**)!!!

Nego Ney B)