Notas de aula 08/04

PILHAS

Uma pilha é um tipo de lista onde inserções e retiradas são feitas sempre no final da lista. Imagine uma pilha de pratos: quando você guarda um prato, normalmente coloca o prato no topo da pilha. Quando você precisa pegar um prato, normalmente você retira o prato que está no topo da pilha.

Desta forma, o item colocado mais recentemente na pilha fica no topo, enquanto o item colocado menos recentemente fica no fundo. Assim, o último item colocado é o primeiro a ser retirado. Por essa característica a pilha é dita como uma estrutura do tipo LIFO (*last in, first out* - último a entrar, primeiro a sair).

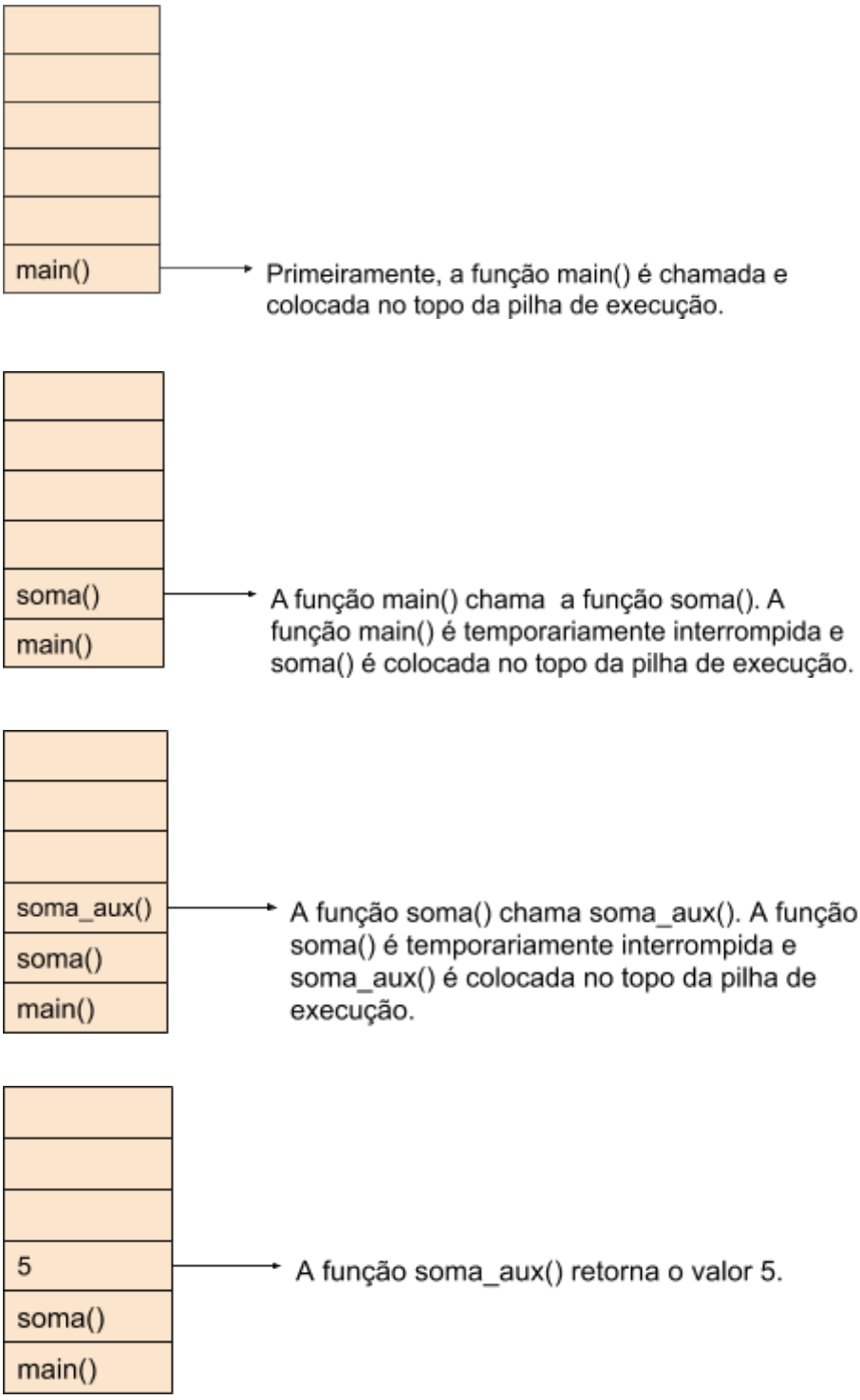
Uma aplicação muito importante de pilhas está na própria na execução dos nossos programas. Todo programa em C consiste de uma ou mais funções. Em um programa com chamadas para várias funções, o computador usa uma pilha de execução para administrar as chamadas de funções que vão sendo feitas.

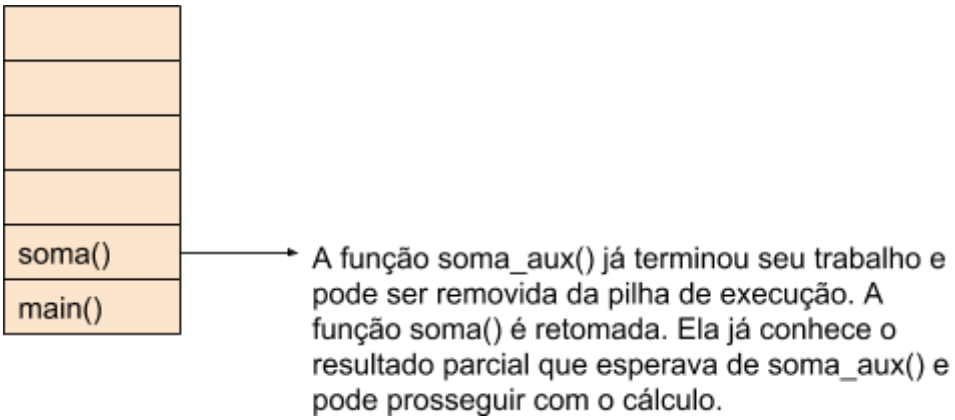
Considere o programa:

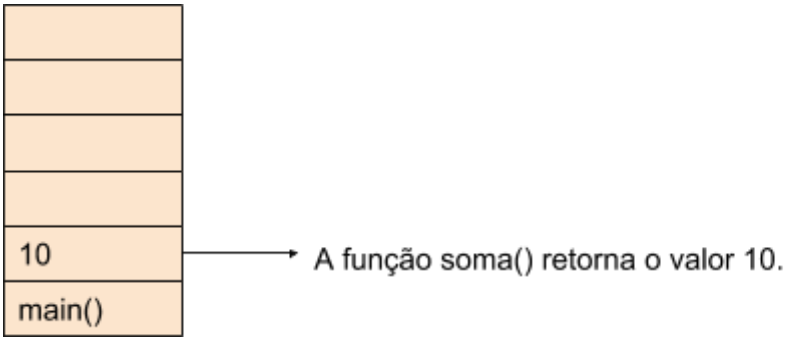
```c
#include <stdio.h>

int soma(int x, int y, int z) {
  int s;
  s = soma_aux(x, y);
  s = s + z;
  return s;
}

int soma_aux(int x, int y) {
  return x + y;
}

main() {
  printf("%d + %d + %d = %d", 2, 3, 5, soma(2, 3, 5));
}
```

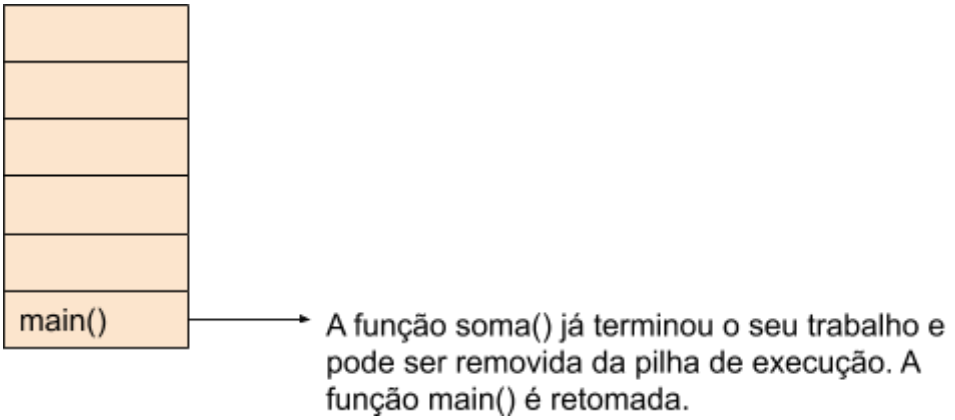
- Na função principal temos o objetivo de somar 3 números. Para isso é chamada a função soma().
- A função soma() tem o objetivo de somar 3 números, mas para isso ela precisa chamar a função soma_aux();
- A função soma_aux() soma 2 números e retorna o resultado para a função soma().
- A função soma() adiciona a esse resultado parcial o terceiro número e retorna para a função principal.

Pilha de execução:


main()
Primeiramente, a função main() é chamada e colocada no topo da pilha de execução.
soma()
main()
A função main() chama a função soma(). A função main() é temporariamente interrompida e soma() é colocada no topo da pilha de execução.
soma_aux()
soma()
main()
A função soma() chama soma_aux(). A função soma() é temporariamente interrompida e soma_aux() é colocada no topo da pilha de execução.
5
soma()
main()
A função soma_aux() retorna o valor 5.







Cada chamada de função (cada retângulo da pilha) é composta por um *frame* que contém os parâmetros da função, suas variáveis locais e endereço de retorno (para que a função soma_aux(), por exemplo, saiba que tem que retornar a resposta do cálculo para a função soma()).

Tipo abstrato de dados Pilha

Geralmente, uma pilha precisa das seguintes operações:

- Criar uma pilha vazia;
- Inserir um novo elemento no final (empilhar);
- Retirar o elemento do final (desempilhar);
- Verificar se a pilha está vazia;
- Verificar se a pilha está cheia;
- Imprimir a pilha;

Implementações

A pilha é um tipo de lista. Vimos que listas podem ser implementadas usando vetores ou usando ponteiros. Vamos primeiramente estudar uma pilha usando vetores.

Implementação por arranjos ou vetores

Podemos implementar uma pilha de forma muito parecida com a implementação da lista das aulas passadas. Na verdade, a implementação será bem mais simples, pois as inserções e retiradas de itens são feitas sempre no final da pilha. Ver implementação no github.

FILAS

Uma fila é um tipo de lista onde inserções são feitas ao final da lista e retiradas são feitas no início da lista. Você pode pensar em uma fila de pessoas no banco: uma nova pessoa sempre deve entrar no final da fila e a pessoa que chegou primeiro será a primeira a ser atendida.

Como o primeiro item a entrar na fila é o primeiro a ser retirado, uma fila é conhecida como uma estrutura do tipo FIFO (*first in, first out* - primeiro a entrar, primeiro a sair).

Filas são estruturas úteis quando queremos processar itens de acordo com a ordem de chegada. Elas são fundamentais para o funcionamento de computadores. Sistemas operacionais usam filas para regular a ordem na qual as tarefas devem receber processamento ou recursos.

Tipo abstrato de dados Fila

Geralmente uma fila contém as seguintes operações:

- Criar uma fila vazia;
- Inserir um novo elemento no final (enfileirar);
- Retirar o elemento do início (desenfileirar);
- Verificar se a fila está vazia;
- Verificar se a fila está cheia;
- Imprimir a fila;

Implementações

Assim como as pilhas, filas podem ser implementadas usando arranjos/vetores ou ponteiros. Primeiramente estudaremos as filas usando arranjos/vetores.

Antes de partirmos para códigos, é preciso analisar alguns pontos. Por causa da característica da fila, a operação de enfileirar faz aumentar a parte de trás da fila, enquanto a operação desenfileirar faz contrair a parte da frente da fila. Desta forma, a fila tende a caminhar pela memória, ocupando espaço da parte de trás e descartando espaço da parte da frente. Isso faz com que em poucas operações a fila vá ao encontro do limite de memória reservado para ela.

Exemplo de uma fila contendo 3 itens e com capacidade máxima de 6 itens.

23	14	19			

- Remoção do item 23:

	14	19			

- Inserção do item 45:

	14	19	45		

- Remoção do item 14:

		19	45		

- Inserção do item 30:

		19	45	30	

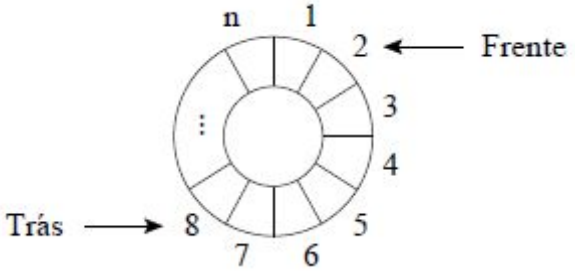
- Remoção do item 19:

			45	30	

- Inserção do item 67, atingindo o limite da fila:

			45	30	67

A fila fica inutilizável pois seu limite foi atingido, apesar de termos posições vazias na parte da frente da fila. Uma solução para este inconveniente é imaginar a fila com um círculo:



Implementação de fila por meio de arranjos/vetores em C

As estruturas para representar a fila são semelhantes às vistas anteriormente. Porém, conforme a ilustração acima, podemos ver a necessidade de guardar quem são o primeiro e último item da fila. Para facilitar também podemos armazenar o tamanho da fila:

```c
typedef struct item Item;
typedef struct fila Fila;

#define MAXTAM 5

struct item {
  int chave;
  // demais campos
};
```

```
struct fila {
  Item item[MAXTAM];
  int primeiro;
  int ultimo;
  int tamanho;
};
```

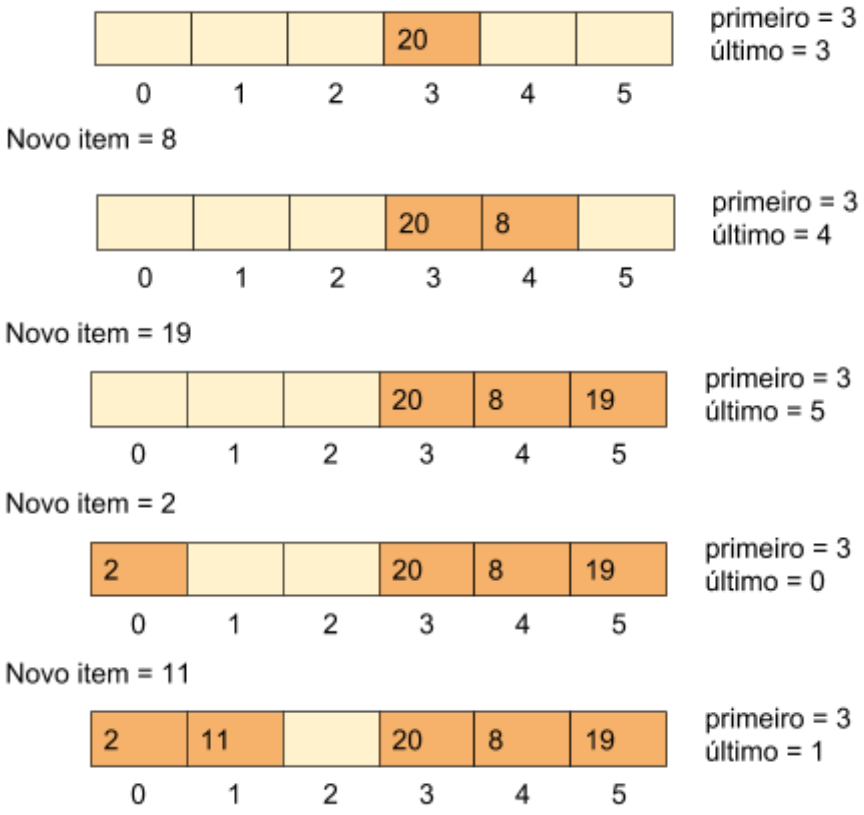
Na função para criar a fila vazia, podemos considerar que o primeiro item é igual ao último item, que é igual a 0 (vamos partir de 0 em vez de -1 pois usaremos primeiro e último como índices). O tamanho da fila também deve ser 0:

```
Fila * cria_fila_vazia() {
  Fila *f = malloc(sizeof(Fila));
  f->primeiro = 0;
  f->ultimo = 0;
  f->tamanho = 0;
  return f;
}
```

Com o campo tamanho fica fácil verificar se a fila está vazia ou cheia:

```
int verifica_fila_vazia(Fila *f) {
  return f->tamanho == 0;
}

int verifica_fila_cheia(Fila *f) {
  return f->tamanho == MAXTAM;
}
```

Vamos agora pensar na função que insere um item no final da fila. Devido a implementação circular, isso requer alguns cuidados a mais em relação a pilha. Considere o cenário abaixo:



- Ao adicionar o item de chave 8, o último passa a ser 4;
- Ao adicionar o item de chave 19, o último passa a ser 5;
- Ao adicionar o item de chave 2, o último passa a ser 0;
- Ao adicionar o item de chave 11, o último passa a ser 1.

Perceba que esses índices caminhando de forma circular podem ser obtidos pela fórmula:
ultimo = (ultimo + 1) % MAXTAM

- Ao adicionar o item de chave 8, o último passa a ser 4:
  ultimo = (3 + 1) % 6 = 4 % 6 = **4**
- Ao adicionar o item de chave 19, o último passa a ser 5:
  ultimo = (4 + 1) % 6 = 5 % 6 = **5**
- Ao adicionar o item de chave 2, o último passa a ser 0:
  ultimo = (5 + 1) % 6 = 6 % 6 = **0**
- Ao adicionar o item de chave 11, o último passa a ser 1:
  ultimo = (6 + 1) % 6 = 7 % 6 = **1**

Assim, podemos implementar a função de inserir um item no final da lista da seguinte forma:

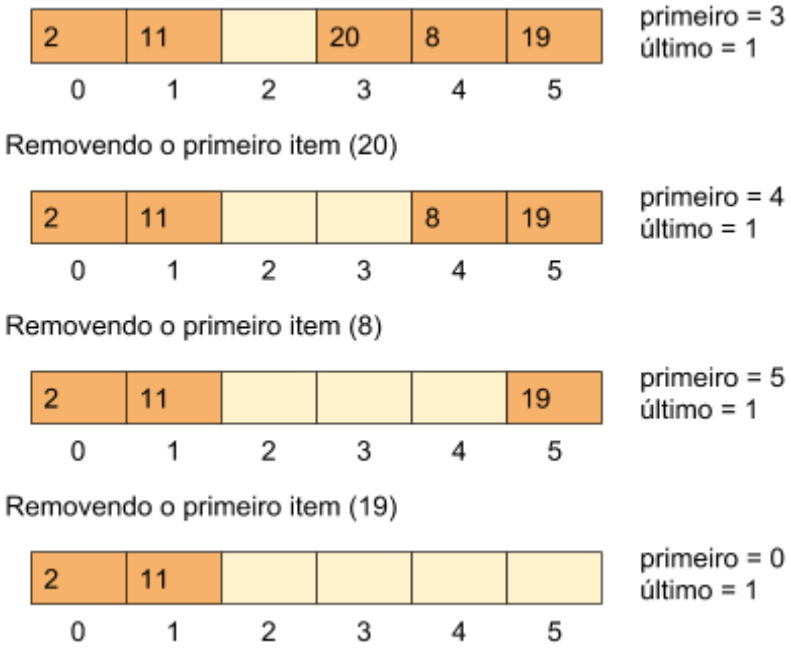
```
void enfileira(Fila* f, int chave) {
  Item novo_item;
```

```
  if(verifica_fila_cheia(f)) {
       printf("Erro: a fila está cheia.\n");
      return;
  }
    else {
      novo_item.chave = chave;
       f->item[f->ultimo] = novo_item;
       f->ultimo = (f->ultimo + 1) % MAXTAM;
       f->tamanho++;
    }
}
```

Caminhando de forma circular para imprimir a fila:

```
void imprime(Fila* f) {
    int i = f->primeiro;
    int t = f->tamanho;
    while (t > 0) {
       printf("Chave: %d\n", f->item[i].chave);
       i = (i+1) % MAXTAM;
       t--;
    }
}
```

A mesma lógica pode ser usada na função de remoção do início para calcular o novo índice do primeiro item da fila:



primeiro = (primeiro + 1) % MAXTAM

- Ao remover o item de chave 20, o primeiro passa a ser 4:
  ultimo = (3 + 1) % 6 = 4 % 6 = **4**
- Ao remover o item de chave 8, o último passa a ser 5:
  ultimo = (4 + 1) % 6 = 5 % 6 = **5**
- Ao remover o item de chave 19, o último passa a ser 0:
  ultimo = (5 + 1) % 6 = 6 % 6 = **0**

```
void desenfileira(Fila* f) {
    if(verifica_fila_vazia(f)) {
      printf("Erro: a fila está vazia.\n");
      return;
  }
    else{
       f->primeiro = (f->primeiro + 1) % MAXTAM;
       f->tamanho--;
    }
}
```

Links interessantes:
Aplicação interativa sobre funcionamento de pilha. Use o botão *push* para colocar um novo item na pilha e o botão *pop* para tirar um item da pilha:
https://www.cs.usfca.edu/~galles/visualization/StackArray.html.

Aplicação interativa sobre funcionamento de fila. Use o botão *enqueue* para colocar um novo item na fila e o botão *dequeue* para tirar um item da fila.
https://www.cs.usfca.edu/~galles/visualization/QueueArray.html