Notas de aula - 29/04

PILHA E FILA ENCADEADA

Estrutura da PILHA ENCADEADA

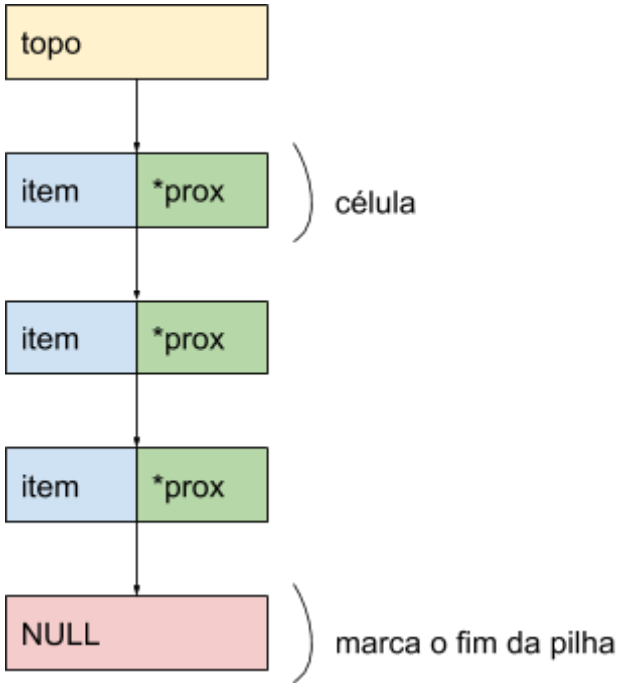
Podemos definir a estrutura da pilha de maneira semelhante a da lista vista anteriormente:

```
typedef struct item Item;
typedef struct celula Celula;
typedef struct pilha Pilha;

struct item {
  int chave;
  // demais campos
};
struct celula {
  Item item;
  Celula *prox;
};
struct pilha {
  Celula *topo;
};
```

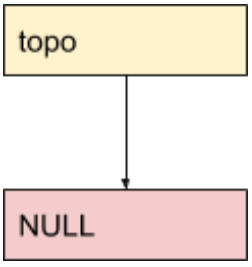
Temos uma estrutura do tipo Item para guardar o item e seus campos relevantes, uma estrutura do tipo Celula, que guarda um item e um ponteiro para quem será a próxima célula da pilha e por fim uma estrutura do tipo Pilha, que contém nada mais do que um ponteiro para a célula que é o topo da pilha.

Podemos visualizar da seguinte forma:



Criação de pilha vazia

Também fica muito parecida com a criação da nossa lista. Se ela está vazia, não temos um endereço para o qual “topo” deve apontar. Assim, podemos fazer “topo” apontar para NULL.



Um exemplo de implementação seria:

```
Pilha * cria_pilha_vazia() {
  Pilha *p = malloc(sizeof(Pilha));
  p->topo = NULL;
  return p;
}
```

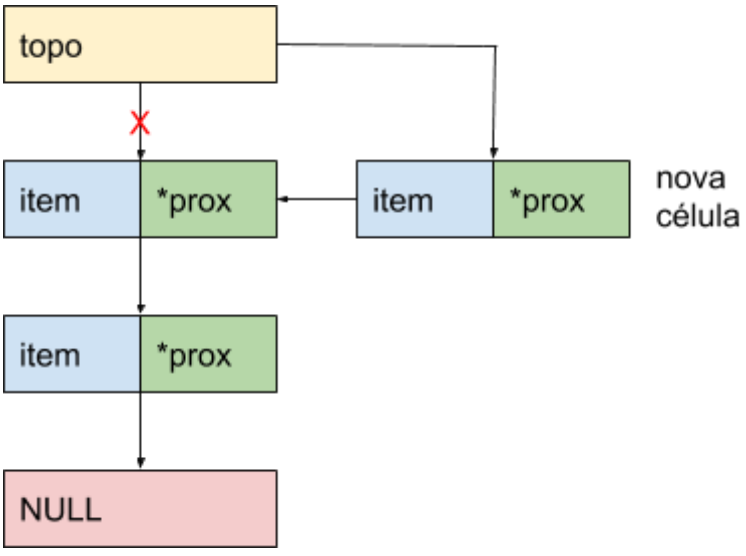
Verificar lista vazia

Para verificar se a pilha está vazia basta verificar se o topo ainda é igual a NULL:

```
int verifica_pilha_vazia(Pilha *p) {
  return p->topo == NULL;
}
```

Empilha

Essa função ficará semelhante a inserção no início da lista. A próxima célula da nova célula adicionada será aquela que era o topo. O topo, por sua vez, agora deve apontar para a nova célula:

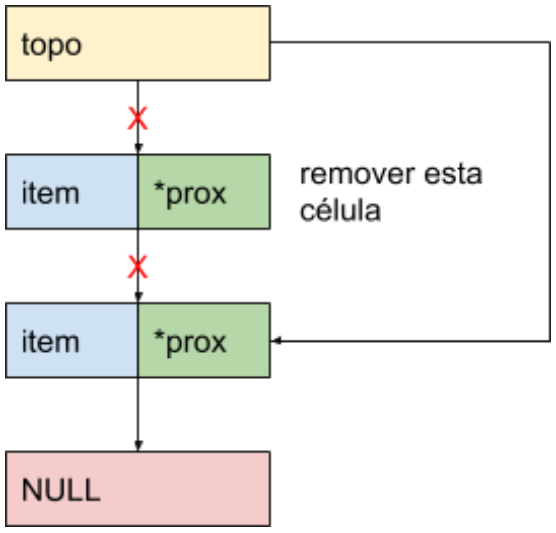


Uma forma de implementação seria:

```
void empilha(Pilha *p, int chave) {
  Item novo;
  novo.chave = chave;
  Celula *nova = malloc(sizeof(Celula));
  nova->item = novo;
  nova->prox = p->topo;
  p->topo = nova;
}
```

Desempilha

Vamos desempilhar uma célula, desde que a pilha não esteja vazia. Para isso precisamos "desligar" aquela célula que estava no topo e ligar o apontador topo com a próxima:



Podemos implementar da seguinte forma:

```
void desempilha(Pilha *p) {
  if(verifica_pilha_vazia(p)) {
    printf("Erro: pilha vazia!\n");
    return ;
  }
  Celula *remover = p->topo;
  p->topo = remover->prox;
  free(remover);
  }
}
```

Liberando a pilha

Um laço deve ser feito para liberar cada célula ocupada e então o apontador para o topo é liberado:

```
void libera_pilha(Pilha *p) {
  Celula *aux = p->topo;
  Celula *liberar;
  while(aux != NULL) {
      liberar = aux;
      aux = aux->prox;
      free(liberar);
  }
  free(p);
}
```

Outra funções úteis

A função imprime pilha é fundamental para acompanhar o estado da pilha. Para imprimir a pilha é necessário percorrer toda a pilha partindo do topo:

```
void imprime(Pilha *p) {
      Celula *aux;
      for(aux = p->topo; aux != NULL; aux = aux->prox)
         printf("chave = %d\n", aux->item.chave);
}
```

Um laço semelhante com um contador dentro pode ser feito para contar a quantidade de itens da pilha:

```
int tamanho_pilha(Pilha *p) {
   Celula *aux = p->topo;
   int i = 0;
   while(aux != NULL) {
      aux = aux->prox;
      i++;
   }
   return i;
}
```

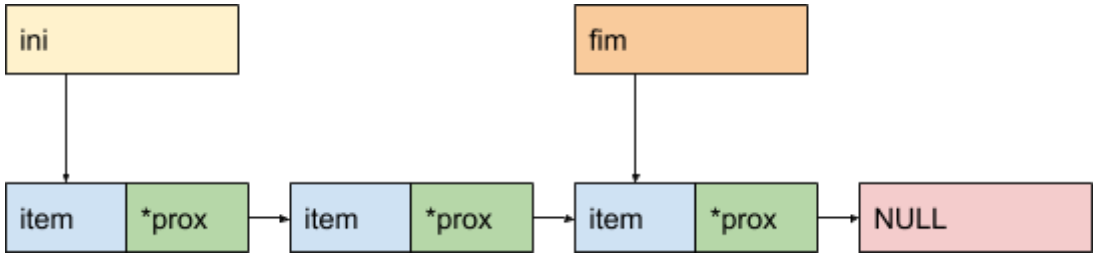
Estrutura da FILA ENCADEADA

Assim como na pilha, temos uma estrutura tipo Item e uma estrutura tipo tipo Celula que aponta para quem é a próxima célula de fila. A diferença agora é que na estrutura fila é conveniente guardar o início e o fim da fila, já que vamos remover do início (desenfileirar) e adicionar no fim (enfileirar):

```
typedef struct item Item;
typedef struct celula Celula;
typedef struct fila Fila;
```

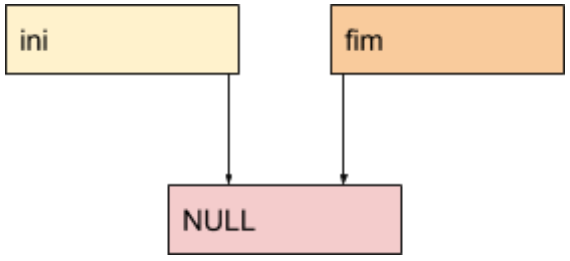
```c
struct item {
  int chave;
  // demais campos
};
struct celula {
  Item item;
  Celula *prox;
};
struct fila {
  Celula *ini;
  Celula *fim;
};
```

Podemos visualizar a fila assim:



Criação de fila vazia

Para criar a fila vazia, tanto ini quanto fim devem apontar para NULL:



Em C, temos:

```c
Fila * cria_fila_vazia() {
  Fila *f = malloc(sizeof(Fila));
  f->ini = NULL;
  f->fim = NULL;
  return f;
}
```

Verificar fila vazia

Podemos dizer que a fila está vazia se o seu início ainda aponta para NULL:

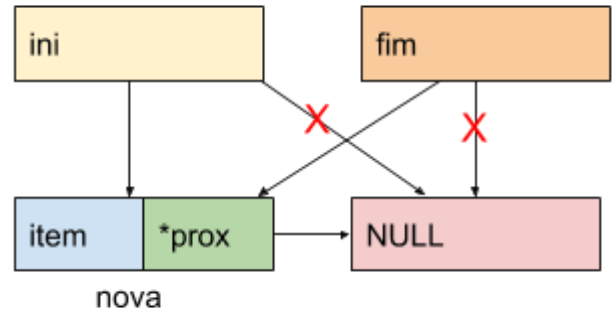
```c
int verifica_fila_vazia(Fila *f) {
  return f->ini == NULL;
}
```

Enfileira

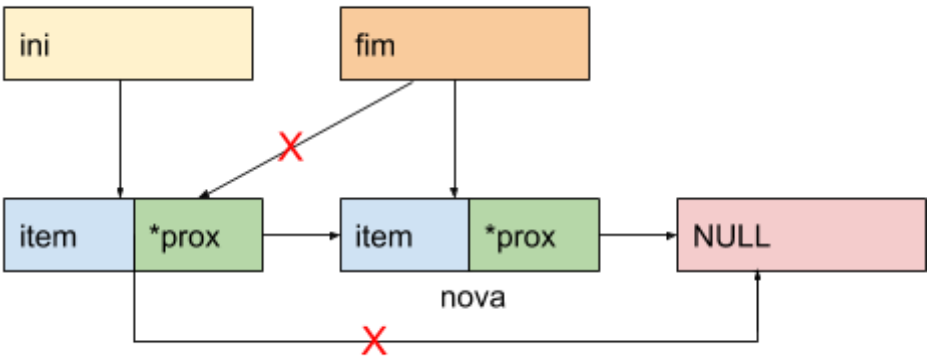
Uma nova célula será adicionada no final da fila, portanto ela deverá apontar para NULL. Feito isso, precisamos tratar duas situações:

- se a fila está vazia, quem deve apontar para a nova célula é o apontador ini;
- se a fila não está vazia, quem deve apontar para a nova célula é a célula que era apontada pelo fim.
  Em ambos os casos, após a correta ligação da nova célula, o apontador fim deve apontar para a nova célula.

Enfileirar na fila vazia:



Enfileirar na fila não vazia:



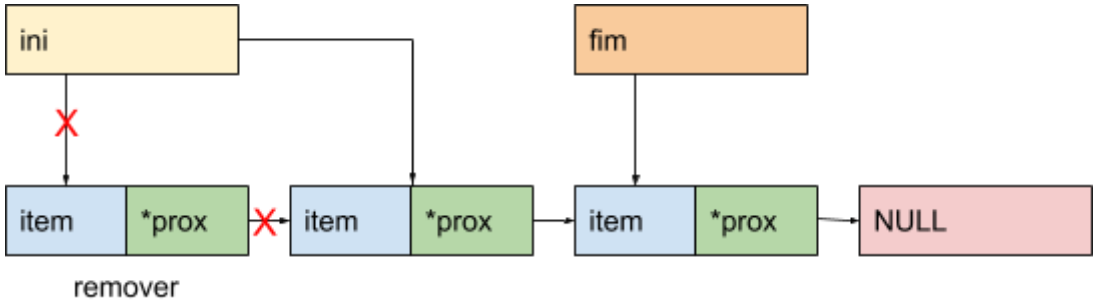
Nó código abaixo, o if/else serve para tratar as situações discutidas acima:

```c
void enfileira(Fila *f, int chave) {
  Item novo_item;
  novo_item.chave = chave;
  Celula *nova = malloc(sizeof(Celula));
  nova->item = novo_item;
  nova->prox = NULL;
  if(verifica_fila_vazia(f)) // se está vazia adiciona no início
      f->ini = nova;
```

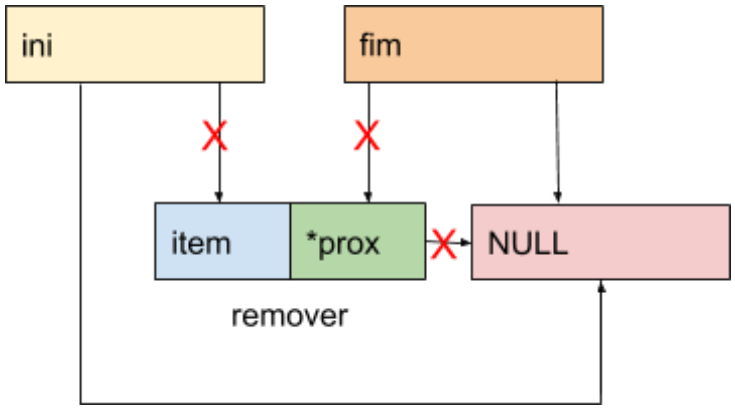
```
  else
      f->fim->prox = nova;
  f->fim = nova;
}
```

Desenfileira

Como de costume, esta operação irá acontecer se a fila não estiver vazia. A célula que vai ser removida é aquela que foi adicionada menos recentemente, ou seja, a célula apontada por ini. O apontador ini deve agora apontar para a célula que era a próxima da primeira célula:



Antes de finalizar temos que tomar cuidado para ver se a fila não ficou vazia após a remoção. Caso tenha ficado vazia, é necessário apontar fim para NULL.



Em C, temos:

```
void desenfileira(Fila *f) {
  if(verifica_fila_vazia(f)) {
      printf("Erro: a fila está vazia.\n");
      return;
  }
  Celula *remover = f->ini;
  f->ini = remover->prox;
  free(remover);
  if(verifica_fila_vazia(f)) // se ficou vazia, fim aponta para NULL
      f->fim = NULL;
}
```

Liberando a fila

Pode ser implementado de maneira semelhante ao processo de liberar a pilha:

```
void libera_fila(Fila *f) {
  Celula *aux = f->ini;
  Celula *liberar;
  while(aux != NULL) {
      liberar = aux;
      aux = aux->prox;
      free(liberar);
  }
  free(f);
}
```

Outras funções úteis

Imprimir a fila:

```
void imprime_fila(Fila *f) {
  Celula *aux = f->ini;
  while(aux != NULL) {
      printf("Chave: %d\n", aux->item.chave);
      aux = aux->prox;
  }
}
```

Contar quantos elementos tem na fila (poderia também ter criado um indicador de tamanho na estrutura fila, como foi feita na implementação por vetor. Assim, não precisaria de função, bastaria incrementar este tamanho na função de enfileirar e decrementar na função de desenfileirar):

```
int tamanho_fila(Fila *f) {
  Celula *aux = f->ini;
  int cont = 0;
  while(aux != NULL) {
      aux = aux->prox;
      cont++;
  }
  return cont;
}
```

Obs.: como neste tipo de fila os itens não ficam em posições contíguas de memória (eles vão sendo alocados dinamicamente conforme a necessidade) não precisamos nos preocupar em implementar uma fila circular (a não ser que o problema a ser resolvido exija que o final da fila aponte para o início).

Links interessantes:

Vídeo aulas sobre pilhas e filas encadeadas do canal Linguagem C Programação Descomplicada, com prof. André Backes:

https://www.youtube.com/watch?v=9GGJH2sjOac

https://www.youtube.com/watch?v=4YXnrKJCWrE