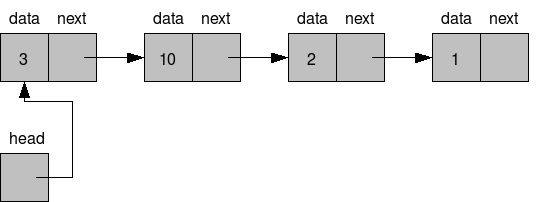
FACIMP – FACULDADE DE IMPERATRIZ

**Professor**: Nonilton

**Aluno**: Rafael Sousa

**O que é uma Lista em C (Lists)**

Use o bom senso e responda: o que é uma lista?  
Vamos supor que você fez uma lista de compras.  
Ou seja, tem um pedaço de papel com diversos elementos, em uma dada ordem.  
Provavelmente essa ordem é a que você vai comprar, do primeiro para o último. Geralmente elas tem uma sequência lógica, como elementos do mesmo setores estarem adjacentes.  
Depois você vai trabalhar com essa lista, seguir a ordem dela, marcar se comprou ou não, anotar o preço etc.  
A analogia é parecida para uma lista em C.  
Em programação, lista é uma série de elementos ligados. O primeiro é ligado no segundo, que é ligado no terceiro etc.



**Listas encadeadas** são estruturas de dados lineares e dinâmicas, a grande vantagem que elas possuem em relação ao uso de vetor é o fato de terem tamanho máximo relativamente infinito (o tamanho máximo é o da memória do computador), ao mesmo tempo que podem ter o tamanho mínimo de 1 elemento evitando o desperdício de memória.

Primitivas

Não existe nenhuma normalização quanto as primitivas usadas para a manipulação de uma lista.  
Abaixo você pode ver uma lista com algumas delas .

* Colocar o índice sobre o primeiro elemento da lista.
* Colocar o índice sobre o último elemento da lista .
* Colocar o índice sobre o elemento que segue o elemento atual .
* Colocar o índice sobre o elemento que precede o elemento atual .
* Verificar se a lista está vazia : Se a lista estiver vazia retorna verdadeiro, se não, falso.
* Verificar se é o primeiro elemento : Retorna verdadeiro se o elemento atual é o primeiro, se não, falso.
* Verificar se é o último elemento  : Retorna verdadeiro se o elemento atual é o último, se não, falso.
* Verificar o número de elementos da lista : Retorna o número de elementos da lista.
* Adicionar um elemento no início : Adicionar um elemento antes do primeiro elemento da lista .
* Adicionar um elemento no fim : Adicionar um elemento depois do último elemento da lista .
* Inserção : Inserir um elemento antes do elemento atual .
* Troca : Trocar o elemento atual .
* Remoção : Remover o elemento atual .
* Listar todos os elementos da lista .

## Lista encadeada linear

Cada nó ou elemento de uma lista encadeada irá possuir guardar o valor do nó e o endereço do próximo nó. Em uma lista encadeada linear o ultimo elemento aponta para NULL .

**struct** No{

char \*p\_dados;

**struct** No \*p\_prox;

};

## Iniciar uma lista

A função abaixo demonstra como iniciar uma lista criando o espaço da raiz na memória.

void criar\_Lista(**struct** No \*\*p\_Raiz){

\*p\_Raiz = NULL;

}

## Inserção

Existem 3 tipos de inserção em uma lista, pode-se inserir no começo, no final ou entre dois elementos da lista.

### Inserção no início

int inserir\_No\_Inicio (**struct** No \*\*p\_Raiz, char \*p\_String){

**struct** No \*p\_Novo;

*/\*\* Alocação dinâmica da memoria \*/*

p\_Novo = (**struct** No \*) malloc(**sizeof**(**struct** No));

**if**( p\_Novo == NULL ){

puts( "Falta Memoria**\n**");

**return** -1 ;

}

p\_Novo->p\_dados = p\_String;

p\_Novo->p\_prox = \*p\_Raiz;

\*p\_Raiz = p\_Novo;

}

### Inserção no fim

int inserir\_No\_Fim(**struct** No \*\*p\_Raiz, char \*p\_String){

**struct** No \*p\_Novo;

**struct** No \*e\_atual;

p\_Novo = (**struct** No \*) malloc(**sizeof**(**struct** No));

**if**(p\_Novo == NULL){

puts("Falta Memoria**\n**");

**return** -1 ;

}

p\_Novo->p\_dados = p\_String;

p\_Novo->p\_prox = NULL;

**if**(\*p\_Raiz == NULL) *//Lista vazia*

\*p\_Raiz = p\_Novo;

**else**{

e\_atual = \*p\_Raiz; */\*@ Primeiro elemento\*/*

**while**(e\_atual->p\_prox != NULL){

e\_atual = e\_atual->p\_prox;

}

e\_atual->p\_prox = p\_Novo;

}

}

## Remoção

Assim como na inserção também existem 3 tipos de remoção, no início, no fim ou entre dois elementos da lista.

### Remoção no início

void remover\_No\_Inicio(**struct** No \*\*p\_Raiz){

**if**(\*p\_Raiz == NULL) printf("**\n**A lista ja esta vazia**\n**");

**else**{

**struct** No \*p\_atual;

p\_atual = \*p\_Raiz;

\*p\_Raiz = (\*p\_Raiz)->p\_prox;

free(p\_atual);

}

}

### Remoção no fim

void remover\_No\_Fim(**struct** No \*\*p\_Raiz){

**if**(\*p\_Raiz == NULL) printf("**\n**A lista ja esta vazia");

**else**{

**struct** No \*p\_atual, \*p\_anterior ;

p\_atual = \*p\_Raiz;

**while**(p\_atual->p\_prox != NULL){

p\_anterior = p\_atual ;

p\_atual = p\_atual->p\_prox;

}

p\_anterior->p\_prox = NULL;

free(p\_atual);

}

}

## Exibição

### Do fim para a raiz

void mostrar\_Do\_Fim\_Para\_Raiz(**struct** No \*p\_Raiz){

**if**(p\_Raiz == NULL) printf("**\n**Lista vazia");

**else**{

**struct** No \*p\_Atual\_Corredor, \*p\_Atual\_Fim;

p\_Atual\_Corredor = p\_Raiz;

p\_Atual\_Fim = p\_Raiz;

**while**(p\_Atual\_Fim->p\_prox != NULL){ *//ir para o ultimo elemento*

p\_Atual\_Fim = p\_Atual\_Fim->p\_prox;

}

**while**(p\_Atual\_Corredor != p\_Atual\_Fim){

**if**(p\_Atual\_Corredor->p\_prox == p\_Atual\_Fim){

printf(" <- %s", p\_Atual\_Fim->p\_dados);

p\_Atual\_Fim = p\_Atual\_Corredor;

p\_Atual\_Corredor = p\_Raiz;

}

**else** p\_Atual\_Corredor = p\_Atual\_Corredor->p\_prox;

}

printf(" <- %s", p\_Atual\_Fim->p\_dados);

}

}

### Da raiz para o fim

void mostrar\_Da\_Raiz\_Para\_Fim(**struct** No \*p\_Raiz){

**if**(p\_Raiz == NULL) printf("**\n**Lista vazia");

**else**{

**struct** No \*p\_atual;

p\_atual = \*p\_Raiz;

**while**(p\_atual != NULL){

printf("%s", p\_atual->p\_dados);

p\_atual = p\_atual->p\_prox;

}

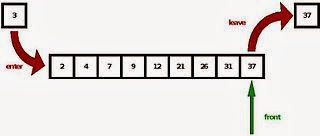
}

}

**Filas em C (Queue)**

Outro importante conceito de estrutura de dados dinâmica são as filas, que são exatamente iguais as do mundo real.  
Como funciona a fila de um caixa eletrônico?

Chega a primeira pessoa, é atendida. Já a segunda, fica na fila, e será atendida depois da primeira.  
A terceira a chegar só vai ser atendida depois da segunda, e assim sucessivamente.  
Ou seja, os primeiros elementos a chegar serão os primeiros a serem atendidos.  
Um termo muito conhecido para designar tal tipo de ideia é FIFO - First In, First Out (Primeiro que entra, primeiro que sai).  
Em termos de programação, dizemos que os elementos que chegam vão para a cauda da fila, ou seja, para o final, serão atendidos por último. Os elementos que serão primeiro atendidos são os que estão na cabeça da fila (na frente).  
Também chamamos o ato de colocar algo na fila de ENQUEUE, e de tirar de DEQUEUE.



# Fila

Uma fila ou queue em inglês é uma estrutura de dados que usa o método FIFO(acrônimo para First In, First Out, que em português significa primeiro a entrar, primeiro a sair).  
A idéia fundamental da fila é que só podemos inserir um novo elemento no final da fila e só podemos retirar o elemento do início.  
**Exemplo de fila em C:**

#include *<stdio.h>*

#include *<string.h>*

#include *<stdlib.h>*

void q\_enter(void);

void q\_list(void);

int q\_store(char \*ptr);

int q\_delete(void);

int count = 0; *//contador*

int store = 0; *// proxima posição na fila*

int retrieve = 0; *// recupera a posição da fila*

char \*queue[100]; *// vetor da fila*

int main()

{

int i = 0;

**for** ( i = 0; i < 100; i++ )

{

queue[i] = NULL;

}

q\_enter(); *// entra os dados na fila*

printf("**\n\n**Todos os dados da fila (FIFO):**\n**");

q\_list();

q\_delete(); *// Apaga a primeira entrada da fila*

printf("**\n\n**A fila depois delete(FIFO):**\n**");

q\_list();

printf("**\n\n**Numero de elementos restantes na fila: %i **\n**", count);

getchar(); *// espera*

**return** 0;

}

void q\_enter(void)

{

**static** char str[100], \*ptr;

puts("Pressione a tecla ENTER sem nome pra sair**\n**");

**do** {

printf("Entre o nome:");

gets(str);

ptr = (char \*) malloc(strlen(str)); *//alocar um espaço na memória*

strcpy(ptr,str);

**if** (\*str)

{

count++;

q\_store(ptr); *// Guarda o endereço da seqüência de caracteres*

}

} **while** (\*str); *//Sair se não houver uma entrada*

}

*// listar a fila*

void q\_list(void)

{

int k;

**for** (k = retrieve; k < store; k++)

{

printf("Elemento %d : %s **\n**",k+1,queue[k]);

}

}

*// Guarda os itens na fila*

int q\_store(char \*ptr)

{

**if** (store == 100) {

printf("**\n**A lista esta cheia!**\n**");

**return** 0 ;

}

queue[store] = ptr;

store++; *// próximo índice da fila*

}

*// Apaga um item da fila*

int q\_delete(void)

{

**if** (store == retrieve) {

printf("**\n**A fila esta vazia!");

**return** 0 ;

}

count--;

retrieve++;

}

**Pilhas em C (Stack)**

[](http://1.bp.blogspot.com/-LfkJb6wrX4w/UlVs1ZfwhHI/AAAAAAAAAj4/E0y_PIRJspo/s1600/Pilha-stack.jpg)Outra importante estrutura dinâmica de dados são as pilhas (stacks, em inglês), que tem um funcionamento contrário ao das filas. São ditas do tipo LIFO - Last In, First Out (Ultimo a entrar, primeiro a sair).  
  
Para entender esse tipo de estrutura, podemos imaginar uma pilha de pratos.  
Você come uma coisa, e guarda o prato.  
Come a segunda coisa, e põe o segundo prato em cima do primeiro.  
Come a terceira e põe este prato em cima do segundo, e assim sucessivamente.  
  
Quando você for lavar, que prato vai retirar primeiro? O de cima.  
E qual o último prato a ser retirado? O primeiro.  
Ou seja, você vai tirando os pratos de cima, que são os pratos que chegaram por último na pilha.  
Já os pratos que chegaram primeiro na fila serão os últimos a saírem dela.  
  
Dois termos comum nesse tipo de ordenação de estruturas são PUSH e POP.  
PUSH é colocar um elemento na pilha, e POP é tirar.

## Pilha

Pilha ou stack é uma lista linear em que todas as inserções e remoções de elemento só podem ser feitos em uma extremidade chamada topo.As pilhas também são chamadas de estruturas LIFO (Last In First Out) ou seja o último elemento inserido é o primeiro removido.

## Construção do protótipo de um elemento da lista

**typedef** **struct** Elemento\_da\_lista{

char \*dados;

**struct** Elemento\_da\_lista \*proximo;

}Elemento;

**struct** Localizar{

Elemento \*inicio;

int tamanho;

} Pilha;

## Inicialização

void iniciar (Localizar \*monte){

monte->inicio = NULL;

monte->tamanho = 0;

}

## Inserir um elemento na pilha(push)

**Algoritmo:**  
Declaração do elemento(s) a ser inserido.  
Alocação da memória para o novo elemento  
Inicializar o campo de dados.  
Preencher o ponteiro inicio com o primeiro elemento  
Colocar em dia o tamanho da pilha.

int empilhar(Localizar \* monte, char \*dados){

Elemento \*novo\_elemento;

**if** ((novo\_elemento = (Elemento \*) malloc (**sizeof** (Elemento))) == NULL)

**return** -1;

**if** ((novo\_elemento->dados = (char \*) malloc (50 \* **sizeof** (char)))

== NULL)

**return** -1;

strcpy (novo\_elemento->dados, dados);

novo\_elemento->proximo = monte->inicio;

monte->inicio = novo\_elemento;

monte->tamanho++;

}

## Retirar um elemento da pilha (pop)

int desempilhar (Localizar \*monte){

Elemento \*p\_elemento;

**if** (monte->tamanho == 0)

**return** -1;

p\_elemento = monte->inicio;

monte->inicio = monte->inicio->proximo;

free (p\_elemento->dados);

free (p\_elemento);

monte->tamanho--;

**return** 0;

}

## Imprimir os elementos da pilha

void mostrar(Localizar \* monte){

Elemento \*atual;

int i;

atual = monte->inicio;

**for**(i=0;i<monte->tamanho;++i){

printf("**\t\t**%s**\n**", atual->dados);

atual = atual->proximo;

}

}