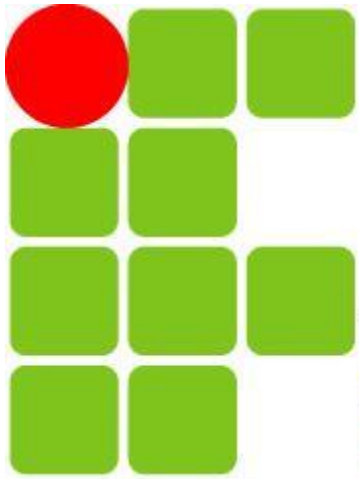


INSTITUTO FEDERAL
ESPÍRITO SANTO
Campus Cachoeiro de Itapemirim

# Listas Simplesmente Encadeadas

## Programação 1

Rafael Vargas Mesquita

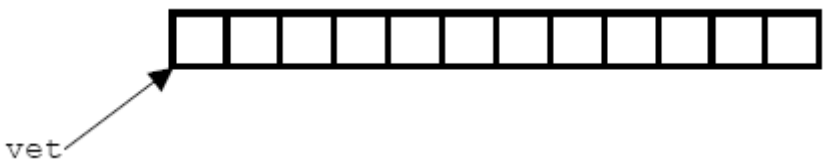
http://www.ci.ifes.edu.br
ftp://ftp.ci.ifes.edu.br/informatica/rafael/

# Vetores

- Para representarmos um grupo de dados, já vimos que podemos usar um vetor em C. O vetor é a forma mais primitiva de representar diversos elementos agrupados.

```
#define MAX 12
...
main(){
   int vet[MAX];
...
}
```

  - Ao declararmos um vetor, reservamos um espaço contíguo(sequencial) de memória para armazenar seus elementos, conforme ilustra a figura abaixo.



# Vetores

- O fato do vetor ocupar um espaço contíguo na memória, nos permite acessar qualquer um de seus elementos a partir do ponteiro para o primeiro elemento.
- De fato, o símbolo vet, após a declaração do slide anterior, representa um ponteiro para o primeiro elemento do vetor, isto é, o valor de vet é o endereço da memória onde o primeiro elemento do vetor está armazenado.
- De posse do ponteiro para o primeiro elemento, podemos acessar qualquer elemento do vetor através do operador de indexação vet[i].

# Vetores

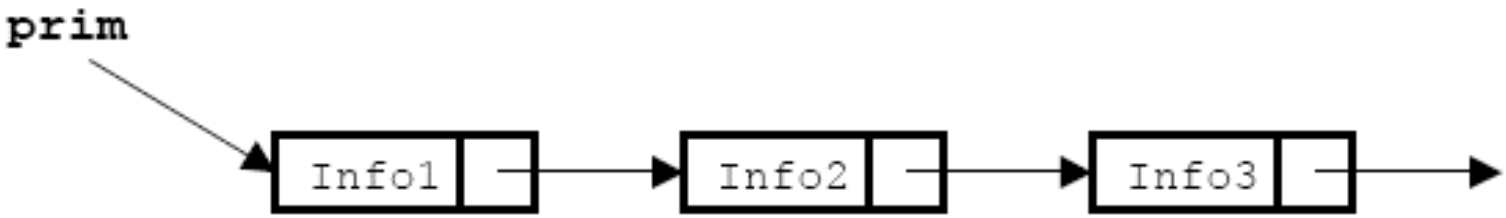
- No entanto, o vetor não é uma estrutura de dados muito flexível, pois precisamos dimensioná-lo com um número máximo de elementos. Se o número de elementos que precisarmos exceder a dimensão do vetor, teremos um problema, pois não existe uma maneira simples e barata (computacionalmente) para alterarmos a dimensão do vetor em tempo de execução.
- Por outro lado, se o número de elementos que precisarmos armazenar no vetor for muito inferior à sua dimensão, estaremos subutilizando o espaço de memória reservado.
- Uma possível solução é a utilização de vetores dinâmicos, mas essa abordagem ainda obriga o conhecimento do tamanho do vetor a ser utilizado.

# Listas Simplesmente Encadeadas

- Definição
  - É uma estrutura de dados cujo espaço alocado para ela é proporcional ao número de elementos nela armazenado. No entanto não é possível acessar seus elementos por um ou mais índices diretamente.

# Listas Simplesmente Encadeadas

- Representação Gráfica
  - Uma lista simplesmente encadeada pode ser representada como na figura abaixo:



# Listas Simplesmente Encadeadas

- Declaração

```
typedef struct no {
    int info;
    struct no* prox;
} No;

typedef struct lista {
    No* prim;
} Lista;
```

# Listas Simplesmente Encadeadas

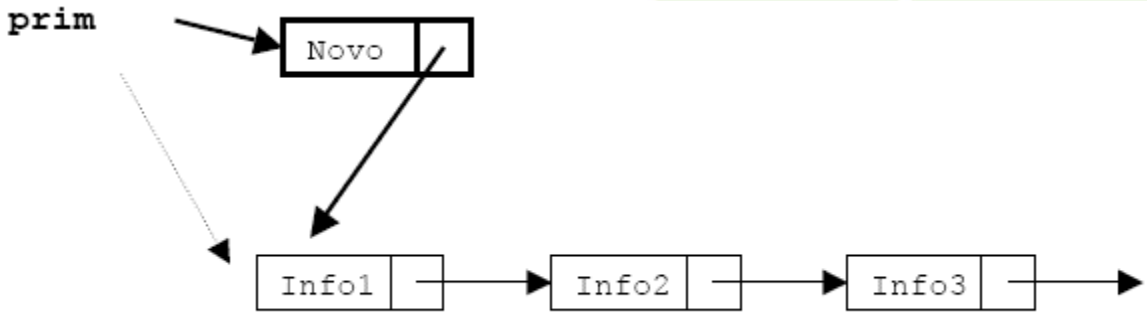
- Funções
  - Função de Inicialização de uma Lista

```
/* função de inicialização: retorna lista vazia */
Lista *inicializa (void){
  Lista *nova = (Lista *) malloc(sizeof(Lista));
  nova->prim = NULL;
  return nova;
}
```

# Listas Simplesmente Encadeadas

- Funções
  - Função de Inserção de Elementos na Lista

```
// insere elemento novo na lista (no início)
void insere (Lista *l, int v){
   No *novo = (No*) malloc(sizeof(No));
   novo->info = v;
   novo->prox = l->prim;
   l->prim = novo;
}
```



# Listas Simplesmente Encadeadas

- Funções
  - A seguir um pequeno trecho de código utilizando as funcionalidades descritas até esse slide:

```
int main (){
    Lista* l; /* declara uma lista não inicializada */
    l = inicializa(); /* inicializa lista como vazia */
    insere(l, 23); /* insere na lista o elemento 23 */
    insere(l, 45); /* insere na lista o elemento 45 */
    ...
    return 0;
}
```

# Listas Simplesmente Encadeadas

- Funções
  - Função de Impressão de uma Lista

```
void imprime (Lista *l){
   No *aux; /* variável auxiliar para percorrer a lista*/
   for (aux = l->prim; aux != NULL; aux = aux->prox)
       printf("\t\tInfo = %d\n", aux->info);
}
```

# Listas Simplesmente Encadeadas

- Funções
  - Função de Verificação de Lista Vazia

```
// retorna 1 se vazia ou 0 se não vazia
int vazia (Lista *l){
   return (l->prim == NULL);
}
```

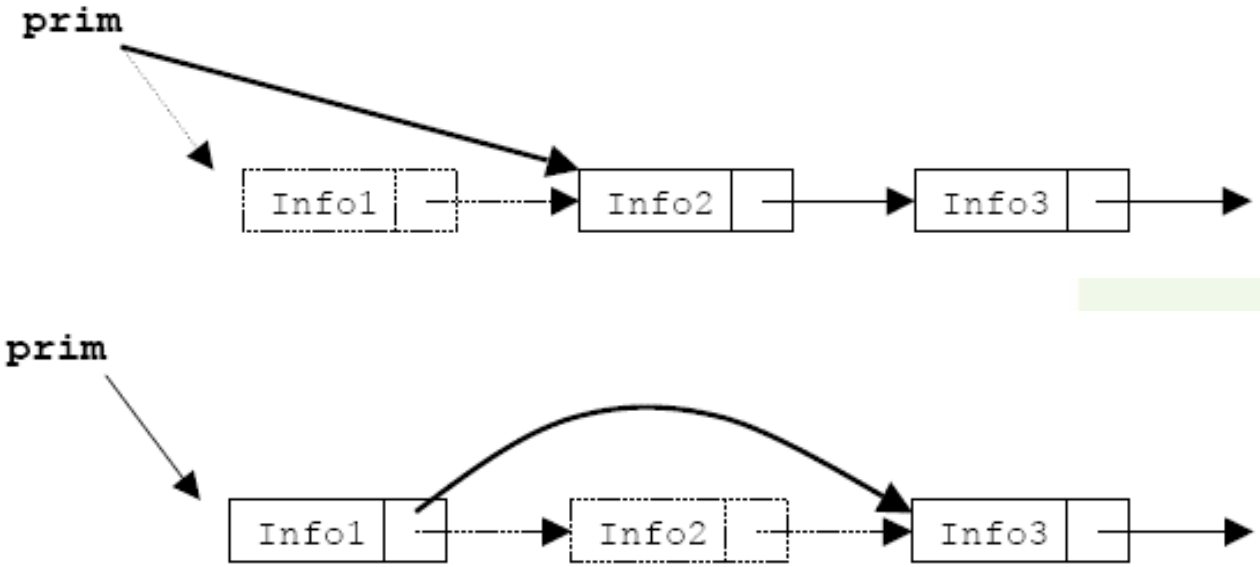
# Listas Simplesmente Encadeadas

- Funções
  - Função de Busca de Elementos

```
int busca (Lista *l, int v){
   No *aux; // variável auxiliar para percorrer a pilha
   for (aux = l->prim; aux != NULL; aux = aux->prox)
       if (aux->info == v)
              return 1;
   // não achou o elemento
   return 0;
}
```

# Listas Simplesmente Encadeadas

- Funções
  - Função de Remoção de Elementos da Lista
  - Graficamente



# Listas Simplesmente Encadeadas

- Funções: Função de Remoção de Elementos da Lista

```
int retira (Lista* l, int v){
        No *ant = NULL; /* ponteiro para elemento anterior */
        No *aux = l->prim; /* ponteiro para percorrer a lista*/
        /* procura elemento na lista, guardando anterior */
        while (aux != NULL && aux->info != v) {
                ant = aux;
                aux = aux->prox;
        }

        if (aux == NULL) /* verifica se achou elemento */
                return 0; /* não achou: retorna lista original */
        /* retira elemento */
        if (ant == NULL) { /* retira elemento do inicio */
                l->prim = l->prim->prox;
        }
        else {/* retira elemento do meio da lista */
                ant->prox = aux->prox;
        }
        free(aux);
        return 1;
}
```

# Listas Simplesmente Encadeadas

- Funções
  - Função de Liberação de uma Lista

```
void libera (Lista* l){
  Lista* p = l;
  while (p != NULL) {
    /* guarda referência para o próximo elemento */
    Lista* t = p->prox;
    free(p); /* libera a memória apontada por p */
    p = t; /* faz p apontar para o próximo */
  }
}
```

# Listas Simplesmente Encadeadas

- Funções
  - A seguir um pequeno trecho de código utilizando as funcionalidades descritas até esse slide:

```
#include <stdio.h>
int main (void) {
  Lista* l; /* declara uma lista não iniciada */
  l = inicializa(); /* inicia lista vazia */
  insere(l, 23); /* insere o elemento 23 */
  insere(l, 45); /* insere o elemento 45 */
  insere(l, 56); /* insere o elemento 56 */
  insere(l, 78); /* insere o elemento 78 */
  imprime(l); /* imprimirá: 78 56 45 23 */
  retira(l, 78);
  imprime(l); /* imprimirá: 56 45 23 */
  retira(l, 45);
  imprime(l); /* imprimirá: 56 23 */
  libera(l);
  return 0;
}
```

# Bibliografia


- SANTOS, Henrique José. Curso de Linguagem C da UFMG, apostila.
- FORBELLONE, André Luiz. Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados. São Paulo: MAKRON, 1993.