



Estrutura_módulo_1_1_2_3
_(C,_Vetores,_Struct)

Prof. MSc. Olavo Ito

Ambiente da prática

Google Colab

O Colab, ou “Colaboratory”, permite escrever e executar Python no navegador e conta com o seguinte:

- Nenhuma configuração necessária.
- Acesso gratuito a GPUs.
- Compartilhamento fácil.
- Trabalha em ambiente Linux.
- Utiliza os recursos da linguagem C do Colab.

Iniciando o ambiente

- Carregando o notebook
- Editar em célula de código
- Editor
- Compilação
- Comandos auxiliares (`rm -rf ./` e `ls -l`)
- Estrutura módulo 1-1:2:3 (C, Vetores, *Struct*)

Entrada e saída em C

Entrada e saída

- `stdio.h`
- `printf(formato, argumentos);` Função para saída de valores segundo um determinado formato.
- `scanf(formato, lista de endereços);` Função para capturar e armazenar valores fornecidos via teclado.

Máscara	tipo
%c	char
%d	int
%u	unsigned int
%f	double ou float
%e	double ou float (científico)
%s	cadeia de caracteres
\n	quebra de linha
\t	tabulação
\”	caractere ”
\\	caractere \

Vetores em C

Vetores e matrizes

- Declaração
 - `int v[10];`
 - `int v [5] = { 5, 10, 15, 20, 25 };`
 - Ou
 - `int v[] = { 5, 10, 15, 20, 25 };`
- O acesso a cada elemento do vetor é feito por meio de uma indexação da variável `v`.
- Observamos que, em C, a indexação de um vetor varia de zero a $n-1$, em que n representa a dimensão do vetor.

Vetor de caracteres e *Strings*

Vetor de caracteres e Strings

- As cadeias de caracteres em C são representadas por vetores do tipo char terminadas, obrigatoriamente, pelo caractere nulo ('\0').
- Para armazenarmos uma *string*, devemos reservar uma posição adicional para o caractere de fim da cadeia.
- Todas as funções que manipulam cadeias de caracteres recebem como parâmetro um vetor de char, processam caractere por caractere, até encontrarem o caractere nulo, que sinaliza o final da cadeia.

Estrutura em C

Estrutura em C

- Em C, o tipo registro é conhecido como estrutura. Uma estrutura em C serve basicamente para agrupar diversas variáveis dentro de um único contexto.
- A estrutura, uma vez criada, passa a ser utilizável dentro do programa atuando como variáveis.

Link

- <https://tinyurl.com/edpratica01>

ATÉ A PRÓXIMA!



Estrutura_módulo _1_3A_(Funções)

Prof. MSc. Olavo Ito

Funções

Nó

Em C, tudo é feito por meio de funções. Já vínhamos utilizando alguns como os da biblioteca `stdio.h` para entrada e saída de dados. A forma geral para definir uma função é:

```
<tipo retornado> <nome da função> (<lista de parâmetros>...) {  
    corpo da função....;  
    return <variável do tipo declarado no início da função>;  
}
```

- Em caso de procedimentos, o tipo deverá ser `void`.

Variáveis locais e globais


Variáveis locais e globais

```
#include <stdio.h>
```

```
Int a, b,c;
```

```
float d, e,f;
```

Variáveis globais

Two blue arrows originate from the text 'Variáveis globais' and point to the declarations 'Int a, b,c;' and 'float d, e,f;'.

```
int A(int n){
```


```
    int x,y;
```

```
    x=a+y+n;
```

```
    return x ;
```

```
}
```

Variáveis locais da função A

Two blue arrows originate from the text 'Variáveis locais da função A' and point to the declarations 'int x,y;' and 'return x;' inside function A.


```
float B(int n){
```

```
    float x,y;
```

```
    x=d+e/n;
```

```
    return x ;
```

Variáveis locais da função B

Two blue arrows originate from the text 'Variáveis locais da função B' and point to the declarations 'float x,y;' and 'return x;' inside function B.

Link

- <https://tinyurl.com/edpratica02>

ATÉ A PRÓXIMA!



Estrutura_módulo _1_4(Ponteiros)

Prof. MSc. Olavo Ito

Ponteiro de variáveis

Ponteiro de variáveis

- A linguagem C tem uma maneira especial de uma variável armazenar endereços

`<tipo>* nome;`

`int *p;`

- O programa reserva um espaço na memória para uma variável chamada p que irá guardar um endereço, e nesse endereço armazenado conterá uma informação do tipo inteiro. O símbolo * identifica que uma variável é do tipo ponteiro.
- & ("endereço de"), aplicado a variáveis, resulta no endereço da posição da memória reservada para a variável.
 - * ("conteúdo de"), aplicado a variáveis do tipo ponteiro, acessa o conteúdo do endereço de memória armazenado pela variável ponteiro.

Passagem de valores por valor e por referência

Passagem de valores por valor e por referência

- Passagem por valor

```
void troca(int a, int b){  
  
}  
  
int main(){  
    troca(a,b);  
}
```

- Passagem por referência:

```
void troca(int *a, int *b){  
  
}  
  
void main(){  
    troca(&a,&b);  
}
```

Alocação de memória

Alocação de memória

```
<tipo>* var = (<tipo> *) malloc(sizeof(<tipo>));
```

```
float * n = (float *) malloc(sizeof(float));
```

```
int * n = (int *) malloc(sizeof(int));
```

```
registro * r = (registro *) malloc(sizeof(registro));
```

Link

- <https://tinyurl.com/edpratica03>

ATÉ A PRÓXIMA!



Estrutura_módulo _2_1_(lista)

Prof. MSc. Olavo Ito

Listas Ligadas


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

info	proximo

Inserção

Inserindo 1 nó

```
No* lista;  
lista=inicia();  
printf("\nendereço alocado para lista %x conteudo %x ",&lista,lista);
```

```
No* novo_no;  
int tamanho=sizeof(No);  
novo_no = (No*) malloc(tamanho);
```

```
novo_no->info=23;  
novo_no->proximo=lista;
```

Inserindo mais um nó

```
No* lista;
lista=inicia();
printf("\nendereço alocado para lista %x ",lista);
No* novo_no;
int tamanho=sizeof(No);
novo_no = (No*) malloc(tamanho);
novo_no->info=23;
novo_no->proximo=lista;

lista=novo_no;
printf("\nendereço alocado para lista %x conteudo %d proximo:%x",lista,lista->info,
lista->proximo);

novo_no = (No*) malloc(tamanho);
novo_no->info=45;
novo_no->proximo=lista;
```

```
No* insere(No* lista, int num){
    No* novo_no;
    int tamanho=sizeof(No);

    novo_no = (No*) malloc(tamanho);
    novo_no->info=num;
    novo_no->proximo=lista;
    return novo_no;
}
```

```
No* lista;
lista=inicia();
printf("\nendereço alocado para lista %x ",lista);
lista=insere(lista,23);
printf("...",lista,lista->info,lista->proximo);

lista=insere(lista,45);
printf("...",lista,lista->info,lista->proximo);

lista=insere(lista,56);
printf("...",lista,lista->info,lista->proximo);

lista=insere(lista,95);
printf("...",lista,lista->info,lista->proximo);

return 0;
```

Remoção

```

No* retira(No* lista, int num) {
    No* anterior = NULL; /* ponteiro para o no anterior */
    No* atual= lista; /* ponteiro auxiliar para percorrer lista */
    /* procura o no na lista, guardando o no anterior*/
    for (atual=lista;atual!=NULL;atual=atual->proximo){
        if (atual->info==num){
            break; // achou          /* Verifica se achou o no */
        }else{
            anterior=atual;
            if(atual== NULL) {
                return lista; /* naoachou, retorna lista original */
            }
            /* achou, retira o no da lista */
            if(anterior == NULL) {
                lista= atual->proximo; /* retira no do inicio da lista */
            }
            else{
                anterior->proximo= atual->proximo; /* retira no do meio */
            }
            free(atual);
            return lista;
        }
    }
}

```

Link

- <https://tinyurl.com/edpratica03>

ATÉ A PRÓXIMA!



Estrutura_módulo_2_2_3_ Pilhas-Filas

Prof. MSc. Olavo Ito

Pilhas

Criação

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

No
```

info	proximo

```
typedef struct pilha{
    No* topo;
} Pilha;
```

topo

```
void main(){
    Pilha* p = cria();
    .
    .
}
```

Pilha* p = *cria()*;

topo
NULL

```
Pilha* cria(void){
    Pilha* nova_pilha =
(Pilha*)malloc(sizeof(Pilha));
    nova_pilha->topo = NULL;
    return nova_pilha;
}
```

Inserção

```
void push(Pilha* p, int v){  
    p->topo = insere(p->topo, v);  
}
```

```
No* insere(No* lista, int num){  
    No* novo_no = (No*)malloc(sizeof(No));  
    novo_no->info = num;  
    novo_no->proximo = lista;  
    return novo_no;  
}
```

```
void main(){  
    Pilha* p = cria();  
    push(p, 1);  
    push(p, 2);  
    push(p, 3);  
    .  
    .}
```

Inserção

```
void main(){  
    Pilha* p = cria();  
    push(p, 1);  
    push(p, 2);  
    push(p, 3);  
    .  
    .}  
}
```

```
void push(p, 1){  
    p->topo = insere(NULL, 1);  
}  
No* insere(lista NULL, num 1){  
    novo_no #a01 = (No*)malloc(sizeof(No));  
    novo_no #a01 ->info = num 1;  
    novo_no #a01 ->proximo = lista NULL;  
    return novo_no #a01;  
}
```

```
void push(p, 2){  
    p->topo = insere(#a01, 2);  
}  
No* insere(lista #a01, num 2){  
    novo_no #a05 = (No*)malloc(sizeof(No));  
    novo_no #a05 ->info = num 2;  
    novo_no #a05 ->proximo = lista #a01;  
    return novo_no #a05;  
}
```

```
void push(p, 3){  
    p->topo = insere(#a05, 3);  
}  
No* insere(lista #a05, num 3){  
    novo_no #a09 = (No*)malloc(sizeof(No));  
    novo_no #a09 ->info = num 3;  
    novo_no #a09 ->proximo = lista #a05;  
    return novo_no #a09;  
}
```

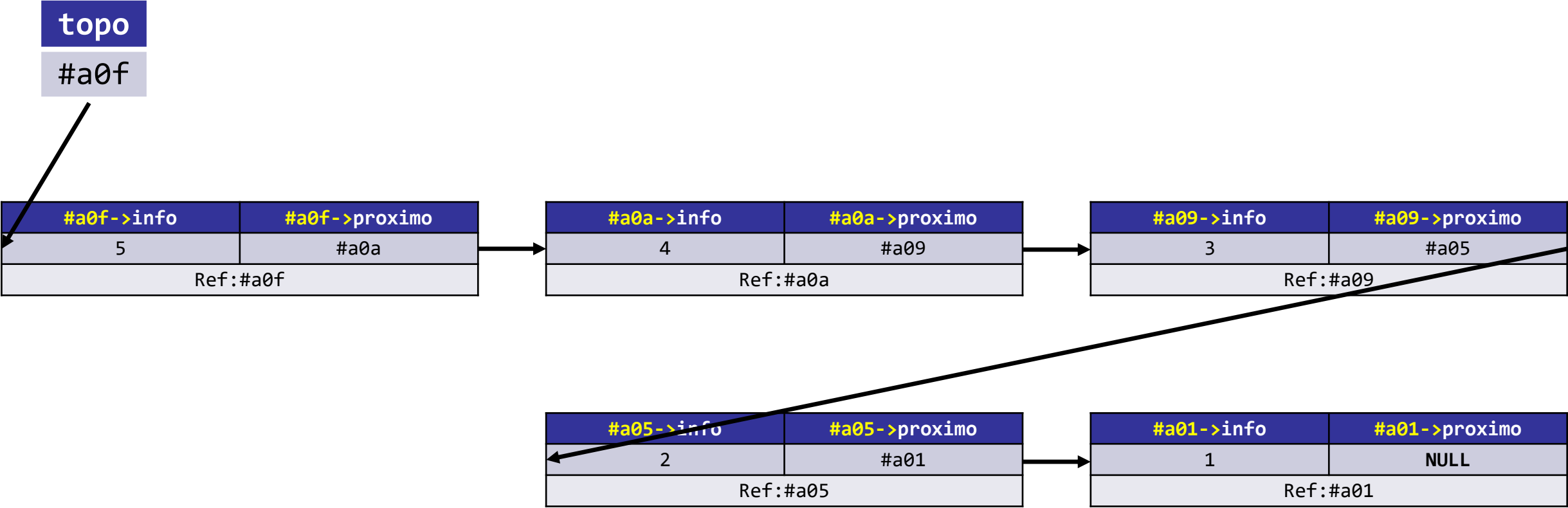
topo
NULL
#a01
#a05
#a09

#a01->info	#a01->proximo
1	NULL
Ref:#a01	

#a05->info	#a05->proximo
2	#a01
Ref:#a05	

#a05->info	#a05->proximo
3	#a05
Ref:#a09	

Inserção

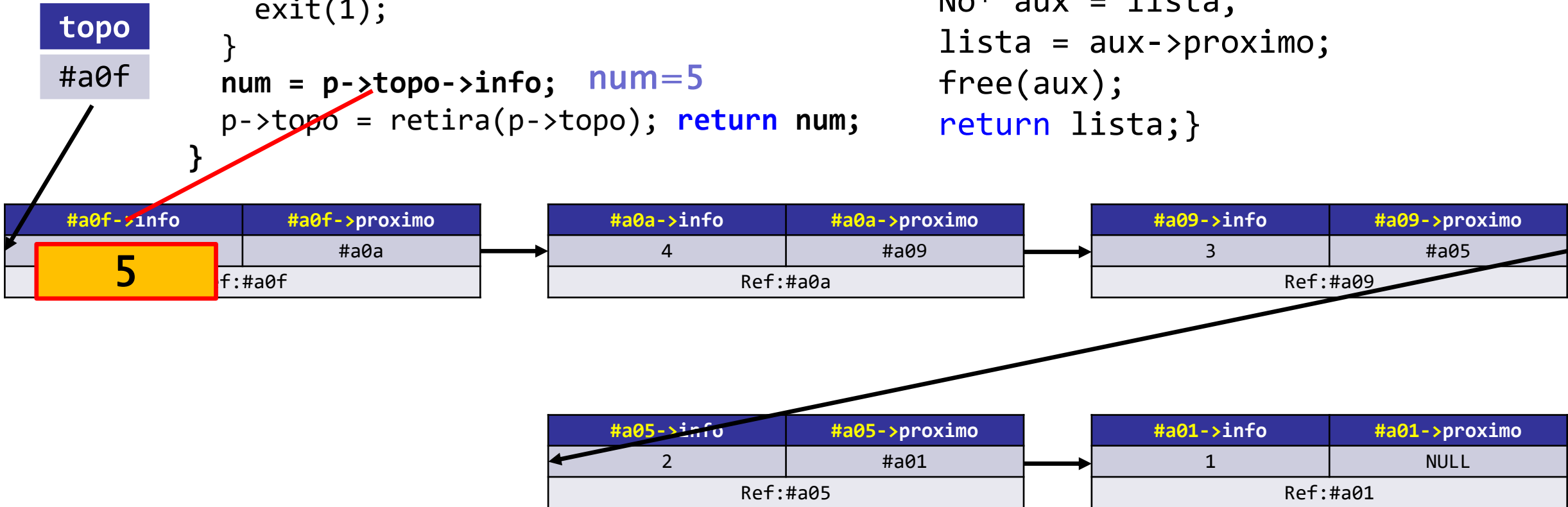


Remoção

```
int pop(Pilha* p){
    int num;
    if(vazia(p)){
        printf("Pilha vazia.\n");
        exit(1);
    }
    num = p->topo->info; num=5
    p->topo = retira(p->topo); return num;
}
```

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

No* retira(No* lista){
    No* aux = lista;
    lista = aux->proximo;
    free(aux);
    return lista;}
```

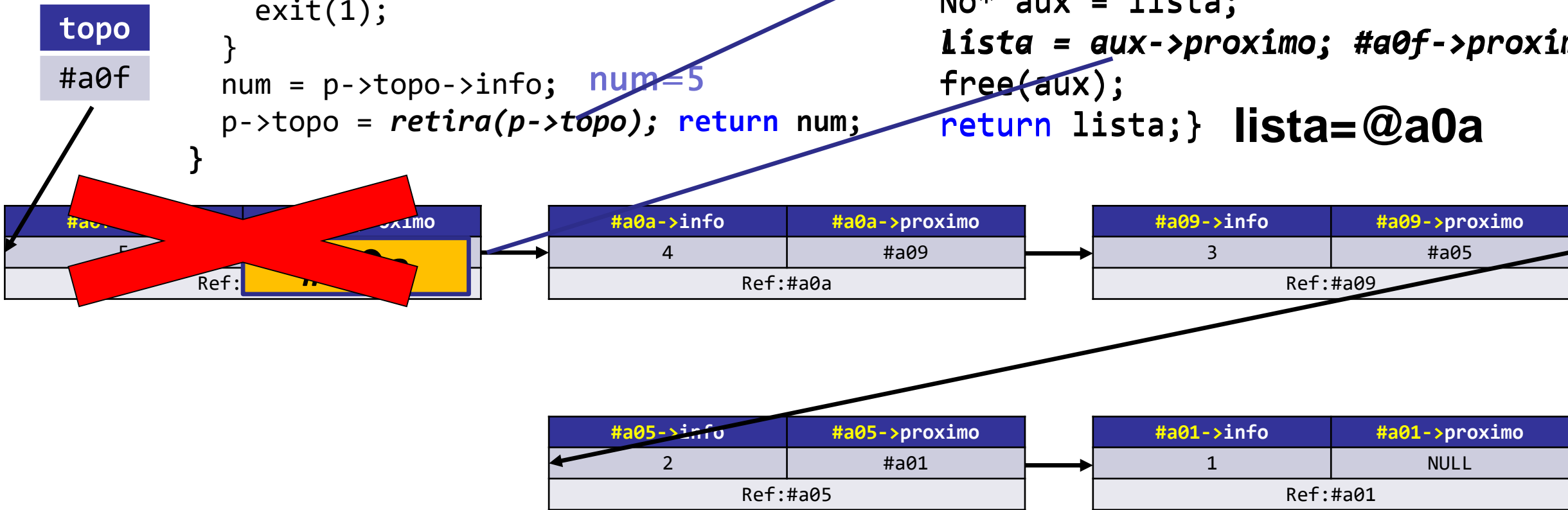


Remoção

```
int pop(Pilha* p){
    int num;
    if(vazia(p)){
        printf("Pilha vazia.\n");
        exit(1);
    }
    num = p->topo->info; num=5
    p->topo = retira(p->topo); return num;
}
```

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

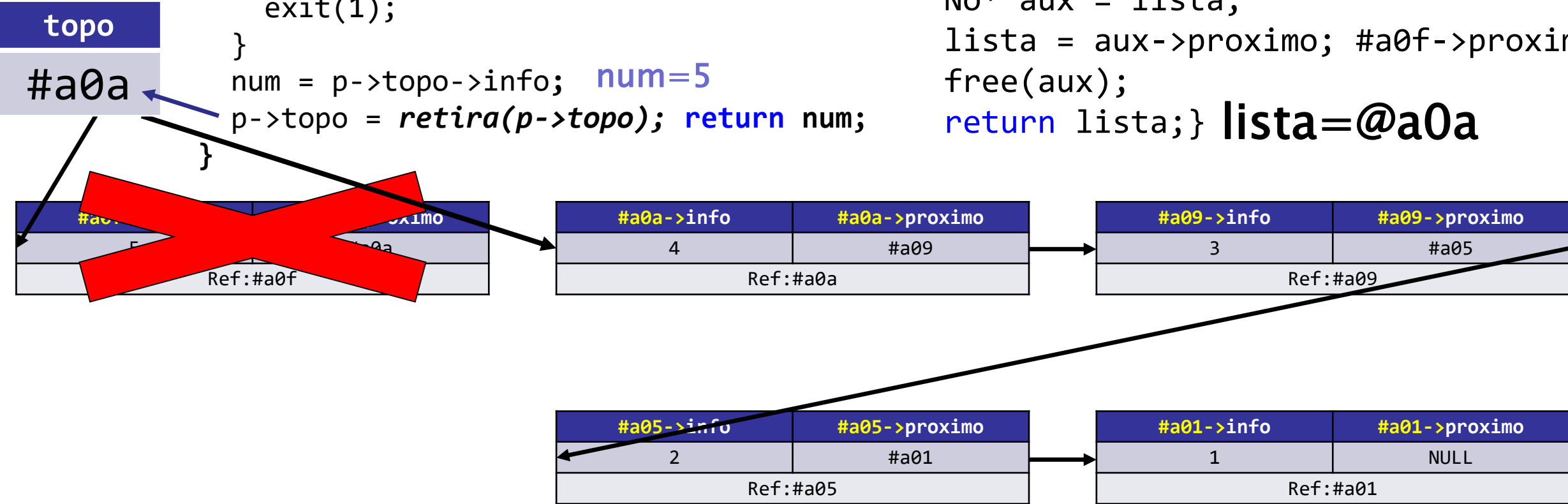
No* retira(No* lista){
    No* aux = lista;
    lista = aux->proximo; #a0f->proximo
    free(aux);
    return lista;} lista=@a0a
```



Remoção

```
int pop(Pilha* p){
    int num;
    if(vazia(p)){
        printf("Pilha vazia.\n");
        exit(1);
    }
    num = p->topo->info; num=5
    p->topo = retira(p->topo); return num;
}
```

```
int vazia(Pilha* p){
    return(p->topo == NULL);
}
No* retira(No* lista){
    No* aux = lista;
    lista = aux->proximo; #a0f->proximo
    free(aux);
    return lista;} lista=@a0a
```



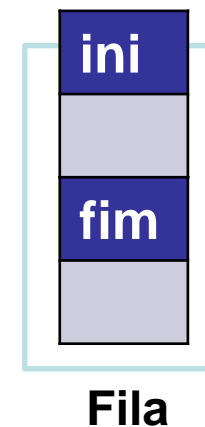
Filas

Estrutura de fila

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

info	proximo
Ref:	

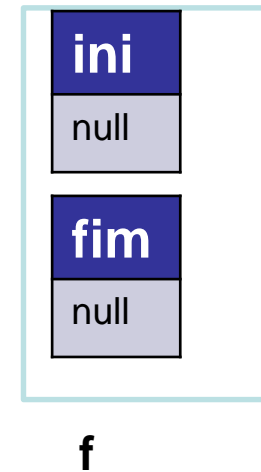
```
typedef struct fila{  
    No* ini;  
    No* fim;  
} Fila;
```



Inicialização

```
Fila* cria(void){  
    Fila* f = (Fila*)malloc(sizeof(Fila));  
    f->ini = f->fim = NULL; /* inicia a fila vazia */  
    return f;  
}
```

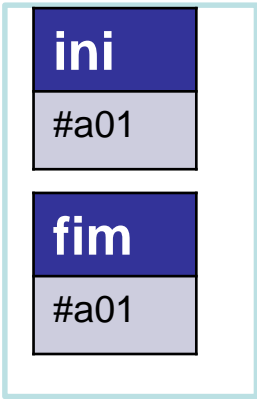
```
void main(){  
    fila* f = cria();  
    .  
    .  
}
```



Inserção

```
int main(void) {
    Fila* f = cria();
    entra(f, 1);
    entra(f, 2);
}

void entra(Fila* f, int v) {
    No* novo_no = insere(f->fim, v);
    novo_no->proximo = NULL;
    if (f->fim != NULL) {
        f->fim->proximo = novo_no;
    }
    f->fim = novo_no;
    if (f->ini == NULL) {
        f->ini = f->fim;
    }
}
```

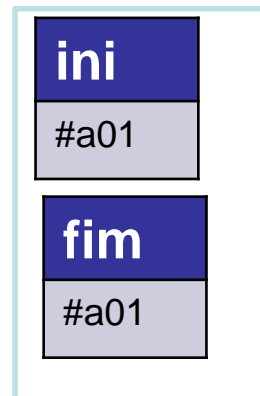


f

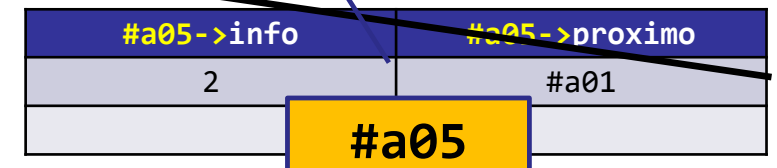
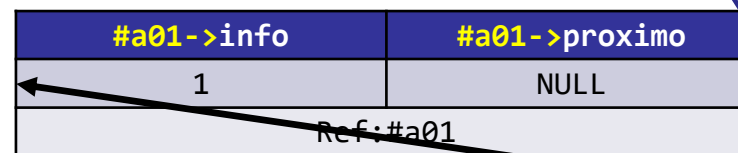
#a01->info	#a01->proximo
1	NULL
Ref:#a01	

Inserção

```
int main(void) {  
    Fila* f = cria();  
    entra(f, 1);  
    entra(f, 2);  
}  
  
void entra(Fila* f, int v) {  
    No* novo_no = insere(f->fim, v);  
    novo_no->proximo = NULL;  
    if (f->fim != NULL) {  
        f->fim->proximo = novo_no;  
    }  
    f->fim = novo_no;  
    if (f->ini == NULL) {  
        f->ini = f->fim;  
    }  
}
```

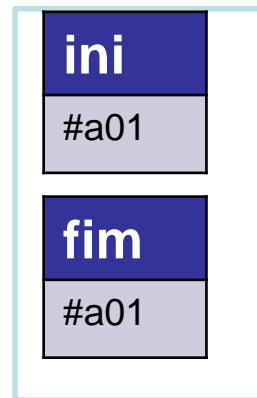


f



Inserção

```
int main(void) {  
    Fil* f = cria();  
    entra(f, 1);  
    entra(f, 2);  
}  
  
void entra(Fila* f, int v) {  
    No* novo_no = insere(f->fim, v);  
    novo_no->proximo = NULL;  
    if (f->fim != NULL) {  
        f->fim->proximo = novo_no;  
    }  
    f->fim = novo_no;  
    if (f->ini == NULL) {  
        f->ini = f->fim;  
    }  
}
```



f

#a01->info	#a01->proximo
1	NULL
Ref: #a01	

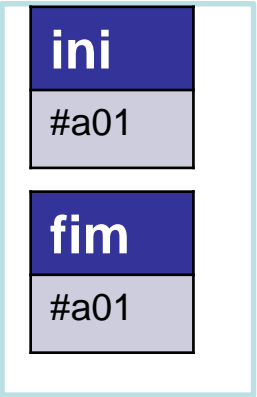
#a05->info	#a05->proximo
2	NULL
Ref: #a05	

Inserção

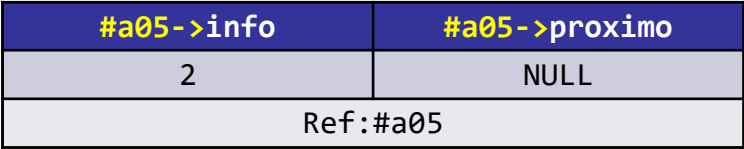
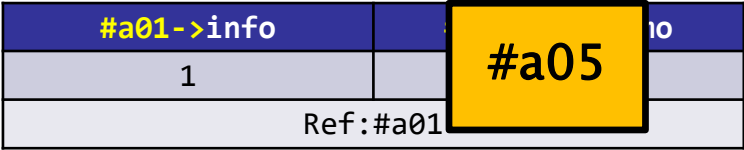
```
int main(void) {
    Fila* f = cria();
    entra(f, 1);
    entra(f, 2);
}

void entra(Fila* f, int v) {
    No* novo_no = insere(f->fim, v);
    novo_no->proximo = NULL;
    if (f->fim != NULL) {
        f->fim->proximo = novo_no;
    }
    f->fim = novo_no;
    if (f->ini == NULL) {
        f->ini = f->fim;
    }
}

No* insere(No* lista, int num) {
    No* novo_no = (No*)malloc(sizeof(No));
    novo_no->info = num;
    novo_no->proximo = lista;
    return novo_no;
}
```



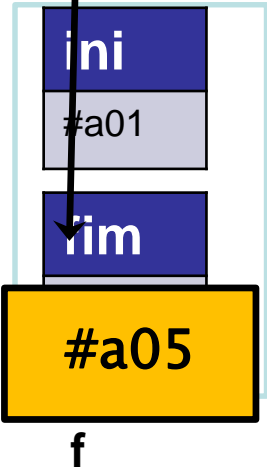
f



Inserção

```
int main(void) {
    Fila* f = cria();
    entra(f, 1);
    entra(f, 2);
}

void entra(Fila* f, int v) {
    No* novo_no = insere(f->fim, v);
    novo_no->proximo = NULL;
    if (f->fim != NULL) {
        f->fim->proximo = novo_no;
    }
    f->fim = novo_no;
    if (f->ini == NULL) {
        f->ini = f->fim;
    }
}
```

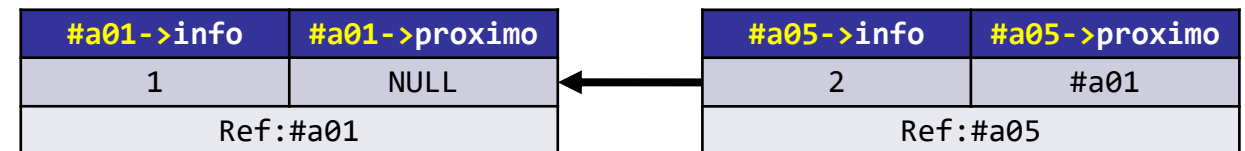
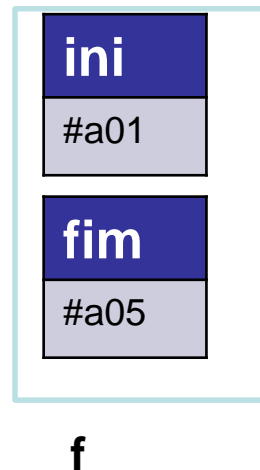


#a01->info	#a01->proximo
1	#a05
Ref: #a01	

#a05->info	#a05->proximo
2	NULL
Ref: #a05	

Inserção

```
int main(void) {  
    Fila* f = cria();  
    entra(f, 1);  
    entra(f, 2);  
}  
  
void entra(Fila* f, int v) {  
    No* novo_no = insere(f->fim, v);  
    novo_no->proximo = NULL;  
    if (f->fim != NULL) {  
        f->fim->proximo = novo_no;  
    }  
    f->fim = novo_no;  
    if (f->ini == NULL) {  
        f->ini = f->fim;  
    }  
}  
  
No* insere(No* lista, int num) {  
    No* novo_no = (No*)malloc(sizeof(No));  
    novo_no->info = num;  
    novo_no->proximo = lista;  
    return novo_no;  
}
```



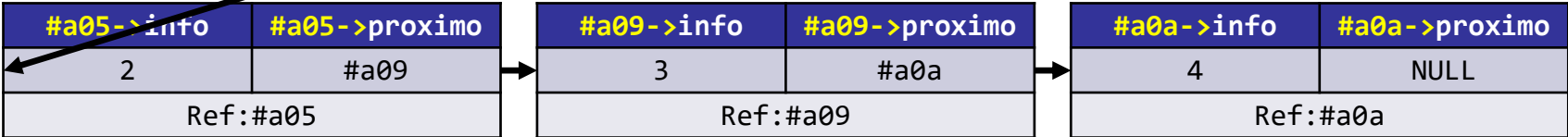
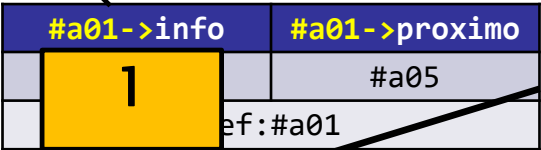
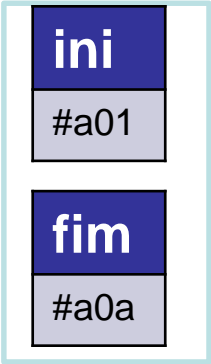
Remoção

```
void main(void){
  Fila* f = cria();
  entra(f, 1);
  entra(f, 2);
  entra(f, 3);
  entra(f, 4);
  printf("1º da fila %d\n", sai(f));
  .
  .
}
```

num=1

```
int sai(Fila* f){
  int num;
  if(vazia(f)){
    printf("Fila vazia.\n");
    exit(1); /* aborta programa */
  }
  num = f->ini->info; f->ini=#a01
  f->ini = retira(f->ini); f->ini <-#a05
  if(f->ini == NULL){ /* Fila ficou vazia? */
    f->fim = NULL;
  }
  return num;
}
```

```
No* retira(No* lista){ #a01
  No* aux = lista;
  lista = aux->proximo;
  free(aux);
  return lista; #a05
}
```



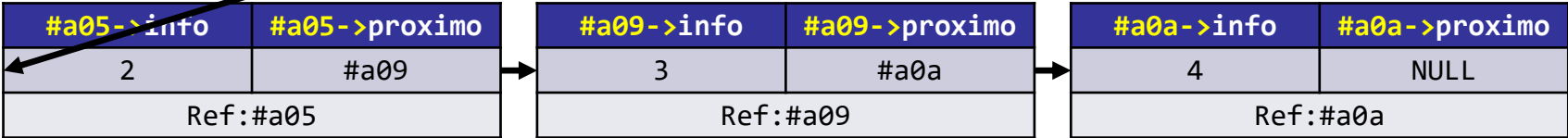
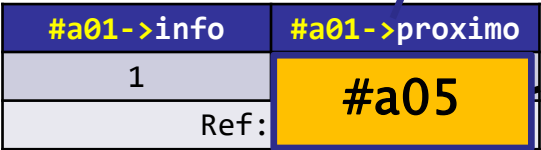
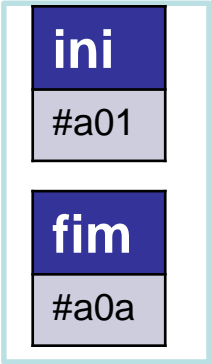
Remoção

```
void main(void){
    Fila* f = cria();
    entra(f, 1);
    entra(f, 2);
    entra(f, 3);
    entra(f, 4);
    printf("1º da fila %d\n", sai(f));
    .
    .
}
```

```
int sai(Fila* f){
    int num;
    if(vazia(f)){
        printf("Fila vazia.\n");
        exit(1); /* aborta programa */
    }
    num = f->ini->info; f->ini=#a01
    f->ini = retira(f->ini); f->ini <-#a05
    if(f->ini == NULL){ /* Fila ficou vazia? */
        f->fim = NULL;
    }
    return num;
}
```

```
No* retira(No* lista){ #a01
    No* aux = lista;
    lista = aux->proximo;
    free(aux);
    return lista; #a05
}
```

Lista=#a05

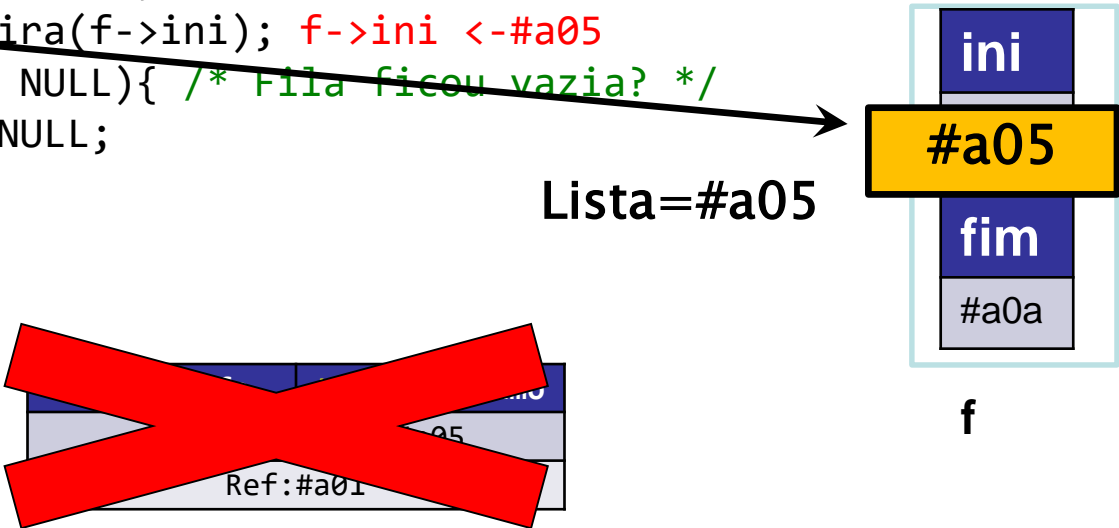


Remoção

```
void main(void){
    Fila* f = cria();
    entra(f, 1);
    entra(f, 2);
    entra(f, 3);
    entra(f, 4);
    printf("1º da fila %d\n", sai(f));
    .
    .
}
```

```
int sai(Fila* f){
    int num;
    if(vazia(f)){
        printf("Fila vazia.\n");
        exit(1); /* aborta programa */
    }
    num = f->ini->info; f->ini=#a01
    f->ini = retira(f->ini); f->ini <-#a05
    if(f->ini == NULL){ /* Fila ficou vazia? */
        f->fim = NULL;
    }
    return num;
}
```

```
No* retira(No* lista){ #a01
    No* aux = lista;
    lista = aux->proximo;
    free(aux);
    return lista; #a05
}
```



Lista=#a05

#a05->info	#a05->proximo	#a09->info	#a09->proximo	#a0a->info	#a0a->proximo
2	#a09	3	#a0a	4	NULL
Ref:#a05		Ref:#a09		Ref:#a0a	

ATÉ A PRÓXIMA!



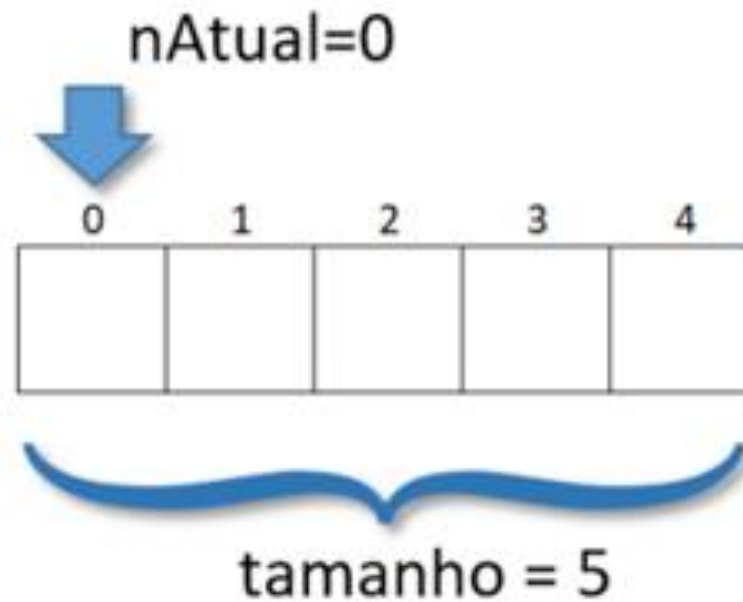
Estrutura_módulo_2_4_ Lista Sequencial

Prof. MSc. Olavo Ito

Listas sequenciais

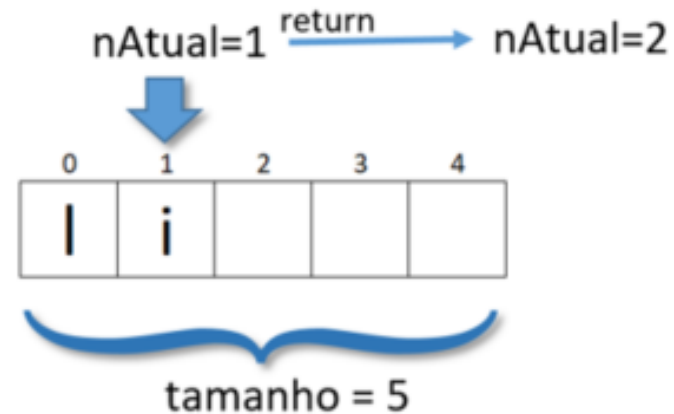
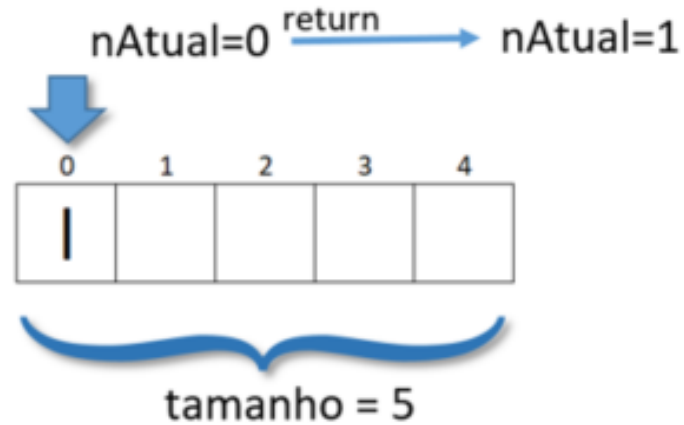
Inicialização

```
#define tamanho 5  
char vetor[tamanho];  
int nAtual = 0;
```



Inserção

```
int main() {  
    char valor;  
    valor = 'l';  
    nAtual = inserir(vetor, valor, nAtual);  
    nAtual = inserir(vetor, 'i', nAtual);  
    nAtual = inserir(vetor, 'v', nAtual);  
    nAtual = inserir(vetor, 'r', nAtual);  
    nAtual = inserir(vetor, 'o', nAtual);  
    valor = 'v';  
    int p = consulta(vetor, valor, nAtual);  
    valor = 'i';  
    nAtual = remover(vetor, valor, nAtual);  
    imprime(vetor, nAtual);  
    return 0;  
}
```



Link

- <https://tinyurl.com/edpratica06b>

ATÉ A PRÓXIMA!

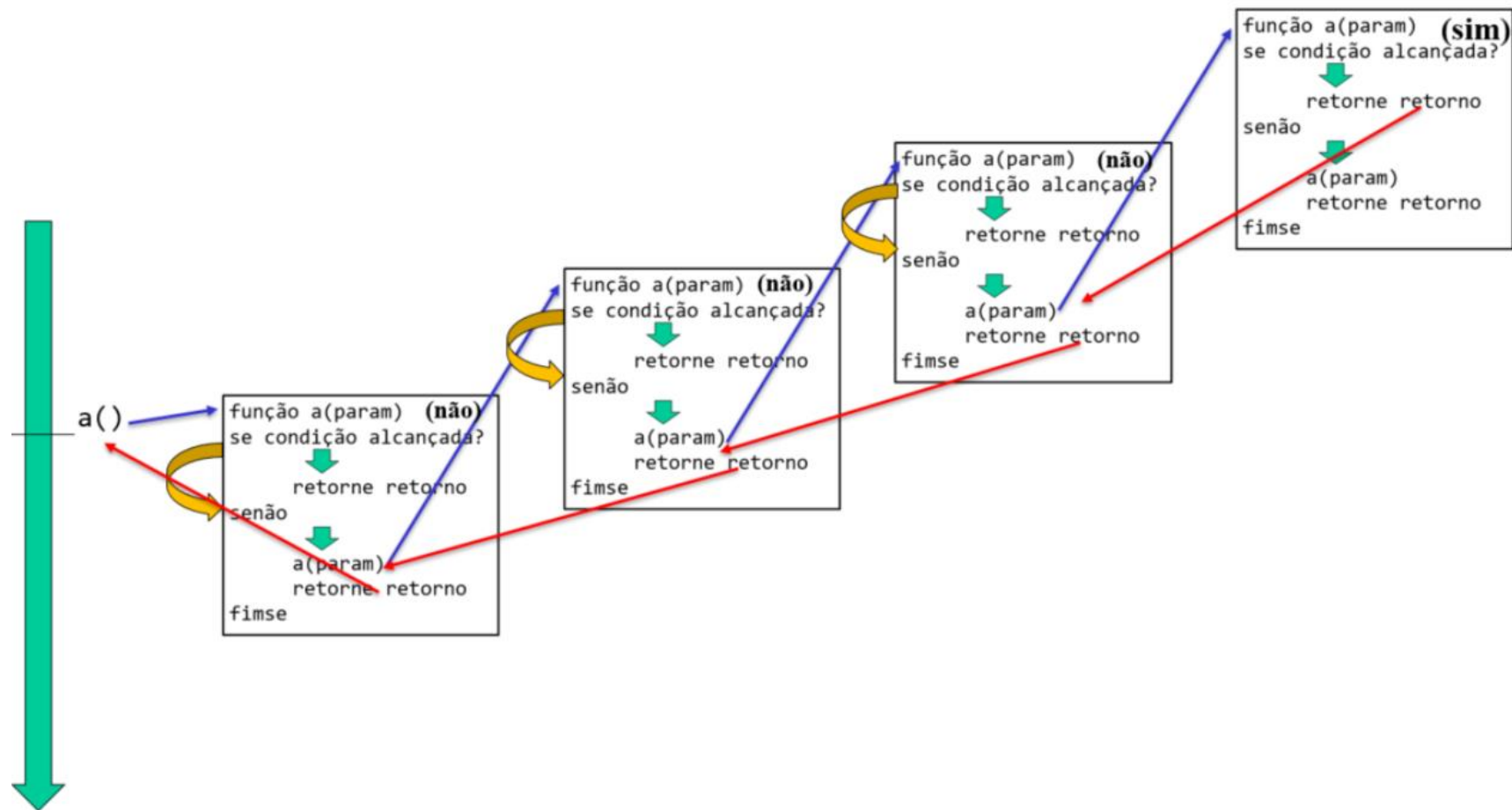


Estrutura_módulo_3_1_ Recursividade

Prof. MSc. Olavo Ito

Recursividade

Conceito



Exemplo fatorial

```
1  #include <stdio.h>
2  int fat(n) {
3      //Função que calcula, recursivamente, o fatorial de n.
4      int result;
5      if ((n == 1) || (n == 0)) {
6          printf("\nponto de parada atingido");
7          return 1;
8      }
9      else {
10         printf("\nchamando fat(%d-1)", n);
11         result = fat(n - 1);
12         result = result * n;
13         return result;
14     }
15 }
16 int main() {
17     int n=4;
18     printf("\nO fatorial de %d é %d \n", n, fat(n));
19     return 0;
20 }
```

Exemplo: Labirinto

```
#define DIM 8  
#define DI 4  
#define DJ 4
```

```
/* matriz que define o 'labirinto'*/  
int m[DIM][DIM] =  
{ { '#', '#', '#', '#', '#', '#', '#', '#'},  
  { '#', ' ', ' ', ' ', ' ', ' ', '#', ' '},  
  { '#', '#', ' ', ' ', '#', '#', ' ', '#'},  
  { '#', ' ', ' ', ' ', ' ', ' ', '#', ' '},  
  { '#', ' ', '#', '#', ' ', '#', ' ', '#'},  
  { '#', ' ', ' ', '#', '#', '#', ' ', '#'},  
  { '#', '#', ' ', ' ', ' ', ' ', ' ', '#'},  
  { '#', '#', '#', '#', '#', '#', '#', '#'} };
```

```
tenta(1, 1)
```

	0	1	2	3	4	5	6	7
0	#	#	#	#	#	#	#	#
1	#	X				#		#
2	#	#		#	#	#		#
3	#					#		#
4	#		#	#	T	#		#
5	#			#	#	#		#
6	#	#						#
7	#	#	#	#	#	#	#	#

Link

- <https://tinyurl.com/edpratica07>

ATÉ A PRÓXIMA!



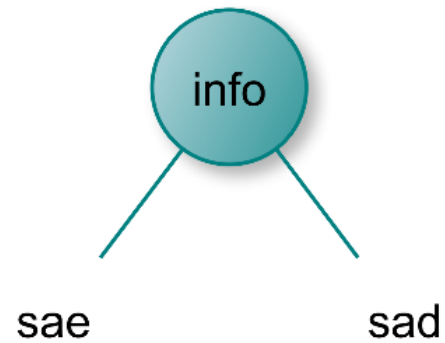
Estrutura_módulo_3_2_ Árvores

Prof. MSc. Olavo Ito

Árvore

```
struct arv {  
    int info;  
    struct arv* sae;  
    struct arv* sad;  
};  
typedef struct Arv;
```

```
typedef struct arv {  
    int info;  
    struct arv* sae;  
    struct arv* sad;  
}Arv;
```



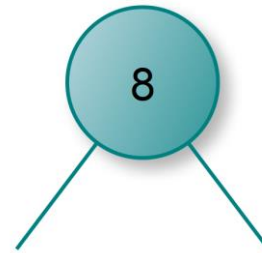
ou



Criação de nó

```
Arv* cria(int c, Arv* sae, Arv* sad) {  
    Arv* p = (Arv*)malloc(sizeof(Arv));  
    p->info = c;  
    p->sae = sae;  
    p->sad = sad;  
    return p;  
}
```

```
void main() {  
    Arv* a1 = cria(8, inicializa(), inicializa());  
    .  
    .  
}
```



Árvore Binária de Busca

Inserção

```
Arv* insere(Arv* a, int v)
{
    if (a == NULL) {
        a = (Arv*)malloc(sizeof(Arv)); a->info = v;
        a->sae = a->sad = NULL;
    }
    else
    {
        if (v < a->info)
            a->sae = insere(a->sae, v);
        else/* v < a->info */
            a->sad = insere(a->sad, v);
    }
    return a;
}
```

Busca

```
Arv* busca(Arv* r, int v)
{
    if (r == NULL)
        return NULL;
    else
        if (r->info > v)
            return busca(r->sae, v);
        else
            if (r->info < v)
                return busca(r->sad, v);
            else
                return r;
}
```

Remoção

- Nenhum e um filho

```
/* achou o elemento nem a maior nem menor*/
if (r->sae == NULL && r->sad == NULL) { /* é folha */
    free(r);
    r = NULL;
}
else if (r->sae == NULL) { /* só tem filho à direita */
    Arv* t = r;
    r = r->sad;
    free(t);
}
else if (r->sad == NULL) { /* só tem filho à esquerda */
    Arv* t = r;
    r = r->sae;
    free(t);
}
else { /* tem os dois filhos */
    Rotina de exclusão dois filhos
}
}
```

Remoção

2 filhos

```
Arv* pai = r;  
Arv* f = r->sad;  
while (f->sae != NULL) {  
    pai = f;  
    f = f->sae;  
}  
r->info = f->info; /* troca as informações */  
f->info = v;  
r->sad = retira(r->sad, v);
```

Link

- <https://tinyurl.com/edpratica08>

ATÉ A PRÓXIMA!