Alocação Dinâmica de Memória

Exercício:

Com base no que vimos, construa um programa que aloque dinamicamente memória para um vetor de strings, o números de elementos do vetor e o comprimento máximo das strings pertencentes a este, serão fornecidos pelo usuário, através da entrada padrão. O vetor deve ser inicializado, através da entrada padrão, e posteriormente, impresso na a saída padrão.

UNIVASF

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
   char* vetor_strings;
   int num_elementos, comprimento_max, i;
   printf ("Entre com o numero de elementos do vetor de strings: ");
   scanf ("%d",&num_elementos);
   printf ("Entre com o comprimento maximo das strings do vetor: ");
   scanf ("%d",&comprimento_max);
   vetor_strings= (char*) calloc (num_elementos, (comprimento_max+1)*sizeof(char));
   if (! vetor_strings) {
        printf (“\nERRO!\n”);
        exit (1);
    }
```

```c
    for (i=0;i<num_elementos;i++)
    {
        printf ("Entre com a string[%d]: ",i+1);
        scanf ("%s",vetor_strings+(i*(comprimento_max+1)));
    }
    printf ("\nStrings contidas no vetor:\n");
    for (i=0;i<num_elementos;i++)
        printf ("\nString[%d]: %s",i+1,vetor_strings+(i*(comprimento_max+1)));
}
```

Alocação Dinâmica de Memória
Parte 2 – Funções realloc e free

UNIVASF

Alocação Dinâmica de Memória

- realloc

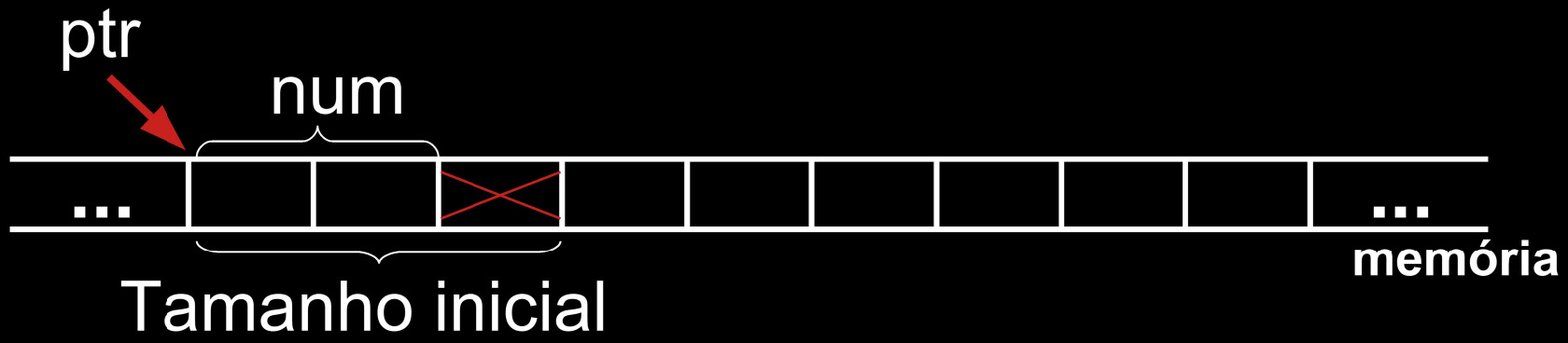
A função **realloc()** serve para realocar memória e tem a seguinte forma:

void *realloc (void *ptr, unsigned int num);

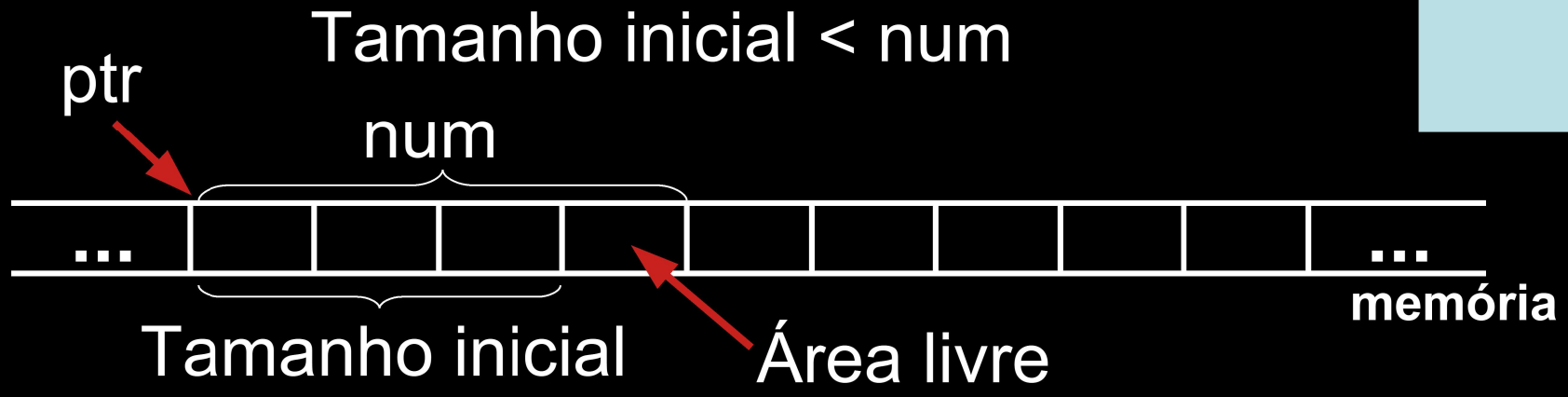
A função modifica o tamanho da memória previamente alocada apontada por ***ptr** para aquele especificado por **num**.

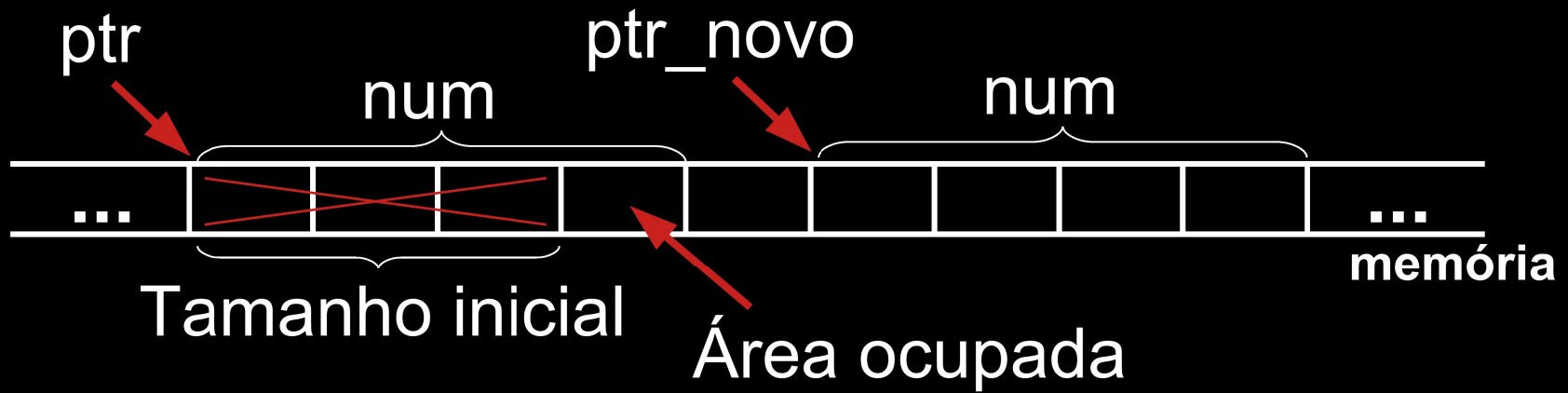
Situações possíveis:

Tamanho inicial > num



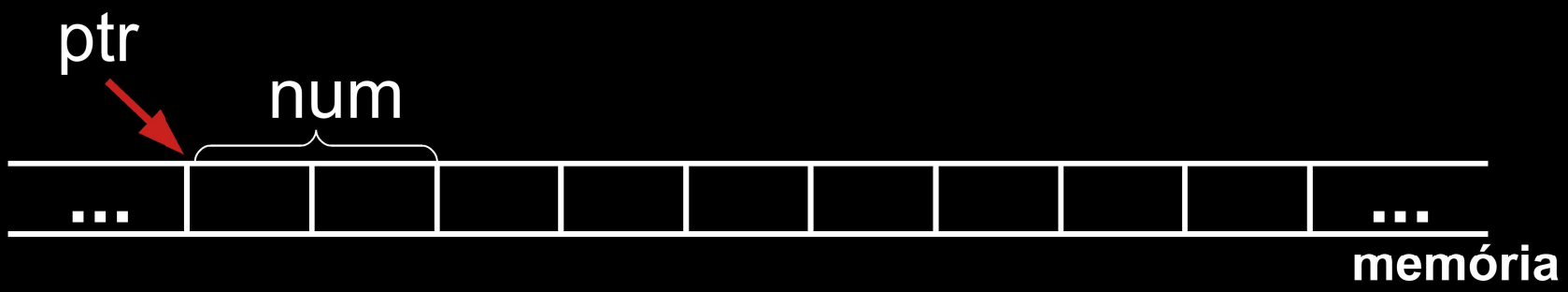
Situações possíveis:





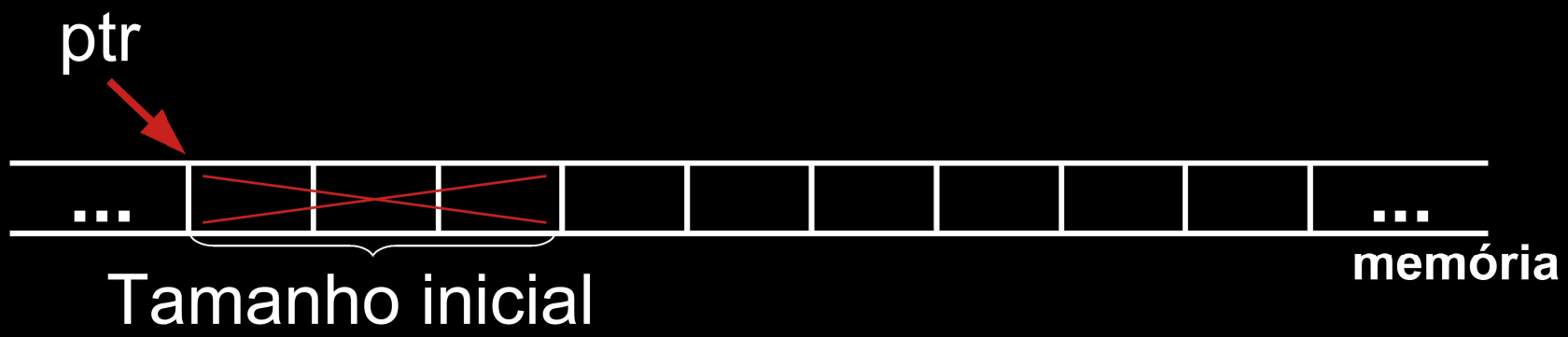
Situações possíveis:

ptr == NULL



Situações possíveis:

num == 0 (zero)



```c
#include <stdio.h>
#include <stdlib.h>
main (void) {
    int *p, a, i;
    /* ... */
    a = 30;
    p = (int *) malloc (a*sizeof(int));
    if (!p) {
        printf ("** Erro: Memoria Insuficiente **");
        exit (1);
    }
    for (i=0; i<a ; i++)
        p[i] = i*i;
    a = 100;
    p = (int *)realloc (p, a*sizeof(int));
    if (!p) { printf (“\nERRO!\n”); exit (1); }
    for (i=30; i<a ; i++)
        p[i] = a*i*(i-6);
    ...
}
```

Alocação Dinâmica de Memória

- free

Quando alocamos memória dinamicamente é necessário que a mesma seja liberada quando esta não for mais necessária.

Para isto existe a função **free()** cuja forma é:

void free (void *p);

```c
#include <stdio.h>
#include <stdlib.h>
int main (void)
{
    int *p, a;
    /* ... */
    p=(int *)malloc(a*sizeof(int));
    if (!p) {
        printf ("** Erro: Memoria Insuficiente **");
        exit (1);
    }
    /* ... */
    free(p);
    /* ... */
}
```

Alocação Dinâmica de Memória

Exercício:

Escreva um programa em C que manipule um vetor de inteiros não nulos alocado dinamicamente. O programa recebe inteiros, através da entrada padrão, e os insere no vetor. A cada inteiro que é inserido a área de memória necessária para armazenar um inteiro é incrementada ao número de bytes necessários para armazenar o vetor. O vetor não ocupa memória inicialmente. Quando o usuário entrar com o inteiro 0 (zero), o programa será finalizado e o mesmo não pertencerá ao vetor. Após o processo de inserção o vetor deve ser impresso na saída padrão. Libere a memória utilizada antes do final do processamento.

UNIVASF

```c
#include <stdio.h>
#include <stdlib.h>
int main () {
    int *vetor=NULL, num_elementos=0, num;
    do {
        printf ("\nEntre com o número inteiro que deseja inserir%s",
        " no vetor (entre com zero para finalizar o programa): ");
        scanf("%d",&num);
        if (num) {
            num_elementos++;
            vetor =(int*)realloc(vetor,
            num_elementos*sizeof(int));
            if (!vetor) { printf ("\nERRO!\n"); exit (1); }
            vetor[num_elementos-1]=num;
        }
    }while(num);
    printf("\nOs elementos do vetor são:");
    for (;num<num_elementos;num++)
        printf("\nO %d° elemento do vetor eh %d", num+1, *(vetor+num));
    free(vetor);
}
```

Alocação Dinâmica de Memória

Exercício:

Escreva um programa em C que manipule vetores de inteiros não nulos alocados dinamicamente. O programa recebe inteiros não nulos, através da entrada padrão, e os insere no vetor. A cada inteiro que é inserido a área de memória necessária para armazenar um inteiro é incrementada ao número de bytes necessários para armazenar o vetor. Um vetor não ocupa memória inicialmente. Quando ocorre o fornecimento do inteiro 0 (zero), o programa percebe que o mesmo não pertence ao vetor e que o vetor já teve todos os valores de seus elementos fornecidos.

UNIVASF

Alocação Dinâmica de Memória

O programa então apresenta o vetor na saída padrão, com layout adequado, e continua a leitura de valores para outro vetor, procedendo da mesma forma. O fornecimento de um vetor sem nenhum elemento indica que o programa deve finalizar seu processamento. Após a apresentação de cada vetor na saída padrão a memória alocada dinamicamente para armazená-lo deve ser liberada. Obs.: As entradas não devem ser solicitadas ao usuário.

UNIVASF

Alocação Dinâmica de Memória

Exemplo de entrada:

8
15
2
7
34
0
3
9
0
0

Exemplo de Saída:

Vetor = { 8, 15, 2, 7, 34 }
Vetor = { 3, 9 }

Manipulação de Arquivos

Manipulação de Arquivos

Devemos iniciar nossa explanação pelo conceito de arquivo:

Arquivo é uma unidade lógica utilizada para armazenar dados em disco ou em qualquer outro dispositivo externo de armazenamento. Pode-se abrir, fechar, ler, escrever ou apagar um arquivo.

A linguagem C manipula tanto arquivos quanto dispositivos de I/O, se utilizando do conceito de "ponteiro para arquivo". Sendo disponibilizada uma série de funções para trabalhar com este conceito, cujos protótipos estão reunidos em **stdio.h**.

UNIVASF

Manipulação de Arquivos

A definição do “ponteiro para arquivo” também está no arquivo **stdio.h**.

Podemos declarar um ponteiro para arquivo da seguinte maneira:

FILE *p;

Os arquivos podem ser classificados em:

- binários;
- texto.

UNIVASF

Manipulação de Arquivos

- fopen()

Esta é a função de abertura de arquivos. Seu protótipo é:

FILE *fopen (char *nome_do_arquivo,char *modo);

Modo	Significado
"r"	Abre um arquivo texto para leitura. O arquivo deve existir antes de ser aberto.
"w"	Abrir um arquivo texto para gravação. Se o arquivo não existir, ele será criado. Se já existir, o conteúdo anterior será destruído.
"a"	Abrir um arquivo texto para gravação. Os dados serão adicionados no fim do arquivo ("append"), se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.

UNIVASF

Manipulação de Arquivos

Modo	Significado
"rb"	Abre um arquivo binário para leitura. Igual ao modo "r" anterior, só que o arquivo é binário.
"wb"	Cria um arquivo binário para escrita, como no modo "w" anterior, só que o arquivo é binário.
"ab"	Acrescenta dados binários no fim do arquivo, como no modo "a" anterior, só que o arquivo é binário.
"r+"	Abre um arquivo texto para leitura e gravação. O arquivo deve existir e pode ser modificado.
"w+"	Cria um arquivo texto para leitura e gravação. Se o arquivo existir, o conteúdo anterior será destruído. Se não existir, será criado.
"a+"	Abre um arquivo texto para gravação e leitura. Os dados serão adicionados no fim do arquivo se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
"r+b"	Abre um arquivo binário para leitura e escrita. O mesmo que "r+" acima, só que o arquivo é binário.
"w+b"	Cria um arquivo binário para leitura e escrita. O mesmo que "w+" acima, só que o arquivo é binário.
"a+b"	Acrescenta dados ou cria uma arquivo binário para leitura e escrita. O mesmo que "a+" acima, só que o arquivo é binário

Manipulação de Arquivos

Poderíamos então, abrir um arquivo binário para escrita, da seguinte forma :

```
#include <stdio.h>
...
FILE *fp;
fp=fopen ("exemplo.bin","wb");
if (!fp) /*fp==NULL*/
    printf ("Erro na abertura do arquivo.");
```

Manipulação de Arquivos

Uma vez aberto um arquivo, vamos poder ler ou escrever nele utilizando as funções disponíveis em **stdio.h**.

Toda vez que estamos trabalhando com arquivos, há uma espécie de posição atual no arquivo. Esta é a posição de onde será lido ou escrito o próximo dado. Normalmente, num acesso sequencial a um arquivo, não temos que mexer nesta posição. Pois, quando lemos um dado a posição no arquivo é automaticamente atualizada. Num acesso randômico teremos que mexer nesta posição.

UNIVASF

Manipulação de Arquivos

- fclose

Quando acabamos de usar um arquivo que abrimos, devemos fechá-lo. Para tanto, usa-se a função **fclose()**, cujo protótipo é:

int fclose (FILE *fp);

O ponteiro **fp** passado à função **fclose()** determina o arquivo a ser fechado. A função retorna zero no caso de sucesso.

UNIVASF

Manipulação de Arquivos

Fechar um arquivo faz com que qualquer dado que tenha permanecido no buffer associado ao fluxo de saída seja gravado.

A função **exit()** fecha todos os arquivos que um programa tiver aberto.

Veremos a seguir algumas funções utilizadas na manipulação de arquivos.

UNIVASF

Manipulação de Arquivos

- putc

A função **putc** é uma função de escrita em arquivo. Seu protótipo é:

int putc (int ch, FILE *fp);

A referida função escreve um caractere no arquivo apontado por **fp**.

O que **putc()** retorna? e para que é útil este retorno?

```c
/*Exemplo de manipulação de arquivo*/
#include <stdio.h>
#include <stdlib.h>
int main()  {
    FILE *fp;
    char c;
    fp = fopen("arquivo.txt","w");
    if(!fp)  {
        printf( "Erro na abertura do arquivo");
        exit(1);
    }
    printf("Entre com um caractere para ser gravado no arquivo: ");
    scanf("%c", &c);
    putc(c, fp); /*Deveria ter sido analisado o retorno de putc()?*/
    fclose(fp);
    return 0;
}
```

```c
/*Exemplo de manipulação de arquivo*/
#include <stdio.h>
#include <stdlib.h>
int main()  {
    FILE *fp;
    char c;
    fp = fopen("arquivo.txt","w");
    if(!fp)  {
        printf( "Erro na abertura do arquivo");
        exit(1);
    }
    printf("Entre com um caractere para ser gravado no arquivo: ");
    scanf("%c", &c);
    if (c == (char) putc(c, fp))
        printf("Tudo deu certo!");
    else
        printf("Ocorreu um erro!");
    fclose(fp);
    return 0;
}
```

Manipulação de Arquivos

Exercício:

Com o que vimos até o momento sobre manipulação de arquivos. Construa uma função em C que possua a capacidade de escrever uma string em um arquivo texto. Escreva um programa que se utiliza adequadamente da função que você implementou.

UNIVASF