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 }

```c
#include <stdio.h>
#include <stdlib.h>
int main () {
    int *vetor, num_elementos, num;
    do {
        vetor=NULL;
        num_elementos=0;
        do {
            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);
        if (num_elementos) {
            printf("Vetor = { ");
            for (;num<num_elementos;num++)
                printf(" %d, ", *(vetor+num));
            printf("\b\b }\n");
            free(vetor);
        }
    }while(num_elementos);
    return 0;
}
```

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.

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?

UNIVASF

```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

```c
#include <stdio.h>
#include <stdlib.h>
void putStrInFile (FILE *, char*);
int main()
{
    FILE *fp;
    char string[100];
    int i;
    fp = fopen("arquivo.txt","w");
    if(!fp)
    {
        printf( "Erro na abertura do arquivo");
        exit(1);
    }
    printf("Entre com a string a ser gravada no arquivo:");
    scanf("%99[^\n]", string);
    putStrInFile (fp, string) ;
    fclose(fp);
    return 0;
}
```

```c
void putStrInFile (FILE *fp, char* string)
{
   int i;
   for(i=0; string[i]; i++)
      putc(string[i], fp);
    /*Não testei se o caractere foi escrito com sucesso.*/
}
```

Manipulação de Arquivos

- getc

Seu protótipo é:

int getc (FILE *fp);

A função retorna um caractere lido do arquivo apontado por **fp**. **OBS.: Equivalente à fgetc().**

Exercício:

Construa um função em C que possua a capacidade de ler e apresentar na saída padrão uma string contida em um arquivo texto.

Manipulação de Arquivos

- feof

EOF ("End Of File") indica o fim de um arquivo. Às vezes, é necessário verificar se um arquivo chegou ao fim. Para isto, podemos usar a função **feof()**. Ela retorna não-zero se o arquivo chegou ao EOF, caso contrário retorna zero.

Seu protótipo é:

int feof (FILE *fp);

Outra forma de se verificar se o final do arquivo foi atingido é comparar o caractere lido por **getc()** com **EOF**. O programa a seguir abre um arquivo já existente e o lê, caractere por caractere, até que o final do arquivo seja atingido. Os caracteres lidos são apresentados na tela:

UNIVASF

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE *fp;
    int c;
    fp = fopen("arquivo.txt","r");
    if(!fp) {
        printf( "Erro na abertura do arquivo");
        exit(1);
    }
    while((c = getc(fp) ) != EOF)
        printf("%c", (char)c);
    fclose(fp);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE *fp;
    fp = fopen("arquivo.txt","r");
    if(!fp) {
        printf( "Erro na abertura do arquivo");
        exit(1);
    }
    while(!feof(fp))
        printf("%c", (char)getc(fp));
    fclose(fp);
    return 0;
}
```

```c
/*Resposta do exercício*/
void get_string(FILE *fp)
{
    int c;
    while ((c = getc(fp)) != EOF)
        printf("%c", (char)c);
}
```

Manipulação de Arquivos

- Arquivos pré-definidos

Quando se começa a execução de um programa, o sistema automaticamente abre alguns arquivos pré-definidos:

stdin: dispositivo de entrada padrão (geralmente o teclado);

stdout: dispositivo de saída padrão (geralmente o vídeo);

stderr: dispositivo de saída de erro padrão (geralmente o vídeo);

stdaux: dispositivo de saída auxiliar (em muitos sistemas, associado à porta serial);

stdprn : dispositivo de impressão padrão (em muitos sistemas, associado à porta paralela).

UNIVASF

Manipulação de Arquivos

Cada uma destas constantes pode ser utilizada como um ponteiro para FILE, para acessar periféricos associados a eles.

Desta maneira, pode-se, por exemplo, usar:

```
char ch = (char)getc(stdin);
```

ou:

```
putc(ch, stdout);
```

Manipulação de Arquivos

Com base no que vimos, podemos agora entender melhor o processo de leitura de valores fornecidos através da entrada padrão.

Sempre que nos utilizamos de uma função para leitura de valores através do teclado, estamos solicitando que seja efetuada uma leitura na **stream** (**buffer**) que representa o arquivo apontado por **stdin**.

Quando não é encontrado algo na **stream** ocorre uma espera pelo fornecimento de valores através do teclado, ou melhor, ocorre uma espera por pressionamento(s) de tecla(s) no teclado.

UNIVASF

Programa make

Programa make

Já vimos alguns conceitos associados ao processo de compilação. Porém, dispersos entre os demais conteúdos.

Vamos relembrá-los agora:

- Definição de macros, quando vimos o **#define**;
- Diretivas para compilação, quando vimos o **-o** e o **-lm**;
- Arquivos de cabeçalho, quando vimos os arquivos **.h**;
- Divisão do projeto em arquivos separados, quando trabalhamos a definição de bibliotecas de ligação estática.

UNIVASF

Programa make

O último tópico de nosso conteúdo programático é: Compilação separada com o uso do programa make.

O make é uma ferramenta que controla a geração de executáveis e outros arquivos não-fonte de um programa a partir de arquivos fonte do programa.

O make recebe o seu conhecimento sobre como construir o seu programa a partir de um arquivo chamado makefile, que lista cada um dos arquivos não-fonte e como gerá-lo de outros arquivos.

UNIVASF

Programa make

É interessante que o programador ao escrever um programa, escreva um makefile para ele, de modo que seja possível utilizar o make para construir e instalar o programa.

FIM!