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

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

- 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 os periféricos associados a eles.

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

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

ou:

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

UNIVASF

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!