Operador *sizeof*

- ✓ Usado para se saber o tamanho de variáveis ou de tipos.
- ✓ Retorna o tamanho do **tipo** ou **variável** em bytes.
- ✓ O sizeof é chamado um operador porque ele é substituído pelo tamanho do tipo ou variável no momento da compilação.
- ✓ Não é uma função.
- ✓ O sizeof admite duas formas:
  - sizeof nome_da_variável
  - sizeof (nome_do_tipo)
- ✓ Se quisermos então saber o tamanho de um float fazemos sizeof(float).
- ✓ Se declararmos a variável f como float e quisermos saber o seu tamanho faremos sizeof f.

Fonte: http://www.mtm.ufsc.br/~azeredo/cursoC/aulas/cb50.html

Função gets e fgets

- A função *gets* permite colocar na variável todos os caracteres introduzidos, sem encerrar com o branco e colocando '\0' como enter.
  Se o conteúdo digitado ultrapassar o sizeof da variável, a área de memória subjacente será preenchida!
- A função *fgets* estabelece o número máximo de caracteres a serem lidos, porém o enter faz com que sejam incluídos os caracteres '\n' e '\0'.
  Se o conteúdo digitado ultrapassar o limite, então apenas os caracteres estabelecidos são aceitos (é truncado).

Funções *gets* e f

```
//testa funcao gets e fgets
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main( )
{
    char nome[10];
    printf("\nUsando gets:\n");
    puts("\nNome:");
    gets(nome);        // digita 23 caracteres
    printf("\nTamanho de Nome = %d, nome = %s\n",strlen(nome), nome);
    fflush(stdin);
    printf("\nUsando fgets, com digitado maior que variável:\n");
    puts("\nNome:");
    fgets(nome, sizeof(nome),stdin); // digita 23 caracteres
    printf("\nTamanho de Nome = %d, nome = %s\n",strlen(nome), nome);
    fflush(stdin);
    printf("\nUsando fgets, com digitado menor que variável:\n");
    puts("\nNome:");
    fgets(nome, sizeof(nome),stdin); // digita 5 caracteres
    printf("\nTamanho de Nome = %d, nome = %s\n",strlen(nome), nome);
    system("PAUSE");
    return 0;
}
```

```
Nome:
Maria da Silva Palhares
Tamanho de Nome = 23, nome = Maria da Silva Palhares
Usando fgets, com digitado maior que variÃvel:
Nome:
Maria da Silva Palhares
Tamanho de Nome = 9, nome = Maria da
Usando fgets, com digitado menor que variÃvel:
Nome:
Maria
Tamanho de Nome = 6, nome = Maria
Pressione qualquer tecla para continuar. . .
```

Maria + '\n'!!!

Função fgets – leitura de *strings*

Lê de um arquivo uma *string* de um tamanho máximo informado.
Para leitura de *strings* a partir do dispositivo padrão de entrada, o formato geral de *fgets* é:

fgets(*string* (=vetor de caracteres), tamanho máximo da *string*, *stdin*)

Além de ler *strings* de arquivos, a função *fgets* difere da função *gets* em duas questões importantes:
1) com *fgets*, a *string* é lida até o tamanho máximo indicado - 1 (para inserção do \0), com *gets* não há controle de limite da *string*, podendo ocorrer "estouro de *buffer*";
2) em *fgets*, o caractere de nova linha \n, se dentro do tamanho máximo previsto, também integrará a *string*, em *gets* isso não ocorre.

Arquivos

- Repositórios permanentes de dados (informações) que possibilitam a comunicação dos programas com o ambiente.
- Podem ser armazenados em dispositivos de memória auxiliar (discos).
- A alteração de um arquivo durante a execução de um programa torna permanente (além do tempo de execução do programa) o processamento realizado.
- Em arquivos é possível armazenar-se bem mais informações do que na memória.

Conceito em C de *streams*

- ✓ Conjunto seqüencial de bytes, sem qualquer tipo de estrutura interna.
- ✓ São vistos por programas ou comandos como um fluxo de caracteres.

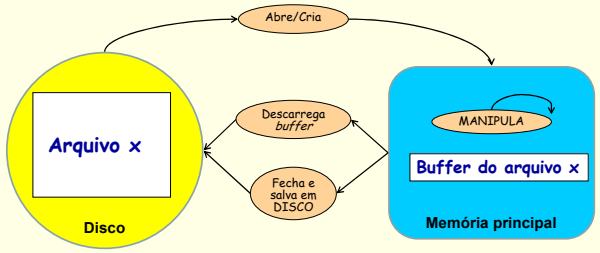
Arquivos em C

✓ No C, arquivos podem ser definidos como repositórios permanentes de *streams*, armazenados em dispositivos de memória auxiliar de forma permanente, a partir do uso de periféricos que aceitam tanto a leitura como a gravação de dados (disco, *pendrives*, etc).

✓ No C, toda a entrada ou saída de dados é processada através do processamento de *streams*, independentemente do periférico utilizado – *device independent*.

✓ As operações de entrada e saída de dados de um arquivo (incluídos os arquivos padrão stdin - de entrada e stdout - de saída) são apoiadas pelo armazenamento temporário destes *streams* em áreas da memória principal denominadas *buffers*.

Esquema de utilização de arquivos



• arquivo: entidade externa ao programa;
• armazenamento permanente (não se perdem ao término do programa) das informações em memória secundária (periféricos).

→ Movimentação de dados
○ Funções

Tipos de arquivos do C

✓ Existem 2 tipos de arquivo em C:

➡ **Binário**: composto por uma seqüência (fluxo binário) de bytes lidos, sem tradução, diretamente do dispositivo externo. Não ocorre nenhuma tradução e existe uma correspondência um para um entre os dados do dispositivo e os que estão no fluxo.

➢ **Texto**: composto de uma seqüência de caracteres perceptíveis para os usuários (como letras, dígitos e caracteres especiais utilizados na escrita) e pelos separadores *branco*, *tab* e ***new line*** (indicando final de linha e, dependendo do ambiente, utilizando para isso 1 ou 2 caracteres ASCII), que pode ou não ser dividida em linhas terminadas por um caractere de final de linha.

Arquivo Binário

• agregado sequencial de *bytes;*
• **logicamente, o stream pode visto como um conjunto de dados simples ou estruturados;**
• não tem limitação de tamanho;
• **comprimento do arquivo**: número de elementos que apresenta no momento;
• elementos não tem nome;
• somente um elemento é acessível a cada momento
→ **elemento corrente;**
• Acesso aos dados: seqüencial ou randômico.

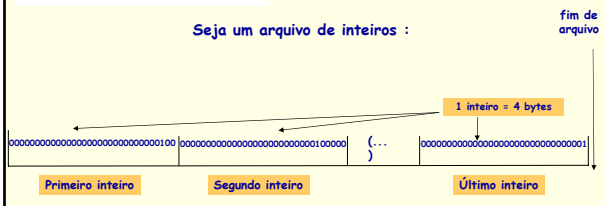
Arquivo

✓ Característica física:
➢ *stream* (fluxo de bytes).

✓ Característica lógica:
➢ Na memória principal, o buffer pode ser visualizado e manipulado pelo programa como uma estrutura (registro).

Arquivos: armazenamento

As separações entre *bytes* são apenas ilustrativas, no disco não existem!



Esquema de utilização de arquivos em *C*

A utilização de arquivo em C envolve os seguintes passos:

1. **Declaração** do arquivo através da forma

```
FILE *point_arq
```

ponteiro para referências ao arquivo neste programa

2. **Abertura** do arquivo, onde o ponteiro declarado é associado fisicamente ao arquivo externo através de

```
point_arq = fopen ("nome_do_arqu","acesso_desejado");
```

3. **Operações com arquivo**, através de funções de manipulação de arquivos (aplicadas ao ponteiro).
4. **Fechamento** do arquivo, através de

```
fclose (point_arq);
```


Declaração de um Arquivo

```
FILE *point_arq
```

- Para que um arquivo possa ser usado pelo programa é necessária que seja declarado através da estrutura global FILE, onde cada arquivo a ser usado pelo programa é associado a uma variável do tipo apontador.

```
// Programa que utiliza arquivos
#include <stdio.h> // e demais biblioteca necessárias
FILE *arq;  //declaração global de um "ponteiro de arquivo"
FILE *arq1, *arq2;
...
int main( )
{
.....
}
```

Para cada arquivo utilizado deve ter sido ou ser definido um ponteiro.

Observar que **FILE** é sempre escrito em maiúsculas!!!

Funções de manipulação de arquivo

- **fopen**
- **fclose**
- **fflush**
- **fread**
- **fwrite**
- **feof**
- **ferror**

O sucesso ou fracasso da execução destas funções pode ser verificado a partir do valor retornado. É altamente recomendado que os programas incluam esta verificação!

Função fopen

Protótipo: **FILE *_fopen_ (char *nome_de_arquivo, char *modo de abertura)**

Abre,cria ou recria um arquivo e retorna um ponteiro tipo FILE associado a este arquivo.

Ex.:

```
p_tes = fopen("teste", "wb");
p_exe = fopen("c:\\exemplo.bin", "rb");
```

Atenção: **dentro de programas, nomes de arquivos que incluam \ devem ser escritos com duas \\ se integrarem uma *string*.**

Abertura de arquivo

```
point_arq = fopen (nome do arquivo, modo de abertura)
```

- Abre/cria/recria um arquivo com o **nome** (completo, do arquivo físico) e **modo de abertura** desejado passados como parâmetro do tipo string.
- A função retorna um apontador para e estrutura FILE.
- Se não for possível abrir o arquivo, ocorre erro e o valor retornado em point_arq é NULL.

OBS: após abertura, o programa deve sempre verificar se não houve erro!

Modos de abertura para arquivos binários

"rb" – abre um arquivo binário para leitura. Se o arquivo não existir ocorrerá um erro!

"wb" – cria um arquivo binário para escrita. Se o arquivo já existir, a gravação ocorrerá sobre os dados existentes.

"ab" – se o arquivo não existir, é criado um novo e funciona tal qual o modo "wb"; mas se já existir, os novos dados são acrescidos no final do arquivo, a partir dos já existentes e de modo seqüencial.

"r+b" – abre um arquivo binário para leitura/escrita; se o arquivo não existir, ele é criado;

"w+b" – cria/recria um arquivo binário para escrita/leitura; se o arquivo não existir, ele é criado;

"a+b" – acrescenta dados na escrita e permite a leitura; se o arquivo não existir , ele é criado.

Obs.: os modos r+b, w+b e a+b também podem ser escritos rb+, wb+ e ab+.

Abertura de arquivo

Exemplo:

Retorna um apontador para uma estrutura FILE.

```
#include <stdio.h>      // p/ uso e manipulação de streams
#include <stdlib.h>
FILE *arq1, *arq2;  // declaração da estrutura
int main( )
{
 arq1 = fopen("c:\\exemplo.bin","rb"); /*abertura para leitura */
  if (!arq1)      // verifica se funcionou: retorno NULL indica erro
      printf("Erro na abertura do arquivo 1.");
  // abertura para leitura, junto à análise do resultado da operação:
  if (!(arq2 = fopen("c:\\exemplo.dat","rb"))
      printf("Erro na abertura do arquivo 2.");
....
}
```

19

Internamente ao programa, nomes de arquivos que incluam \ devem ser escritos com duas \\

Funções de manipulação de arquivo

fclose (point_arq)

Fecha um *stream* de arquivo
Obs.: ao fechar o *stream* do arquivo, todos os dados que ainda estavam em *buffer* são salvos em disco.

fflush (point_arq)

Descarrega o conteúdo em *buffer* para o arquivo, sem fechar o *stream*, corespondente a funções de gravação (como *fwrite*) que ainda não tenham sido efetivadas.

Funções de manipulação de arquivo

Exemplo:

```
...
   FILE *arq;
   arq = fopen("exemplo.bin","rb"); // se ocorrer erro, arq recebe null
   if (!arq)
      printf("Erro na abertura do arquivo.");
      /* na seqüência encerra a execução do programa */
   else
      {
        ...
        fclose(arq); /* fecha o arquivo */
      }
...
```


Arquivos binários acesso sequencial

Funções de manipulação de arquivo

fread (&varbuffer,numbytes,quant, point_arq)

Lê dados do arquivo para a memória principal
Se valor retornado ≠ **quant**, em geral é indicação de erro!

- lê o dado a partir do elemento corrente e o armazena em varbuffer;
- numbytes é o tamanho da unidade a ser lida;
- quant indica quantas unidades de numbytes serão lidas: logo, o número total de *bytes* que poderão ser lidos é igual a numbytes*quant;
- varbuffer pode se do mesmo tamanho que numbytes ou um ponteiro para uma região da memória;
- a quantidade - quant - de numbytes efetivamente lida é retornada pela função, podendo ser menor se o final do arquivo for alcançado previamente ou se ocorrer erro.

Funções de manipulação de arquivo

fwrite (&varbuffer,numbytes,quant, point_arq)

Grava dados no arquivo
Se valor retornado ≠ **quant**, em geral é indicação de erro!

- o funcionamento é similar a anterior, só que para escrita;
- a quantidade - quant - de numbytes efetivamente escrita (podem ocorrer erros ou buffer estar incompleto) é retornada pela função.

Funções de manipulação de arquivo

feof (point_arq)

Indica se um arquivo chegou ao fim.
Retorna verdadeiro (não-zero) caso o final tenha sido atingido, falso (zero) caso contrário.
Observe que o **feof** só é detectado através da leitura (mal sucedida) do final do arquivo: esta leitura deve ser ignorada !

```
...
  FILE *arq;
  ...
  while(!feof(arq)) /* Enquanto não for o fim faça... */
    { // cuidado: EOF é detectado através de leitura!!!
        ...
    }
  ...
...
```


Funções de manipulação de arquivo

ferror(point_arq)

Indica se a última operação com um arquivo produziu um erro.
Retorna verdadeiro (não-zero) se ocorrer algum erro, caso contrário, retorna falso (zero).
Cada nova operação com o arquivo modifica a condição de erro.

```
...
  FILE *arq;
  float pilido;
  arq = fopen(...,"rb");
  ...
  fread(&pilido,...,...,arq);
  if(ferror(arq))
    printf("Erro na Leitura!");
  ...
...
```


Exercícios

1. Faça um programa para gravar um arquivo composto por estruturas que contenham **nome**, **idade** e **altura** de atletas.
2. Faça outro programa, agora para listar um arquivo com estas estruturas.
3. Faça um terceiro programa, para imprimir a altura de um determinado atleta, cujo nome foi lido do teclado.
4. Faça outro programa que insere um novo registro no final do arquivo.

Definições necessárias

```
// Programa de manipulação de arquivos:
#include <stdio.h> // e demais biblioteca necessárias
#include <string.h>
#include <stdlib.h>
FILE *arq;  //declaração global de um "ponteiro de arquivo"
struct atleta
   { /*declaração da estrutura utilizada*/
     char nome[31];
     int idade;
     float altura;
   };
...
int main()
{

......

}
```


1. Programa para gravar um arquivo com nome, idade e altura de atletas:

```
int main( )  // se fosse função: void criainsere( )
  {
    struct atleta buffer;
    char nome[15];
    int op;
    printf("Nome do arquivo: ");
    scanf("%s",&nome);
    if(!(arq = fopen(nome,"wb"))) //cria ou grava por cima
        printf("Erro criacao");
    else
       {do
          { //coleta dados do usuário e grava no arquivo
            printf("\nNome: ");
            fgets(&buffer.nome, sizeof(buffer.nome)-1,stdin);
            fflush(stdin);
            printf("\nIdade: ");
            scanf("%d",&buffer.idade);
            printf("\nAltura: ");
            scanf("%f",&buffer.altura);
            fwrite(&buffer,sizeof(struct atleta),1,arq);
            printf("\n1-InserirNovo, 2-Encerrar\n");
            scanf("%i",&op);
          } while(op != 2);
        fclose(arq);  //fecha
       }
..... // comandos de encerramento normal de programa
}
```