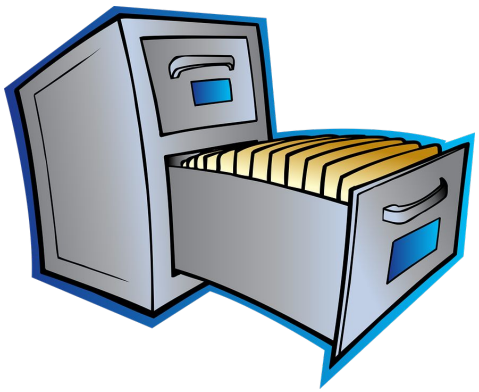
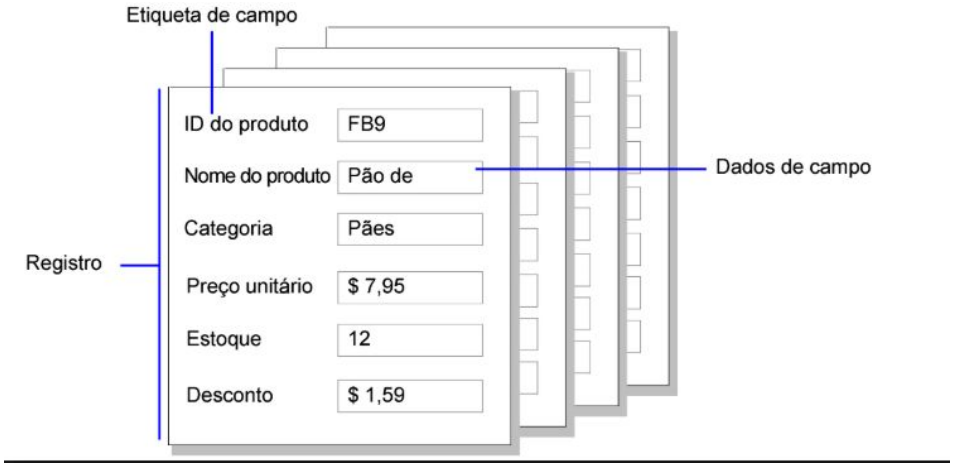
Algoritmos e Programação II

Prof. Joilson dos Reis Brito

Registros




Etiqueta de campo
ID do produto
FB9
Nome do produto
Pão de
Dados de campo
Categoria
Pães
Registro
Preço unitário
$ 7,95
Estoque
12
Desconto
$ 1,59

REGISTROS

Vetores e Matrizes são variáveis compostas homogêneas porque todos os seus elementos são do mesmo tipo(caracter, inteiro, real ou lógico).

Vetor Idades					
18	88	46	21	10	64

Vetor NomeEscritor								
S	t	a	n		L	e	e	\0

Matriz Notas alunos			
8,9	7,4	4,7	10,0
0,5	5,6	4,0	3,0
10,0	9,5	9,8	8,9
6,5	7,0	6,5	8,1

REGISTROS

Variáveis compostas heterogêneas são uma coleção de uma ou mais variáveis, sendo possível ser de tipos diferentes, colocadas juntas sob um único nome.

Em algoritmo são chamadas de Registros e em C são criadas com o comando **STRUCT**.

Variáveis compostas heterogêneas são utilizadas para representar um conjunto de informações que estão logicamente relacionadas.

Super-herói	Data da criação
Superman	1938

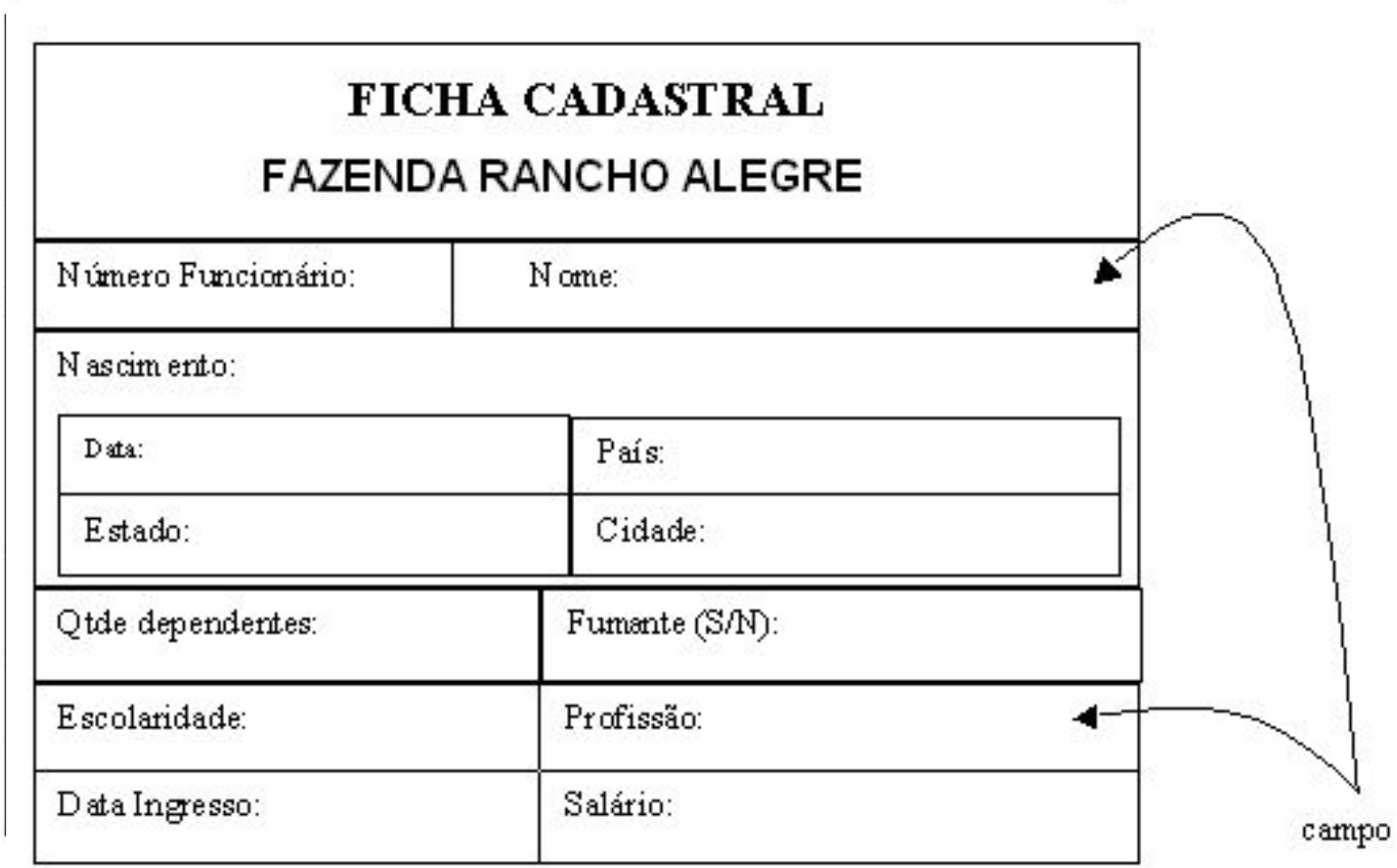
REGISTROS

Exemplos de informações que são armazenadas em registros:

- Uma ficha de cadastro de uma empresa tem as informações sobre um funcionário, necessárias para a empresa.
- Um cartão de ponto de uma empresa possui informações sobre um funcionário e seus horários de entrada e saída.
- Uma ficha de um consultório médico possui dados e características do estado de saúde dos pacientes.

REGISTROS

Suponha que uma fazenda chamada Rancho Alegre possui uma ficha de cadastro dos funcionários que trabalham na fazenda conforme formato abaixo:



REGISTROS

Percebe-se que a ficha possui um conjunto de informações relacionadas, ou seja, todas são dados de um funcionário pertencente a fazenda Rancho Alegre. A cada informação dar-se-á o nome de **campo**.

Algumas informações da ficha possuem o mesmo tipo de dado, por exemplo:

- nome, cidade, estado, país, profissão e fumante são do tipo **cadeia de caracteres**.

- Qtde dependentes é um valor do tipo **int** (Ex.: 10 dependentes).

- Salário é uma informação do tipo **float** (Ex.: R$ 1000,00)

REGISTROS

Para representar esse tipo de estrutura utilizamos o conceito de variáveis compostas heterogêneas ou registro (**struct** em C).

Variáveis compostas heterogêneas (registros) são um conjunto de informações (campos) relacionadas, onde cada campo pode ser de um tipo de dado diferente (**int, double, char**).

REGISTROS

Criação de um novo tipo:

Até o momento são conhecidos os tipos de dados básicos: inteiro (int), real (float ou double), caracter (char).

Quando for necessário utilizar um registro há a necessidade de definir um novo tipo.

REGISTROS

Sintaxe:

```
struct NOME_DO_NOVO_TIPO
 {
    tipo1 campo11, campo12, ..., campo1N;
    tipo2 campo21, campo22, ..., campo2N;
     ...
    tipoM campoM1, campoM2, ... campo MN;
 };
```

Onde:

NOME_DO_NOVO_TIPO é o identificador que especifica o nome do registro criado (novo tipo de dados). A partir desse nome podem ser criadas variáveis desse tipo.

tipo1, tipo2, tipo3, ... tipoM são os tipos de cada um dos campos do registro.

campo11, campo12, ..., campoMN são os nomes dos campos do registro.

REGISTROS

Exemplo:

```
struct FichaFuncionario
{
   int Numero, Nascimento, Dependentes,DataIngresso;
   char Nome[40],Pais[40],Estado[40],Cidade[40];
   char Fumante,Escolaridade[40],Profissao[40];
   float Salario;
};
```

REGISTROS

Declaração da variável do tipo registro:

NOME_DO_NOVO_TIPO regVar1, regVar2, ..., regVarN;

Onde:

regVar1, regVar2, ..., regVarN são as variáveis do tipo **NOME_DO_NOVO_TIPO** definido anteriormente.

Exemplo:

FichaFuncionario Funcionario;

REGISTROS

Acesso a um campo do registro:

O acesso aos campos das variáveis do tipo registro é realizado através do operador ponto " . ", conforme abaixo:

Exemplos:

Funcionario.Nascimento=01041989;

gets(Funcionario.Nome);

printf("Nome: %s Data Nascimento: %i", Funcionario.**Nome,** Funcionario.**Nascimento);**

REGISTROS

RESUMINDO

FichaFuncionario é um tipo criado no programa no qual o registro é criado.

Funcionario é uma variável do tipo FichaFuncionario.

Funcionario.Nascimento é um campo da variável Funcionario que armazena a data de nascimento de um funcionário.

VETORES DE REGISTROS

Mas se fosse necessário ler as informações de 100 funcionários, eu iria criar 100 variáveis ?

VETORES DE REGISTROS

Exemplo com o registro de funcionários:

```
FichaFuncionario Funcionarios[100];
```

Vetores de Registros

O acesso aos campos será feita normalmente com o índice do vetor:

Funcionarios[Indice].Nome

Funcionarios[Indice].Salario

Exemplo

Programa que lê os nomes e salários de vários funcionários da Fazenda Rancho Alegre e escreve o nome do com o maior salário.

Exemplo

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

#define Tamanho 3
// Criação do tipo com as informações do registro
struct FichaFuncionario
{
  int Numero, Nascimento, Dependentes,DataIngresso;
  char Nome[40],Pais[40],Estado[40],Cidade[40];
  char Fumante,Escolaridade[40],Profissao[40];
  float Salario;
};
```

Exemplo

```
int main()
{
    setlocale(LC_ALL, "Portuguese");
    FichaFuncionario Funcionarios[Tamanho];
    int I,IndiceMaior;
    float MaiorSalario;
    for(I=0;I<Tamanho;I++)
    {
        printf("Funcionário %d\n", I+1);
        fflush(stdin);
        printf("Nome: ");gets(Funcionarios[I].Nome);
        printf("Salario: ");scanf("%f",&Funcionarios[I].Salario);
        system("cls");
    }
    MaiorSalario = Funcionarios[0].Salario;
    IndiceMaior = 0;
```

Exemplo

```
    for(I=1;I<Tamanho;I++)
      if(Funcionarios[I].Salario>MaiorSalario)
        {
            MaiorSalario = Funcionarios[I].Salario;
            IndiceMaior = I;
        }
    printf("Funcionário com maior salário: %s \n",Funcionarios[IndiceMaior].Nome);
    system("pause");
    return 0;
}
```

Exemplo de programa com vetor de registros

Faça um programa que cria tipo chamado HistoricoSuperheroi que vai armazenar o nome e o ano de criação de um superheroi. Depois o programa deve criar um vetor de 7 posições do tipo HistoricoSuperheroi e ler os nomes e o ano de criação dentro do vetor. Por fim o programa deverá imprimir a relação dos superherois classificados pelo ano de criação.

PLANO DE TESTE

ENTRADA	SAÍDA
Super-herói: Quarteto Fantástico Ano Criação: 1961 Super-herói: Lanterna Verde Ano Criação: 1940 Super-herói: Batman Ano Criação: 1938 Super-herói: Homem-Aranha Ano Criação: 1962 Super-herói: Capitão América Ano Criação: 1941 Super-herói: Superman Ano Criação: 1938 Super-herói: Mulher-Maravilha Ano Criação: 1941	1938 Superman 1938 Batman 1940 Lanterna Verde 1941 Mulher-Maravilha 1941 Capitão América 1961 Quarteto Fantástico 1962 Homem-Aranha

```
struct HistoricoSuperHeroi
{
      char Nome[40];
      int AnoCriacao;
};

HistoricoSuperHeroi Menor;

HistoricoSuperHeroi SuperHerois[7];
```

Menor

Nome	AnoCriacao

SuperHerois

	Nome	AnoCriacao
0		
1		
2		
3		
4		
5		
6		

Após leitura:

SuperHerois

	Nome	AnoCriacao
0	Quarteto Fantástico	1961
1	Lanterna Verde	1940
2	Batman	1938
3	Homem-Aranha	1962
4	Capitão América	1941
5	Superman	1938
6	Mulher-Maravilha	1941

Ordenação:

I = 0;

Menor = SuperHerois[I];

SuperHerois

	Nome	**AnoCriacao**
0	Quarteto Fantástico	**1961**
1	Lanterna Verde	**1940**
2	Batman	1938
3	Homem-Aranha	1962
4	Capitão América	1941
5	Superman	1938
6	Mulher-Maravilha	1941

Menor

Nome	**AnoCriacao**
Quarteto Fantástico	**1961**

J = I + 1;

Compara SuperHeroi[1].AnoCriacao comMenor.AnoCriacao

Ordenação:

I = 0;

Menor = SuperHerois[I];

SuperHerois

	Nome	**AnoCriacao**
0	Quarteto Fantástico	**1961**
1	Lanterna Verde	**1940**
2	Batman	1938
3	Homem-Aranha	1962
4	Capitão América	1941
5	Superman	1938
6	Mulher-Maravilha	1941

Menor

Nome	**AnoCriacao**
Quarteto Fantástico	**1961**

Ordenação:

I = 0;

SuperHerois

	Nome	AnoCriacao
0	Quarteto Fantástico	**1961**
1	Lanterna Verde	**1940**
2	Batman	1938
3	Homem-Aranha	1962
4	Capitão América	1941
5	Superman	1938
6	Mulher-Maravilha	1941

Menor

Nome	AnoCriacao
Quarteto Fantástico	**1961**

J = I + 1; J = 1;

Compara SuperHeroi[1].AnoCriacao com Menor.AnoCriacao.

É menor!

Ordenação:

SuperHerois

	Nome	AnoCriacao
0	Quarteto Fantástico	**1961**
1	Lanterna Verde	**1940**
2	Batman	1938
3	Homem-Aranha	1962
4	Capitão América	1941
5	Superman	1938
6	Mulher-Maravilha	1941

Menor

Nome	AnoCriacao
Lanterna Verde	**1940**

o Registro Menor recebe o registro armazenado na posição 1.
Aux = 1;

Ordenação:

J++; J = 2

Compara SuperHeroi[2].AnoCriacao comMenor.AnoCriacao.

É menor!

SuperHerois

	Nome	AnoCriacao
0	Quarteto Fantástico	**1961**
1	Lanterna Verde	**1940**
2	Batman	**1938**
3	Homem-Aranha	1962
4	Capitão América	1941
5	Superman	1938
6	Mulher-Maravilha	1941

Menor

Nome	AnoCriacao
Lanterna Verde	**1940**

o Registro Menor recebe o registro armazenado na posição 2.

Ordenação:

SuperHerois

	Nome	AnoCriacao
0	Quarteto Fantástico	**1961**
1	Lanterna Verde	**1940**
2	Batman	**1938**
3	Homem-Aranha	1962
4	Capitão América	1941
5	Superman	1938
6	Mulher-Maravilha	1941

Menor

Nome	AnoCriacao
Batman	**1938**

o Registro Menor recebe o registro armazenado na posição 2.
Aux = 2;

Ordenação:

J++; J = 3

Compara SuperHeroi[3].AnoCriacao comMenor.AnoCriacao.

Não é menor!

SuperHerois

	Nome	**AnoCriacao**
0	Quarteto Fantástico	**1961**
1	Lanterna Verde	**1940**
2	Batman	**1938**
3	Homem-Aranha	**1962**
4	Capitão América	1941
5	Superman	1938
6	Mulher-Maravilha	1941

Menor

Nome	**AnoCriacao**
Batman	**1938**

Ordenação:

J++; J = 4

Compara SuperHeroi[4].AnoCriacao comMenor.AnoCriacao.

Não é menor!

SuperHerois

	Nome	AnoCriacao
0	Quarteto Fantástico	**1961**
1	Lanterna Verde	**1940**
2	Batman	**1938**
3	Homem-Aranha	**1962**
4	Capitão América	**1941**
5	Superman	1938
6	Mulher-Maravilha	1941

Menor

Nome	AnoCriacao
Batman	**1938**

Ordenação:

J++; J = 5

Compara SuperHeroi[5].AnoCriacao comMenor.AnoCriacao.

Não é menor!

SuperHerois

	Nome	**AnoCriacao**
0	Quarteto Fantástico	**1961**
1	Lanterna Verde	**1940**
2	Batman	**1938**
3	Homem-Aranha	**1962**
4	Capitão América	**1941**
5	Superman	**1938**
6	Mulher-Maravilha	1941

Menor

Nome	**AnoCriacao**
Batman	**1938**

Ordenação:

J++; J = 6

Compara SuperHeroi[6].AnoCriacao com Menor.AnoCriacao.

Não é menor!

SuperHerois

	Nome	AnoCriacao
0	Quarteto Fantástico	**1961**
1	Lanterna Verde	**1940**
2	Batman	**1938**
3	Homem-Aranha	**1962**
4	Capitão América	**1941**
5	Superman	**1938**
6	Mulher-Maravilha	**1941**

Menor

Nome	AnoCriacao
Batman	**1938**

Após a primeira passagem do segundo for, o programa identifica o primeiro menor ano de criação e faz a troca com o registro da posição 0;

Ordenação:

SuperHerois[2]= SuperHerois[0];
SuperHerois[0] = Menor;

SuperHerois

	Nome	AnoCriacao
0	Batman	**1938**
1	Lanterna Verde	**1940**
2	Quarteto Fantástico	**1961**
3	Homem-Aranha	**1962**
4	Capitão América	**1941**
5	Superman	**1938**
6	Mulher-Maravilha	**1941**

Menor

Nome	AnoCriacao
Batman	**1938**

O processo se repete até que a variável I alcance o limite do vetor.

Programa Completo

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#define TAM 5

struct HistoricoSuperheroi
{
    char Nome[40];
    int AnoCriacao;
};
```

```
int main()
{
    setlocale(LC_ALL,"Portuguese");
    int I, J, Aux;
    HistoricoSuperheroi SuperHerois[TAM];
    HistoricoSuperheroi Menor;
    for(I=0; I<TAM; I++)
    {
        printf("\nSuper Herói %d\n",I+1);
        printf("Nome: ");
        fflush(stdin);
        gets(SuperHerois[I].Nome);
        printf("Ano de criação: ");
        scanf("%d",&SuperHerois[I].AnoCriacao);
    }
    for(I=0;I<TAM - 1; I++)
    {
        Menor = SuperHerois[I];
        Aux = I;
        for(J=I+1; J < TAM; J++)
        {
            if(SuperHerois[J].AnoCriacao < Menor.AnoCriacao)
            {
                Menor = SuperHerois[J];
                Aux = J;
            }
        }
        SuperHerois[Aux]= SuperHerois[I];
        SuperHerois[I] = Menor;
    }
```

```
    printf("Ano de Criação - Nome\n");
    for(I=0; I<TAM; I++)
    {
       printf("    %d     -   %s\n ",SuperHerois[I].AnoCriacao,SuperHerois[I].Nome);
    }
    system("Pause");
    return 0;
}
```

Arquivo de Registros

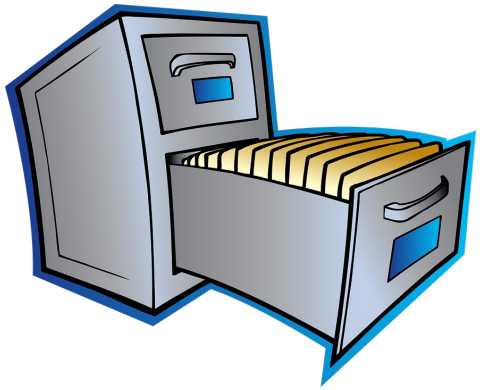
Foram digitadas várias informações para testar esse programa dos Super Heróis…

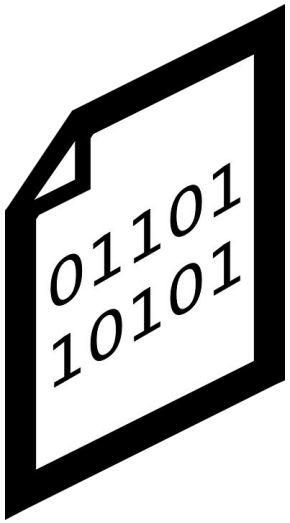
Para executar meu programa vou precisar digitar todos os super heróis novamente?

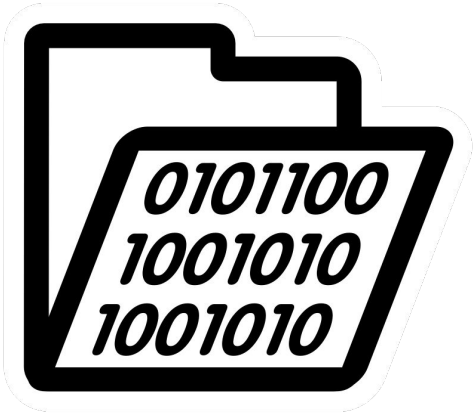
Isso acontece porque o vetor fica na memória RAM do computador.

Arquivo de Registros

ARQUIVOS BINÁRIOS







Arquivos Binários

Até agora, estudamos como processar arquivos de texto em nossos programas. Nos arquivos de texto, existe um formato interno que é identificado pela ocorrência do '**\n**' separando os caracteres encontrados no arquivo por linhas.

Arquivos binários são, em geral, arquivos de dados, executáveis, jogos, etc.

Em arquivos binários não faz sentido ler linha por linha, pelo fato de não estarem organizados dessa forma.

Arquivos Binários

As funções para tratamento de arquivos binários são diferentes das funções usadas para arquivos de texto.

As operações que vamos usar para processar arquivos binários são denominadas **Acesso Direto**, e só podem ser utilizadas para arquivos abertos em modo **binário**.

No acesso direto os dados são escritos em blocos da memória para o disco e lidos em blocos do disco para a memória.

Arquivos Binários

Arquivos Binários permitem escrever um vetor inteiro no disco de uma só vez, e o mesmo acontece para fazer a leitura do vetor também!

Modos de Abertura de Arquivo Binário

Por padrão, a abertura de um arquivo é realizada considerando que o arquivo é de texto.

Para abrir o arquivo em modo binário é necessário acrescentar um **b** aos modos de abertura que já estudamos:

rb, wb, ab, r+b, w+b, a+b.

Leitura e Escrita em Arquivos Binários

As funções que permitem leitura e escrita em arquivos binários são:

fread e **fwrite**

Escrita de Blocos em Arquivos Binários

A função fwrite faz parte do arquivo stdio.h e é responsável por escrever um bloco de bytes existente em memória para um arquivo aberto em modo binário.

Sixtaxe:

int fwrite(const void *ptr, int size, int n, FILE *arq)

Escrita de Blocos em Arquivos Binários

int fwrite(const void *ptr, int size, int n, FILE *arq)

Em que:

prt - é um ponteiro que contém o endereço de memória daquilo que queremos guardar no arquivo (const indica que o parâmetro não será alterado)

size - indica o tamanho em bytes de cada um dos elementos que pretendemos escrever. O comando sizeof(tipo) devolve a quantidade de bytes do endereço para o qual ptr aponta.

n - Indica o número de elementos que queremos escrever.

arq - ponteiro para o arquivo no qual queremos escrever.

A função **fwrite** devolve, após a gravação, o número de itens que conseguiu escrever com sucesso no arquivo.

Exemplo 1 - Escrita de Blocos em Arquivos Binários

Programa que realiza leitura de um vetor com 10 idades a partir do teclado e as armazena do arquivo Dados.dat.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#define TAM 10

int main()
{
        FILE *PontArq;
        int Idades[TAM], I;
        int Validacao;
        setlocale(LC_ALL,"Portuguese");
        printf("Digite as Idades:");
        for(I=0; I<TAM; I++)
        {
                printf("\nIdades[%d]: ",I);
                scanf("%d",&Idades[I]);
        }
        PontArq = fopen("Dados.dat","wb");
        if(PontArq == NULL)
        {
                printf("Impossível Abrir o arquivo!\n");
                exit(1);
        }
        Validacao = fwrite(Idades,sizeof(int),TAM, PontArq);
        if(Validacao != TAM)
          printf("\nErro ao gravar todos os dados!\n");
        else
          printf("Dados Gravados com sucesso no arquivo!\n");
      fclose(PontArq);
        system("Pause");
        return 0;
}
```

A função sizeof(int) devolve o inteiro referente a quantidade de bytes que um int ocupa.

Leitura de Blocos em Arquivos Binários

A função fread também faz parte do arquivo stdio.h e é responsável por ler um bloco de bytes existente em memória para um arquivo aberto em modo binário.

Sixtaxe:

int fread(const void *ptr, int size, int n, FILE *arq)

Leitura de Blocos em Arquivos Binários

int fread(const void *ptr, int size, int n, FILE *arq)

Em que:

prt - é um ponteiro que contém o endereço de memória daquilo que queremos ler do arquivo (const indica que o parâmetro não será alterado)

size - indica o tamanho em bytes de cada um dos elementos que pretendemos ler.

n - Indica o número de elementos que queremos ler.

arq - ponteiro para o arquivo no qual queremos ler.

A função fread retorna o número de itens que conseguiu ler com sucesso no arquivo.

Exemplo 2 - Leitura de Blocos em Arquivos Binários

Programa que carrega um vetor com 10 idades lidas a partir do arquivo Dados.dat gravados na memória pelo programa anterior. Depois de ler os 10 inteiros, o programa deverá mostrar o vetor na tela.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#define TAM 10

int main()
{
	FILE *PontArq;
	int Idades[TAM], I;
	int Validacao;
	setlocale(LC_ALL,"Portuguese");
	PontArq = fopen("Dados.dat","rb");
	if(PontArq == NULL)
	{
	   printf("Impossível Abrir o arquivo!\n");
	  exit(1);
	}
	Validacao = fread(Idades,sizeof(int),TAM, PontArq);
	if(Validacao != TAM)
	  printf("\nErro ao ler todos os dados!\n");
	else
	{
	  for(I=0; I<TAM; I++)
	  {
	     printf("Idades[%d]: %d\n",I, Idades[I]);
	  }
	}
	fclose(PontArq);
	system("Pause");
	return 0;
}
```

Acesso Sequencial e Acesso Direto a Arquivos

O acesso à informação pode ser realizado de duas formas:

Acesso Sequencial: Como temos feito até agora, percorrendo o arquivo até localizarmos aquilo que pretendemos.

Acesso Direto: Colocando-nos na posição que queremos de um arquivo, sem ter que percorrer todos os bytes que se encontram antes dessa posição.

Posicionamento ao Longo do Arquivo

Sempre que se abre um arquivo através da função **fopen** é criada em memória uma estrutura do tipo FILE que contém informações sobre o arquivo que se está processando.

Nessa informação existe a posição em que estamos no arquivo, isso porque as funções de leitura e escrita avançam automaticamente no arquivo à medida que vão lendo ou escrevendo informações.

Posicionamento ao Longo do Arquivo

A linguagem C não vê o arquivo de forma formatada, por isso a posição corrente do arquivo é dada em termos de bytes.

Sempre que um arquivo é aberto (exceto para acrescentar “a”), ele é posicionado antes do primeiro byte do arquivo, isto é, na posição 0 (zero).

Se o arquivo for aberto para acrescentar, ele é posicionado após o último byte do arquivo.

Posicionamento ao Longo do Arquivo

O posicionamento para arquivos só faz sentido para **Arquivos Binários**.

Posicionamento ao Longo do Arquivo

Para saber a posição corrente em um arquivo, existe a função **ftell**.

Sintaxe:

long ftell(FILE *Arq)

Observe que a função retorna um long e não um int, isso porque a dimensão dos arquivos pode ultrapassar o valor máximo representado por um inteiro.

Exemplo 3 - Posição do Arquivo

Programa que carrega de um vetor com 10 idades lidas a partir do arquivo Dados.dat gravados na memória pelo programa anterior. Depois de ler os 10 inteiros, o programa deverá mostrar o vetor na tela e acessa o posicionamento final do arquivo mostrando seu valor.

```
#define TAM 10
int main()
{
	FILE *PontArq;
	int Idades[TAM], I;
	int Validacao;
	setlocale(LC_ALL,"Portuguese");
	PontArq = fopen("Dados.dat","rb");
	if(PontArq == NULL)
	{
		printf("Impossível Abrir o arquivo!\n");
		exit(1);
	}
	 printf("Posição inicial do arquivo: %d\n",ftell(PontArq));
	Validacao = fread(Idades,sizeof(int),TAM, PontArq);
	if(Validacao != TAM)
	  printf("\nErro ao ler todos os dados!\n");
	else
	{
		for(I=0; I<TAM; I++)
	  {
		  printf("Idades[%d]: %d\n",I, Idades[I]);
	  }
	}
	printf("O tamanho d arquivo é de %d Bytes.\n",ftell(PontArq));
	fclose(PontArq);
	system("Pause");
	return 0;
}
```

Posicionamento ao Longo do Arquivo

Independente do local onde nos encontramos no arquivo, é sempre possível voltar a colocar o ponteiro para apontar para o início do arquivo através do **rewind,** evitando fechar o arquivo e abri-lo novamente simplesmente para voltar ao início do arquivo.

A função mais importante de posicionamento num arquivo é a função **fseek**, pois nos permite apontar o ponteiro para qualquer posição do arquivo.

Posicionamento ao Longo do Arquivo

Sintaxe:

int fseek(FILE *Arq, long Salto, int Origem)

Onde:

Arq - Arquivo sobre o qual podemos operar.

Salto - Indica o número de bytes que pretendemos andar. Um valor positivo indica andar para frente, um valor negativo indica que pretendemos andar para trás.

Origem - Indica o local a partir do qual queremos realizar o salto no arquivo. São admitidos apenas três valores, que são definidos como constantes.

Constante	**Valor**	**Significado**
SEEK_SET	0	O salto é realizado a partir da origem do arquivo.
SEEK_CUR	1	O salto é realizado a partir da posição corrente do arquivo.
SEEK_END	2	O salto é realizado a partir do final do arquivo.

A função devolve 0 se o movimento foi realizado com sucesso dentro do arquivo, ou um valor diferente de 0, caso contrário.

Exemplo 4 - Posição do Arquivo

Programa que abre o arquivo Dados.dat e mostra quantos bytes ele tem, obtendo esse valor através de Acesso Direto, isto é, sem ter que percorrer todo o arquivo.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#define TAM 10

int main()
{
	FILE *PontArq;
	int Idades[TAM], I;
	int Validacao;
	setlocale(LC_ALL,"Portuguese");
	PontArq = fopen("Dados.dat","rb");
	if(PontArq == NULL)
	{
		printf("Impossível Abrir o arquivo!\n");
		exit(1);
	}
	fseek(PontArq,0,SEEK_END); //OU fseek(PontArq,0,2);
	printf("O tamanho do arquivo é de %d Bytes.\n",ftell(PontArq));
	fclose(PontArq);
	system("Pause");
	return 0;
}
```

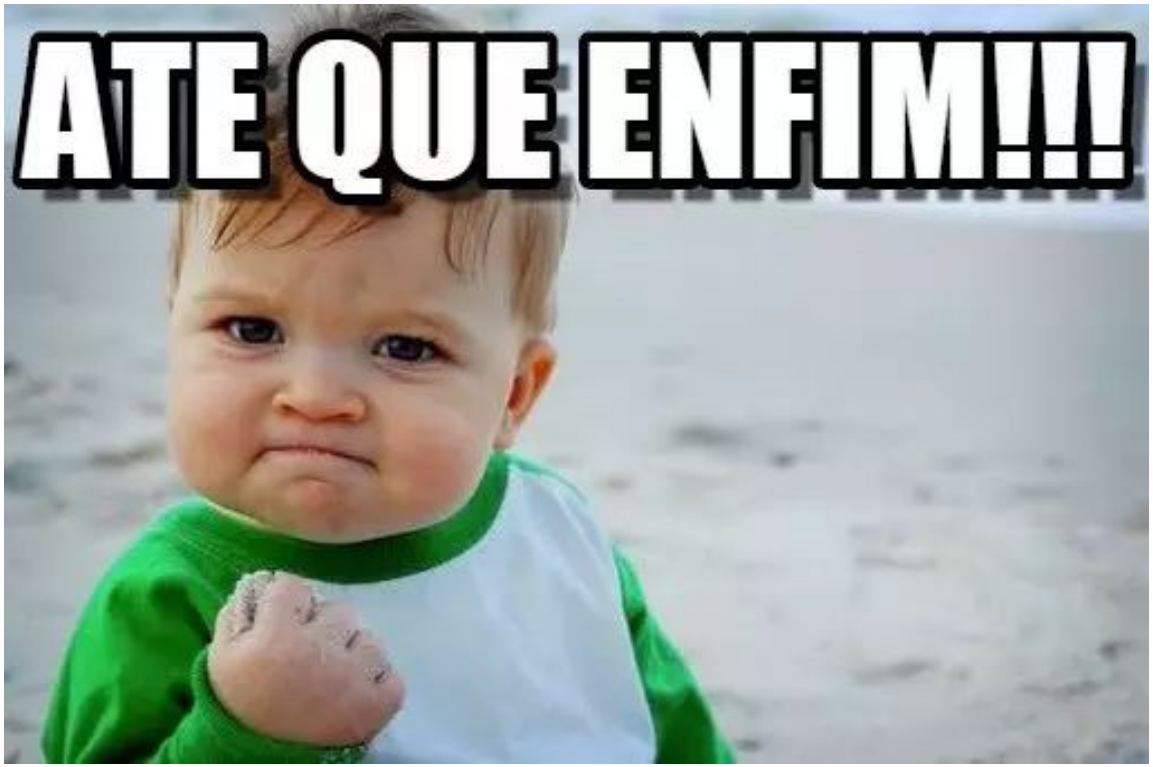
Exemplo 5 - Posição do Arquivo

Programa que abre o arquivo Dados.dat e solicita ao usuário qual a idade que ele deseja pesquisar no arquivo. Mostra também a primeira e última idade do arquivo.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#define TAM 5

int main()
{
	FILE *PontArq;
	int Idades[TAM], I;
	int OrdemIdade, IdadePesquisada;
	setlocale(LC_ALL,"Portuguese");
	PontArq = fopen("Dados.dat","rb");
	if(PontArq == NULL)
	{
		printf("Impossível Abrir o arquivo!\n");
		exit(1);
	}
	printf("Qual a ordem da idade que deseja visualizar?\n");
	scanf("%d",&OrdemIdade);
	fseek(PontArq,(OrdemIdade-1)*sizeof(int),SEEK_SET);
	fread(&IdadePesquisada,sizeof(int),1,PontArq);
	printf("\nA %dª Idade do arquivo é: %d\n",OrdemIdade,IdadePesquisada);
	rewind(PontArq);
	fread(&IdadePesquisada,sizeof(int),1,PontArq);
	printf("\nA 1ª Idade do arquivo é: %d\n",IdadePesquisada);
	fseek(PontArq,- sizeof(int),SEEK_END);
	fread(&IdadePesquisada,sizeof(int),1,PontArq);
	printf("\nA 10ª Idade do arquivo é: %d\n",IdadePesquisada);
	fclose(PontArq);
	system("Pause");
	return 0;
}
```

ARQUIVOS DE REGISTROS (struct)



Arquivos de Registros (struct)

O processamento de structs armazenadas em arquivos é sempre realizado recorrendo a arquivos abertos em modo binário.

As structs são, em geral, lidas e escritas através de funções fread e fwrite.

Exemplo 6 - Programa dos Super Heróis

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#define TAM 3

struct HistoricoSuperheroi
{
	char Nome[40];
	int AnoCriacao;
};


int main()
{
	setlocale(LC_ALL,"Portuguese");
	int I, J, Aux;
	HistoricoSuperheroi SuperHerois[TAM];
	HistoricoSuperheroi Menor;
	FILE *PontArq;
	PontArq = fopen("ArquivoSuperHerois.dat","w+b");
	 if(PontArq == NULL)
	  {
	   printf("Impossível abrir o arquivo.\n");
	   exit(1);
	   }
```

Exemplo 6 - Programa dos Super Heróis

```
for(I=0; I<TAM; I++)
{
	printf("\nSuper Herói %d\n",I+1);
	printf("Nome: ");
	fflush(stdin);
	gets(SuperHerois[I].Nome);
	printf("Ano de criação: ");
	scanf("%d",&SuperHerois[I].AnoCriacao);
}
fwrite(SuperHerois,sizeof(HistoricoSuperheroi),TAM,PontArq);
```

Exemplo 6 - Programa dos Super Heróis

```
fclose(PontArq);
PontArq = fopen("ArquivoSuperHerois.dat","rb");
if(PontArq == NULL)
{
      printf("Impossível abrir o arquivo.\n");
      exit(1);
 }
 fread(SuperHerois,sizeof(HistoricoSuperheroi),TAM,PontArq);
 fclose(PontArq);
 for(I=0;I<TAM - 1; I++)
    {
          Menor = SuperHerois[I];
          Aux = I;
          for(J=I+1; J < TAM; J++)
          {
                if(SuperHerois[J].AnoCriacao < Menor.AnoCriacao)
                {
                      Menor = SuperHerois[J];
                      Aux = J;
                }
          }
          SuperHerois[Aux]= SuperHerois[I];
          SuperHerois[I] = Menor;
    }
fclose(PontArq);
```

Exemplo 6 - Programa dos Super Heróis

```
    PontArq = fopen("ArquivoSuperHerois.dat","wb");
    if(PontArq == NULL)
    {
         printf("Impossível abrir o arquivo.\n");
         exit(1);
       }
    fwrite(SuperHerois,sizeof(HistoricoSuperheroi),TAM,PontArq);
    PontArq = fopen("ArquivoSuperHerois.dat","r+b");
    if(PontArq == NULL)
    {
         printf("Impossível abrir o arquivo.\n");
         exit(1);
     }
     fread(SuperHerois,sizeof(HistoricoSuperheroi),TAM,PontArq);
     printf("Ano de Criação - Nome\n");
     for(I=0; I<TAM; I++)
      {
          printf("    %d     -   %s\n ",SuperHerois[I].AnoCriacao,SuperHerois[I].Nome);
      }
    fclose(PontArq);
    system("Pause");
    return 0;
}
```