Estrutura de Dados

Profa.: Marcela

Revisão C

Etapas para desenvolver o programa

- Análise: nesta etapa estuda-se o enunciado do problema, quais são os dados de entrada, o processamento e a saída;
- Algoritmo: ferramentas como descrição narrativa, fluxograma e português estruturado são utilizados para descrever o problema e suas soluções;
- Codificação: o algoritmo é transformado em código de linguagem de programação para se trabalhar;

Estrutura do Programa

```
int main()
{
printf("primeiro programa em c");
return(0);
}
```

E se eu quiser passar argumentos para a função main?

```
int main(int argc, char * argv[])
...
```


Tipos de Variáveis

- Em C existem 5 tipos básicos:

tipo	bit	byte	escala
char	8	1	-128 a 127
int	16	2	-32768 a 32767
float	32	4	3.4E-38 a 3.4E+38
double	64	8	1.7E-308 a 1.7E+308
void	0	0	sem valor

Modificadores

short

unsigned

long

O tamanho e o intervalo coberto por cada tipo varia de acordo com o tipo de computador.

O **int** tem sempre o tamanho da palavra de máquina. Se em um computador a palavra tem 16 bits, então o inteiro tem 16 bits de tamanho.

short int

long int

unsigned int

long double

A função *printf*

- Sintaxe: **printf(exp,lista de argumentos)**

Exemplo: **printf("nossa idade varia de: \n %s a %d" anos, "zero" ,100);**

Principais caracteres especiais:
\n =nova linha
\t =TAB
\" =aspas
**** =barra
**0 ** =nullo

Principais códigos para a impressão formatada de *printf*

%c =caracter

%d =decimal

%e =notação científica

%f =float

%s =string (cadeia de caracteres)

A função *scanf*

•Função de E/S usada para ler dados formatados da entrada padrão.
- •Sintaxe: scanf("cod. formatação", lista argumentos);
- •Lista de argumentos: endereço de variáveis.

Cod. formatação

código	Função ler
%c	Um caractere
%d	Inteiro decimal
%e	Notação científica
%f	Ponto Flutuante
%s	Inteiro octal
%x	Cadeia de caracteres
%u	Decimal sem sinal
%l	Inteiro longo
%lf	double

Exercício: Faça um programa que peça sua idade e exiba o número de horas vividas.

A função getche()

- Com *scanf* é necessário pressionar [enter] para terminar a leitura;
- ***getche*:** lê um caractere e o retorna. Não é necessário pressionar enter.

```
void main(void)
{
  char ch;
  printf("Digite um caractere: ");
  ch=getche();
  printf("\nVoce digitou o caracter %c.",ch);
}
```

Saída:
Digite um caractere: r
Voce digitou o caracter r.

A função getch()

- *getch*: Semelhante a *getche*, porém não permite que o caractere seja impresso na tela

```
void main(void)
{
  char ch;
  printf("Digite um caractere: ");
  ch=getch();
  printf("\nVoce digitou o caracter %c.",ch);
}
```

Saída:
Digite um caractere:
Voce digitou o caracter r.

Precedência de Operadores

Ordem de Precedência
- (menos unário)
%
*, /
+,-
<,<=, >=, >,!=

A) Qual é a saída de:

```
void main(void) {
  int exp= 2*9%9;
  printf("%d",exp); }
```

Saída: 0;1

D) Faça um programa que calcule a temperatura em Celsius. Dado a temperatura de entrada em Fahrenheit.

C = (F-32)* 5/9

O que acontece se eu tirar o parênteses da expressão anterior?

B) Qual é a saída de:

```
void main(void) {
  int exp= 2*-9+8/2*2+8*3/2-1 ;
  printf("%d",exp);}
```


Operadores de incremento Pré-fixado e Pós-fixado

- **Pré-fixado:** primeiro o valor de seu argumento é atualizado e posteriormente a expressão é executada.
- **Pós-fixado:** primeiro a expressão é atualizada e posteriormente o valor de seu argumento é atualizado.
- Em expressões que o operador aparece sozinho na instrução não faz diferença se o operador é pré-fixado ou pós-fixado.

```
void main(void){
  int a=10;
  a= a+4*++a;
  printf("%d\n",a);
}
```

```
void main(void){
  int a=10, int b;
  b= a+4*a++;
  printf("%d\n",b);
}
```

```
void main(void){
  int a=10;
  a= a+4*a++;
  printf("%d\n",a);
}
```

Saída: 55;50;51

Operadores Lógicos

- && E lógico (binário)
- || Ou lógico (binário)
- ! Negação Lógica (unário)

Comandos Condicionais/laços

- if, switch-case
- for, while, do-while

Operador Condicional Ternário ? :

- Maneira compacta de expressar uma instrução ***if*** e ***else***.
  condição? expressão 1: expressão 2

- Exemplos:

int max = (num1>num2)?num1:num2

equivale a:

```
if(num1>num2)
  max = num1;
else
  max = num2;
```

13

Funções

- A transmissão de informações para uma função é feita através de seus argumentos.
- Funções que não retornam nada são do tipo void

```
int abs(int x){
 return( x>0 ? x : -x);
}
```

Faça uma função que calcule a potência de um número.
Exemplo: pot(5,3) = 53= 5x5x5=125

```
int pot(int a, int b){
  int i,pow = 1;
  for(i=1; i<=b ;i++){
    pow*=a;
  }
  return pow;
}
```

14

Funções Recursivas

- Uma função é chamada recursiva se ela é definida em termos de ela mesmo.
- Exercício:
  - Implemente a função *fatorial* que calcula o fatorial. A função *main* deve solicitar um número x e, se x for positivo, chamar a função *fatorial* para calcular o *fatorial* desse número. Se x for negativo o programa é finalizado.

```
int fat(int r)
{
  if(r == 0)
    return 1;
  return r* fat(r-1);
}
```

Como funciona a recursão?

15

Funções Recursivas Exercício

- Faça uma função recursiva que calcule a potência x^y, para y≥0.
  - Exemplo: pot(5,3) = 5^3= 5x5x5=125

```
int pow(int x,int y)
{
  if(y == 0)
    return 1;
  return x* pow(x,y-1);
}
```

16

Classes de Armazenamento

- As variáveis em C podem ser
  - auto: automáticas
  - extern: externas
  - static: estáticas
  - register: em registradores

17

Classe auto

- As variáveis locais são da classe **auto**:
  - são criadas quando a função é chamada
  - destruídas quando a função termina sua execução.
  - as variáveis declaradas dentro da função são automáticas por default.
  - a classe de variável auto pode ser explicitada usando a palavra chave auto:

auto int n;

18

Classe extern

- Em C, todas as funções e todas as variáveis declaradas fora de qualquer função é da classe de armazenamento **extern.**
- Essas variáveis **extern** são conhecidas por todas as funções declaradas depois dela.
- Variáveis **extern** retém seus valores durante toda a execução do programa. Mantendo sua posição de memória alocada.
- A palavra **extern** indica que a função usará uma variável externa. Essa declaração não é obrigatória se a variável foi declarada no mesmo arquivo.

Classe extern
Exemplo

```
int a;
void func1(){
   extern int a;
   a=4;
}
void func2(){
  a=2;
}
```

```
void func3(){
  int a;
  a=3;
}
```

Saída:
4
2
2

```
int main(int){
  a=100;
  func1();
  printf("\n%d", a);
  func2();
  printf("\n%d", a);
  func3();
  printf("\n%d", a);
   return(0);
}
```

Qual é a saída?

Todas as funções acessaram a mesma variável externa?

Classe static

- São conhecidas no escopo em que são declaradas:
  - local: se declarada dentro da função
  - externa: se declarada fora da função
- Mantém seus valores mesmo após a execução da função (no caso de ser uma variável local).
- Variável externa estática: acesso a mesma restrito ao mesmo arquivo fonte.

Classe static
Exercício

- Qual a diferença da saída desses programas?

```
void inc()
{
  static int a=0;
  a++;
  printf("%d ",a);
}

void main(void)
{
  inc();
  inc();
  inc();
  getch();
}
```

Saída: 1 2 3

```
void inc()
{
  static int a;
  a=0;
  a++;
  printf("%d ",a);
}

void main(void)
{
  inc();
  inc();
  inc();
  getch();
}
```

Saída: 1 1 1

Classe register

- A classe de armazenamento **register** indica que a variável deve ser guardada fisicamente em uma memória de acesso muito rápida chamada registrador.
- O tamanho do registrador varia de acordo com o hardware, podendo armazenar variáveis int e char.
- São semelhantes as automáticas, porém restritas ao tipo int e char.

Variáveis com mesmo nome

- Variáveis com mesmo nome e endereço diferentes são variáveis diferentes.
- Uma variável local é limitada ao bloco em que foi declarada.
- Se duas variáveis compartilham o mesmo nome, a que foi declarada no bloco atual tem precedência sobre a que foi declarada num bloco diferente.

Variáveis com mesmo nome -Exercício

- Qual é a saída do programa abaixo?

```
int a=0;
void inc()
{
  a++;
  printf("\n%d",a);
}
main(void)
{
   int a = 10;
   printf("\n%d",a);
   inc();
   getch();
}
```

Saída:
10
1

25

O pré-processador C

- É um programa que examina o código fonte C e executa certas modificações nele, baseado em instruções chamadas diretivas.
- O pré-processador faz parte do compilador.
- O pré-processador é executado automaticamente antes da compilação.
- Linhas normais do programa são instruções para o micro-processador.
- Diretivas do pré-processador são instruções para o compilador.
- As diretivas são iniciadas pelo símbolo # e podem ser colocadas em qualquer parte do programa.

26

A diretiva #define

- Usada para definição de constantes:

```
#define PI 3.14159
...
area = PI*10;
```

- Quando o compilador encontra #define, ele substitui cada ocorrência de PI por 3.14159
- Um comando #define está restrito a uma linha
- Não há ponto-e-vírgula no final.

27

A diretiva #define

```
#define ERRO  printf("Erro")
...
if(x<0)
ERRO;
...
```

28

Macros
Exemplo

- A diretiva #define com argumentos é chamada macro:
- Não colocar espaço entre o identificador e o argumento da Macro.

```
#define PI 3.14
#define AREA(X)( 4*PI*X )

void main(void)
{
   printf("\n%.2f", AREA(10) );
}
```

29

Macros- Cuidado ao passar expressões

- Os parâmetros da macro são passados do "jeito" que são escritos;
- Qual é a saída do programa abaixo?

```
#define AREA(X)( 4*PI*X )
#define MULT(X,Y) (X*Y)

main(void)
{
   printf("\n%d",MULT(10+1,3));
   printf("\n%d",MULT((10+1),3));
   getch();
}
```

30

Macros

- + Não necessita especificar o tipo de argumento e nem de retorno.
- Macros são substituições do código, que ocorre toda a vez que é chamada.
- +Programa mais rápido.
- -Código do programa executável maior.

A diretiva #include

- Causa a inclusão de um programa fonte em outro.

pascal.h

```
#define program main()
#define begin {
#define write(x) printf(x)
#define end getch();}
```

pascal.c

```
#include "pascal.h"
program
  begin
    write("Teste");
  end
```


Outras Diretivas

- #undef: força o compilador a abandonar a macro. Exemplo:

```
#define PI 3.14
#undef PI
```

- #ifdef,#ifndef #else, #endif.
  - Exemplo:

```
#ifndef PI
#define PI 3.14
#else
#define OUTRO_PI 3.147
#endif
```


Matrizes

- Faça um programa que leia 5 notas fornecidas pelo usuário e imprima a média delas.

```
Exemplo:
int notas[5];
int i;
  for(i=0;i<5;i++){
    printf("\nDigite a nota%d: ", i);
    scanf("%d",&notas[i]);
}
```


Passando Arrays como argumentos de função

- **protótipo:**

```
void carregaDados(int[]);
```

- **função:**

```
void leDados(int notas[])
{
  for(i=0;i<5;i++)
      notas[i]=i*2;
}
```


Passando Arrays como argumentos de função

Não é feita a cópia de um vetor quando ele é passado para a função. A função acessa os dados do vetor original.

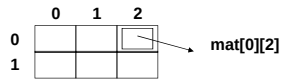
```
int main(void){
  int notas[5];
  void leNotas(int notas[]);
  printf("notas[4]=%d", notas[i]);
}
```

Matrizes de mais de uma dimensão

A cada par de colchetes na declaração aumentamos em um a dimensão da matriz.

Exemplo:

int mat[2][3]



int mat[3][2][5]; // matriz de 3 dimensoes

Inicializando Matrizes de três dimensões

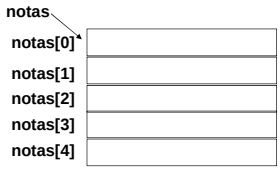
```
int trevet[3][2][4]={
  { {50,10,5,2}, {50,10,5,2}},
  { {50,10,5,2}, {50,10,5,2}},
  { {50,10,5,2}, {50,10,5,2}}
  };
```


O que representa o nome da matriz sem os colchetes?

- É o endereço da matriz!.

Considere a matriz abaixo:

notas é igual &nota[0]



Chamada por Valor X Chamada por referência

- Quando a matriz é passada como argumento de uma função, ela é passada por referência.
- Como as matrizes podem ser bastante grandes é mais eficiente manter uma cópia da matriz na memória, não importa o número de funções que a acessam.
- As alterações feitas na matriz dentro da função alteram a própria matriz.

String

- String é um vetor do tipo char terminada pelo caractere '\0'.
- Sempre que o C encontrar algo entre “” ele reconhece que se trata de uma string constante, isto é, o caractere entre aspas + o '\0'.
- Cada caractere tem um byte. O caractere '\0' **(null)** é o caractere 0 em ASCII.

Exemplo

- Implemente o programa abaixo:

```
Int main(){
  char a[]="teste";
  int i;
  for(i=0;i<=strlen(a);i++)
    printf("\nchar= %c, cod asc=%d", a[i],a[i]);
  getch();
}
```

Qual é o tamanho da string retornado por strlen(a)? resp. 5

Porque a string tem um caractere a mais do que esse tamanho?

Matriz de Strings

- Fazer um programa que le 5 nomes, armazena em uma matriz e posteriormente imprime os nomes lidos.

```
main(){
 char nome[5][81];//5 nomes de no maximo 80 caracteres
 int i;
 for(i=0;i<5;i++){
   printf("\nDigite o nome: ");
   gets(nome[i]);// recebe a string digitada pelo usuário até que a tecla
                 //[enter] ser pressionada.

 }
 printf("\nOs 5 nomes digitados foram:\n");
 for(i=0;i<5;i++)
   puts(nome[i]);//escreve e pula uma linha
 getch();
}
```


Ponteiro

- Ponteiro Constante é um endereço.
  Exemplo: nome de uma matriz representa um ponteiro constante
- Ponteiro Variável é um lugar para guardar endereços.

Um ponteiro variável A aponta para B se ele contém o endereço de B.

Passando endereços para a função

```
main(){
  void muda(int *a);
  int x=10;
  muda(&x);
  printf("\n%d",x);
  muda(&x);
  printf("\n%d",x);
  getch();
}

void muda(int *a){
  *a = *a + 2;
}
```

um ponteiro para uma variável

passa o endereço da variável x para a função muda

altera o conteúdo do endereço passado como parâmetro.

Declarando variáveis ponteiros

int *px ;

- A instrução acima declara que px é um endereço e o conteúdo desses endereço é um valor inteiro.
- C oferece dois operadores para trabalhar com ponteiros:
  - & →retorna o endereço de memória da váriavel.
  - * →retorna o conteúdo (valor) da variável localizada no endereço (ponteiro).

	Memória
End	
4206596	23
	20
	65
	11

```
main(){
  int * a;
  *a = 20;
  printf("\n%u",a);
  printf("\n%u",*a);
  getch();
}//Qual é a saída?
```


O endereço do próprio ponteiro

- O nome do ponteiro retorna o endereço para o qual ele aponta.
- O operador & junto ao nome do ponteiro retorna o endereço do ponteiro.
- O operador * junto ao nome do ponteiro retorna o conteúdo da variável apontada.

Incrementando um ponteiro

- ptr++;
- Incrementar um ponteiro significa movimentá-lo para o próximo que tem o mesmo tipo de dado apontado.

	End	Memória
ptr→	1000	23
ptr++→	1004	20
	1008	65
		11

Adicionar e subtrair valores de ponteiros

- ptr+3

	End	Memória
ptr→	1000	23
	1004	20
	1008	65
ptr+3→	1012	11

Atenção exemplo com ponteiros para inteiros!

- py-px =3

	End	Memória
px→	1000	23
	1004	20
	1008	65
py→	1012	11

Ponteiros e Matrizes

Operações com matrizes podem ser feitas usando ponteiros:

printf("\n%d",nums[d]); equivale a printf("\n%d",*(nums + d));

* (matriz + indice) é o mesmo que matriz[indice]

Existe duas maneiras de referenciar o endereço de um elemento da matriz: matriz + índice ou &matriz[indice]

Inicialização de uma matriz de ponteiros para String

- Faça um programa que peça para o usuário digitar um nome. Se o nome estiver em uma lista predefinida o programa exibe: "Pode entrar", senão o programa exibe "guardas prendam esse sujeito".

```
main(){
 char *ptr[]={
    "maria",
    "marcela",
    "sergio",
    "renata",
    "pedro"};
 char nome[40];
 int i,entra=0;
puts("Digite o seu nome: ");
gets(nome);
```

```
for(i=0;i<TAM;i++){
   if(strcmp(ptr[i],nome)==0){
     puts("Entre honrado!");
     entra=1;
     break;
   }
 }
 if(!entra)
   puts("Guardas, prendam-no intruso!");
 getch();
}
```


Diferenças

Versao char matriz[5][10]

m	a	r	c	e	l	a			
m	a	r	i	a					
p	e	d	r	o					
s	e	r	g	i	o				
a	n	a							

Versao char *matriz[5]

m	a	r	c	e	l	a
m	a	r	i	a		
p	e	d	r	o		
s	e	r	g	i	o	
a	n	a				

Estruturas

- Permite agrupar dados de tipos desiguais.
- Exemplo de estrutura: um registro de um funcionário:
  - número funcionário (inteiro)
  - nome (string)
  - salário (float)
- A estrutura é um tipo de dado cujo formato é definido pelo programador

Exemplo Estrutura

```
int main(){
  struct dado{
    char nome[TAM];
    int idade;
    float salario;
  };
```

Declaração de uma variável do tipo da estrutura

struct dado a;

Acessando membros de uma estrutura

a.salario = 200.0;

Declarando estruturas com typedef

Typedef permite que tipos de dados sejam definidos em C. Inclusive estruturas. Ele simplifica a sintaxe declaração de variáveis do tipo estrutura.

```
typedef struct aluno{
  char nome[TAM];
  float nota;
};

aluno a;
a.nome = 10;
```


Atribuição de Estruturas

- quando uma estrutura é atribuída a outra, todos os elementos de uma estrutura são realmente atribuídos de uma vez para os correspondentes elementos da outra estrutura.
- essa instrução de atribuição tão simples não pode ser usada para uma matriz, que deve ser atribuída elemento por elemento.

b=a;

Estruturas

Passando Estruturas para Funções

```
void printLivro(struct livro l){
   printf("\nTitulo: %s",l.nome);
   printf("\nCodigo: %d",l.cod);
}
```

Matrizes de Estruturas
struct livro lista[TAM];

Ponteiros para Estruturas:
struct livro *ptr;

ptr = &lista[0];

Heap

- Heap memória disponível, que não está em uso.
- A linguagem C permite a alocação e a liberação de memória do heap.
- As funções malloc, calloc e free são usadas para esse propósito.

Alocando memória, a função Malloc

```
typedef struct aluno{
  char nome[TAM];
  float nota;
};
aluno * a = (aluno *) malloc(sizeof(aluno));
```

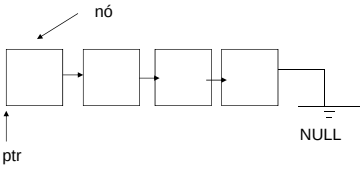
```
struct aluno{
   ...
};
struct aluno * a = (struct aluno *) malloc(sizeof(struct aluno));
```


A função *free*

- A função *free* é um complemento de *malloc()*
- Recebe como argumento um ponteiro para a área de memória previamente alocada por *malloc.* Libera essa área para uma possível utilização futura.
- É extremamente importante liberar memória após o seu uso.
- Exemplo:

```
long * ptr =(long *) calloc(100,sizeof(long));
free(ptr);
```

Lista Encadeada



NULL é uma constante definida no stdio.h como 0

Uma estrutura que possui um ponteiro para outra estrutura.

```
typedef struct node{
int value;
node * next;
“}
```

Exemplo: Fazer um programa que insere os livros fornecidos pelo usuário em uma lista encadeada e lista todos os nomes inseridos.

Possível Resposta

```
#include <stdio.h>
struct no{
  int cod;
  char titulo[80];
  struct no *proximo;
};
void insere();
void lista();
struct no *primeiro, *atual, *novo;
int main(){
  char opc;
```

```
while(1){
    printf("\nDigite a opcao:");
    printf("\n1.Inserir livro.");
    printf("\n2.Listar todos os livros.");
    printf("\n3.Sair\n");
    opc= getche();
    switch(opc){
      case '1':
         insere();
         break;
      case '2':
         lista();
         break;
      case '3':
         return;
    }//end switch
  } //end while
}
```


Continuação Possível Resposta

```
void insere(){
   novo = (struct no*) malloc(sizeof(struct
no));
   printf("\nForneca o nome:");
   scanf("%s",novo->titulo);
   printf("\nForneca o codigo:");
   scanf("%d",&novo->cod);
   novo->titulo;
   novo->proximo = NULL;
   if(!primeiro)
     primeiro = novo;
   if(atual)
     atual->proximo = novo;
   atual = novo;
}
```

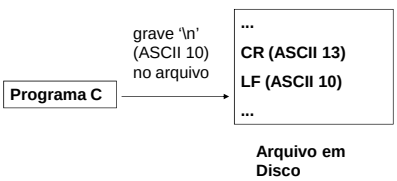
```
void lista(){
  novo = primeiro;
  printf("\n");
  while(novo){
    printf("\ntitulo: %s",novo-> titulo);
    printf("\ncodifo: %d",novo-> cod);
    novo = novo->proximo;
  }
  printf("\n");
}
```


Arquivos

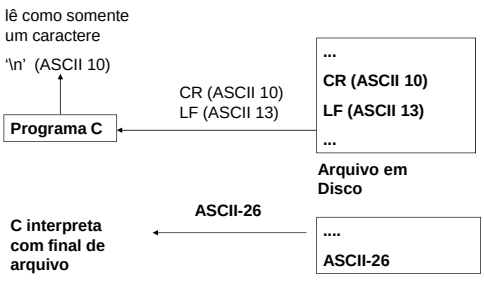
- Coleção de bytes referenciadas por um nome.
- Os arquivos em C podem ser gravados e lidos em modo texto e em binário.

Modo Texto

- Modo Texto: os dados são gravados e lidos como uma cadeia de caracteres.
- Ao gravar o caractere ‘\n’ (nova linha) em um programa em C, ele será gravado como 2 caracteres no disco CR/LF



Modo Texto



Modo Binário

- Os dados da memória são transferidos sem fazer nenhuma conversão.
- Os dados do arquivos são lidos sem nenhuma conversão.
- Os números são guardados de acordo com sua representação em memória. Exemplo: 4 bytes para float.

Trabalhando com arquivos

- Informações específicas do arquivo deve estar presente antes que o programa possa acessar
- Essas informações geralmente ficam guardadas em uma estrutura chamada FILE definida no arquivo <stdio.h>

A estrutura FILE armazena vários parâmetros usados nas operações com arquivos como:

•o identificador do arquivo;

•tamanho do buffer usado na transferência;

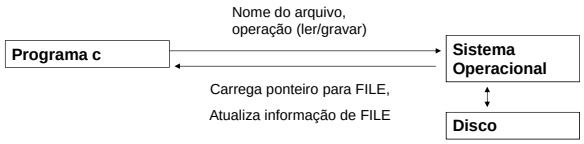
•ponteiro para o buffer de transferência de dados;

•posição corrente de acesso ao arquivo

O Programa C e O Sistema Operacional trabalhando juntos

•O programa C faz requisições para o Sistema Operacional para realizar operações com arquivos.

•É o sistema operacional quem realiza o acesso ao disco.



Abrindo um arquivo no modo padrão (texto)

```
#include<stdio.h>
#include<conio.h>
main(){
  FILE *f;
  char ch;
  f = fopen("arq01.txt","w");
   while((ch=getche())!= '\r') //'/r' é o
enter
    putc(ch,f);
  fclose(f);
}
```

fopen:

• chama SO e inicializa a estrutura FILE;

• retorna um ponteio para a estrutura FILE;

• retorna NULL se o arquivo não pode ser aberto.

putc(ch,f):

• grava o char *ch* no arquivo correspondente a *f*

fclose(f):

•grava tudo o que está no buffer no arquivo

O uso de buffer

- O quão ineficiente é gravar um caractere por vez no disco. Para cada caractere acessar o disco, posicionar a cabeça de gravação na trilha correta e encontrar o bloco correto.
- Quando a função *putc* é executada, o caractere é temporariamente armazenado no buffer. Quando o buffer está completamente preenchido, o seu conteúdo é escrito no disco todo de uma vez.
- O programa força o buffer a ser gravado no disco quando o arquivo é fechado.

A função *fopen*

- FILE * f = fopen(“arq.txt”,”w”);
- o segundo parâmetro específica o tipo de abertura do arquivo:

"w"
"r"
"rb"
"wb"

  - r : leitura
  - w: gravação
  - *b:* para informar que o modo de leitura é binário

Lendo um arquivo

```
#include<stdio.h>
#include<conio.h> //getch()
#include<stdlib.h> //exit(0)

void erro(){
  printf("\nErro");
  getch();
  exit(0);
}
main(){
  FILE *f;
  char ch;
  if( (f=fopen("arq01.txt","r"))== NULL)
    erro();
  while((ch=getc(f))!= EOF)
       printf("%c",ch);
  fclose(f);
  getch();
}
```

exit – fecha todos os arquivos em aberto, termina o programa e devolve o controle para o SO.

getc(f)- le o próximo caractere do arquivo indicado por f

Gravando e Lendo Strings

Escrevendo strings no arquivo

```
#include<stdio.h>
main(){
  FILE *f;
  char * string;
  f = fopen("arq01.txt","w");
  while(strlen(gets(string))>0){
    fputs(string,f);
    fputs("\n",f);
    }
  fclose(f);
}
```

Escreve a string no arquivo. Não adiciona automaticamente o caractere nova linha.

Lendo strings do arquivo (reproduz o comando type do DOS

```
#include<stdio.h>
main(int c,char * argv[]){
   FILE *f;
   char string[200];
   if( (f=fopen(argv[1],"r"))== NULL)
      erro();
  while(fgets(string,200,f)!=NULL)
     printf("%s",string);
  fclose(f);
  getch();
}
```

Lê uma linha por vez do arquivo

Inclui na string o caractere nova linha

Gravando um arquivo na forma formatada (modo texto)

Os programas anteriores mostraram como gravar e ler caracteres.

Mas como manusear os dados numéricos?

```
#include <stdio.h>
#include <conio.h>
main(){
 FILE *f;
 char nome[100];
 int idade;
 int renda;
 char resp='s';
 f = fopen("out.txt","w");
 while(resp=='s'){
     printf("\nDigite o nome,idade e renda:");
     scanf("%s %d %d",nome,&idade,&renda);
     fprintf(f,"%s %d %d ",nome,idade,renda);
     printf("Continuar (s/n)? ");
     resp = getche();
 }
 fclose(f);
}
```

```
#include<stdio.h>
#include<conio.h>
main(){
FILE * f;
char nome[30];
int idade;
int renda;
f = fopen("out.txt","r");
while(fscanf(f,"%s %d %d", nome, &idade, &renda)!= EOF)
  printf("%s %3d %3d \n",nome,idade,renda);
fclose(f);
getch();
}
```


A maneira mais eficiente de armazenar registros é no modo binário.

Escrevendo Registros
fwrite

```
fwrite(&pessoa,sizeof(pessoa),1,f)
```

- ponteiro para a localização de memória do dado a ser gravado
- tamanho do dado a ser gravado
- quantos itens de mesmo tipo serão gravados
- ponteiro para FILE do arquivo que queremos gravar

```
#include<stdio.h>
struct Pessoa{
  char nome[30];
  int idade;
  float renda;
 }pessoa;

 main(){
   FILE * f;
   if( (f= fopen("out.bin", "wb") )==NULL)
       erro( );
   do{
      printf("\nNome: "); scanf("%s",pessoa.nome);
      printf("\nIdade: "); scanf("%i",&pessoa.idade);
      printf("\nRenda: "); scanf("%f",&pessoa.renda);
      fwrite(&pessoa,sizeof(pessoa),1,f);
      printf("\n Adiciona outra pessoa (s/n)? ");
   } while(getche()=='s');
   fclose(f);
 }
```


Lendo Registros com *fread*

```
fread(&pessoa,sizeof(pessoa),1,f)
```

- ponteiro para a localização de memória do dado a ser lido
- tamanho do tipo de dados a ser lido
- quantos itens de mesmo tipo serão lidos
- ponteiro para FILE do arquivo a ser lido

```
#include <stdio.h>
 main(){
   struct Pessoa{
      char nome[30];
      int idade;
      float renda;
   }pessoa;
   FILE * f;
   if( (f= fopen("out.bin", "rb") )==NULL){
      printf("Nao e possivel abrir o arquivo");
      exit(1);
   }
   while(fread(&pessoa,sizeof(pessoa),1,f)==1)
      printf("\nnome=%s; idade=%d; renda=%.2f",
  pessoa.nome, pessoa.idade, pessoa.renda);
   fclose(f);
   getch();
 }
```

Retorna o número de itens lidos. Normalmente este número é igual ao terceiro argumento. Se for encontrado o fim de arquivo, o número será menor.

fseek

Posiciona o ponteiro de leitura para a posição desejada.

```
fseek(f,offset,0)
```

- retorna 0 conseguiu posicionar
- ponteiro para FILE
- modo (tipo long int) número de bytes de deslocamento a partir da posição especificada pelo terceiro argumento

opções de modo

modo	deslocamento a partir de
0	começo do arquivo
1	posição corrente do arquivo
2	fim do arquivo

```
#include <stdio.h>
main(){
 struct Pessoa{
    char nome[30];
    int idade;
    float renda;
 }pessoa;
 FILE * f;
 int numreg;
 long offset; /* deve ser long*/
 if( (f= fopen("out.bin", "rb") )==NULL)
    erro():
 printf("\nDigite o numero do registro: "); scanf("%d",&numreg);
 offset = numreg*sizeof(pessoa);
 if(fseek(f,offset,0)!=0)
    erro();
  if(fread(&pessoa,sizeof(pessoa),1,f)<1)
    erro();
 printf("nome=%s; idade=%d
 renda=%.2f",pessoa.nome,pessoa.idade,pessoa.renda);
 fclose(f);
 getch();
}
```

posicionar eu consegui, mas tenho de verificar se foi possível ler!

ftell()

- retorna a posição de um ponteiro de um arquivo binário em relação ao seu começo.
- recebe como argumento um ponteiro para FILE;
- retorna um valor long que corresponde ao número de bytes do começo do arquivo até a sua posição inicial.

lendo o próximo registro

```
//colocar esse trecho no codigo anterior antes de //fclose();
printf("\nDeseja ler o proximo registro?");
 if(getche()=='s'){
   if(fseek(f,ftell(f),0)!=0)
     erro();
   if(fread(&pessoa,sizeof(pessoa),1,f)<1)
     erro();
   printf("\nnome=%s; idade=%d;
   renda=%.2f",pessoa.nome,
   pessoa.idade,pessoa.renda);
    }
 }
```

Existe uma maneira mais simples de acessar o próximo registro?

- Referência: Mizrahi, V. V., Treinamento em Linguagem C++, Módulo 1