C
Arquivos

Adriano Cruz

Instituto de Matemática
Departamento de Ciência da Computação
UFRJ

15 de fevereiro de 2016

Section Summary

Bibliografia

1. Adriano Cruz. *Curso de Linguagem* `C`, Disponível em `http://equipe.nce.ufrj.br/adriano`
2. Ulysses de Oliveira. *Programando em* `C`, Editora Ciência Moderna.

Section Summary

Fluxos de Dados

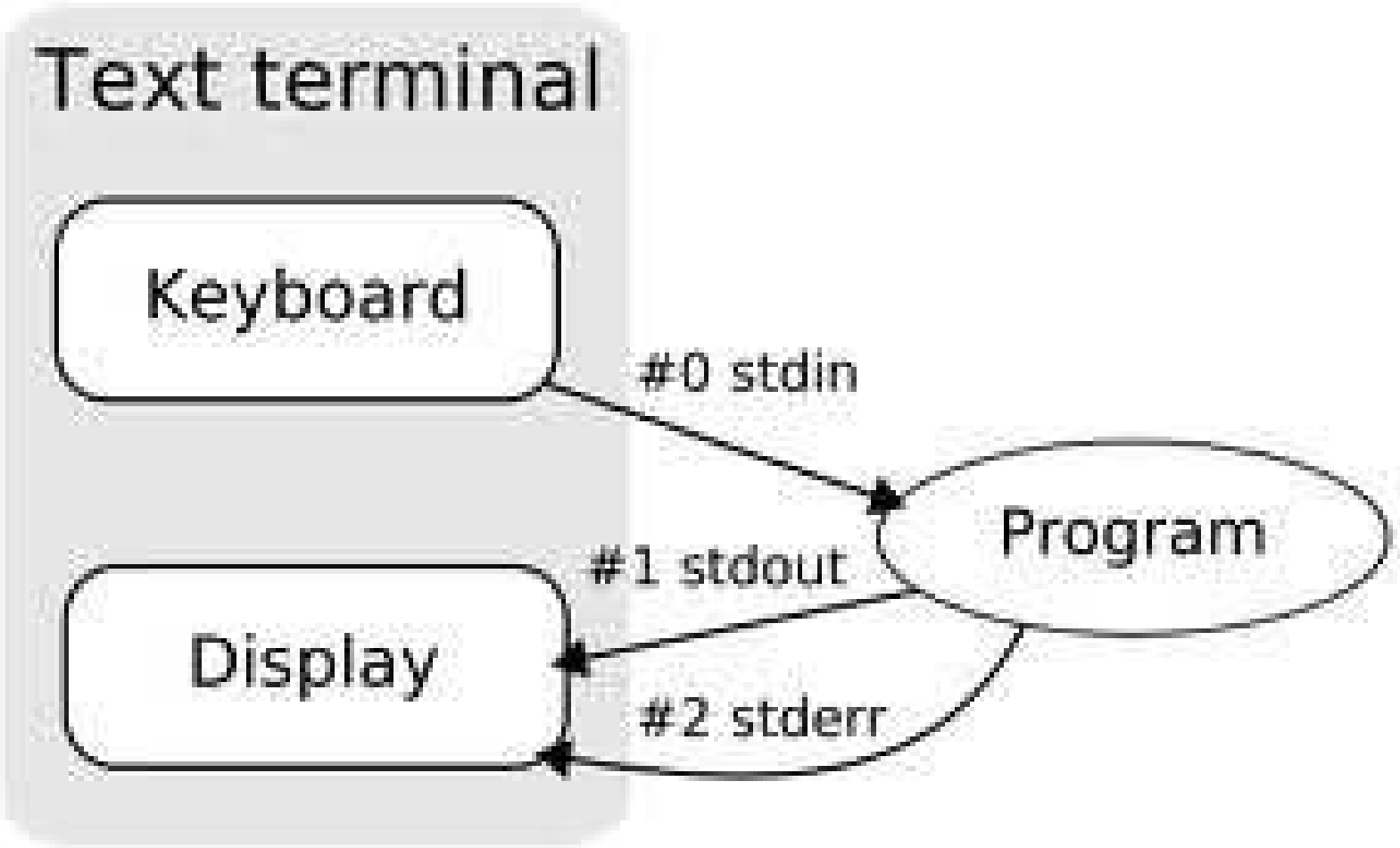
- Para isolar os programadores dos problemas de manipular os vários tipos de dispositivos de armazenamento e seus diferentes formatos a linguagem `C` utiliza o conceito de fluxo de dados (*stream*).
- Todos os sistemas de arquivos se comportam da mesma maneira.
- Dados podem ser manipulados em dois diferentes tipos de fluxos: fluxos de texto e fluxos binários.

Periféricos

Fluxos de Texto

- Um fluxo de texto é composto por uma seqüência de caracteres, que pode dividida em linhas terminadas por um caractere de final de linha.
- Fluxos de dados padrão:
  - `stdin`, para entrada de dados, normalmente associado ao teclado;
  - `stdout` para saída de dados, normalmente associado ao vídeo.
- Ao iniciar todo programa em `C` é automaticamente associado a estes dois fluxos de dados.
- A definição de que periféricos estarão associados a estes fluxos depende do sistema operacional.

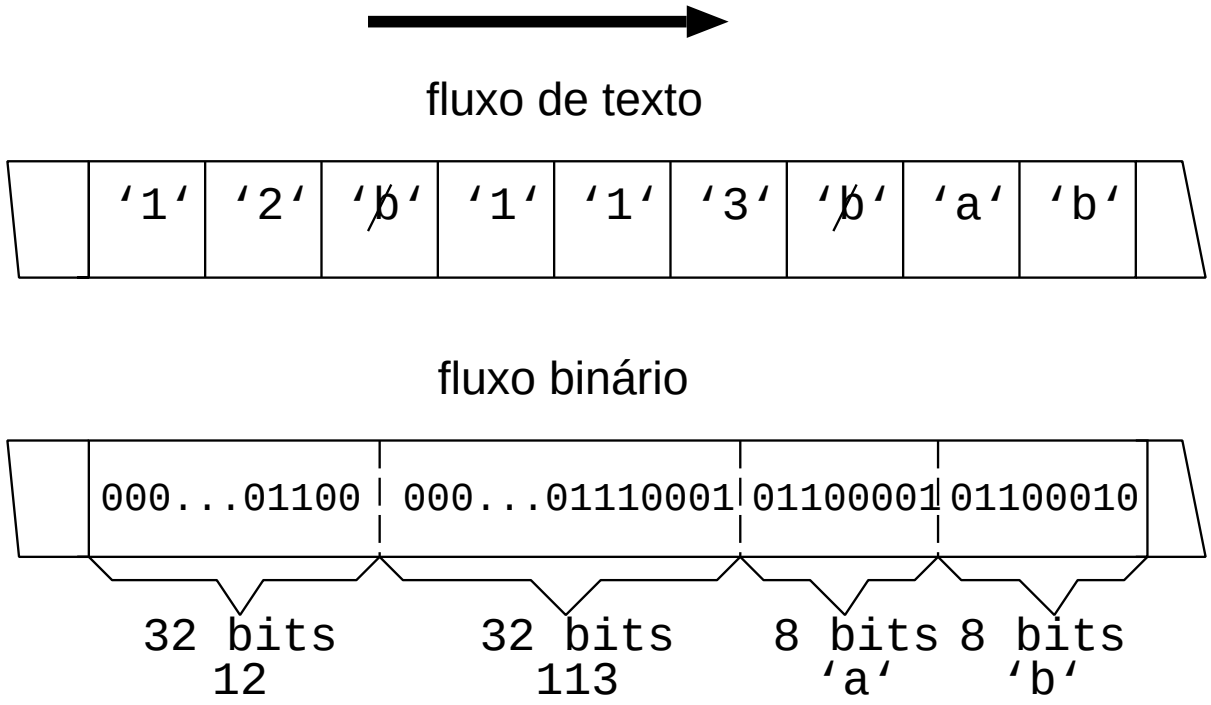
Fluxos


Text terminal
Keyboard
#0 stdin
Program
#1 stdout
Display
#2 stderr

Fluxo Binário

- Um fluxo binário é composto por uma seqüência de bytes lidos, sem tradução, diretamente do dispositivo externo.
- Existe uma correspondência um para um entre os dados do dispositivo e os que estão no fluxo que o programa manipula.
- No fluxo de texto os dados são armazenados como caracteres sem conversão para representação binária.
- No fluxo binário cada número inteiro ocupa 32 bits e é armazenado na forma binária.

Fluxos

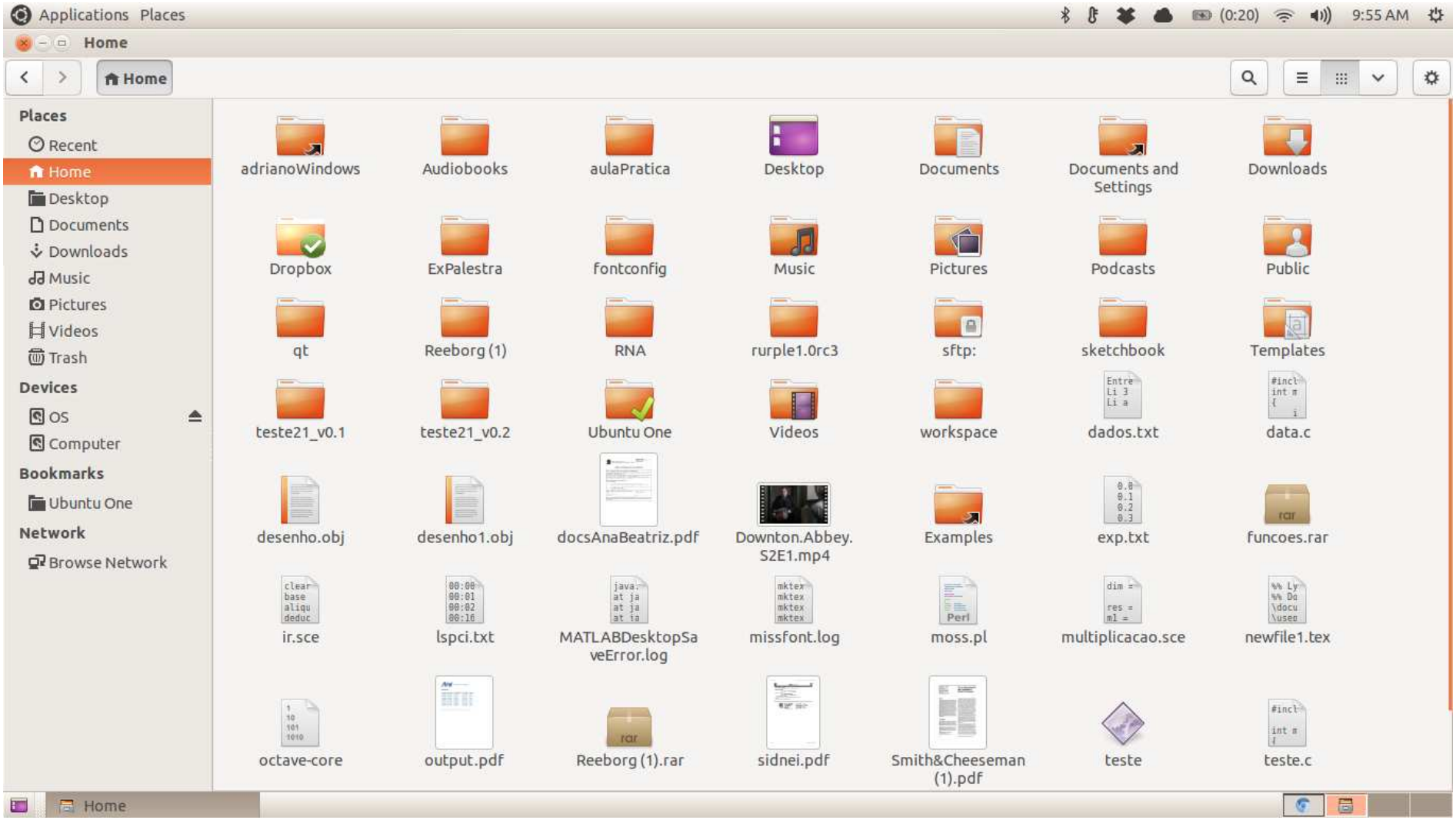

fluxo de texto
'1' '2' '\b' '1' '1' '3' '\b' 'a' 'b'
fluxo binário
000...01100 000...01110001 01100001 01100010
32 bits
12
32 bits
113
8 bits
'a'
8 bits
'b'

Section Summary

Arquivos

- Um arquivo pode estar associado à qualquer dispositivo de entrada e saída.
- Por exemplo: impressora, teclado, disquete, disco rígido etc.
- No entanto, os programas vêem os arquivos através de fluxos.
- Para que um determinado arquivo em um periférico seja associado a um fluxo é necessário que o arquivo seja aberto.
- Normalmente a interação entre o programa e os arquivos é feita por meio de buffers que intermediam a transferência dos dados entre os programas e os periféricos.

Arquivos


Applications Places
(0:20)
9:55 AM
Home
Home
Places
Recent
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Devices
OS
Computer
Bookmarks
Ubuntu One
Network
Browse Network
adrianoWindows
Audiobooks
aulaPratica
Desktop
Documents
Documents and Settings
Downloads
Dropbox
ExPalestra
fontconfig
Music
Pictures
Podcasts
Public
qt
Reeborg (1)
RNA
rurple1.0rc3
sftp:
sketchbook
Templates
teste21_v0.1
teste21_v0.2
Ubuntu One
Videos
workspace
dados.txt
data.c
desenho.obj
desenho1.obj
docsAnaBeatriz.pdf
Downton.Abbey.S2E1.mp4
Examples
exp.txt
funcoes.rar
ir.sce
lspci.txt
MATLABDesktopSaveError.log
missfont.log
moss.pl
multiplicacao.sce
newfile1.tex
octave-core
output.pdf
Reeborg (1).rar
sidnei.pdf
Smith&Cheeseman (1).pdf
teste
teste.c
Home

Operações em Arquivos

- abertura de arquivos;
- fechamento de arquivos;
- remover um arquivo;
- leitura e escrita de um caractere ou byte;
- procurar saber se o fim do arquivo foi atingido;
- posicionar o arquivo em um ponto determinado.

Arquivos

Operações em Arquivos

- Algumas dessas funções não se aplicam a todos os tipos de dispositivos.
- Em uma impressora pode não ser possível ir para o início do arquivo.
- Um arquivo em disco permite acesso aleatório enquanto um teclado não.
- Ao final das operações nos arquivos o programa deve fechá-los.
- Ao final do programa todos os arquivos associados são fechados automaticamente e os conteúdos dos *buffers* são descarregados para o dispositivo externo.

Section Summary

Operações em Arquivos

stdio.h

As funções de Entrada e Saída normalmente utilizadas pelos programadores estão armazenadas na biblioteca `stdio.h`.

O que fazer?

Usar #include<stdio.h> no início do programa.

Funções

Função	Descrição
`fopen()`	Abre um arquivo
`fclose()`	Fecha um arquivo
`fputc()`	Escreve um caractere em um arquivo
`getc()`, `fgetc()`	Lê um caractere de um arquivo
`fprintf()`	Equivalente a `printf()`
`sscanf()`	Equivalente a `scanf()`. Lê de uma cadeia de caracteres
`fscanf()`	Equivalente a `scanf()`
`fseek()`	Posiciona o arquivo em um ponto específico
`rewind()`	Posiciona o arquivo no início
`feof()`	Retorna verdade se chegou ao fim do arquivo
`ferror()`	Verifica a ocorrência de um erro
`fflush()`	Descarrega o *buffer* associado ao arquivo
`fread()`	Leitura de dados no modo binário
`fwrite()`	Escrita de dados no modo binário

Começando do começo

- Para ter acesso aos dados em um arquivo é necessário a definição de um ponteiro do tipo especial `FILE`.
- Este tipo também está definido na biblioteca `stdio.h`.
- Um ponteiro deste tipo permite que o programa tenha acesso a uma estrutura que armazena informações importantes sobre o arquivo.
- Para definir uma variável deste tipo o programa deve conter a seguinte declaração

```
FILE *arq;
```

onde `arq` é o nome do ponteiro que será usado para executar as operações no arquivo.

Section Summary

Abrindo arquivos

- Antes de qualquer operação o arquivo deve ser "aberto".
- Esta operação associa um fluxo de dados a um arquivo.
- Um arquivo pode ser aberto de diversas maneiras:
  - leitura,
  - escrita,
  - leitura/escrita,
  - adição de texto.
- A função para abrir o arquivo é `fopen()` e tem o seguinte protótipo:

```
FILE *fopen (const char *parq, const char *modo)
```

- A função retorna um ponteiro nulo (`NULL`) se o arquivo não puder ser aberto.
- O teste de sucesso na abertura do arquivo deve ser sempre executado.

Modos de abertura de arquivo

"r": Abre um arquivo para leitura, o arquivo deve existir ou um erro ocorre.

"w": Cria um arquivo vazio para escrita, caso um arquivo com o mesmo nome exista o seu conteúdo é apagado.

"a": Adiciona ao final de um arquivo. O arquivo é criado caso ele não exista.

"r+": Abre um arquivo para leitura e escrita. O arquivo deve existir ou um erro ocorre.

"w+": Cria um arquivo vazio para leitura e escrita. Se um arquivo com o mesmo nome existe o conteúdo é apagado.

"a+": Abre um arquivo para leitura e adição. Todas as operações de escrita são feitas no final do arquivo. É possível reposicionar o ponteiro do arquivo para qualquer lugar em leituras, mas as escritas moverão o ponteiro para o final do arquivo. O arquivo é criado caso não exista.

Não se esqueça

Atenção

Observar que se um arquivo for aberto com permissão de escrita todo o seu conteúdo anterior será apagado.

Exemplo

```
FILE *pa; /* declaracao do ponteiro para arquivo */

/* nome externo associado ao interno */
pa = fopen ("arquivo.txt", "w");
if (pa == NULL) { /* verifica erro na abertura */
    printf("Arquivo nao pode ser aberto.");
    return 1;
}
```

Fechando

- Um arquivo deve ser fechado com a função `fclose()` cujo protótipo é

```
int fclose (FILE *parq);
```

- Todos os *buffers* internos associados com o fluxo de dados do arquivo são descarregados.
- O conteúdo de qualquer *buffer* não escrito é escrito e dados não lidos de *buffers* são perdidos.
- Em muitos sistemas operacionais uma operação de escrita em um arquivo não ocorre imediatamente à emissão da ordem de escrita.
- O sistema operacional pode executar a ordem no momento que achar mais conveniente.
- Um valor zero de retorno significa que a operação foi executada com êxito, qualquer outro valor implica em erro.

Fim de arquivo

- A função `feof()` indica que um arquivo chegou ao seu final.
- O protótipo da função:

```
int feof(FILE *parq)
```

- *Se já existe o valor `EOF` para indicar o final de arquivo, por que precisamos de uma função extra do tipo `feof()`?*
- `EOF` é um valor inteiro, e em arquivos binários este valor pode ser parte do arquivo e não o final do arquivo.
- A função `feof()` serve para indicar que o final de um arquivo binário foi encontrado.
- Naturalmente esta função pode ser aplicada também a arquivos texto. Um valor diferente de zero é retornado no caso de ter sido atingido o final do arquivo. O valor zero indica que ainda não se chegou ao final do arquivo.

Exemplo EOF

```
#include<stdio.h>
int main (void) {
    char c;

    c = getchar();
    while (c != EOF) {
        putchar(c);
        c = getchar();
    }
    return 0;
}
```

Exemplo feof()

```
#include<stdio.h>
int main (void) {
    char c;

    c = getchar();
    while (!feof(stdin)) {
        putchar(c);
        c = getchar();
    }
    return 0;
}
```

Voltando para o Início

- `rewind()` recoloca o indicador de posição de arquivo no início do arquivo.
- O protótipo da função é o seguinte:

```
void rewind(FILE *parq)
```

- O arquivo deve estar aberto em um modo que permita a execução das operações desejadas.
- Por exemplo, um arquivo aberto somente para "escrita" e em seguida reposicionado para o início, não irá permitir outra operação que não seja "escrita".

Section Summary

Lendo Caracteres

- As operações mais simples em arquivos são a leitura e escrita de caracteres.
- Para ler um caractere de um arquivo, que foi previamente aberto, pode-se usar as funções `getc()` e `fgetc()`, que são equivalentes.
- Os protótipos destas funções são os seguintes:

```
int fgetc (FILE *parq); int getc (FILE *parq);
```

- As funções `getc()` e `fgetc()` são equivalentes.
- A função lê o caractere como um unsigned char mas retorna o valor como um inteiro, onde o byte mais significativo vale zero.
- O apontador do arquivo avança um caractere e passa a apontar para o próximo caractere a ser lido.

Lendo Caracteres

- A função devolve o código `EOF` ao chegar ao final do arquivo ou caso um erro ocorra.
- O valor `EOF` também é um inteiro válido e portanto ao usar arquivos binários é necessário que a função `feof()` seja utilizada para verificar o final do arquivo.
- A função `ferror()` pode ser usada para determinar se um erro ocorreu.

Escrevendo caracteres

- Para escrever caracteres há duas funções definidas `putc()` e `fputc()`.
- Os protótipos das funções são os seguintes:

```
int putc(int ch, FILE *parq); int fputc(int ch, FILE *parq)
```

onde `parq` é um ponteiro de arquivo para o arquivo que foi previamente aberto por meio da função `fopen()` e `ch` é o caractere a ser escrito.

Exemplo I

```
#include<stdio.h>
#include<stdlib.h>
int main (void )  {
    int c;
    FILE *pa;
    char *nome = "texto.txt";
    if (( pa = fopen(nome, "w+")) == NULL) {
        printf("Nao foi possivel abrir o arquivo.\n");
        exit(1);
    }
    c = getchar();
    while (!feof(stdin)) {
        fputc(c, pa);
        c = getchar();
    }
    rewind(pa);  /* volta ao inicio do arquivo */
    printf("\nTerminei de escrever, agora vou ler.\n");
```

Exemplo II

```
    c = fgetc(pa);
    while (!feof(pa)) {
        putchar(c);
        c = fgetc(pa);
    }
    fclose(pa);
    return 0;
}
```

Outro exemplo I

```
#include<stdio.h>
int  main (void)  {
    int c;
    FILE *pa;
    char *nome = "texto.txt";
    if (( pa = fopen(nome, "w")) == NULL) {
        printf("\n\nErro ao abrir o arquivo.\n");
        return 1;
    }
    c = getchar();
    while (!feof(stdin)) {
        fputc(c, pa);
        c = getchar();
    }
    fclose(pa);
    printf("Terminei de escrever, agora vou ler.\n");
    if (( pa = fopen(nome, "r")) == NULL) {
```

Outro exemplo II

```
        printf("Erro ao abrir o arquivo.\n");
        return 1;
    }
    c = fgetc(pa);
    while (!feof(pa)) {
        putchar(c);
        c = fgetc(pa);
    }
    fclose(pa);
    return 0;
}
```

Section Summary

Funções I

- As funções `fgets()` e `fputs()` servem para ler e escrever cadeias de caracteres em arquivos.
- Os protótipos das funções são:
- `int fputs(char *str, FILE *parq);`
- `int fgets(char *str, int comp, FILE *parq);`

Exemplo I

```
#include<stdio.h>
#define MAX 80
int main (void)  {
    char linha[MAX];
    FILE *pa;
    char *nome = "texto.txt";

    if (( pa = fopen(nome, "w+")) == NULL) {
        printf("Nao foi possivel abrir o arquivo.\n");
        return 1;
    }
    fgets(linha, MAX, stdin);
    while (!feof(stdin)) {
        fputs(linha, pa);
        fgets(linha, MAX, stdin);
    }
    rewind(pa);  /* volta ao inicio do arquivo */
```

Exemplo II

```
    printf("Terminei de escrever, agora vou ler.\n");
    fgets(linha, MAX, pa);
    while (!feof(pa)) {
        puts(linha);
        fgets(linha, MAX, pa);
    }
    fclose(pa);
    return 0;
}
```

Section Summary

Exemplo I

- As funções `fprintf()` e `fscanf()` são equivalentes as funções `printf()` e `scanf()`.
- `int fprintf(FILE *parq, const char *formatacao, ...);`
- `int fscanf(FILE *parq, const char *formatacao, ...);`

Exemplo I

```c
#include<stdio.h>
int main (void ) {
    char palavra[20];
    int i; float f;
    FILE *pa;
    char *nome = "format.txt";
    if (( pa = fopen(nome, "w+")) == NULL) {
        printf("Nao foi possivel abrir o arquivo.\n");
        return 1;
    }
    puts ("Palavra?"); scanf ("%s", palavra);
    puts ("Inteiro."); scanf("%d", &i);
    puts ("Flutuante."); scanf("%f", &f);
    /* Escreve os dados no arquivo */
    fprintf(pa, "%s %d %f", palavra, i, f);
    rewind(pa);  /* volta ao inicio do arquivo */
    printf("\nTerminei de escrever, agora vou ler.\n");
```

Exemplo II

```
    fscanf(pa, "%s %d %f", palavra, &i, &f);
    printf("Palavra lida: %s\n", palavra);
    printf("Inteiro lido: %d\n", i);
    printf("Float  lido: %f\n", f);
    fclose(pa);
    return 0;
}
```

Section Summary

Lendo e escrevendo binário

- As funções `fread` e `fwrite` são empregadas para leitura e escrita de dados em modo binário.
- Os protótipos das funções são:
- `size_t fread (void *ptr, size_t size, size_t nmemb, FILE *parq);`
- `size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *parq);`

fread

- `size_t fread (void *ptr, size_t size, size_t nmemb, FILE *parq);`
- `fread` lê `nmemb` objetos, cada um com `size` bytes de comprimento, do fluxo apontado por `stream` e os coloca na localização apontada por `ptr`.
- Ela retorna o número de itens que foram lidos com sucesso.
- Caso ocorra um erro, ou o fim do arquivo foi atingido o valor de retorno é menor do que `nmemb` ou zero.

- `size_t fread (void *ptr, size_t size, size_t nmemb, FILE *parq);`
- `fwrite` escreve `nmemb` elementos de dados, cada um com `size` bytes de comprimento, para o fluxo apontado por `stream` obtendo-os da localização apontada por `ptr`.
- Ela retorna o número de itens que foram lidos com sucesso.
- Caso ocorra um erro, ou o fim do arquivo foi atingido o valor de retorno é menor do que `nmemb` ou zero.

Exemplo I

```
#include<stdio.h>
int main (void ) {
    int inum=10; float fnum=2.5;
    double pi=3.141516;     char c='Z';
    FILE *pa; char *nome = "texto.bin";

    if (( pa = fopen(nome, "w+")) == NULL) {
        perror("fopen: ");
        return 1;
    }
    fwrite(&inum, sizeof(int), 1, pa);
    fwrite(&fnum, sizeof(float), 1, pa);
    fwrite(&pi, sizeof(double), 1, pa);
    fwrite(&c, sizeof(char), 1, pa);
    rewind(pa);
    fread(&inum, sizeof(int), 1, pa);
    fread(&fnum, sizeof(float), 1, pa);
```

Exemplo II

```
    fread(&pi, sizeof(double), 1, pa);
    fread(&c, sizeof(char), 1, pa);
    printf("%d, %f, %f, %c\n", inum, fnum, pi, c);
    fclose(pa);
    return 0;
}
```

- int fseek(FILE *stream, long offset, int whence);
- fseek() posiciona o indicador de posição no arquivo.
- A nova posição, MEDIDA EM BYTES, é obtida adicionando-se offset bytes to the position specified by whence.
- Se whence é igual a SEEK_SET, SEEK_CUR, or SEEK_END, o offset é relativo ao início do arquivo, a posição atual ou ao final do arquivo respectivamente.
- SEEK_SET relativo à posição inicial.
- SEEK_CUR relativo à posição atual.
- SEEK_END relativo à posição final.

Exemplo I

```c
#include <stdio.h>
#define TAM 10
int main(int argc, char **argv) {
    FILE *binario;
    char *nome = "binario.bin";
    int v[TAM];
    int i;
    int novo;

    for (i = 0; i < TAM; i++) {v[i] = i;}
    binario = fopen(nome, "r+");
    if (!binario) puts("erro");
    fwrite(v, sizeof(int), TAM, binario);
    for (i = 0; i < TAM; i++) {printf("%d\n", v[i]);}
    novo = 999;
    rewind(binario);
    /* Escreve no terceiro elemento do arquivo */
```

Exemplo II

```
    fseek(binario, 2*sizeof(int), SEEK_SET);
    fwrite(&novo, sizeof(int), 1, binario);
    fseek(binario, 0, SEEK_SET); // = rewind(binario)
    fread(v, sizeof(int), TAM, binario);
    for (i = 0; i < TAM; i++) {printf("%d\n", v[i]);}
    fclose(binario);
    return 0;
}
```

The End