UNIVERSIDADE FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM DESENVOLVIMENTO E ANÁLISE DE SISTEMAS
LINGUAGEM DE PROGRAMAÇÃO ESTRUTURADA
PROF. ME. ANDREIA DE JESUS

ARQUIVOS NA LINGUAGEM C

0. Streams: é um canal através do qual os dados fluem de/para o disco, teclado, impressora. É o gerenciamento de E/S(I/O) da linguagem C.

1. Arquivos de Dados: nos permitem armazenar informações permanentemente e acessar e alterar essas informações sempre que necessário. Um arquivo pode ser acessado de forma sequencial ou aleatória.

2. Tipos de Arquivos:
   a) Texto: Armazenam caracteres. Os dados são gravados exatamente como seriam impressos na tela, ou seja, os dados são gravados como caracteres de 8 bits utilizando a tabela ASCII. Para isso, existe uma conversão de dados.

Exemplo: Considere um número inteiro com 8 dígitos
int n=76391603; (neste exemplo considerar int com 4 bytes. Logo ocupa 32 bits – 4 bytes na memória)

Em um arquivo texto cada dígito será convertido para seu caractere ASCII, ou seja, 8 bits por dígito.

“76391603” = 64 bits – 32 bytes

Observação: um int sempre será 4 bytes independente da quantidade de dígitos que o número tiver. Já no arquivo o tamanho varia de acordo com a quantidade de dígitos.
Por exemplo:
int n=16; // ocupará 32 bits - 4 bytes em memória
// já em um arquivo texto ocupará 16 bits – 2 bytes

b) Binário: os dados são gravados exatamente como estão organizados na memória do computador. Não existe etapa de conversão de dados.

Exemplo: Considere um número inteiro com 8 dígitos
int n=76391603; (neste exemplo considerar int com 4 bytes. Logo ocupa 32 bits – 4 bytes na memória)
Em um arquivo binário o conteúdo da memória será copiado diretamente para o arquivo sem conversão.

“76391603” // 32 bits – 4bytes (codificado)

Arquivo Texto	Arquivo Binário
Tem conversão de dados	Sem conversão de dados
Arquivos maiores	Arquivos em geral menores
Leitura e escrita mais lentas	Leitura e escrita mais rápida
São de mais fácil manipulação	Mais difíceis de serem manipulados
Podem ser lidos por diferentes programas (editores de texto).	Somente pode ser lido pelo programa que o gravou.
Tanto arquivo de texto quanto arquivos binários são gravados e acessados sequencialmente.	

3. Funções mais comuns de arquivos com buffer: (biblioteca stdio.h)

Nome	Função
fopen( )	Abre um arquivo.
fclose( )	Fecha um arquivo.
putc( )/ fputc( )	Escreve um caractere em um arquivo.
getc( )/ fgetc( )	Lê um caractere em um arquivo.
fprintf( )	É para um arquivo o que o printf é para o console.
fscanf( )	É para um arquivo o que o scanf é para o console.
feof( )	Devolve verdadeiro se é final de arquivo.
ferror( )	Devolve verdadeiro se ocorre um erro.
rewind( )	Repõe o indicador de posição no início do arquivo.
remove( )	Apaga um arquivo.
fseek( )	Posiciona um arquivo em um byte específico.

4. O ponteiro de arquivo: é um ponteiro para informações que definem várias coisas sobre o arquivo, incluindo seu nome, status e a posição atual do arquivo. Este ponteiro identifica um arquivo específico em disco.
OBS: Um ponteiro de arquivo é uma variável ponteiro do tipo FILE.
FILE é definido em stdio.h.

Exemplo: FILE *fp;

5. Especificação para tipo de arquivo:

Modo	Significado
r	Abre um arquivo texto para leitura.
w	Cria um arquivo texto para escrita.
a	Anexa a um arquivo texto.
rb	Abre um arquivo binário para leitura.
wb	Cria um arquivo binário para escrita.
ab	Anexa a um arquivo binário.
r+	Abre um arquivo texto para leitura/escrita
w+	Cria um arquivo texto para leitura/escrita
a+	Anexa ou cria um arquivo texto para leitura/escrita
r + b	Abre um arquivo binário para leitura/escrita
w + b	Cria um arquivo binário para leitura/escrita
a + b	Anexa ou cria um arquivo binário para leitura/escrita

6. Operações básicas sobre Arquivo:

Abrindo e fechando um arquivo

```
void main( )
{
   FILE * ftp;
   ……..
   if((ftp = fopen("arquivo.txt", "w+")) == NULL)
   {
       printf("O arquivo não pode ser aberto!");
       exit(0);
   }
   ……
   fclose(ftp);
}
```

Criando e escrevendo em um arquivo de dados – um caractere por vez

```
void main()
{
   FILE * ftp;
   char ch;

   if((ftp = fopen("prova.txt", "w"))== NULL)
   {
       printf("O arquivo não pode ser aberto!");
       exit(0);
   }
```

```
 do{
        ch = getchar();
        putc(ch, ftp);
   }while(ch != '$');
}
```

Obs: outra função que escreve um caractere em uma arquivo é a **fputc**.

Lendo um arquivo de Dados – um caractere por vez

```
void main()
{
   FILE * ftp;
   char ch;

   if((ftp = fopen("prova.txt", "r"))== NULL)
   {
       printf("O arquivo não pode ser aberto!");
       exit(0);
   }
```

```
do{
      ch= getc(ftp);
      putchar(ch);
   }while(ch != '$');
   fclose(ftp);
}
```

Obs: outra função que lê um caractere em uma arquivo é a **fgetc**.

Criando e escrevendo em um arquivo de dados – com a função fprintf

```
#include <stdio.h>

void main(void)
{
        float nota, nota1, nota2, nota3, nota4, media=0.0;
        char str[10];
        FILE *arq;

        if((arq = fopen("arq_strings.dat", "w+"))== NULL)
         {
                printf("O arquivo não pode ser aberto!");
                exit(0);
         }
         printf("Nota de Programação Linguagem Estruturada: ");
        scanf("%f", &nota);
        fprintf(arq, "%.2f\n", nota);
        media += nota;

        printf("Nota de Estrutura de Dados: ");
        scanf("%f", &nota);
        fprintf(arq, "%.2f\n", nota);
        media += nota;

        printf("Nota de Algoritmos: ");
        scanf("%f", &nota);
        fprintf(arq, "%.2f\n", nota);
        media += nota;

        media /= 3;
        fprintf(arq, "%.2f\n", media);

        fclose(arq);
}
```

Pesquise sobre as funções fscanf e fgets para leitura de dados e strings de um arquivo.

7. Função fread( ) e fwrite( ):

Estas funções podem ler e escrever tipos de dados maiores que um byte. Elas permitem a leitura e a escrita de blocos de qualquer tipo de dados.
Uma das mais úteis aplicações de fread( ) e fwrite( ) envolve ler e escrever tipos de dados definidos pelo usuário, especialmente estruturas.

Seus protótipos são:

size_t fread(void *buffer, size_t num_bytes, size_t count, FILE *fp);
size_t fwrite(const void *buffer, size_t num_bytes, size_t count, FILE *fp);

Para fread( ) buffer é um ponteiro para um região de memória que receberá os dados do arquivo. Para fwrite, buffer é um ponteiro para as informações que serão escritas no arquivo O número de bytes a ler ou a escrever é especificado por num_bytes. O argumento count determina quantos itens (cada um de comprimento num_byte) serão lidos ou escritos. Finalmente fp é um ponteiro para um stream aberta anteriormente. Os protótipos das duas funções estão definidas na bibliotecas stdio.h.

```
struct Aluno
{
    char nome[50];
    char curso[60];
    foat media;
};

void main( )
{
     struct Aluno academico;
     FILE *arq;

     printf(“\n Nome:”);
     gets(academico.nome);

     printf(“\n Curso:”);
     gets(academico.curso);

     printf(“\n media:”);
     scanf(“%f”, &academico.media);

      if((arq = fopen("arq_strings.dat", "w+"))== NULL)
      {
          printf("O arquivo não pode ser aberto!");
          exit(0);
      }

      fwrite(&academico, 1, sizeof(struct Aluno), arq);

      …….

      rewind(arq);
      fwrite(&academico, 1, sizeof(struct Aluno), arq);

      fclose();
}
```

8. Função fseek e E/S com acesso aleatório:

Esta função permite modificar o identificador de posição do arquivo.

Prototype da função:

int fseek(FILE *fp, long numbytes, int origin);

O fp é um ponteiro de arquivo devolvido por uma chamada à fopen( ). O numbytes, um inteiro longo, é o número de bytes a partir de origin, que se tornará a nova posição corrente, e origin é uma das seguintes macros definidas em sdtio.h:

Origin	Nome da Macro
Início do arquivo	SEEK_SET
Posição atual	SEEK_CUR
Final do Arquivo	SEEK_END

Exemplo:

```
.......
fseek(fp, 9 * sizeof(struct Aluno), SEEK_SET);
.....…
```