Laboratório de programação
Arquivos de Acesso Aleatório

Universidade Estadual Vale do Acaraú – UVA

Paulo Regis Menezes Sousa

paulo_regis@uvanet.br

Arquivos de acesso aleatório

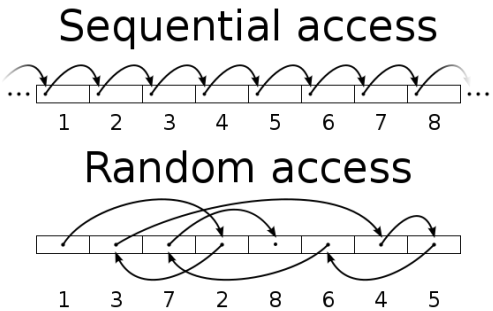
- Escrita
- Leitura

Excluindo arquivos

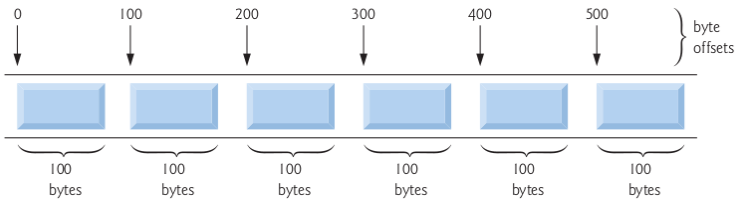
Argumentos em linha de comando

Executando outros programas com `system()`

- Arquivos de acesso aleatório são arquivos nos quais a leitura e gravação de dados não é realizada de forma sequencial.

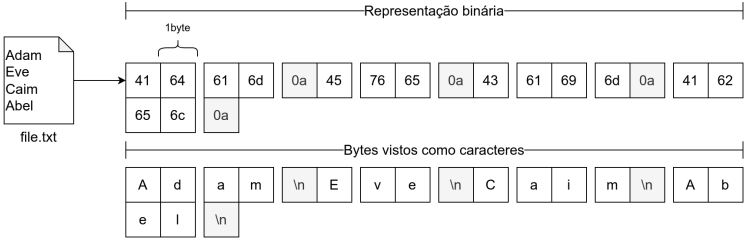


- A linguagem C não impõe nenhum tipo de estrutura de arquivo. Assim para usar esse tipo de arquivo precisamos definir uma forma de criá-los.
- A forma mais simples de implementação de arquivos de acesso aleatório é a imposição de um *registro de tamanho fixo* de

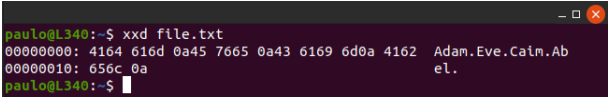


- Usando registros de mesmo tamanho e de tamanho fixo é fácil calcular a localização exata de cada registro em relação ao inicio do arquivo.

- Em um arquivo de texto a informação é registrada inteiramente como uma sequência de bytes que representam caracteres.



- Você pode criar uma visualização semelhante a esta usando o programa `xxd` no terminal de uma distribuição Linux.



rb	abre um arquivo binário existente para leitura (read)
wb	abre um arquivo binário para escrita (write). Se o arquivo já existe, seu conteúdo é descartado. Senão, um novo arquivo vazio é criado
ab	abre um arquivo binário para concatenação (append). Se o arquivo já existe, seu conteúdo é preservado e as escritas serão concatenadas no final do arquivo. Senão, um novo arquivo vazio é criado
r+b ou **rb+**	abre um arquivo binário existente para leitura e escrita. O conteúdo anterior do arquivo é preservado e o ponteiro é posicionado no início do arquivo
w+b ou **wb+**	abre um arquivo binário para leitura e escrita. Se o arquivo já existe, seu conteúdo é descartado. Senão, um novo arquivo vazio é criado
a+b ou **ab+**	abre um arquivo binário para escrita e concatenação. Se o arquivo já existe, seu conteúdo é preservado e as escritas serão concatenadas no final do arquivo. Se não, um novo arquivo binário vazio é criado. O ponteiro de leitura é posicionado no início do arquivo; as escritas são efetuadas no seu final

- Para escrever em um arquivo um blocos de bytes usa-se a função `fwrite()`, cujo protótipo é:

```
1 int fwrite(void *buffer, int nbytes, int count, FILE *fp)
```

- A função `fwrite()` recebe quatro parâmetros de entrada:
  - **buffer** : um ponteiro genérico para a região de memória que contém os dados que serão gravados no arquivo.
  - **nbytes** : tamanho, em bytes, de cada unidade de dado a ser gravada.
  - **count** : total de unidades de dados que devem ser gravadas.
  - **fp** : o ponteiro para o arquivo em que se deseja trabalhar.
- O valor do retorno da função será igual ao valor de **count**, a menos que ocorra algum erro na gravação dos dados.

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

int main(){
    FILE *arq;
    int totalGravado, v[5] = {1,2,3,4,5};

    arq = fopen("file.lab","wb");
    if(arq == NULL){
        printf("Problemas na CRIACAO do arquivo\n");
    }
    else{
        /* grava todo o array no arquivo (5 posições) */
        totalGravado = fwrite(v, sizeof(int), 5, arq);
        if(totalGravado != 5){
            printf("Erro na escrita do arquivo!");
        }
        fclose(arq);
    }
    return 0;
}
```

- Para ler de um arquivo um blocos de bytes usa-se a função `fread()`, cujo protótipo é:

```
1 int fread(void *buffer, int nbytes, int count, FILE *fp)
```

- A função `fread()` recebe quatro parâmetros de entrada:
  - **buffer** : um ponteiro genérico para a região de memória que contém os dados que serão gravados no arquivo.
  - **nbytes** : tamanho, em bytes, de cada unidade de dado a ser gravada.
  - **count** : total de unidades de dados que devem ser gravadas.
  - **fp** : o ponteiro para o arquivo em que se deseja trabalhar.
- O valor do retorno da função será igual ao valor de **count**, a menos que ocorra algum erro na gravação dos dados.

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    FILE *arq;
    int i, totalLido, v[5];
    arq = fopen("../arquivo-acesso-aleatorio/file.lab","rb");
    if(arq == NULL){
        printf("Problemas na ABERTURA do arquivo\n");
    }
    else{
        totalLido = fread(v, sizeof(int), 5, arq);
        if(totalLido != 5){
            printf("Erro na leitura do arquivo!");
        }
        else{
            for(i = 0; i < 5; i++)
                printf("v[%d] = %d\n",i,v[i]);
        }
        fclose(arq);
    }
    return 0;
}
```

- Como uma *string* pode ser gravada juntamente com o seu tamanho.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define T 20
int main(){
    FILE *arq;
    char str[T] = "Hello World!";
    int t = T;
    arq = fopen("file.lab","wb");
    if(arq == NULL){
        printf("Erro\n");
        return 1;
    }
    fwrite(&t,  sizeof(int),  1, arq);
    fwrite(str, sizeof(char), t, arq);
    fclose(arq);
    return 0;
}
```

- Como pode ser lida uma *string* gravada juntamente com o seu tamanho.

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    FILE *arq;
    char str[20];
    int t;
    arq = fopen("file.lab", "rb");
    if(arq == NULL){
        printf("Erro\n");
    }
    fread(&t, sizeof(int), 1, arq);
    fread(str,sizeof(char), t, arq);
    str[t] = '\0';
    printf("%s\n",str);
    fclose(arq);
    return 0;
}
```

- A linguagem C permite realizar operações de leitura e escrita randômica. Para isso, usa-se a função `fseek()`, cujo protótipo é:

```
1 int fseek(FILE *fp, long numbytes, int origem)
```

- A função `fseek()` recebe três parâmetros de entrada:
  - **fp**: o ponteiro para o arquivo em que se deseja trabalhar.
  - **nbytes**: é o total de bytes a partir de origem a ser pulado.
  - **origem** : determina a partir de onde os *nbytes* de movimentação serão contados.
- A função retorna um valor inteiro igual a ZERO quando a movimentação dentro do arquivo for bem-sucedida. Um valor de retorno diferente de zero significa que houve um erro durante a movimentação.

- Os valores possíveis para o parâmetro origem são definidos por constante na biblioteca `stdio.h` e são:

Constante	Valor	Significado
SEEK_SET	0	Início do arquivo
SEEK_CUR	1	Ponto atual no arquivo
SEEK_END	2	Fim do arquivo

Portanto, para movermos nbytes a partir do início do arquivo, a origem deve ser SEEK_SET. Se quisermos mover a partir da posição atual em que estamos no arquivo, devemos usar a constante SEEK_CUR. Por fim, se quisermos mover a partir do final do arquivo, a constante SEEK_END deverá ser usada.

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

struct cadastro{
    char nome[20];
    char rua[20];
    int  idade;
};

int main(){
    FILE *f;
    struct cadastro c, cad[4] = {{"Ricardo","Rua 1",31},
                                 {"Carlos" ,"Rua 2",28},
                                 {"Ana"    ,"Rua 3",45},
                                 {"Bianca" ,"Rua 4",32}};
```

```
1       f = fopen("file.cad","wb");
2       if(f == NULL){
3           printf("Erro na abertura\n");
4           return 1;
5       }
6       fwrite(cad, sizeof(struct cadastro), 4, f);
7       fclose(f);
8
9       f = fopen("file.cad","rb");
10      if(f == NULL){
11          printf("Erro na abertura\n");
12          return 1;
13      }
14      fseek(f, sizeof(struct cadastro), SEEK_CUR);
15      fread(&c,  sizeof(struct cadastro), 1, f);
16      printf("%s\n%s\n%d\n", c.nome, c.rua, c.idade);
17      fclose(f);
18
19      return 0;
20  }
```

- Além de permitir manipular arquivos, a linguagem C também permite excluí-los do disco rígido. Isso pode ser feito facilmente utilizando a função `remove()`, cujo protótipo é:

```
1 int remove(char *nomeArquivo)
```

- A função `remove()` recebe como parâmetro de entrada o caminho e o nome do arquivo a ser excluído do disco rígido, e não um ponteiro para **FILE**.
- Como resultado, essa função retorna um valor inteiro igual a ZERO quando houver sucesso na exclusão do arquivo. Um valor de retorno diferente de zero significa que houve um erro durante a sua exclusão.

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

int main(){
    int status;

    status = remove("file.txt");
    if(status != 0){
        perror("Erro na remocao do arquivo");
        return 1;
    }
    else
        printf("Arquivo removido com sucesso.\n");

    return 0;
}
```

- Em linguagem C podemos passar argumentos através da linha de comando para um programa quando ele inicia.
- A função `main` recebe parâmetros passados via linha de comando como vemos a seguir:

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

- Onde:
  - `argc` é um valor inteiro que indica a quantidade de argumentos que foram passados ao chamar o programa.
  - `argv` é um vetor de **char** que contém os argumentos, um para cada *string* passada na linha de comando.
- O primeiro elemento `argv[0]` armazena o nome do programa que foi chamado no *prompt*, sendo assim, `argc` é pelo menos igual a 1, pois no mínimo existirá um argumento.

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

int main(int argc, char* argv[]){
    int i;

    printf("programa: %s\n",argv[0]);

    if(argc == 1)
        printf(" info: nenhum parâmetro.\n");

    for(i=1; i<argc; i++)
        printf(" parametro [%d]: %s\n", i, argv[i]);

    return 0;
}
```

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

int main(int argc, char* argv[]){
    int i, soma;

    printf("programa: %s\n",argv[0]);

    if(argc == 1)
        printf(" info: nenhum parâmetro.\n");

    for(i=1, soma=0; i<argc; i++)
        soma = soma + atoi(argv[i]);

    printf(" soma = %d\n",soma);

    return 0;
}
```

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

int main(int argc, char*argv[]){
    int numCaracteres = 0;

    FILE *f = fopen(argv[1],"r");

    while (fgetc(f) != EOF) /* Ler um char */
        numCaracteres++;

    fclose(f);
    printf("total de caracteres: %d\n", numCaracteres);

    return 0;
}
```

- A função `system()` fornece ao seu programa um pequeno prompt de comando.
- Qualquer comando que você normalmente digita no prompt de comando real pode ser emitido do seu programa diretamente para o sistema operacional através da função `system()`.

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

int main(void){
    puts("Pressione Enter para ver uma lista de arquivos: ");
    getchar();
    system("dir");
    return (0);
}
```

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

int main(int argc, char* argv[]){
    char comando[120];

    sprintf(comando, "gcc %s.c -o %s", argv[1], argv[1]);
    printf("%s\n", comando);
    system(comando);

    return 0;
}
```

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

int main(int argc, char* argv[]){
    char comando[120];

    sprintf(comando, "mv %s novo-%s", argv[1], argv[1]);
    printf("%s\n", comando);
    system(comando);

    return 0;
}
```

Exercício 39

Faça um programa que leia 100 números. Esse programa deverá, em seguida, armazenar esses números em um arquivo binário.

Exercício 40

Elabore um programa que leia um arquivo binário contendo 100 números. Mostre na tela a soma desses números.

Exercício 41

Crie um programa que leia um arquivo binário contendo uma quantidade qualquer de números. O primeiro número lido indica quantos valores existem no arquivo. Mostre na tela o maior e o menor valor lido.

Exercício 42

Crie uma estrutura representando um atleta. Essa estrutura deve conter o nome do atleta, seu esporte, idade e altura. Agora, escreva um programa que leia os dados de cinco atletas e os armazene em um arquivo binário.

Exercício 43

Considerando a estrutura atleta do exercício anterior, escreva um programa que leia um arquivo binário contendo os dados de cinco atletas. Calcule e exiba o nome do atleta mais alto e do mais velho.