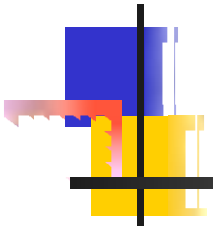
Linguagens de Programação II

Arquivos

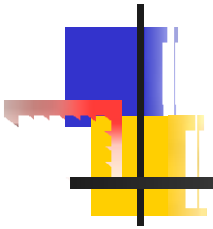
Dany Sanchez Dominguez

dsdominguez@gmail.com
Sala 1 - NBCGIB



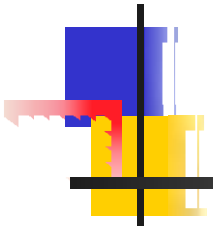
Roteiro

- Introdução
  - Arquivos ASCII vs Binários
- Hierarquia de dados
- Arquivos e fluxos (streams)
- Fluxos automáticos
- Declarar, abrir e fechar fluxos (arquivos)
- Arquivos de acesso sequencial
- Arquivos de acesso aleatório
- Comparação entre AS e AA



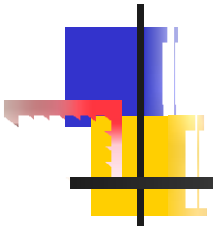
Introdução

- As variáveis de um programa são armazenadas na memória dinâmica do computador (memória principal),
- Porem, o armazenamento de dados em variáveis é temporário,
- Todos os dados são perdidos quando o programa termina,
- Para resolver este problema dispositivos de armazenamento definitivo devem ser utilizados.



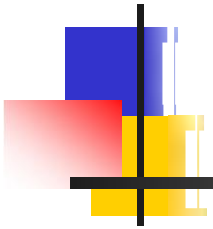
Introdução

- Arquivos são utilizados para armazenamento permanente dos dados,
- Arquivos são gravados na memória estática do computador (memória secundaria, discos, memórias flash)
- Os arquivos podem ser classificados como arquivos ASCII ou arquivos binários,



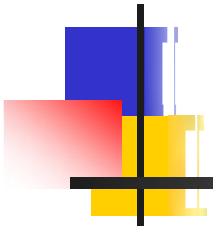
Introdução

- Arquivos ASCII
  - também conhecidos como arquivos de texto,
  - utilizam a representação ASCII (ou outras UFT8, ISO...) para armazenar os dados (caracteres ASCII),
  - são legíveis por humanos,
  - podem ser lidos e interpretados diretamente.



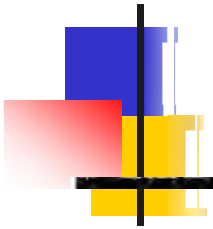
Introdução

- Arquivos binários
  - armazenam os dados usando representação binária (interna ao PC), (zeros e uns),
  - não são legíveis por humanos,
  - Arquivos binários devem ser interpretados por máquinas (Ex. computadores).



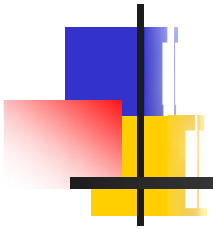
Introdução

- O sistema de arquivos da linguagem C é extremamente poderoso e flexível,
- Permite trabalhar com arquivos binários e arquivos texto,
- É possível criar arquivos que satisfazem qualquer necessidade,
- O trabalho com arquivos em C é considerado operações de E/S.



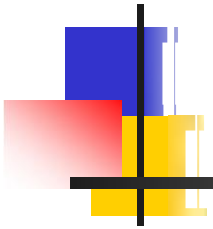
Hierarquia de dados

- Todos os itens de dados processados por um computador são reduzidos a combinações de zeros e uns,
- Os menores itens de dados no computador podem assumir o valor 0 ou 1,
- Este item de dado é chamado de **bit** (binary digit),
- Torna-se complicado para os programadores trabalhar com dados na forma de bits (baixo nível),



Hierarquia de dados

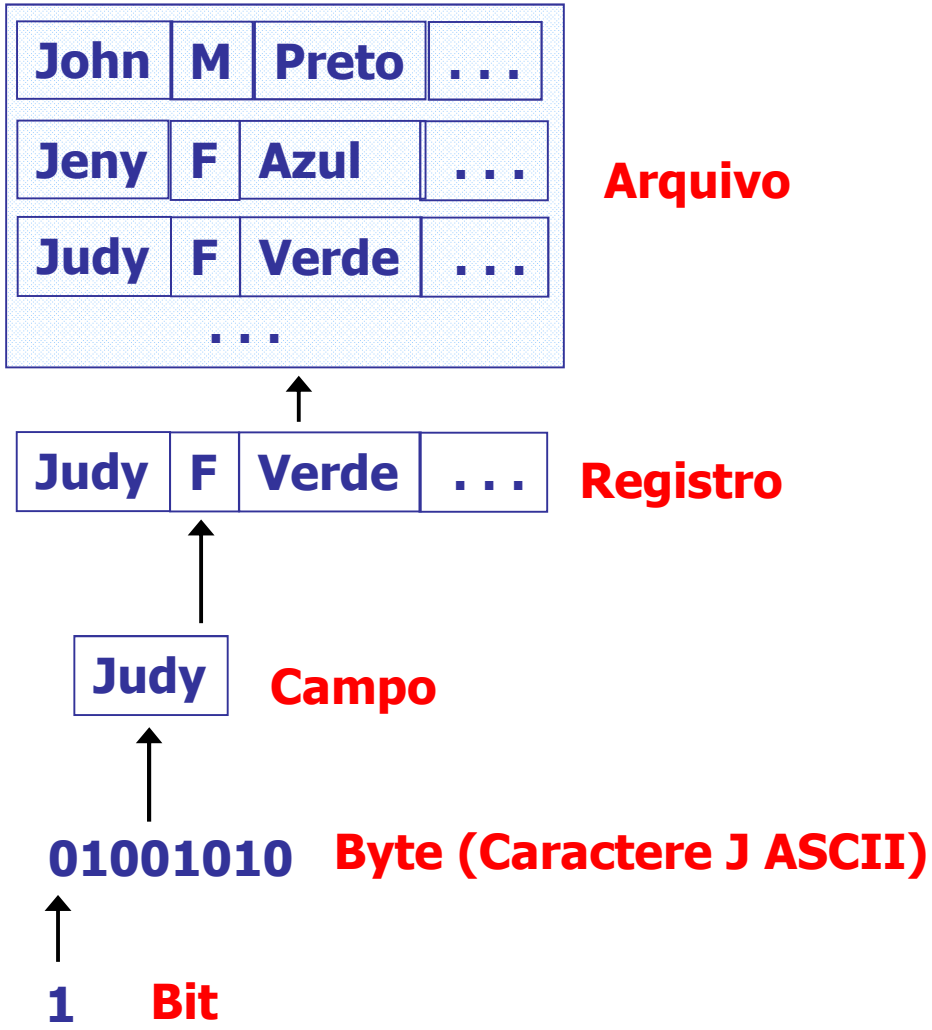
- Os programadores trabalham com o conjunto de caracteres da maquina (alto nível),
- Conjunto de caracteres da máquina
  - letras
  - dígitos
  - símbolos especiais
- Um **caractere** é representado por 1 byte (agrupação de 8 bits),
- Vários caracteres podem ser agrupados para formar um **campo** de dados,



Hierarquia de dados

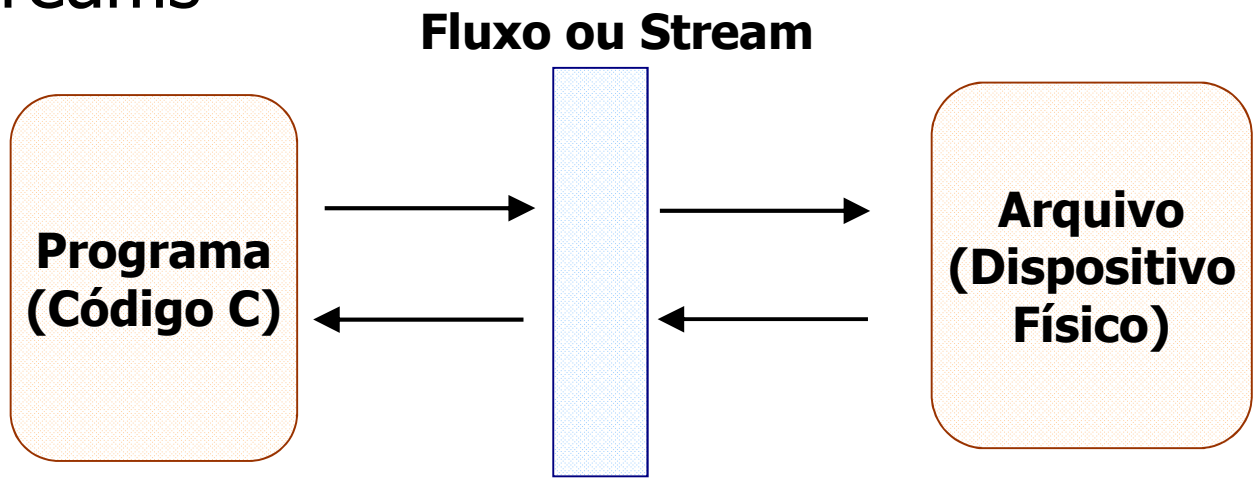
- Um **registro** é formado por vários campos (Ex. uma struct de C),
- A agrupação de vários registros nos levaria a um **arquivo**,
- Os itens de dados processados pelos computadores formam *hierarquia de dados*,
- Onde os dados se tornam maiores e mais complexos na medida que evoluímos de bits para arquivos,

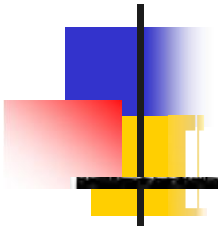
Hierarquia de dados



Arquivos e Fluxos (Streams)

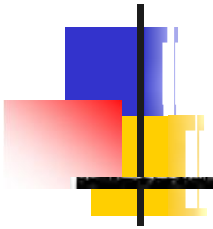
- O sistema de arquivos da linguagem C prevê a interação entre três entidades:
  - o programa em C,
  - o arquivo,
  - e um dispositivo lógico chamado de fluxo ou streams





Arquivos e Fluxos (Streams)

- A linguagem C fornece uma interfase que independe do dispositivo real que é acessado,
- Provê ao programador um nível de abstração entre seu programa e o arquivo que esta sendo acessado,
- O sistema de arquivos de C é projetado para trabalhar com uma ampla variedade de dispositivos: terminais, impressoras, discos, memórias flash, fitas.

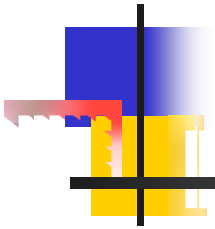


Arquivos e Fluxos (Streams)

- Embora existam grandes diferencias entre os dispositivos,
- Todos eles são transformados no mesmo dispositivo lógico (fluxo),
- Para o programador todos os fluxos tem o mesmo comportamento,
- Existem dois tipos de fluxos:
  - fluxos de textos
  - fluxos binários

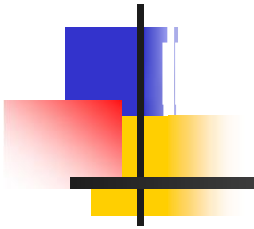
Arquivos e Fluxos (Streams)

- Fluxos de textos:
  - Um fluxo de texto é uma seqüência de caracteres,
  - Normalmente o fluxo é organizado em um conjunto de linhas que terminam no caractere nova linha,
  - Dependendo do conjunto de caracteres do sistema podem ocorrer "traduções" nos caracteres,
  - Podem existir diferencias entre a informação lida ou escrita (no fluxo) e a informação presente no dispositivo externo.



Arquivos e Fluxos (Streams)

- Fluxos binários:
  - Um fluxo binário é uma seqüência de bytes,
  - Existe uma total correspondência entre os dados (lidos ou escritos) e o dispositivo externo,
  - Não ocorre nenhuma tradução de caracteres.

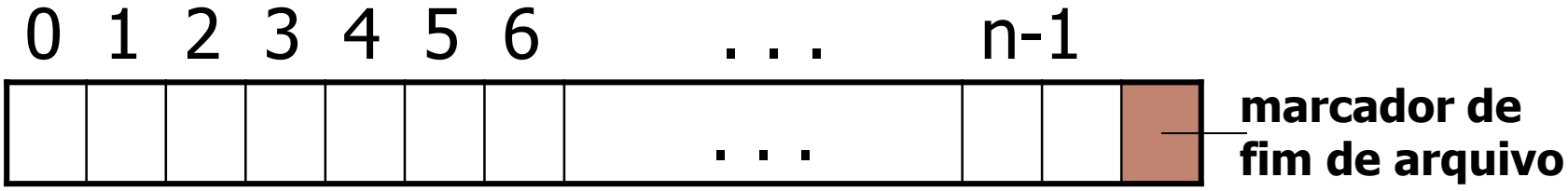


Arquivos e Fluxos (Streams)

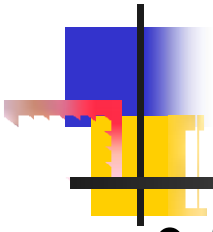
- Em C, um arquivo pode ser qualquer dispositivo (arquivo em disco, terminal ou teclado),
- Um fluxo é associado a um arquivo com uma operação de abertura de arquivo,
- Uma vez aberto o arquivo informações podem ser trocadas entre ele e seu programa,
- Nem todos os arquivos apresentam os mesmos recursos (Ex: acesso aleatório (disco = sim), (teclado ou fita = não))

Arquivos e Fluxos (Streams)

- No sistema de arquivos de C, todos os fluxos são iguais, mas não todos os arquivos,
- A linguagem C visualiza cada arquivo como um fluxo seqüencial de bytes:

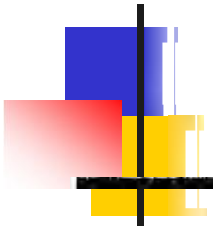


- Cada arquivo termina com um marcador final de arquivo.



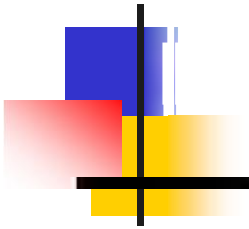
Arquivos e Fluxos (Streams)

- O sistema de arquivos de C, é um sistema com buffer (armazenamento temporário),
- O intercambio de informação entre o fluxo e o arquivo não acontece em forma continua,
- A informação é armazenada no fluxo até que um determinado tamanho de bloco seja alcançado, então o bloco é transferido para o arquivo,
- O uso do buffer aumenta o desempenho do sistema de arquivos.



Arquivos e Fluxos (Streams)

- Um arquivo é desassociado de um fluxo mediante uma operação de fechamento,
- Quando um arquivo é fechado, o conteúdo (se houver) do fluxo associado é escrito no dispositivo externo,
- Esse processo é conhecido como descarga (flushing) do fluxo,
- Garantindo que nenhuma informação seja acidentalmente deixada no fluxo.



Fluxos Automáticos

- Três arquivos e seus respectivos fluxos são abertos automaticamente quando inicia a execução de um programa:
  - entrada padrão (`stdin`) ligado ao teclado,
  - saída padrão (`stdout`) ligado à tela,
  - saída de erro padrão (`stderr`) ligado à tela.
- Estes fluxos permitem a interação do usuário com o programa, operações I/O,

Fluxos Automáticos

- Os fluxos automáticos apresentam o mesmo comportamento que os outros fluxos,
- Eles podem ser utilizados como parâmetros de fluxo em qualquer função de C,
- Eles não podem ser abertos ou fechados manualmente (programa),
- Eles podem ser redirecionados para outros arquivos,
- Eles são fechados automaticamente ao finalizar a execução do programa.

Declarando Fluxos (Streams)

- Um fluxo é declarado como um ponteiro a uma estrutura do tipo `FILE`,
- A estrutura `FILE` (`struct _iobuf`) é definida no arquivo de cabeçalho `<stdio.h>`,
- E contem as informações necessárias para processar o arquivo:
  - descritor de arquivos: índice para a tabela de arquivos abertos do sistema operacional,
  - BCA: bloco de controle de arquivos,
  - indicador de posição no arquivo (cursor).

Abrindo Arquivos (Streams)

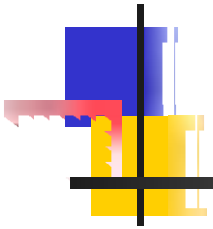
- Declaramos um fluxo (ponteiro a arquivo):

```
FILE *PFile
```

- Para abrir um arquivo usamos a função `fopen()`

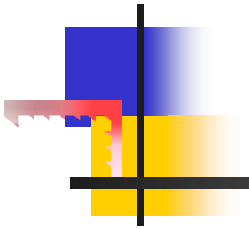
```
FILE *fopen(const char* nomearq,
            const char* modo)
```

- A função `fopen()` abre um fluxo para uso e associa um arquivo a ela,
- `fopen()` retorna um ponteiro a arquivo ou `NULL` se a operação falha,



Abrindo Arquivos (Streams)

- `fopen()` recebe dois argumentos:
  - `nomearq`, é um ponteiro para uma cadeia de caracteres que representa um nome de arquivo válido (pode incluir caminho),
  - `modo`, é uma cadeia de caracteres que determina como o arquivo será aberto.
- O modo determina as operações que poderão ser feitas com o arquivo.

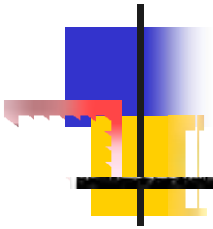


Abrindo Arquivos (Streams)

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 escrita/leitura
a+	Anexa a um arquivo-texto/escrita
rb+	Abre um arquivo binário para leitura/escrita
wb+	Cria um arquivo binário para escrita/escrita
ab+	Anexa a um arquivo binário/escrita

Abrindo Arquivos (Streams)

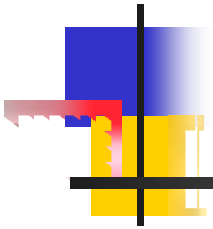
- Modos de abertura de arquivo:
  - **r**, abre um arquivo para leitura, o cursor é colocado no inicio do arquivo,
  - **w**, cria um arquivo para escrita, se o arquivo existe toda a informação é apagada,
  - **a**, abre um arquivo para escrita, se o arquivo existe a informação é mantida, o cursor é colocado no final do arquivo,
  - **b**, refere-se a arquivos binários
  - **+**, permite realizar operações de leitura e escrita no mesmo fluxo.



Abrindo Arquivos (Streams)

```
FILE *fp;

if ((fp = fopen("teste.txt", "w"))==NULL){
  printf("Erro ao abrir o arquivo!!!\n");
  exit(1);
}
```

- O sucesso de fopen deve ser verificado antes de realizar qualquer outra operação sobre o arquivo,
- Prevendo erros como: disco cheio, arquivo protegido, arquivo inexistente, ...



Fechando Arquivos

- Para fechar um arquivo usamos `fclose()`

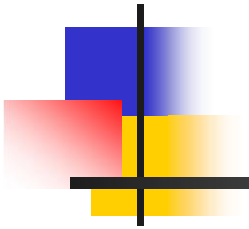
```
int fclose(FILE *fp)
```

- A função `fclose()` libera o fluxo associado ao arquivo deixando-o disponível para ser utilizado,
- Existe um limite do sistema operacional para o número de arquivos abertos simultaneamente,
- Feche um arquivo quando não for mais usar-lo,
- Os dados no buffer do fluxo são descarregados ao arquivo antes de liberar-lo.

Fechando Arquivos

- A função `fclose()` retorna um inteiro,
- Um valor zero no retorno indica que o fechamento foi bem sucedido,
- Qualquer valor diferente de zero indica um erro,

```
FILE *fp;

if ((fp = fopen("teste.txt", "w"))==NULL){
  printf("Erro ao abrir o arquivo!!!\n");
  exit(1);
}

//Usar o arquivo

fclose(fp);
```

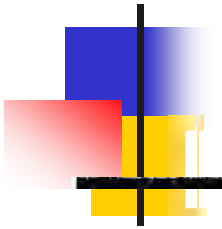


Verificação de Fim de Arquivo

- Para verificar se o fim de um arquivo foi alcançado usamos `feof()`

```
int feof (FILE *fp);
```

- A função `feof()` retorna verdadeiro (diferente de 0) se o final do arquivo foi alcançado, caso contrario retorna falso (0),
- O parâmetro de `feof()` é o fluxo para o qual desejamos verificar o fim de arquivo,



Verificação de Fim de Arquivo

- A função `feof()` é utilizada quando realizamos leitura de arquivos,
- `feof()` pode ser utilizado com o fluxo de entrada padrão `stdin`,
- Para simular o indicador de fim de arquivo via teclado usamos a combinação de teclas `<ctrl>z`.

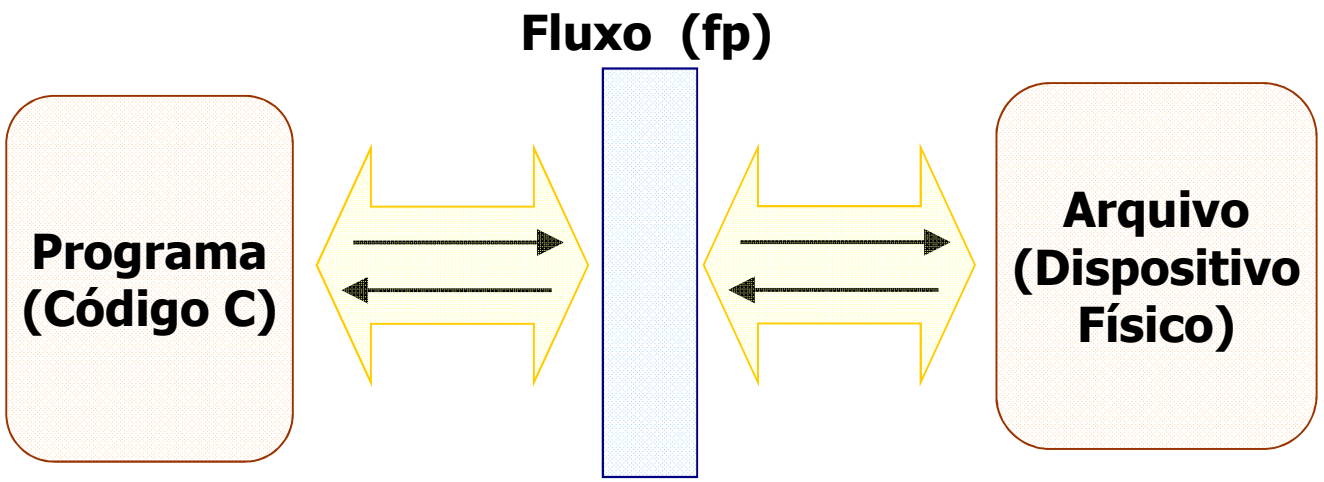
Exemplo

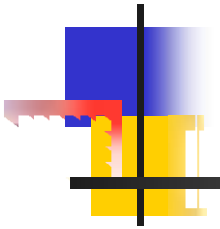
```
FILE *fp;
```

```
if ((fp = fopen("teste.txt", "r"))==NULL){
  printf("Erro ao abrir o arquivo!!!\n");
  exit(1);
}
```

```
while(!feof(fp)){
  //processa os elementos do arquivo
}
```

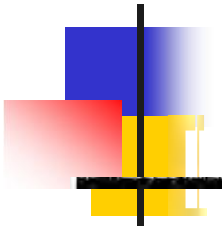
```
fclose(fp);
```





Arquivos de Acesso Seqüencial

- O acesso seqüencial a arquivos esta intimamente relacionado ao uso de arquivos textos,
- A linguagem C não impõe estrutura a um arquivo,
- O programador deve fornecer qualquer estrutura de arquivos visando satisfazer as exigências de uma aplicação específica,
- A escrita em arquivos seqüenciais é similar a escrita na tela.



Arquivos de Acesso Seqüencial

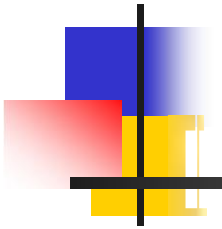
- Funções para escrever em arquivos seqüenciais:

```
int fprintf(FILE *fp, const char *format, variaveis);
```

  - Similar a função `printf()`,
  - Escreve no fluxo especificado e avança o cursor do arquivo,
  - O primeiro argumento é o fluxo de escrita,

```
int fputc(int ch, FILE *fp);
```

  - Similar a função `putc()`,
  - Escreve o caractere ch no fluxo fp e avança o cursor um caractere.



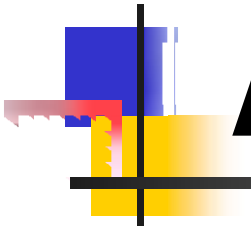
Arquivos de Acesso Seqüencial

- Funções para escrever em arquivos seqüenciais:

```
int fputs(const char *str, FILE *fp);
```

  - Similar a função puts(),
  - Escreve a string str no fluxo fp e avança o cursor até o final do arquivo.

- **Exemplo:** *Crie um programa que gere aleatoriamente uma matriz de inteiros com dimensão indicada pelo usuário e armazene a matriz no arquivo "mat.txt"*



Arquivos de Acesso Seqüencial

- **Exemplo ...**

Estrutura do arquivo de saída

```
Dimensao da Matriz A - n
a[0][0]       a[0][1]       a[0][2]       . . . a[0][n-1]
a[1][0]       a[1][1]       a[1][2]       . . . a[1][n-1]
                . . .         . . .
a[n-1][0]     a[n-1][1]     a[n-1][2]     . . . a[n-1][n-1]
```

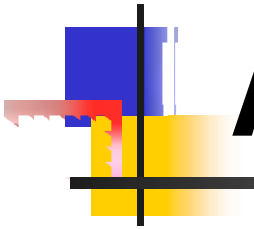
```c
#include <stdio.h>
#include <time.h>
//Funções auxiliares
int** aloca_matriz(int);
void libera_matriz(int **, int);
void gera_matriz(int **, int);
void grava_matriz(int **, int);
//Função main()
int main(){
  int n, **mat;

  printf("Digite a dimensao da matriz: ");
  scanf("%d", &n);

  mat = aloca_matriz(n);
  srand(time(NULL));
  gera_matriz(mat, n);
  grava_matriz(mat, n);
  libera_matriz(mat, n);

  return 0;
}
```

```c
void gera_matriz(int **m, int d){
  int i, j;

  for(i=0;i<d;i++)
    for(j=0;j<d;j++)
      m[i][j] = rand();
}
```

```
void grava_matriz(int **m, int d){
  int i, j;
  FILE *f;

  if ((f=fopen("mat.txt","w"))==NULL){
    printf("Erro ao criar arquivo!!!\n");
    exit(1);
  }

  fprintf(f, "%d\n", d);
  for(i=0;i<d;i++){
    for(j=0;j<d;j++)
      fprintf(f, "%7d", m[i][j]);
    fprintf(f, "\n");
  }

  fclose(f);
}
```

Arquivo: mat.txt

```
4
   1252   19773    3202    3127
    318   32680   32285    7955
   2784   12377   18051   27830
   2742   24288   30739   19159
```



Arquivos de Acesso Seqüencial

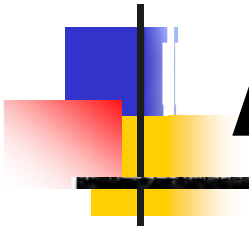
- Funções para leitura em arquivos seqüenciais:

```
int fscanf(FILE *fp, const char *format, variaveis);
```

  - Similar a função `scanf()`,
  - Lê dados no fluxo especificado (fp) e avança o cursor do arquivo,
  - O primeiro argumento é o fluxo para leitura,

```
int fgetc(FILE *fp);
```

  - Similar a função `getc()`,
  - Retorna o primeiro caractere do fluxo fp e avança o cursor um caractere.



Arquivos de Acesso Seqüencial

- Funções para leitura em arquivos seqüenciais:

```
char * fgets(const char *str, int num, FILE *fp);
```

  - Similar a função gets(),
  - Lê no máximo num-1 caracteres no fluxo fp e os coloca na cadeia de caracteres str,
  - Avança o cursor de arquivo num-1 caracteres.

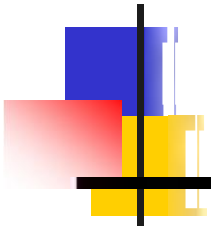
- **Exemplo:** *Crie um programa que leia e mostre na tela o conteúdo do arquivo "mat.txt" criado no exemplo anterior.*

```c
#include <stdio.h>
//Funções auxiliares
int** aloca_matriz(int);
void libera_matriz(int **, int);
int** le_matriz(int *);
void prn_matriz(int **, int);
//Função main()
int main(){
  int n, **mat;

  mat = le_matriz(&n);
  prn_matriz(mat, n);
  libera_matriz(mat, n);

  system("PAUSE");
  return 0;
}
```

```c
void prn_matriz(int **m, int d){
  int i, j;

  printf("Dimensao: %d\n", d);
  for(i=0;i<d;i++){
    for(j=0;j<d;j++)
      printf("%7d", m[i][j]);
    printf("\n");
  }
}
```

```
int ** le_matriz(int *d){
  int i, j, **m;
  FILE *f;

  if ((f=fopen("mat.txt","r"))==NULL){
    printf("Erro ao abrir arquivo!!!\n");
    exit(1);
  }

  fscanf(f, "%d\n", d);
  m = aloca_matriz(*d);
  for(i=0;i<*d;i++){
    for(j=0;j<*d;j++)
      fscanf(f, "%d", &m[i][j]);
    fscanf(f, "\n");
  }

  fclose(f);
  return m;
}
```

```
4
   1252  19773   3202   3127
    318  32680  32285   7955
   2784  12377  18051  27830
   2742  24288  30739  19159
```

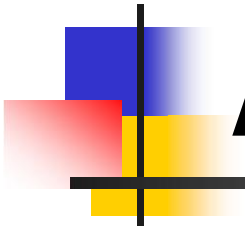


Arquivos de Acesso Seqüencial

- Outras funções:

```
int fflush(FILE *fp);
```

  - Descarga o conteúdo do buffer do fluxo para o arquivo,
  - Recebe como parâmetro o fluxo de arquivo,
  - Uma chamada com parâmetro NULL descarga o fluxo em todos os arquivos abertos,
  - Quando um arquivo é fechado, uma chamada a `fflush()` é feita.



Arquivos de Acesso Seqüencial

- Outras funções:

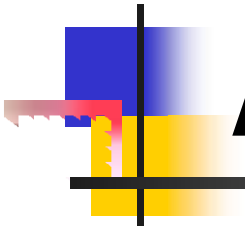
```
void rewind(FILE *fp);
```

  - Posiciona o cursor do arquivo no início do arquivo,
  - Recebe como parâmetro o fluxo de arquivo.

Arquivos de Acesso Seqüencial

- **Exemplo:** Crie um programa que receba um arquivo com os dados das contas de clientes de uma empresa (contas.txt). Seu programa deve mostrar um menu com as opções:
  1. Listar contas com saldo zero,
  2. Listar contas com saldo credor,
  3. Listar contas com saldo devedor,
  4. Encerrar programa.
- A arquivo tem a estrutura:

Conta	Titular	Saldo
100	Pedro	32.27
200	Pachuca	122.16
250	Santos	-50.16
320	Orlando	0.00



Arquivos de Acesso Seqüencial

- **Exemplo...**
  - Estrutura `TConta`:
    - Numero de conta
    - Titular da conta
    - Saldo
  - Função `void mostra_menu(void);`
  - Função `void lista_contas(FILE *, int);`
  - Função `void le_conta(FILE *, TConta *);`
  - Função `void prn_conta(const TConta *);`

```c
#include <stdio.h>

typedef struct{
  int nc;
  char titular[50];
  float saldo;
}TConta;

void mostra_menu(void);
void lista_contas(FILE *, int);
void le_conta(FILE *, TConta *);
void prn_conta(const TConta *);

int main(){
int op = 0;
  FILE *fp;

  if((fp = fopen("contas.txt","r"))==NULL){
    printf("Erro ao abrir arquivo!!!\n");
    system("PAUSE");
    return -1;
  }
```

```
  mostra_menu();
  while((op=getchar())!='4'){
    switch(op){
      case '1': case '2': case '3':
        lista_contas(fp, op);
      case '\n': case ' ':
        break;
      default:
        printf("Opcao incorreta.\n");
        system("PAUSE");
        break;
      }
    mostra_menu();
  }

  fclose(fp);

  return 0;
}
```

```
void mostra_menu(void){
  system("CLS");
  printf("Digite a opcao:\n");
  printf("(1) Contas com saldo zero\n");
  printf("(2) Contas com saldo credor\n");
  printf("(3) Contas com saldo devedor\n");
  printf("(4) Encerrar programa\n");
}
```

```
void le_conta(FILE *f, TConta *ct){
  fscanf(f, "%d", &(ct->nc));
  fscanf(f, "%s", &(ct->titular));
  fscanf(f, "%f", &(ct->saldo));
}
```

```
void prn_conta(const TConta *ct){
  printf("%5d %-10s %-6.2f\n",
              ct->nc, ct->titular, ct->saldo);
}
```

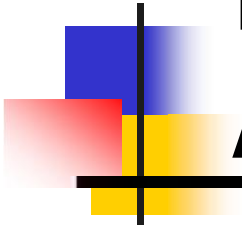
```
void lista_contas(FILE *f, int op){
  TConta dados;
  char aux[255];

  system("CLS");
  rewind(f);
  fgets(aux, 255, f);
  while(!feof(f)){
    le_conta(f, &dados);

    if (op=='1' && !dados.saldo)
      prn_conta(&dados);
    else if (op=='2' && dados.saldo>0)
      prn_conta(&dados);
    else if (op=='3' && dados.saldo<0)
      prn_conta(&dados);
  }
  system("PAUSE");
}
```

Arquivos de Acesso Seqüencial

- Que acontece no arquivo "contas.txt" se tentarmos modificar um registro intermédiario?
- Qual seria o tempo de acesso ao primeiro registro?
- Qual seria o tempo de acesso ao último registro?

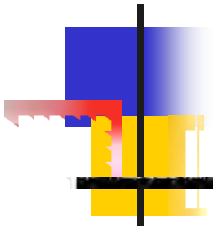
Conta	Titular	Saldo
100	Pedro	32.27
200	Pachuca	122.16
250	Santos	-50.16
320	Orlando	0.00

Limitações dos Arquivos de Acesso Seqüencial



- Impossível modificar registros intermédios sem afetar outros registros,
- O tempo de acesso a um registro depende da posição do registro no arquivo,
- A leitura (escrita) de um conjunto de registros é feita um registro de cada vez (vetor de registros).

- Uma alternativa as limitações dos arquivos de acesso seqüencial são os arquivos de acesso aleatório.

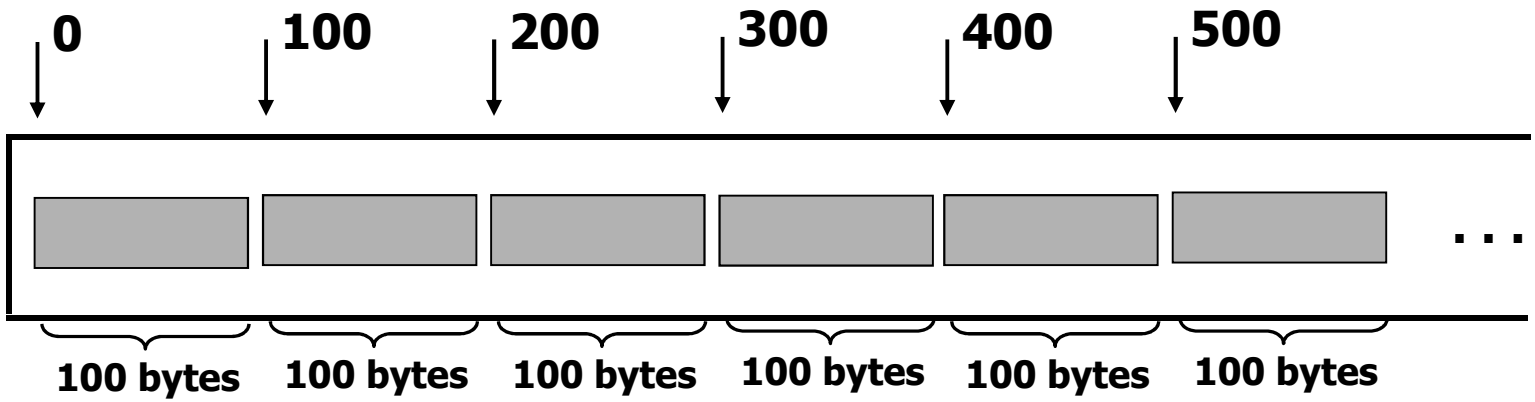


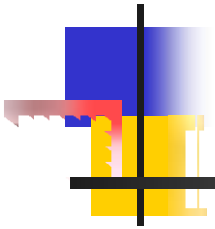
Arquivos de Acesso Aleatório

- Os registros em um arquivo seqüencial criados com E/S formatada, não possuem necessariamente o mesmo tamanho,
- Entretanto, os registros de um arquivo de acesso aleatório geralmente possuem o mesmo tamanho,
- O que permite que cada um dos registros pode ser acessado diretamente (rapidez),

Arquivos de Acesso Aleatório

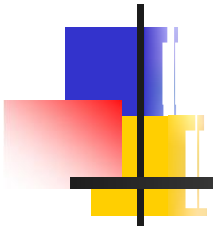
- Como todos os registros tem o mesmo comprimento, a localização de um registro relativa ao inicio do arquivo pode ser calculada facilmente,
- Vista de um arquivo de AA com registros de comprimento fixo (100 bytes)





Arquivos de Acesso Aleatório

- Recomendasse o uso de arquivos de acesso aleatórios em aplicações:
  - que trabalhem com grandes volumes de dados (arquivos grandes),
  - onde a velocidade de acesso a um registro seja crítica,
  - onde os registros precisem ser atualizados com freqüência.

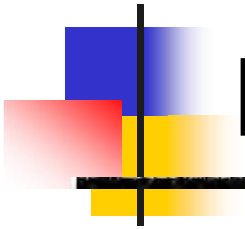


Escrita em Arquivos de AA

- Para escrever em um AAA usamos a função:

```
size_t fwrite(const void *ptr, size_t size,
              size_t count, FILE *fp);
```

  - Parâmetros:
    - `ptr`, ponteiro ao endereço do bloco de memória a ser escrito no arquivo,
    - `size`, Tamanho em bytes de cada elemento a ser escrito,
    - `count`, número de elementos a ser escrito,
    - `fp`, fluxo associado ao arquivo de saída.



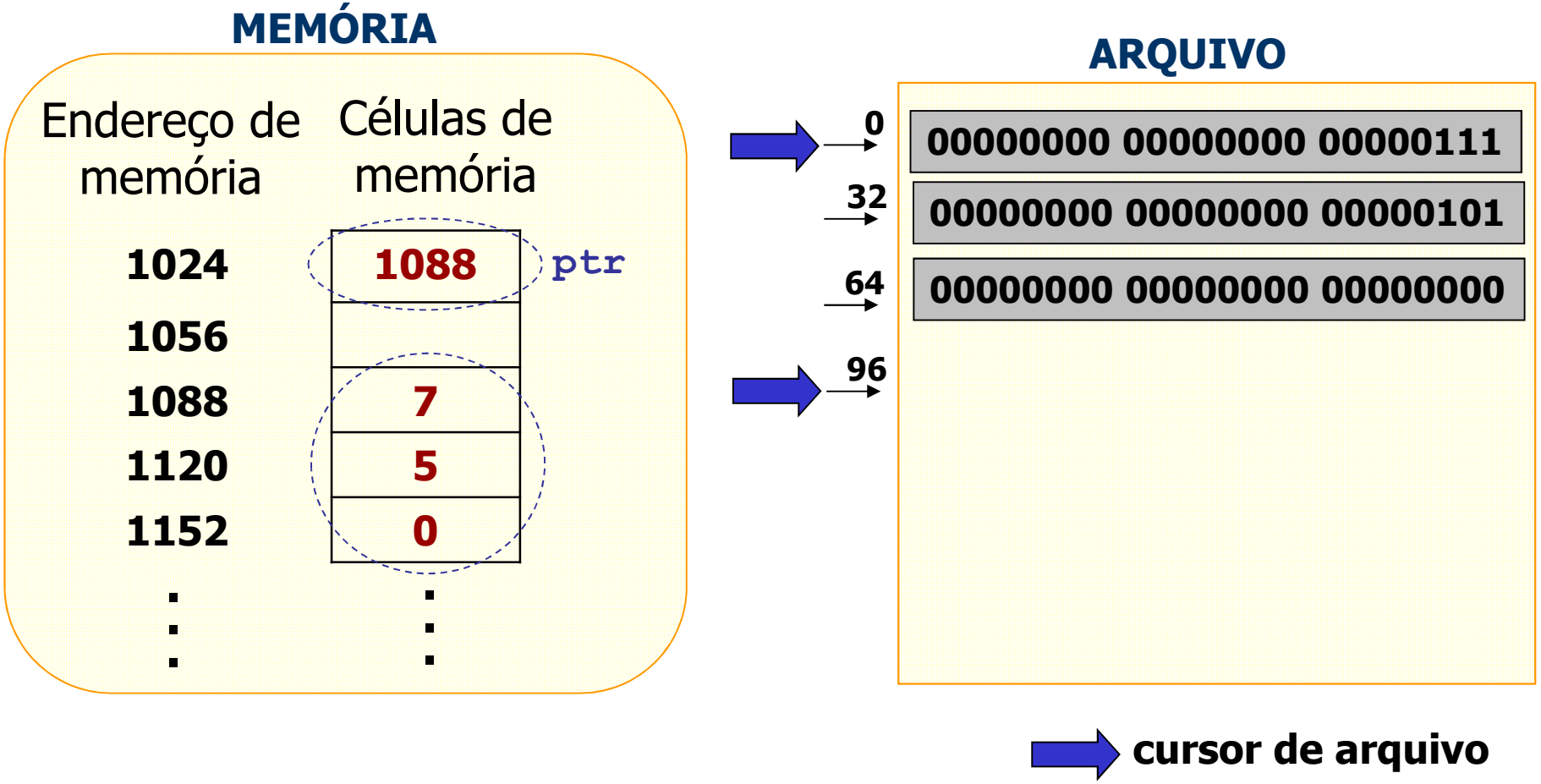
Escrita em Arquivos de AA

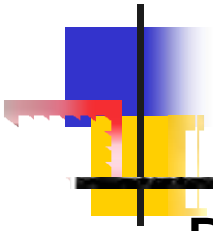
```
size_t fwrite(const void *ptr, size_t size,
              size_t count, FILE *fp);
```

- Escreve um bloco de dados em um fluxo de arquivo,
- Escreve um vetor de `count` elementos, procedente do bloco de memória apontado por `ptr`, na posição do cursor no fluxo `fp`,
- Cada elemento do vetor tem um tamanho de `size` bytes,
- O cursor de arquivo é movimentado até o final do bloco escrito no arquivo.
- Retorna o numero total de elementos escritos, se for diferente de `count` temos uma condição de erro.

Escrita em Arquivos de AA

```
fwrite(ptr, sizeof(int), 3, fp);
```



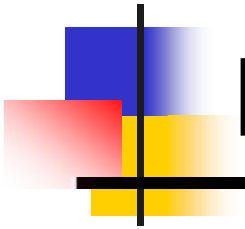


Leitura em Arquivos de AA

- Para ler em um AAA usamos a função:

```
size_t fread(void *ptr, size_t size,
             size_t count, FILE *fp);
```

  - Parâmetros:
    - `ptr`, ponteiro ao endereço do bloco de memória onde serão armazenados os valores lidos,
    - `size`, Tamanho em bytes de cada elemento a ser lido,
    - `count`, número de elementos a serem lidos,
    - `fp`, fluxo associado ao arquivo de entrada.



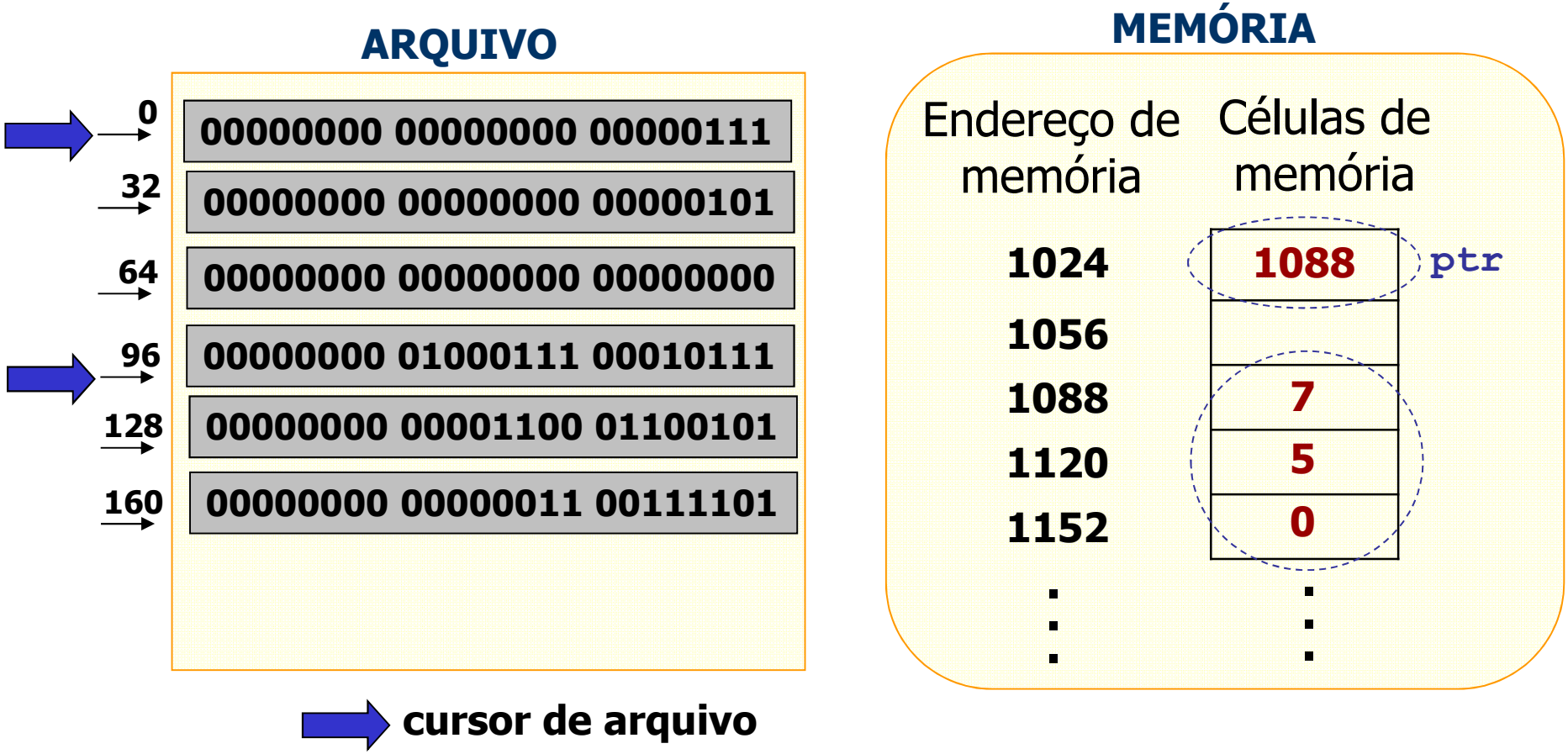
Escrita em Arquivos de AA

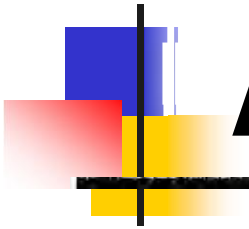
```
size_t fread(void *ptr, size_t size,
             size_t count, FILE *fp);
```

- Lê um bloco de dados de um fluxo de arquivo,
- Lê um vetor de `count` elementos no arquivo `fp` e armazena os dados no bloco de memória apontado por `ptr`,
- Cada elemento do vetor tem tamanho `size` bytes,
- O tamanho do bloco lido é (`size * count`)
- O cursor de arquivo avança até o final do bloco lido,
- A função retorna o número de elementos lidos, se o valor for diferente de `count` uma condição de erro esta presente.

Escrita em Arquivos de AA

```
fread(ptr, sizeof(int), 3, fp);
```



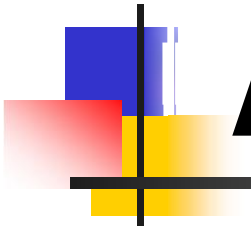


Arquivos de Acesso Aleatório

- Uma das principais vantagens dos AAA é que mais de um bloco de memória pode ser lidos ou escritos em uma única operação,
- Todos os elementos de um vetor podem ser gravados em uma única operação.

- **Exemplo**: Escreva um programa que crie um vetor e grave seus elementos no arquivo binário `vet_bin.dat`.

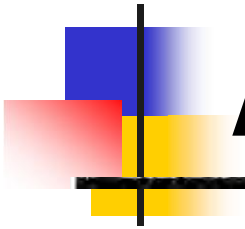
```c
#include <stdio.h>
#define N 5

int main(){
  int vet[N], i;
  FILE *fp;

  for(i=0;i<N;i++)
    vet[i] = i+1;

  if ((fp=fopen("vet_bin.dat","wb"))==NULL){
    printf("Erro ao abrir arquivo!!!\n");
    exit(1);
  }

  fwrite(vet, sizeof(int), N, fp);
  fclose(fp);

  system("PAUSE");
  return 0;
}
```



Arquivos de Acesso Aleatório

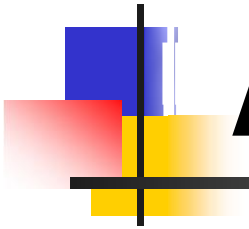
- **Exemplo**: Escreva um programa que lê os elementos de um vetor no arquivo binário `vet_bin.dat` e mostra eles na tela.

```c
#include <stdio.h>
#define N 5

int main(){
  int vet[N], i;
  FILE *fp;

  if ((fp=fopen("vet_bin.dat",“rb"))==NULL){
    printf("Erro ao abrir arquivo!!!\n");
    exit(1);
  }

  fread(vet, sizeof(int), N, fp);
  fclose(fp);

  for(i=0;i<N;i++)
    printf("%d\n",vet[i]);

  system("PAUSE");
  return 0;
}
```



Arquivos de Acesso Aleatório

- Os AAA são utilizados freqüentemente com variáveis de tipo estrutura (registros),
- Todos os campos da estrutura podem ser gravados ou lidos com uma única operação de E/S.
- Em arquivos seqüenciais cada campo da estrutura deve ser lido ou escrito separadamente.



Arquivos de Acesso Aleatório

- **Exemplo**: Escreva um programa que receba os dados de um aluno (nome, matricula e média) e grave eles em um arquivo binário.

- **Exemplo**: Escreva um programa que leia os dados de um aluno (exemplo anterior) de um arquivo e mostre eles na tela.

- **Exemplo**: Escreve estrutura

```
typedef struct{
  char nome[50];
  int matricula;
  float media;
}TAluno;

void le_aluno(TAluno *);

void grava_aluno(const TAluno *);

int main(){
  TAluno al;

  le_aluno(&al);
  grava_aluno(&al);

  system("PAUSE");
  return 0;
}
```

- **Exemplo**: Escreve estrutura ...

```
void le_aluno(TAluno *dad){
  printf("Digite o nome: ");
  gets(dad->nome);
  printf("Digite matricula: ");
  scanf("%d", &(dad->matricula));
  printf("Digite media: ");
  scanf("%f", &(dad->media));
}
```

```
void grava_aluno(const TAluno *dad){
  FILE *fp;

  if ((fp=fopen("aluno.dat","wb"))==NULL){
    printf("Erro!!!\n");
    exit(1);
  }
  fwrite(dad, sizeof(TAluno), 1, fp);
  fclose(fp);
}
```

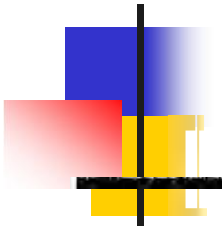
- **Exemplo**: Lê estrutura

```
typedef struct{
  char nome[50];
  int matricula;
  float media;
}TAluno;

void le_aluno(TAluno *);

void prn_aluno(const TAluno *);

int main(){
  TAluno al;

  le_aluno(&al);
  prn_aluno(&al);

  system("PAUSE");
  return 0;
}
```

- **Exemplo**: Lê estrutura ...

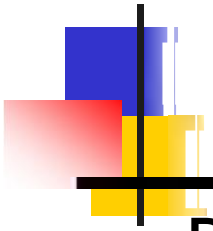
```
void le_aluno(TAluno *dad){
  FILE *fp;

  if ((fp=fopen("aluno.dat","rb"))==NULL){
    printf("Erro ao abrir arquivo!!!\n");
    exit(1);
  }
  fread(dad, sizeof(TAluno), 1, fp);
  fclose(fp);
}
```

```
void prn_aluno(const TAluno *dad){
  printf("Nome: %s\n", dad->nome);
  printf("Matricula: %d\n", dad->matricula);
  printf("Media: %.2f\n", dad->media);
}
```



Arquivos de Acesso Aleatório

- Em AAA dados podem ser inseridos, excluídos ou modificados sem que seja necessário reescrever todo o arquivo,
- Em arquivos de acesso aleatório registros podem ser modificados individualmente,
- Para navegar pelos registros dentro de um arquivo é necessário um mecanismo que permita manipular o cursor do arquivo,

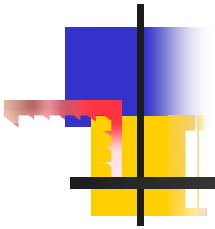


Manipulando o cursor do AA

- Para movimentar o cursor do arquivo usamos a função:

```
int fseek(FILE *fp, long int offset, int origin);
```

  - Desloca o cursor do arquivo associado ao fluxo `fp` a uma nova posição definida pelo valor `offset` e o origem `origin` do deslocamento.
  - Retorna zero se o deslocamento for bem sucedido ou diferente zero caso contrário.



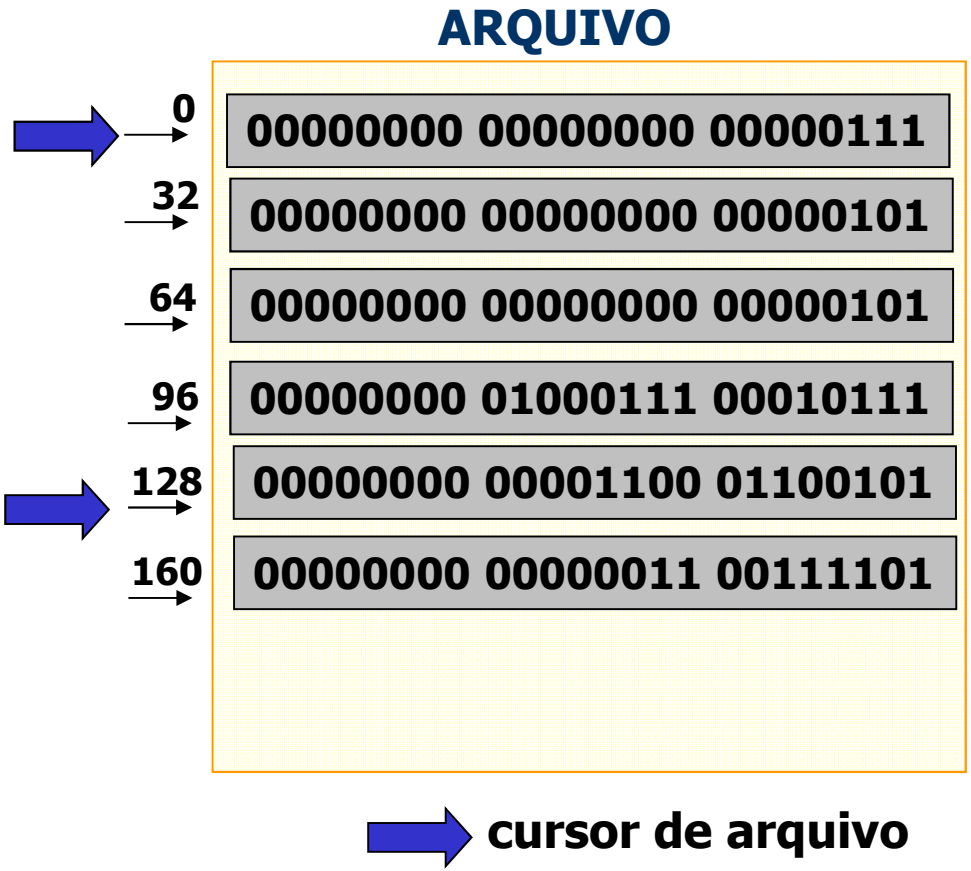
Manipulando o cursor do AA

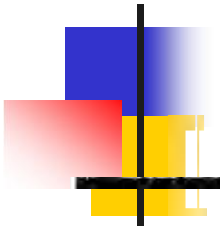
```
int fseek(FILE *fp, long int offset, int origin);
```

- Parâmetros:
  - `fp`, fluxo associado ao arquivo,
  - `offset`, numero de bytes do deslocamento,
  - `origin`, posição a partir da qual o deslocamento é computado.
- Existem três valores possíveis para o parâmetro `origin`
  - `SEEK_SET`, inicio do arquivo
  - `SEEK_CUR`, posição atual do cursor
  - `SEEK_END`, final do arquivo

Escrita em Arquivos de AA

```
fseek(fp, 4*sizeof(int), SEEK_SET);
```





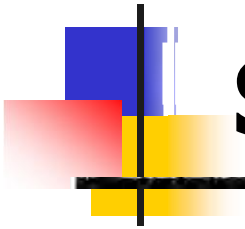
Manipulando o cursor do AA

- **Exemplo**: Considere um arquivo binário que armazena um vetor de 5 elementos (criado em exemplo anterior), escreva um programa que duplique o terceiro elemento e substitua por -1 o último elemento. Seu programa não deve utilizar variáveis de tipo vetor.

```
void mostra_arquivo(FILE *);

int main(){
  int num;
  FILE *fp;

  if ((fp=fopen("vet_bin.dat","rb+"))==NULL){
    printf("Erro ao abrir arquivo!!!\n");
    exit(1);
  }

  mostra_arquivo(fp);

  fseek(fp, 3*sizeof(int), SEEK_SET);
  fread(&num, sizeof(int), 1, fp);
  num *= 2;
  fseek(fp, -sizeof(int), SEEK_CUR);
  fwrite(&num, sizeof(int), 1, fp);

  mostra_arquivo(fp);
```

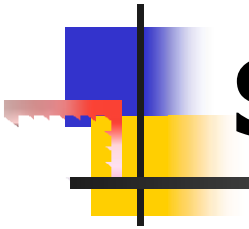
```
  num =-1;
  fseek(fp, -sizeof(int), SEEK_END);
  fwrite(&num, sizeof(int), 1, fp);

  mostra_arquivo(fp);

  fclose(fp);

  system("PAUSE");
  return 0;
}
```

```
void mostra_arquivo(FILE *f){
  int val;

  rewind(f);
  while((fread(&val, sizeof(int), 1, f))==1){
    printf("%d\n", val);
  }
  printf("\n");
  rewind(f);
}
```



Seqüencial vs Aleatório

	Seqüencial	Aleatório
Tam. do Registro	-Cada registro possui um tamanho diferente	-Os registros tem o mesmo tamanho
Tempo de acesso	-O tempo de acesso a um registro depende de sua posição no arquivo (lento)	-O tempo de acesso a um registro independe de sua posição no arquivo (rápido)
Modificação de registros	-Registros não podem ser modificados a menos que o todo o arquivo seja reescrito	-Registros podem ser modificados individualmente



Seqüencial vs Aleatório

	Seqüencial	Aleatório
Gravação de dados	-Dados são gravados usando o conjunto de caracteres da maquina	-Dados são gravados diretamente em formato binário
Tipos de arquivo	-Arquivos texto	-Arquivos binários
Aplicações	-Arquivos pequenos ou médios, dados precisam ser interpretados a olho nu	-Arquivos grandes, dados apenas são interpretados por maquinas