MC102 – Aula27
Recursão IV - MergeSort e Arquivos

Alexandre M. Ferreira

IC – Unicamp

08/06/2017

Roteiro

1. MergeSort
2. Arquivos
   - Introdução a Arquivos em C
   - Nomes e Extensões
   - Tipos de Arquivos
   - Caminhos Absolutos e Relativos
3. Arquivos Textos
   - Ponteiro para Arquivos
   - Abrindo um Arquivo
   - Lendo um Arquivo
   - Escrevendo em um Arquivo
4. Exemplos
5. Informações Extras: fscanf para ler **int**, **double**, etc.

Introdução

- Problema:
  - Temos um vetor **v** de inteiros de tamanho **n**.
  - Devemos deixar **v** ordenado crescentemente.
- Veremos um algoritmo baseado na técnica **dividir-e-conquistar** que usa recursão.

Introdução

- Problema:
  - Temos um vetor **v** de inteiros de tamanho **n**.
  - Devemos deixar **v** ordenado crescentemente.
- Veremos um algoritmo baseado na técnica **dividir-e-conquistar** que usa recursão.

Dividir e Conquistar

- Temos que resolver um problema P de tamanho n.
- **Dividir:** Quebramos P em sub-problemas menores.
- Resolvemos os sub-problemas de forma recursiva.
- **Conquistar:** Unimos as soluções dos sub-problemas para obter solução do problema maior P.

Dividir e Conquistar

- Temos que resolver um problema P de tamanho n.
- **Dividir:** Quebramos P em sub-problemas menores.
- Resolvemos os sub-problemas de forma recursiva.
- **Conquistar:** Unimos as soluções dos sub-problemas para obter solução do problema maior P.

Dividir e Conquistar

- Temos que resolver um problema P de tamanho n.
- **Dividir:** Quebramos P em sub-problemas menores.
- Resolvemos os sub-problemas de forma recursiva.
- **Conquistar:** Unimos as soluções dos sub-problemas para obter solução do problema maior P.

Dividir e Conquistar

- Temos que resolver um problema P de tamanho n.
- **Dividir:** Quebramos P em sub-problemas menores.
- Resolvemos os sub-problemas de forma recursiva.
- **Conquistar:** Unimos as soluções dos sub-problemas para obter solução do problema maior P.

Merge-Sort: Ordenação por intercalação

- O Merge-Sort é um algoritmo baseado na técnica dividir-e-conquistar.
- Neste caso temos que ordenar um vetor de tamanho n.
  - **Dividir:** Dividimos o vetor de tamanho n em dois sub-vetores de tamanho aproximadamente iguais (um de tamanho $\lceil n/2 \rceil$ e outro de tamanho $\lfloor n/2 \rfloor$).
  - Resolvemos o problema de ordenação de forma recursiva para estes dois sub-vetores.
  - **Conquistar:** Com os dois sub-vetores ordenados, construímos um vetor de tamanho n ordenado.

Merge-Sort: Ordenação por intercalação

- O Merge-Sort é um algoritmo baseado na técnica dividir-e-conquistar.
- Neste caso temos que ordenar um vetor de tamanho n.
  - **Dividir:** Dividimos o vetor de tamanho n em dois sub-vetores de tamanho aproximadamente iguais (um de tamanho $\lceil n/2 \rceil$ e outro de tamanho $\lfloor n/2 \rfloor$).
  - Resolvemos o problema de ordenação de forma recursiva para estes dois sub-vetores.
  - **Conquistar:** Com os dois sub-vetores ordenados, construímos um vetor de tamanho n ordenado.

Merge-Sort: Ordenação por intercalação

- O Merge-Sort é um algoritmo baseado na técnica dividir-e-conquistar.
- Neste caso temos que ordenar um vetor de tamanho n.
  - **Dividir:** Dividimos o vetor de tamanho n em dois sub-vetores de tamanho aproximadamente iguais (um de tamanho $\lceil n/2 \rceil$ e outro de tamanho $\lfloor n/2 \rfloor$).
  - Resolvemos o problema de ordenação de forma recursiva para estes dois sub-vetores.
  - **Conquistar:** Com os dois sub-vetores ordenados, construímos um vetor de tamanho n ordenado.

Merge-Sort: Ordenação por intercalação

- O Merge-Sort é um algoritmo baseado na técnica dividir-e-conquistar.
- Neste caso temos que ordenar um vetor de tamanho n.
  - **Dividir:** Dividimos o vetor de tamanho n em dois sub-vetores de tamanho aproximadamente iguais (um de tamanho $\lceil n/2 \rceil$ e outro de tamanho $\lfloor n/2 \rfloor$).
  - Resolvemos o problema de ordenação de forma recursiva para estes dois sub-vetores.
  - **Conquistar:** Com os dois sub-vetores ordenados, construímos um vetor de tamanho n ordenado.

Merge-Sort: Ordenação por intercalação

Conquistar: Dados dois vetores v_1 e v_2 ordenados, como obter um outro vetor ordenado contendo os elementos de v_1 e v_2?

V1	3	5	7	10	11	12							
V2	4	6	8	9	11	13	14						
	3	4	5	6	7	8	9	10	11	11	12	13	14

Merge (Fusão)

- A ideia é executar um laço onde em cada iteração testamos quem é o menor elemento dentre $v_1[i]$ e $v_2[j]$, e copiamos este elemento para o novo vetor.
- Durante a execução deste laço podemos chegar em uma situação onde todos os elementos de um dos vetores (v_1 ou v_2) foram todos avaliados. Neste caso terminamos o laço e copiamos os elementos restantes do outro vetor.

Merge (Fusão)

- A ideia é executar um laço onde em cada iteração testamos quem é o menor elemento dentre $v_1[i]$ e $v_2[j]$, e copiamos este elemento para o novo vetor.
- Durante a execução deste laço podemos chegar em uma situação onde todos os elementos de um dos vetores (v_1 ou v_2) foram todos avaliados. Neste caso terminamos o laço e copiamos os elementos restantes do outro vetor.

Merge (Fusão)

Retorna um vetor ordenado que é a fusão dos vetores ordenados passados por parâmetro:

```
int * merge(int a[], int ta,  int b[], int tb){
  int *c = malloc(sizeof(int)*(ta+tb) );
  int i=0,j=0,k=0; //índice de a, b, e c resp.

  while(i< ta && j< tb){ //Enquanto não processou completamente um
    if(a[i] <= b[j])     //dos vetores, copia menor elem para vet c
      c[k++] = a[i++];
    else
      c[k++] = b[j++];
  }

  while(i<ta) //copia resto de a
    c[k++] = a[i++];
  while(j<tb) //copia resto de b
    c[k++] = b[j++];
  return c;
}
```

Merge (Fusão)

Retorna um vetor ordenado que é a fusão dos vetores ordenados passados por parâmetro:

```
int * merge(int a[], int ta,  int b[], int tb){
  int *c = malloc(sizeof(int)*(ta+tb) );
  int i=0,j=0,k=0; //índice de a, b, e c resp.

  while(i< ta && j< tb){ //Enquanto não processou completamente um
    if(a[i] <= b[j])      //dos vetores, copia menor elem para vet c
      c[k++] = a[i++];
    else
      c[k++] = b[j++];
  }

  while(i<ta) //copia resto de a
    c[k++] = a[i++];
  while(j<tb) //copia resto de b
    c[k++] = b[j++];
  return c;
}
```

Merge (Fusão)

Retorna um vetor ordenado que é a fusão dos vetores ordenados passados por parâmetro:

```
int * merge(int a[], int ta,  int b[], int tb){
  int *c = malloc(sizeof(int)*(ta+tb) );
  int i=0,j=0,k=0; //índice de a, b, e c resp.

  while(i< ta && j< tb){ //Enquanto não processou completamente um
    if(a[i] <= b[j])     //dos vetores, copia menor elem para vet c
      c[k++] = a[i++];
    else
      c[k++] = b[j++];
  }

  while(i<ta) //copia resto de a
    c[k++] = a[i++];
  while(j<tb) //copia resto de b
    c[k++] = b[j++];
  return c;
}
```

Merge (Fusão)

- A função descrita recebe dois vetores ordenados e devolve um terceiro contendo todos os elementos.
- Porém no merge-sort faremos a intercalação de sub-vetores de um mesmo vetor.
- Isto evitará a alocação de vários vetores durante as chamadas recursivas, melhorando a performance do algoritmo.
- Teremos posições **ini, meio, fim** de um vetor e devemos fazer a intercalação dos dois sub-vetores: um de **ini** até **meio**, e outro de **meio+1** até **fim**.
  - Para isso a função utiliza um vetor auxiliar, que receberá o resultado da intercalação, e que no final é copiado para o vetor a ser ordenado.

Merge (Fusão)

- A função descrita recebe dois vetores ordenados e devolve um terceiro contendo todos os elementos.
- Porém no merge-sort faremos a intercalação de sub-vetores de um mesmo vetor.
- Isto evitará a alocação de vários vetores durante as chamadas recursivas, melhorando a performance do algoritmo.
- Teremos posições **ini, meio, fim** de um vetor e devemos fazer a intercalação dos dois sub-vetores: um de **ini** até **meio**, e outro de **meio+1** até **fim**.
  - Para isso a função utiliza um vetor auxiliar, que receberá o resultado da intercalação, e que no final é copiado para o vetor a ser ordenado.

Merge (Fusão)

- A função descrita recebe dois vetores ordenados e devolve um terceiro contendo todos os elementos.
- Porém no merge-sort faremos a intercalação de sub-vetores de um mesmo vetor.
- Isto evitará a alocação de vários vetores durante as chamadas recursivas, melhorando a performance do algoritmo.
- Teremos posições **ini, meio, fim** de um vetor e devemos fazer a intercalação dos dois sub-vetores: um de **ini** até **meio**, e outro de **meio+1** até **fim**.
    - Para isso a função utiliza um vetor auxiliar, que receberá o resultado da intercalação, e que no final é copiado para o vetor a ser ordenado.

Merge (Fusão)

Faz intercalação de pedaços de **v**. No fim **v** estará ordenado entre as posições **ini** e **fim**:

```
void merge(int v[], int ini,  int meio, int fim, int aux[]){
  int i=ini,j=meio+1,k=0; //indices da metade inf, sup e aux respc.

  while(i<=meio && j<=fim){ //Enquanto não processou um vetor inteiro.
    if(v[i] <= v[j])
      aux[k++] = v[i++];
    else
      aux[k++] = v[j++];
  }

  while(i<=meio) //copia resto do primeiro sub-vetor
    aux[k++] = v[i++];

  while(j<=fim)  //copia resto do segundo sub-vetor
    aux[k++] = v[j++];

  for(i=ini, k=0 ; i<= fim; i++, k++) //copia vetor ordenado aux para v
      v[i]=aux[k];
}
```

Merge (Fusão)

Faz intercalação de pedaços de **v**. No fim **v** estará ordenado entre as posições **ini** e **fim**:

```c
void merge(int v[], int ini,  int meio, int fim, int aux[]){
  int i=ini,j=meio+1,k=0; //indices da metade inf, sup e aux respc.

  while(i<=meio && j<=fim){ //Enquanto não processou um vetor inteiro.
    if(v[i] <= v[j])
      aux[k++] = v[i++];
    else
      aux[k++] = v[j++];
  }

  while(i<=meio) //copia resto do primeiro sub-vetor
    aux[k++] = v[i++];

  while(j<=fim)  //copia resto do segundo sub-vetor
    aux[k++] = v[j++];

  for(i=ini, k=0 ; i<= fim; i++, k++) //copia vetor ordenado aux para v
      v[i]=aux[k];
}
```

Merge (Fusão)

Faz intercalação de pedaços de **v**. No fim **v** estará ordenado entre as posições **ini** e **fim**:

```
void merge(int v[], int ini, int meio, int fim, int aux[]){
  int i=ini,j=meio+1,k=0; //indices da metade inf, sup e aux respc.

  while(i<=meio && j<=fim){ //Enquanto não processou um vetor inteiro.
    if(v[i] <= v[j])
      aux[k++] = v[i++];
    else
      aux[k++] = v[j++];
  }

  while(i<=meio) //copia resto do primeiro sub-vetor
    aux[k++] = v[i++];

  while(j<=fim)  //copia resto do segundo sub-vetor
    aux[k++] = v[j++];

  for(i=ini, k=0 ; i<= fim; i++, k++) //copia vetor ordenado aux para v
      v[i]=aux[k];
}
```

Merge-Sort

- O merge-sort resolve de forma recursiva dois sub-problemas, cada um contendo uma metade do vetor original.
- Com a resposta das chamadas recursivas podemos chamar a função **merge** para obter um vetor ordenado.

Merge-Sort

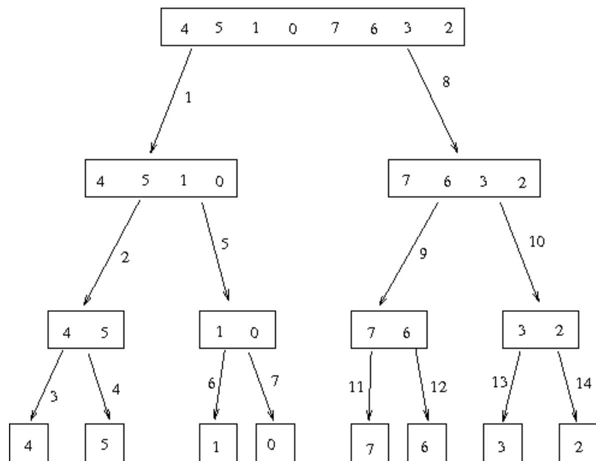
- O merge-sort resolve de forma recursiva dois sub-problemas, cada um contendo uma metade do vetor original.
- Com a resposta das chamadas recursivas podemos chamar a função **merge** para obter um vetor ordenado.

Merge-Sort

```
void mergeSort(int v[],int ini, int fim, int aux[]){
  int meio = (fim+ini)/2;
  if(ini < fim){    //Se tiver pelo menos 2 elementos então ordena
    mergeSort(v,ini,meio,aux);
    mergeSort(v,meio+1,fim,aux);
    merge(v,ini,meio,fim,aux);
  }
}
```

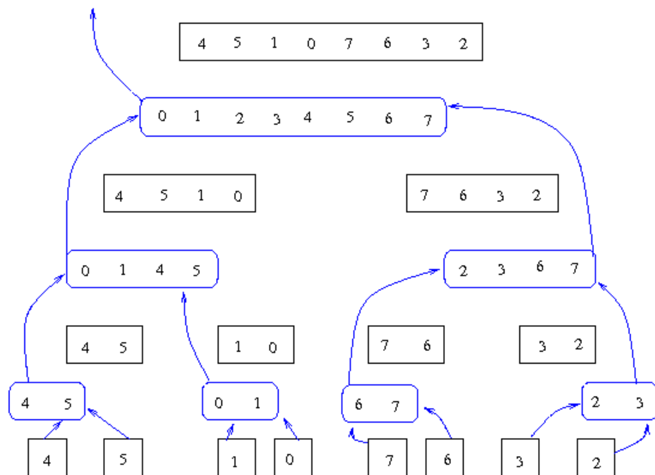
Merge-Sort

Abaixo temos um exemplo com a ordem de execução das chamadas recursivas.



Merge-Sort

Abaixo temos o retorno do exemplo anterior.



Merge-Sort: Exemplo de uso

- Note que só criamos 2 vetores, **v** a ser ordenado e **aux** do mesmo tamanho de **v**.
- Somente estes dois vetores existirão durante todas as chamadas recursivas.

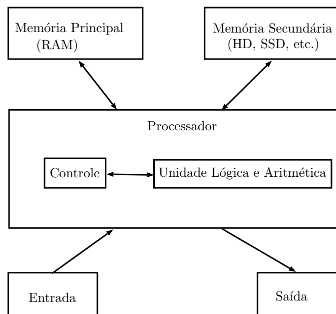
```c
#include "stdio.h"
#include <stdlib.h>

void merge(int v[], int ini, int meio, int fim, int aux[]);
void mergeSort(int v[], int ini, int fim, int aux[]);


int main(){
   int v[]={12,90, 47, -9, 78, 45, 78, 3323, 1, 2, 34, 20};
   int aux[12];
   int i;
   mergeSort(v, 0, 11, aux);
   for(i=0; i<12; i++)
      printf("\n %d",v[i]);
}
```

Exercícios

1. Mostre passo a passo a execução da função merge considerando dois sub-vetores: $(3, 5, 7, 10, 11, 12)$ e $(4, 6, 8, 9, 11, 13, 14)$.
2. Faça uma execução Passo-a-Passo do Merge-Sort para o vetor: $(30, 45, 21, 20, 6, 715, 100, 65, 33)$.
3. Reescreva o algoritmo Merge-Sort para que este passe a ordenar um vetor em ordem decrescente.
4. Considere o seguinte problema: Temos como entrada um vetor de inteiros v (não necessariamente ordenado), e um inteiro x. Desenvolva um algoritmo que determina se há dois números em v cuja soma seja x. Tente fazer o algoritmo o mais eficiente possível. Utilize um dos algoritmos de ordenação na sua solução.

Tipos de Memória

- Quando vimos a organização básica de um sistema computacional, havia somente um tipo de memória.
- Mas na maioria dos sistemas, a memória é dividida em dois tipos:



Tipos de Memória

- A memória principal (Random Access Memory) utilizada na maioria dos computadores, usa uma tecnologia que requer alimentação constante de energia para que informações sejam preservadas.

Tipos de Memória

- A memória secundária (como Hard Disks ou SSD) utilizada na maioria dos computadores, usa uma outra tecnologia que NÃO requer alimentação constante de energia para que informações sejam preservadas.

Tipos de Memória

- Todos os programas executam na RAM, e por isso quando o programa termina ou acaba energia, as informações do programa são perdidas.
- Para podermos gravar informações de forma *persistente*, devemos escrever estas informações em arquivos na memória secundária.
- A memória secundária possui algumas características:
  - É muito mais lenta que a RAM.
  - É muito mais barata que a memória RAM.
  - Possui maior capacidade de armazenamento.
- Sempre que nos referirmos a um arquivo, estamos falando de informações armazenadas em memória secundária.

Tipos de Memória

- Todos os programas executam na RAM, e por isso quando o programa termina ou acaba energia, as informações do programa são perdidas.
- Para podermos gravar informações de forma *persistente*, devemos escrever estas informações em arquivos na memória secundária.
- A memória secundária possui algumas características:
  - É muito mais lenta que a RAM.
  - É muito mais barata que a memória RAM.
  - Possui maior capacidade de armazenamento.
- Sempre que nos referirmos a um arquivo, estamos falando de informações armazenadas em memória secundária.

Tipos de Memória

- Todos os programas executam na RAM, e por isso quando o programa termina ou acaba energia, as informações do programa são perdidas.
- Para podermos gravar informações de forma *persistente*, devemos escrever estas informações em arquivos na memória secundária.
- A memória secundária possui algumas características:
  - É muito mais lenta que a RAM.
  - É muito mais barata que a memória RAM.
  - Possui maior capacidade de armazenamento.
- Sempre que nos referirmos a um arquivo, estamos falando de informações armazenadas em memória secundária.

Tipos de Memória

- Todos os programas executam na RAM, e por isso quando o programa termina ou acaba energia, as informações do programa são perdidas.
- Para podermos gravar informações de forma *persistente*, devemos escrever estas informações em arquivos na memória secundária.
- A memória secundária possui algumas características:
  - É muito mais lenta que a RAM.
  - É muito mais barata que a memória RAM.
  - Possui maior capacidade de armazenamento.
- Sempre que nos referirmos a um arquivo, estamos falando de informações armazenadas em memória secundária.

Nomes e Extensões

- Arquivos são identificados por um nome.
- O nome de um arquivo pode conter uma extensão que indica o conteúdo do arquivo.

Algumas extensões

arq.txt	arquivo texto simples
arq.c	código fonte em C
arq.pdf	*portable document format*
arq.html	arquivo para páginas WWW *(hypertext markup language)*
arq*	arquivo executável (UNIX)
arq.exe	arquivo executável (Windows)

Tipos de arquivos

Arquivos podem ter o mais variado conteúdo, mas do ponto de vista dos programas existem apenas dois tipos de arquivo:

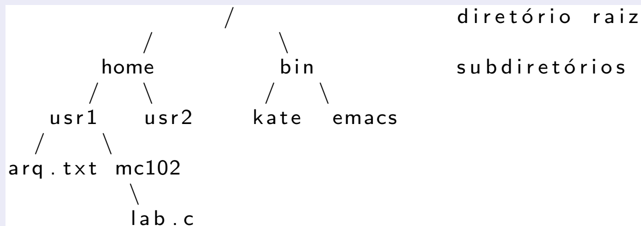
Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples. Exemplos: código fonte C, documento texto simples, páginas HTML.

Arquivo binário: Sequência de bits sujeita às convenções dos programas que o gerou, não legíveis diretamente. Exemplos: arquivos executáveis, arquivos compactados, documentos do Word.

Diretório

- Também chamado de pasta.
- Contém arquivos e/ou outros diretórios.

Uma hierarquia de diretórios



Caminhos Absolutos e Relativos

- O nome de um arquivo pode conter o seu diretório, ou seja, o caminho para encontrar este arquivo a partir da raiz.
- Desta forma o acesso a um arquivo pode ser especificado de duas formas:

Caminho absoluto: descrição de um caminho desde o diretório raiz.

```
/bin/emacs
/home/usr1/arq.txt
```

Caminho relativo: descrição de um caminho a partir do diretório corrente.

```
arq.txt
mc102/lab.c
```

Arquivos texto em C

- Em C, para se trabalhar com arquivos devemos criar um ponteiro especial: **um ponteiro para arquivos.**

```
FILE *nome_variavel;
```

- O comando acima cria um ponteiro para arquivos, cujo nome da variável é o nome especificado.
- Após ser criado um ponteiro para arquivo, podemos associá-lo com um arquivo real do computador usando a função **fopen**.

```
FILE *arq1;
arq1 = fopen("teste.txt","r");
```

- Neste exemplo a variável ponteiro **arq1** fica apontando para o arquivo **teste.txt**.

Arquivos texto em C

```
FILE *arq1;
arq1 = fopen("teste.txt","r");
```

- O primeiro parâmetro para **fopen** é uma string com o nome do arquivo
  - Pode ser absoluto, por exemplo: "/user/eduardo/teste.txt"
  - Pode ser relativo como no exemplo acima: "teste.txt"
- O segundo parâmetro é uma string informando como o arquivo será aberto.
  - Se para leitura ou gravação de dados, ou ambos.
  - Se é texto ou se é binário.
  - No nosso exemplo, o **"r"** significa que abrimos um arquivo texto para leitura.

Abrindo um Arquivo Texto para Leitura

- Antes de acessar um arquivo, devemos abri-lo com a função `fopen()`.
- A função retorna um ponteiro para o arquivo em caso de sucesso, e em caso de erro a função retorna `NULL`.

Abrindo o arquivo `teste.txt`

```
FILE *arq = fopen("teste.txt", "r");
if (arq == NULL)
  printf("Erro ao tentar abrir o arquivo teste.txt.");
else
  printf("Arquivo aberto para leitura.\n");
```

Lendo Dados de um Arquivo Texto

- Para ler dados do arquivo aberto, usamos a função **fscanf()**, que é semelhante à função **scanf()**.
  - **int fscanf(ponteiro para arquivo, string de formato, variáveis)**.
  - A única diferença para o **scanf**, é que devemos passar como primeiro parâmetro um ponteiro para o arquivo de onde será feita a leitura.

Lendo dados do arquivo teste.txt

```
char aux;
FILE *f = fopen ("teste.txt", "r");
fscanf(f, "%c", &aux);
printf("%c", aux);
```

Lendo Dados de um Arquivo Texto

- Quando um arquivo é aberto, um **indicador de posição** no arquivo é criado, e este recebe a posição do início do arquivo.
- Para cada dado lido do arquivo, este indicador de posição é automaticamente incrementado para o próximo dado não lido.
- Eventualmente o indicador de posição chega ao fim do arquivo:
  - A função **fscanf** devolve um valor especial, **EOF** (*End Of File*), caso tente-se ler dados e o indicador de posição está no fim do arquivo.

Lendo Dados de um Arquivo Texto

- Para ler todos os dados de um arquivo texto, basta usarmos um laço que será executado enquanto não chegarmos no fim do arquivo:

Lendo dados do arquivo `teste.txt`

```
char aux;
FILE *f = fopen ("teste.txt", "r");
while (fscanf(f, "%c", &aux) != EOF)
  printf("%c", aux);
fclose(f);
```

- O comando **fclose** (no fim do código) deve sempre ser usado para fechar um arquivo que foi aberto.
  - Quando escrevemos dados em um arquivo, este comando garante que os dados serão efetivamente escritos no arquivo.

Exemplo de programa que imprime o conteúdo de um arquivo texto na tela:

```c
#include <stdio.h>

int main() {
  FILE *arq;
  char aux, nomeArq[100];

  printf("Entre com nome do arquivo:");
  scanf("%s", nomeArq);
  arq =  fopen(nomeArq, "r");
  if (arq == NULL)
    printf("Erro ao abrir o arquivo");
  else{
    printf("------ Dados do arquivo:\n\n");
    while(fscanf(arq,"%c",&aux) != EOF){
      printf("%c",aux);
    }
  }
  fclose(arq);
}
```

Lendo Dados de um Arquivo Texto

- Notem que ao realizar a leitura de um caractere, automaticamente o indicador de posição do arquivo se move para o próximo caractere.
- Ao chegar no fim do arquivo a função **fscanf** retorna o valor especial **EOF**.
- Para voltar ao início do arquivo você pode fechá-lo e abrí-lo mais uma vez, ou usar o comando **rewind**.

```
while(fscanf(arq,"%c",&aux) != EOF){
  printf("%c",aux);
}

printf{"\n\n ------Imprimindo novamente\n\n");
rewind(arq);

while(fscanf(arq,"%c",&aux) != EOF){
  printf("%c",aux);
}
```

Escrevendo Dados em um Arquivo Texto

- Para escrever em um arquivo, ele deve ser aberto de forma apropriada, usando a opção **w**.
- Usamos a função **fprintf()**, semelhante à função **printf()**.
  - **int fprintf(ponteiro para arquivo, string de saída, variáveis)**
  - É semelhante ao **printf** mas notem que precisamos passar o ponteiro para o arquivo onde os dados serão escritos.

Copiando dois arquivos

```
FILE *fr = fopen ("teste.txt", "r"); //Abre arq. para leitura
FILE *fw = fopen ("saida.txt", "w"); //Abre arq. para escrita
while (fscanf(fr, "%c", &c) != EOF)
  fprintf(fw,"%c", c);
fclose(fr);
fclose(fw);
```

Escrevendo Dados em um Arquivo Texto

Exemplo de programa que faz copia de um arquivo texto.

```
int main() {
  FILE *arqIn, *arqOut;
  char aux, nomeArqIn[100], nomeArqOut[100];

  printf("Entre com nome do arquivo de entrada:");
  scanf("%s", nomeArqIn);
  arqIn = fopen(nomeArqIn, "r");
  if (arqIn == NULL){
    printf("Erro ao abrir o arquivo: %s\n",nomeArqIn);  return 1;
  }

  printf("Entre com nome do arquivo de saida:");
  scanf("%s", nomeArqOut);
  arqOut = fopen(nomeArqOut, "w");
  if (arqOut == NULL){
    printf("Erro ao abrir o arquivo: %s\n",nomeArqOut);  return 1;
  }

  while(fscanf(arqIn,"%c",&aux) != EOF){
    fprintf(arqOut,"%c",aux);
  }

  fclose(arqIn);
  fclose(arqOut);
}
```

fopen

Um pouco mais sobre a função **fopen()**.

```
FILE* fopen(const char *caminho, char *modo);
```

Modos de abertura de arquivo texto

modo	operações	indicador de posição começa
r	leitura	início do arquivo
r+	leitura e escrita	início do arquivo
w	escrita	início do arquivo
w+	escrita e leitura	início do arquivo
a	(append) escrita	final do arquivo

fopen

- Se um arquivo for aberto para leitura (**r**) e ele não existir, **fopen** devolve **NULL**.
- Se um arquivo for aberto para escrita ou escrita/leitura (**w ou w+**) e existir ele é apagado e criado;
  Se o arquivo não existir um novo arquivo é criado.
  - No modo **w** você poderá fazer apenas escritas e no modo **w+** você poderá fazer tanto escritas quanto leituras.
- Se um arquivo for aberto para leitura/escrita (**r+**) e existir ele NÃO é apagado;
  Se o arquivo não existir, **fopen** devolve **NULL**.

Exemplo: Lendo um texto na memória

- Podemos ler todo o texto de um arquivo para um vetor (deve ser grande o suficiente!) e fazer qualquer alteração que julgarmos necessária.
- O texto alterado pode então ser sobrescrito sobre o texto anterior.
- Como exemplo, vamos fazer um programa que troca toda ocorrência da letra "a" por "A" em um texto.

Lendo um texto na memória

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

int main(int argc, char *argv[]){
  FILE *arq;
  char nomeArq[100];

  printf("Digite o nome do arquivo: ");
  scanf("%s", nomeArq);
  arq = fopen(nomeArq, "r+");

  if(arq == NULL){
    printf("Arquivo inexistente!\n");
    return 1;
  }

  //Vamos determinar o tamanho do arquivo
  char aux;
  int size=0;
  while(fscanf(arq, "%c", &aux) != EOF)
    size++;

  //Aloca-se string com espaco suficiente para o arquivo
  char *texto = malloc((size+1)*sizeof(char));
   .....
}
```

Lendo um texto na memória

```
int main() {
  ......
  //Carrega o arquivo para a memória
  rewind(arq);
  int i=0;
  while(fscanf(arq, "%c", &aux) != EOF){
    texto[i] = aux;
    i++;
  }
  texto[i] = '\0';
  printf("%s", texto);  //Imprime texto original

  //Escreve o arquivo modificado
  rewind(arq);
  i=0;
  while(texto[i] != '\0'){
    if(texto[i] == 'a')
      texto[i] = 'A';
    fprintf(arq, "%c", texto[i]);
    i++;
  }
  free(texto);
  fclose(arq);
}
```

Resumo para se Trabalhar com Arquivos

- Crie um ponteiro para arquivo: **FILE *parq;**
- Abra o arquivo de modo apropriado associando-o a um ponteiro:
  - **parq = fopen(nomeArquivo, modo);** onde modo pode ser: **r, r+, w, w+**
- Leia dados do arquivo na memória.
  - **fscanf(parq, string-tipo-variavel, &variavel);**
  - Dados podem ser lidos enquanto **fscanf** não devolver **EOF**.
- Altere dados se necessário e escreva-os novamente em arquivo.
  - **fprintf(parq, string-tipo-variavel, variavel);**
- Todo arquivo aberto deve ser fechado.
  - **fclose(parq);**

Informações Extras: fscanf para **int**, **double**, etc.

- Você pode usar o **fscanf** como o **scanf** para ler dados em variáveis de outro tipo que não texto ou char.
  - Pode-se ler uma linha "1234" no arquivo texto para um **int** por exemplo:

```
int i;
fscanf(arq,"%d",&i);
```

- O mesmo vale para o **fprintf** em relação ao **printf**.
  - Neste exemplo é escrito o texto "56" no arquivo.

```
int i=56;
fprintf(arq,"%d",i);
```

- Você pode remover um arquivo usando a função **remove(string-nome-arq)**.