MC-102 — Aula 25
Arquivos em C

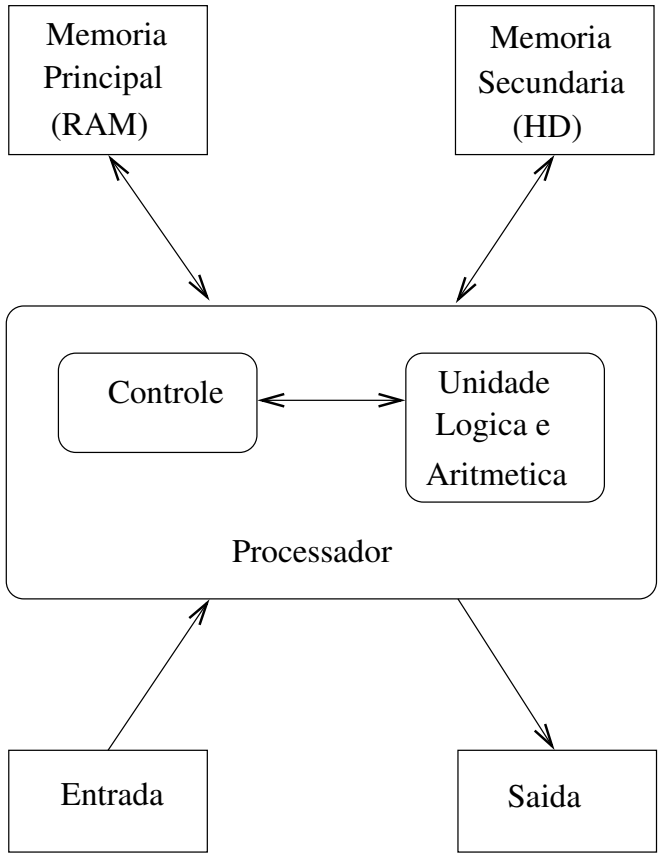
Instituto de Computação – Unicamp

19 de Outubro de 2012

Roteiro

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 secundaria (como Hard Disks) 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:

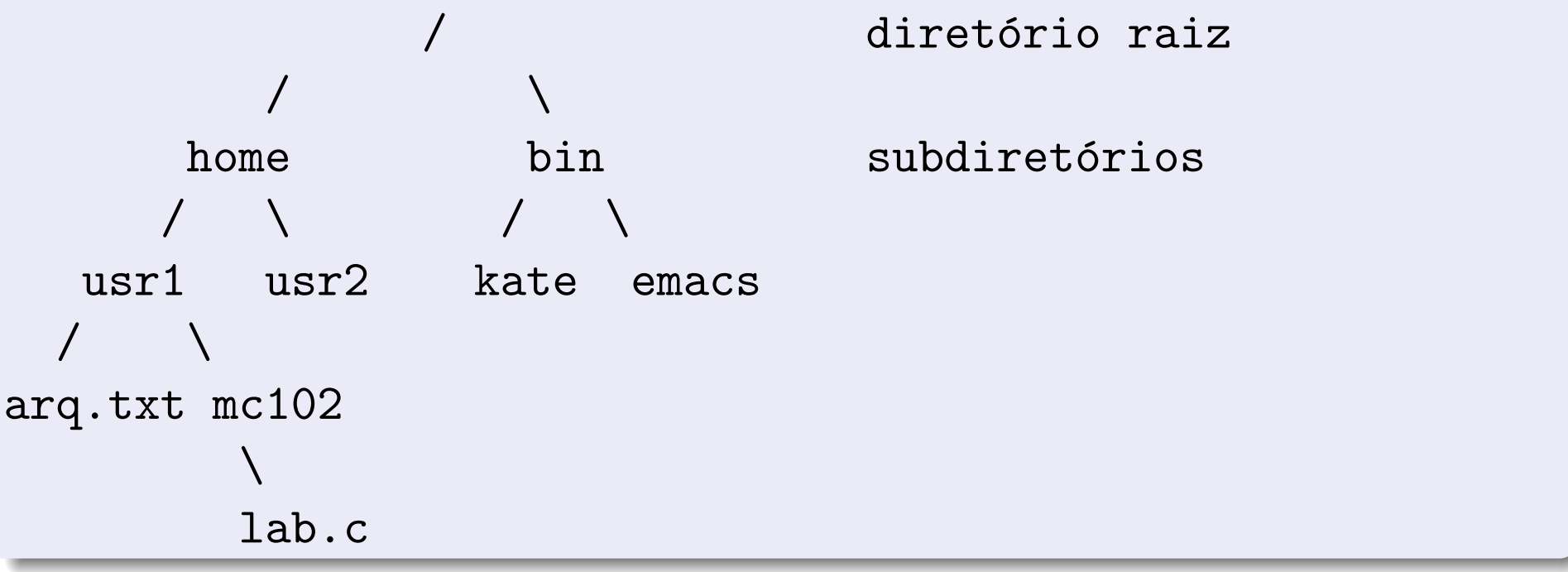
Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples. Exemplos: código fonte C, documento texto simples, páginas HTML.

Arquivo binário: Seqüência 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 ou relativos

O nome de um arquivo pode conter o seu diretório, ou seja, o caminho para encontrar este arquivo a partir da raiz. Os caminhos podem ser especificados 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 feito 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**, 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.

```c
#include <stdio.h>

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

  printf("Entre com nome do arquivo:");
  scanf("%s", nomeArq);
  arq = (FILE *) fopen(nomeArq, "r");
  if (arq == NULL)
    printf("Erro ao abrir o arquivo: teste.txt");
  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**.
- Note que para voltar ao início do arquivo novamente você pode fecha-lo e abri-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 a função `printf()`.
  - **int fprintf(ponteiro para arquivo, texto, 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");
FILE *fw = fopen ("saida.txt", "w");
while (fscanf(fr, "%c", &c) != EOF)
  fprintf(fw,"%c", c);
fclose(fr);
fclose(fw);
```

Escrevendo dados em 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 = (FILE *) fopen(nomeArqIn, "r");
  if (arqIn == NULL){
    printf("Erro ao abrir o arquivo: %s\n",nomeArqIn);  return 0;
  }

  printf("Entre com nome do arquivo de saida:");
  scanf("%s", nomeArqOut);
  arqOut = (FILE *) fopen(nomeArqOut, "w");
  if (arqOut == NULL){
    printf("Erro ao abrir o arquivo: %s\n",nomeArqOut);  return 0;
  }

  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 sobrescrito;
  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**.

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 necessário.
- 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

```
int main() {
  FILE *arq;
  char texto[1001], aux, nomeArqIn[100];
  int i;

  printf("Entre com nome do arquivo de entrada:");
  scanf("%s", nomeArqIn);
  arq = (FILE *) fopen(nomeArqIn, "r");
  if (arq == NULL){
    printf("Erro ao abrir o arquivo: %s\n",nomeArqIn); return 0;
  }

  for(i=0; i<1000 && fscanf(arq,"%c",&aux) != EOF; i++){
    texto[i]  = aux;
   }
  texto[i] = '\0';
  fclose(arq); //fechar para reabri-lo e então  sobreescrever

  //abre arquivo para escrita e o altera
  .....
 }
```

Lendo um texto na memória

```
int main() {

  ......
  //abre arquivo para escrita e o altera
  arq = (FILE *) fopen(nomeArqIn, "w");
  if (arq == NULL){
    printf("Erro ao abrir o arquivo: %s\n",nomeArqIn);
    return 0;
  }

  for(i=0; texto[i] != '\0'; i++){
    if(texto[i] == 'a')
      fprintf(arq,"%c", 'A');
    else
     fprintf(arq,"%c", texto[i]);
  }
  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);**

Outras Informações

- 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)**.