MC102 – Algoritmos e Programação de Computadores

Instituto de Computação

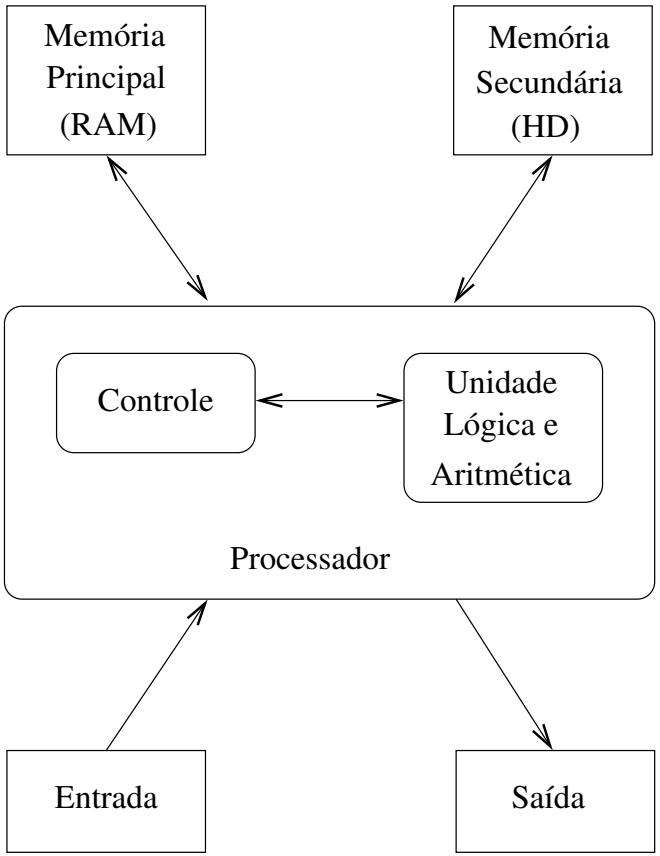
UNICAMP

Primeiro Semestre de 2016

Roteiro

Tipos de Memória

- Quando vimos a organização básica de um sistema computacional, mencionamos apenas um tipo de memória.
- Entretanto, na maioria dos sistemas, a memória é dividida em dois tipos: primária e secundária.



Tipos de Memória

- A memória principal utilizada na maioria dos computadores emprega uma tecnologia que requer alimentação constante de energia para que informações sejam preservadas.

Tipos de Memória

- A memória secundária utilizada na maioria dos computadores emprega uma tecnologia que não requer alimentação constante de energia para que informações sejam preservadas.

Tipos de Memória

- Todos os programas são executados na memória principal, tal que, quando um programa termina ou há interrupção de 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 primária.
  - É mais barata que a memória primária.
  - 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.

Exemplos de extensões:

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

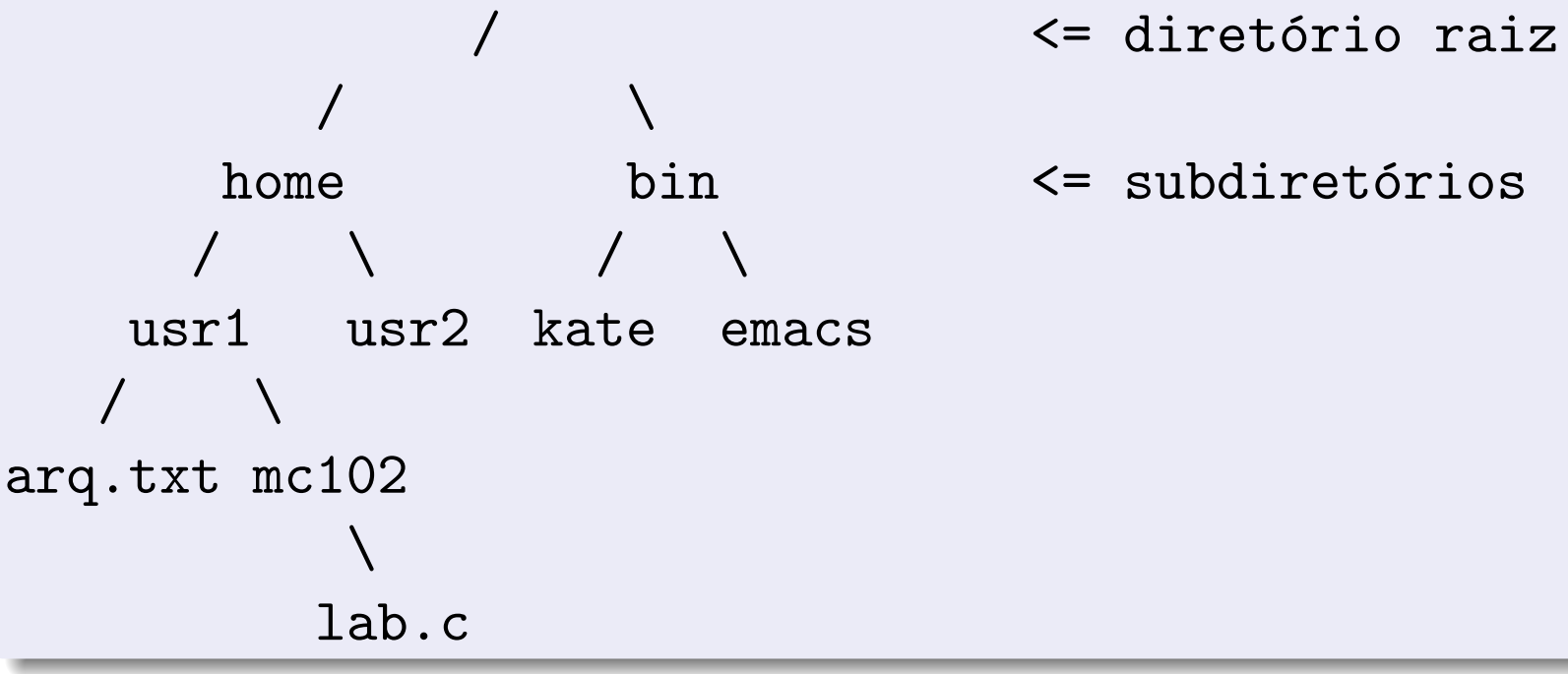
Tipos de arquivos

- Embora arquivos possam ter conteúdos bem distintos, há dois tipos principais de arquivos:
  - 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 do programa que a gerou, não legíveis diretamente. Exemplos:
    - ★ arquivos executáveis
    - ★ arquivos compactados
    - ★ documentos do Adobe Photoshop

Sistemas de arquivos

- Um sistema de arquivos é organizado em diretórios (também chamados de pastas).
- Um diretório pode conter arquivos e/ou outros diretórios.

Exemplo de 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.
- Caminhos podem ser especificados de duas formas:
  - Caminho absoluto: descrição do caminho desde o diretório raiz. Exemplos:

    `/bin/emacs`
    `/home/usr1/arq.txt`
  - Caminho relativo: descrição do caminho a partir do diretório corrente. Exemplos:

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

Abrindo um arquivo texto

- Para se trabalhar com arquivos em C, 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 *arq;
arq = fopen("teste.txt", "r");
```

- Neste exemplo, a variável ponteiro `arq` aponta para o arquivo "`teste.txt`".

Abrindo um arquivo texto

- Note que o comando `fopen` possui dois parâmetros:

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

- O primeiro parâmetro do comando `fopen` é uma string com o caminho de um arquivo:
  - Ou caminho absoluto, por exemplo, "`/user/joao/arq.txt`".
  - Ou caminho relativo, por exemplo, "`teste.txt`".
- O segundo parâmetro é uma string informando o tipo do arquivo e como ele deve ser aberto.
  - Arquivo texto ou binário.
  - Para leitura, gravação ou ambos.
- No nosso exemplo, o parâmetro "`r`" significa que abrimos um arquivo texto para leitura (veremos em breve outras formas de abrir arquivos).

Abrindo um arquivo texto

- Antes de acessar um arquivo, devemos abri-lo com a função `fopen()`.
- Em caso de sucesso, a função retorna um ponteiro para o arquivo aberto. Em caso de erro, a função retorna `NULL`.
- Exemplo:

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

if (arq == NULL)
  printf("Erro ao tentar abrir o arquivo.\n");
else
  printf("Arquivo aberto para leitura.\n");
```

Lendo dados de um arquivo texto

- Para ler dados de um arquivo (aberto de forma adequada), 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.
- Exemplo:

```
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 (a menos que o arquivo seja aberto como `append`, como veremos em breve).
- 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 se tente ler dados e o indicador de posição esteja 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 ao final do arquivo:
- Exemplo:

```
char aux;
FILE *f = fopen("teste.txt", "r");

while (fscanf(f, "%c", &aux) != EOF)
  printf("%c", aux);

fclose(f);
```

Lendo dados de um arquivo texto

- O comando `fclose` deve sempre ser usado para fechar um arquivo que foi aberto com sucesso, de forma que o arquivo possa ser posteriormente aberto por outros programas.
- Quando escrevemos dados em um arquivo, o comando `fclose` garante que os dados serão efetivamente escritos no arquivo.
- Cuidado: antes de fechar um arquivo, verificar se ele foi aberto com sucesso (ou seja, se o ponteiro para o arquivo não é nulo).

Lendo dados de um arquivo texto

```
#include <stdio.h>

int main() {
  FILE *arq;
  char aux, nomeArq[101];

  printf("Entre com nome do arquivo: ");
  scanf("%s", nomeArq);

  arq = fopen(nomeArq, "r");

  ...
```

Lendo dados de um arquivo texto

```
  ...

  if (arq == NULL)
    printf("Erro ao abrir o arquivo: %s\n", nomeArq);
  else {
    printf("----- Inicio do arquivo -----\n");

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

    printf("------ Fim do arquivo -------\n");

    fclose(arq);
  }

  return 0;
}
```

Lendo dados de um arquivo texto

- Note 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 o indicador de posição para o início do arquivo, podemos fechá-lo e abri-lo novamente ou usar o comando `rewind`.
- Exemplo:

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

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

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

Escrevendo dados em um arquivo texto

- Para escrever dados num arquivo (aberto de forma adequada), usamos a função `fprintf()`, que é semelhante à função `printf()`.

```
int fprintf(ponteiro_para_arquivo, string_de_formato, variáveis);
```

- A única diferença para o `printf` é que devemos passar como primeiro parâmetro um ponteiro para o arquivo para onde será feita a escrita.

Copiando dados entre arquivos

```
#include <stdio.h>

int main() {
  FILE *arqIn, *arqOut;
  char aux, nomeArqIn[101], nomeArqOut[101];

  printf("Entre com nome do arquivo de entrada: " );
  scanf("%s", nomeArqIn);

  arqIn = fopen(nomeArqIn, "r");

  if (arqIn == NULL) {
    printf("Erro ao abrir o arquivo de entrada: %s\n", nomeArqIn);
    return 0;
  }

  ...
```

Copiando dados entre arquivos

```
  ...

  printf("Entre com nome do arquivo de saida: ");
  scanf("%s", nomeArqOut);

  arqOut = fopen(nomeArqOut, "w");

  if (arqOut == NULL) {
    printf("Erro ao abrir o arquivo de saida: %s\n", nomeArqOut);
    fclose(arqIn);
    return 0;
  }

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

  fclose(arqIn);
  fclose(arqOut);

  return 0;
}
```

Modos de abertura de arquivos textos

- Como vimos anteriormente, a função `fopen()` é usada para abrir um arquivo:

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

- Um arquivo texto pode ser aberto de vários modos diferentes:

Modo	**Operações**	**Indicador de posição**
`r`	leitura	início do arquivo
`w`	escrita	início do arquivo
`r+`	leitura e escrita	início do arquivo
`w+`	escrita e leitura	início do arquivo
`a`	escrita/append	final do arquivo

Modos de abertura de arquivos textos

- Ao se tentar abrir um arquivo inexistente para leitura (`r`) ou leitura e escrita (`r+`), `fopen` retorna `NULL`. Caso o arquivo exista, o arquivo é aberto e seu conteúdo é preservado.
- Ao se tentar abrir um arquivo inexistente para escrita (`w`) ou escrita e leitura (`w+`), um novo arquivo é criado e então aberto pelo `fopen`. Caso o arquivo exista, seu conteúdo é primeiramente apagado e então aberto para escrita.
- Ao se tentar abrir um arquivo inexistente para escrita/append (`a`), um novo arquivo é criado e então aberto pelo `fopen`. Caso o arquivo exista, o arquivo é aberto e seu conteúdo é preservado.

Removendo um arquivo

- É possível remover um arquivo usando a função pré-definida:

```
int remove(char *caminho);
```

- Note que remover um arquivo é diferente de remover o conteúdo de um arquivo, ou seja, deixá-lo vazio.

Removendo um arquivo

```
#include <stdio.h>

int main() {
  char nomeArq[101];

  printf("Entre com nome do arquivo a ser removido: ");
  scanf("%s", nomeArq);

  if (remove(nomeArq) == 0)
    printf("Arquivo removido com sucesso.\n");
  else
    printf("Nao foi possivel remover o arquivo.\n");

  return 0;
}
```

Lendo um arquivo texto na memória

- Podemos ler todo o texto de um arquivo para um vetor (que deve ser grande o suficiente) e fazer qualquer alteração que julgarmos necessária.
- O texto alterado pode então ser sobrescrito no arquivo original.
- Como exemplo, vamos escrever um programa que troca todas as letras minúsculas de um arquivo pelas letras maiúsculas correspondentes.

Lendo um arquivo texto na memória

```
#include <stdio.h>

int main() {
  FILE *arq;
  char texto[1001], aux, nomeArq[101];
  int i;

  printf("Entre com nome do arquivo de entrada: ");
  scanf("%s", nomeArq);

  arq = fopen(nomeArq, "r+");

  if (arq == NULL) {
    printf("Erro ao abrir o arquivo '%s'.\n", nomeArq);
    return 0;
  }

  ...
```

Lendo um arquivo texto na memória

```
  ...

  /* Copia ate 1000 caracteres do arquivo de entrada */
  for (i = 0; i < 1000 && fscanf(arq, "%c", &aux) != EOF; i++)
    texto[i]  = aux;
  texto[i] = '\0';

  rewind(arq);

  /* Substitui letras minusculas por maiusculas */
  for (i = 0; texto[i]; i++)
    if (texto[i] >= 'a' && texto[i] <= 'z')
      fprintf(arq, "%c", texto[i] - 'a' + 'A');
    else
      fprintf(arq, "%c", texto[i]);

  fclose(arq);

  return 0;
}
```

Lendo e escrevendo outros tipos de valores

- Podemos usar o comando `fscanf`, assim como o `scanf`, para ler outros tipos de valores. Exemplos:

```
int i;
float f;
char s[81];

FILE *arq = fopen("teste.txt", "r+");

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

- Da mesma forma, podemos usar o comando `fprintf`, assim como o `printf`, para escrever outros tipos de valores. Exemplos:

```
fprintf(arq, "%d %f %s", 56, 3.1416,  "Uma mensagem simples");
```

Imprimindo o conteúdo de um arquivo

```
#include <stdio.h>

int main() {
  FILE *arq;
  char aux, nomeArq[101];

  printf("Entre com nome do arquivo: ");
  scanf("%s", nomeArq);

  arq = fopen(nomeArq, "r");

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

    fclose(arq);
  }

  return 0;
}
```

Imprimindo o conteúdo de um arquivo

```
#include <stdio.h>

int main() {
  FILE *arq;
  char aux, nomeArq[101];

  fprintf(stdout, "Entre com nome do arquivo: ");
  fscanf(stdin, "%s", nomeArq);

  arq = fopen(nomeArq, "r");

  if (arq != NULL) {
    while (fscanf(arq, "%c", &aux) != EOF)
      fprintf(stdout, "%c", aux);

    fclose(arq);
  }

  return 0;
}
```

O comando `fgets`

- Ao usar o comando `fscanf` para ler uma string, você deve garantir que foi alocada uma string de tamanho suficiente para armazenar todos os caracteres.
- Caso o programa leia mais caracteres do que o tamanho alocado, um erro ocorrerá durante a execução do programa.
- O comando `fscanf` não é adequado para ler strings contendo espaços em branco.
- Uma alternativa para ler strings é o comando `fgets()`:

```
char *fgets(char *str, int tamanho, FILE *arq);
```

- Sendo que, `str` é o nome da variável usada para armazenar a string, `tamanho` é um inteiro indicando até quantos caracteres devem ser lidos (serão lidos `tamanho-1` caracteres e um caractere extra será reservado para o caractere '\0') e `arq` é o ponteiro para o arquivo (previamente aberto).

Exemplo com fgets

```
#include <stdio.h>

int main() {
  char nomeArq[101], string[81];
  FILE *arq;
  int i = 0;

  printf("Entre com nome do arquivo a ser lido: ");
  scanf("%s", nomeArq);

  /* Abre arquivo para leitura */
  arq = fopen(nomeArq, "r");

  ...
```

Exemplo com fgets

```
  ...

  if (arq == NULL) {
    printf("Erro ao abrir o arquivo '%s' para leitura.\n",
           nomeArq);
    return 0;
  }

  /* Enquanto for possivel ler linhas do arquivo
     (limitadas a 80 caracteres) */
  while (fgets(string, 81, arq))
    printf("%3d: %s", ++i, string);

  fclose(arq);

  return 0;
}
```

Exercícios

- Escreva um programa que leia dois arquivos textos contendo números inteiros e ordenados, e escreva um único arquivo texto com os números ordenados de ambos os arquivos.
- Escreva um programa que leia uma série de números inteiros de um arquivo texto e escreva um arquivo texto contendo estes números ordenados.

  Importante: em ambos os casos, seu programa não deve usar um vetor auxiliar para armazenar os números.