Algoritmos e Programação de Computadores

Instituto de Computação

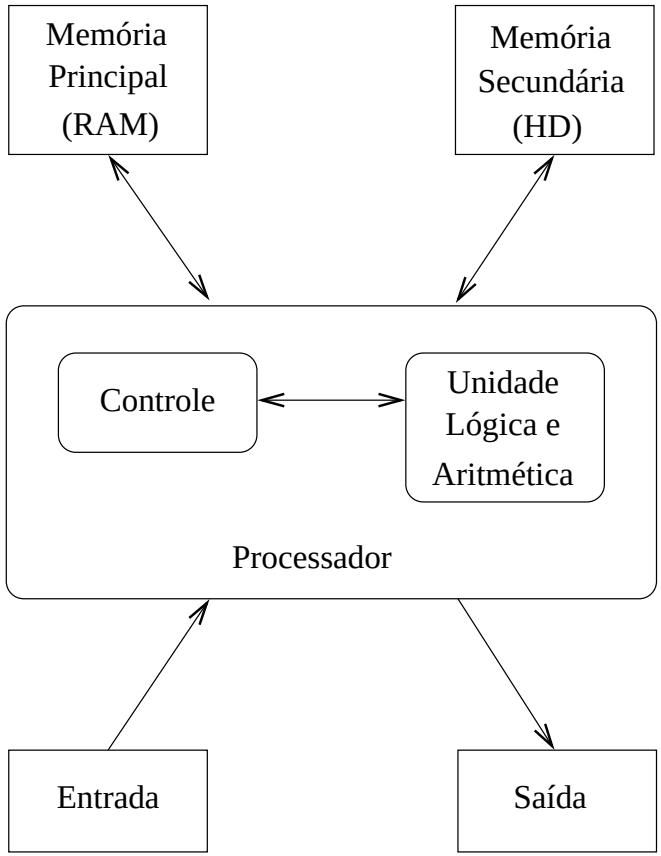
UNICAMP

Primeiro Semestre de 2013

Roteiro

Tipos de Memória

- Quando vimos a organização básica de um sistema computacional, havia somente 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 e, por isso, quando o 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.

Algumas extensões

arq.txt	arquivo texto simples
arq.c	código fonte em C
arq.pdf	*portable document format*
arq.html	arquivo para páginas Web *(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:

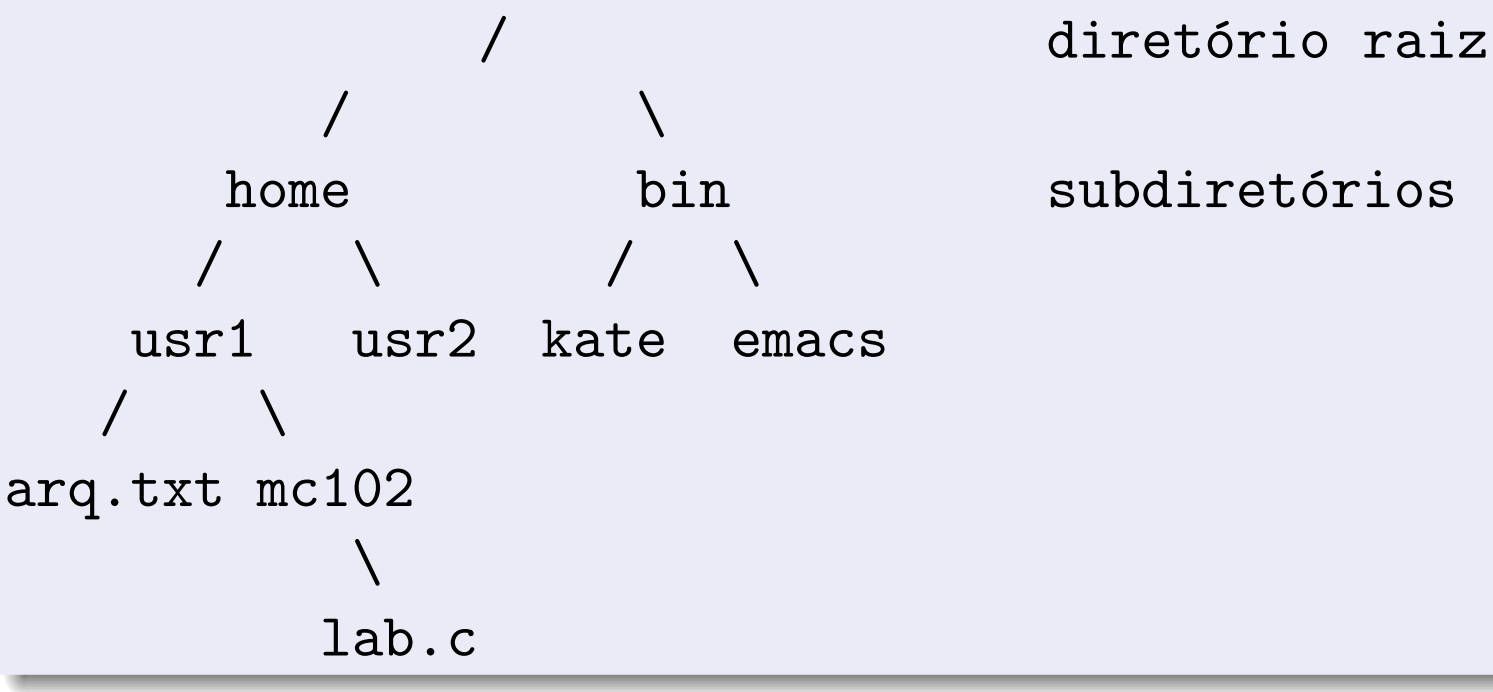
Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples. Exemplos: código fonte C, documento texto simples, páginas HTML.

Arquivo binário: Sequência de bits sujeita às convenções dos programas que o gerou, não legíveis diretamente. Exemplos: arquivos executáveis, arquivos compactados, documentos do Microsoft Word (*.doc) ou do Adobe Photoshop (*.psd).

Diretório

- Um diretório também é chamado de pasta.
- Um diretório pode conter 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 textos em C

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

- Neste exemplo, a variável ponteiro `arq1` aponta para o arquivo `teste.txt`.

Arquivos textos 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/joao/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 parâmetro `r` significa que abrimos um arquivo texto para leitura.

Abrindo um arquivo texto para leitura

- Antes de fazer acesso ao 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.\n");
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` caso se tente 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 ao final 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.
  - Cuidado: antes de fechar um arquivo, verificar se ele foi corretamente aberto.

```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: %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

- Notem que, ao realizar a leitura de um caractere, automaticamente, o ponteiro de leitura 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, podemos fechá-lo e abri-lo novamente ou usar o comando `rewind`.

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

printf{"----- Imprimindo novamente -----\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, texto, variáveis)`
  - É semelhante ao `printf`, porém, notem que precisamos passar o ponteiro para o arquivo onde os dados serão escritos.

Copiando dois arquivos

```
char c;
FILE *fr = fopen("teste.txt", "r");
FILE *fw = fopen("saida.txt", "w");
while (fscanf(fr, "%c", &c) != EOF)
  fprintf(fw, "%c", c);
/* pode ocorrer um erro se os arquivos
   nao foram abertos corretamente */
fclose(fr);
fclose(fw);
```

```c
#include <stdio.h>
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 de entrada: %s\n", nomeArqIn);
    return 0;
  }
  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); /* fecha o arquivo de entrada */
    return 0;
  }
  while (fscanf(arqIn, "%c", &aux) != EOF)
    fprintf(arqOut, "%c", aux);
  fclose(arqIn);
  fclose(arqOut);
  return 0;
}
```

Modos de abertura de arquivos

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

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

Modos de abertura de arquivo

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

- Se um arquivo for aberto para leitura (`r`) e ele não existir, `fopen` retorna `NULL`.
- Se um arquivo for aberto para leitura e escrita (`r+`) e ele existir, então seu conteúdo não é removido. Se o arquivo não existir, `fopen` retorna `NULL`.
- Se um arquivo for aberto para escrita (`w` ou `w+`) e ele existir, então seu conteúdo é primeiramente removido. Se o arquivo não existir, um novo arquivo é criado.
- Se um arquivo for aberto para *append* (`a`) e ele existir, então seu conteúdo não é removido. Se o arquivo não existir, um novo arquivo é criado.

Removendo um arquivo

- É possível remover um arquivo usando a função pré-definida: `int remove(char *nomeArquivo);`
- Note que remover o conteúdo de um arquivo, ou seja, deixá-lo vazio, é diferente de remover o arquivo em si.

```
#include <stdio.h>

int main() {
  char nomeArq[100];

  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 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 toda ocorrência da letra “a” por “A” em um arquivo texto.

Lendo um texto na memória

```
#include <stdio.h>

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

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

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

  for (i = 0; i < 1000 && fscanf(arq, "%c", &aux) != EOF; i++)
    texto[i]  = aux;

  texto[i] = '\0';
  fclose(arq);
  ...
```

Lendo um texto na memória

```
  ...
  /* abre arquivo para escrita */
  arq = fopen(nomeArq, "w");
  if (arq == NULL) {
    printf("Erro ao abrir o arquivo '%s' para escrita.\n", nomeArq);
    return 0;
  }

  for (i = 0; texto[i]; i++)
    if (texto[i] == 'a')
      fprintf(arq, "%c", '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 (`int`, `float`, `string`, etc). Exemplos:

```
int i;
float f;
char s[80];
fscanf(arq, "%d", &i);
fscanf(arq, "%f", &f);
fscanf(arq, "%s", s);
```

- Da mesma forma, podemos usar o comando `fprintf`, assim como o `printf`, para escrever outros tipos de valores (`int`, `float`, `string`, etc). Exemplos:

```
fprintf(arq, "%d", 56);
fprintf(arq, "%f", 3.1416);
fprintf(arq, "%s", "um teste simples");
```

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);
```

onde `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 '\0') e `arq` é o ponteiro para o arquivo (previamente aberto).

Exemplo com fgets

```
#include <stdio.h>

int main() {
  char nomeArq[100], 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");
  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

Intercalação

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.

Ordenação

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.