Engenharia de Computação

Fundamentos de Computação

Aula 14 – Arquivos (binários, texto, fputc, fgetc)

Prof. Fernando Barreto
informatica-ap@utfpr.edu.br

Arquivos

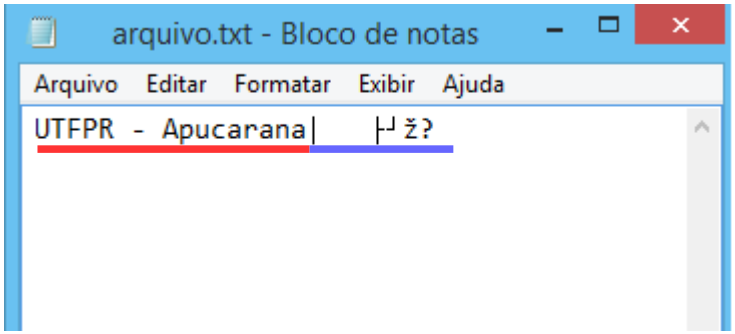
- Coleção de bytes armazenados em um dispositivo de armazenamento secundário (discos, pendrives, cd...)
  - Armazenamento durável e de grande quantidade
- Coleção de bytes é interpretada de acordo com a organização/operação gerada por um programa ao processar o arquivo (ler/escrever)

Arquivos

- Tipos:
  - Texto: armazena caracteres que podem ser mostrados na tela ou modificados por um editor de texto
    - Arquivos maiores (**convertidos** em alguma tabela, geralmente ASCII - necessitam de 8 bits por símbolo). Um exemplo de tabela está em http://pt.wikipedia.org/wiki/ASCII
      - Exemplo: O inteiro 12345678 é representado com 32 bits. Ao representar cada dígito com 8 bits, será necessário 64 bits no arquivo texto...
    - Necessitam de 1 ou mais caracteres especiais dependendo do sistema operacional.
      - Exemplo: Uma nova linha pode utilizar **\n** ou **\r\n** dependendo do sistema...

Arquivos

- Tipos:
  - Binário: armazenados da mesma forma como estão organizados na memória
    - Não tem conversão de dados
      - Exemplo: O inteiro 12345678 é representado em 32 bits. Ao gravar no arquivo, copia-se diretamente sem conversão, ocupando 32 bits.
  - Exemplo de um arquivo com texto e binário junto...



Texto

Binário (ilegível em texto...)

Arquivos em C

- Biblioteca **stdio.h**
  - Funções para manipulação de arquivos
    - Abrir/Fechar e Ler/Escrever bytes ou caracteres
- Programador deve organizar como será feito a manipulação de arquivos
- Ponteiro especial para uma área da memória com informações que representam o arquivo em uso

  FILE *ponteiro_arquivo;
- Para leitura/escrita do/no arquivo, usa-se geralmente um ponteiro para um "buffer" (área de memória onde se armazenará/lerá os dados)

Arquivos

- Abrir um arquivo com função **fopen( )**

  FILE *fopen(*char *nome_arquivo, char *modo_uso***)**

  – Pode retornar:
    - Um ponteiro do tipo FILE para o arquivo aberto.
    - NULL, indicando erro. Dica: usar *exit( )* , para abortar o programa fechando qualquer arquivo aberto do programa

  – Parâmetros:
    - *char *nome_arquivo* aponta para uma string com o caminho do arquivo (pode ser desde a raiz do sistema de arquivos ou apenas o nome do arquivo, se o mesmo estiver na pasta de onde o programa é executado).
      – Ex: "/media/Rede/FD61A/ProjetoX/arquivo.txt" ou "arquivo.txt", se o arquivo.txt estiver na pasta do projeto da sua IDE de nome *ProjetoX*.
    - *char *modo_uso* é uma string com código, veja próximo slide

Arquivos

- Modos possíveis de abrir arquivo

Modo	Arquivo	Função
"r"	Texto	Leitura. Arquivo deve existir.
"w"	Texto	Escrita. Cria arquivo, se não houver. Apaga o anterior, se ele existir.
"a"	Texto	Escrita. Os dados serão adicionados no fim do arquivo ("append").
"rb"	Binário	Leitura. Arquivo deve existir.
"wb"	Binário	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"ab"	Binário	Escrita. Os dados serão adicionados no fim do arquivo ("append").
"r+"	Texto	Leitura/escrita. O arquivo deve existir e pode ser modificado.
"w+"	Texto	Leitura/escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a+"	Texto	Leitura/escrita. Os dados serão adicionados no fim do arquivo ("append").
"r+b"	Binário	Leitura/escrita. O arquivo deve existir e pode ser modificado.
"w+b"	Binário	Leitura/escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a+b"	Binário	Leitura/escrita. Os dados serão adicionados no fim do arquivo ("append").

Fonte: Backes, A. "Linguagem C completa e Descomplicada"

Arquivos

- Fechar um arquivo com função **fclose( )**

int fclose(*FILE *ponteiro_arquivo*)

  - Retorna 0, indicando sucesso
  - Retorna != 0, indicando erro
- "Fechar um arquivo" tem o objetivo de gravar toda informação pendente no "buffer".
  - OBS: Apenas quando o "buffer" está aproximadamente cheio que o mesmo é gravado realmente no arquivo.
- **Boa prática**: sempre fechar o arquivo quando não for utilizar mais.
  - Pode-se forçar gravar o conteúdo do "buffer" de saída do hardware a partir da função:

int fflush(*FILE *ponteiro_arquivo)*

Arquivos


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

```
//início trecho de código....
FILE *meu_arquivo=NULL;
meu_arquivo = fopen("arquivo.txt","w");     //ATENÇÃO ao parâmetro modo_uso !!!
if (meu_arquivo == NULL) {
        printf("Erro ao abrir o arquivo");
        exit(1); //stdlib.h , aborta o programa e fecha qualquer arquivo aberto automaticamente
}
//fim trecho de código....
```

```
//início trecho de código
if (fclose(meu_arquivo) != 0) {
        printf("Erro ao fechar o arquivo");
        exit(1);
}
//fim trecho de código...
```

Arquivo Texto: Operações

- Abrir arquivo com modo texto: "r" ou "w"
- Escrever 1 caracter no arquivo aberto

int fputc(*int caracter, FILE *ponteiro_arquivo***)**

  - Retorna:
    - EOF, se houver erro de escrita. EOF significa *End of File*
  - Parâmetros:
    - *int caracter* é um unsigned char para ser escrito no arquivo
    - *FILE *ponteiro_arquivo* é o ponteiro para o arquivo aberto
- Quebra de linha no arquivo texto ?
  - Com o fputc( ), basta inserir um '\n'
    - OBS: dependo do sistema operacional é necessário inserir '\r' e depois '\n'

Arquivo Texto: Operações

- Ler 1 caracter do arquivo aberto

int fgetc(*FILE *ponteiro_arquivo*)

  - Retorna:
    - EOF, se houver erro de leitura ou final do arquivo atingido
    - Um código int, que pode ser convertido em unsigned char (0-255) que é a representação em ASCII
      - » **Boa prática**: uso da função **feof**(*FILE *ponteiro_arquivo*) como auxiliar na detecção de final do arquivo atingido pelo fgetc(). A função feof() retorna != 0 caso tenha atingido o final do arquivo...
  - Parâmetros:
    - *FILE *ponteiro_arquivo* é o ponteiro para o arquivo aberto

Arquivos

- Observações:
  - O retorno das funções que lidam com arquivos geralmente informam erros que possam ocorrer. No entanto, é recomendável (não obrigatório) tratá-los o que torna o programa mais robusto
  - Um arquivo possui um indicador de posição atual onde irá ler/gravar
    - É atualizado a cada operação de leitura/escrita
    - Avança para uma próxima posição a ser lida/escrita, de acordo com o tamanho de dados lido/escrito
      - Ex: ao ler um caracter do arquivo, lê-se 1 byte, e o indicador de posição avança esse tamanho lido em bytes (1 byte). Se ler 1 inteiro, avança-se 4 bytes. Se ler uma struct molde_x, avança-se 1 sizeof(struct molde_x) bytes..

Arquivos: Operações

- Mover o indicador de posição leitura/escrita do arquivo

  int fseek(*FILE *ponteiro_arquivo, **long** numbytes, **int** origem***)**

  - *numbytes*, total de bytes a partir da *origem* a ser deslocado
    - Pode ser positivo ou negativo dependendo da *origem*
  - *origem*, a partir de onde haverá o deslocamento:
    - SEEK_SET indica início do arquivo
    - SEEK_CUR indica local atual
    - SEEK_END indica final do arquivo
  - Retorna != Zero se ocorreu erro na movimentação
- Saber se uma leitura recente chegou ao final do arquivo

  int feof(*FILE *ponteiro_arquivo***)**

  - Retorna != Zero indicando que já atingiu o final do arquivo

Exercícios

- A partir das funções de arquivo vistas em aula:
  - 1_1) Faça uma função que receba a string abaixo por parâmetro e crie um arquivo que armazenará essa string inserindo '-' entre os chars da string. Se houver ' ' substitua por '#'

    String: "Universidade Tecnologica Federal do Parana"

    Arquivo: "U-n-i-v-e-r-s-i-d-a-d-e-#-T-e-c-n-o-l-o-g-i-c-a-#-F-e-d-e-r-a-l-#-d-o-#-P-a-r-a-n-a"
  - 1_2) Faça outra função que receba por parâmetro uma string, leia o arquivo gerado na função anterior para preencher essa string, substituindo '#' por ' '. Mostre a string final na tela.

    "U-n-i-v-e-r-s-i-d-a-d-e- -T-e-c-n-o-l-o-g-i-c-a- -F-e-d-e-r-a-l- -d-o- -P-a-r-a-n-a"
  - 1_3) Faca outra função parecida com a anterior, mas que gere uma string a partir de 26 bytes de deslocamento da origem do arquivo. "T-e-c-n-o-l-o-g-i-c-a- -F-e-d-e-r-a-l- -d-o- -P-a-r-a-n-a"