Introdução à Programação C
CMT012
Aula 13

Ronald Souza
Instituto de Computação - UFRJ
ronaldsouza@dcc.ufrj.br

Conteúdo de hoje

Arquivos

⟶ Conceito de arquivos
⟶ Definir e diferenciar dados **transientes** e dados **persistentes**
⟶ Definir e diferenciar **arquivos de texto** e **arquivos binários**
⟶ Exemplos de **operações** com arquivos
⟶ Exemplos de funções e programas que usam arquivos

Arquivos

Os arquivos são a forma fundamental de armazenamento de informação na memória não volátil do computador.

⟶ Dados armazenados em arquivos **persistem** no computador **mesmo que seu programa de origem seja finalizado ou o computador seja desligado**.

Dados transientes
X
dados persistentes

→ **Variáveis** são o recurso pelo qual programadores manipulam **dados** ao longo da execução de seus programas.

Internamente, variáveis indicam células da memória principal (RAM).

→ Dados armazenados em variáveis são **transientes** pois não são preservados ao fim da execução do programa.

Dados transientes
X
dados persistentes

⟶ Os computadores teriam utilidade muito reduzida se a informações (i.e. os **dados**) só pudessem ser preservadas durante a execução de seus programas de origem.

⟶ Por isso, existem **dados persistentes**, os quais podem ser lidos e alterados independentemente do programa que os criou estar em execução (processo na memória principal).

⟶ **Dados são persistentes quando residem em algum dispositivo de memória secundária**, como memória flash (ex.: SSD, pen drive), discos rígidos, CDs, etc.

Dados persistentes - Arquivos

Um **arquivo** é, essencialmente, um conjunto de dados persistentes.
⟶ Uma sequência de bytes unicamente identificada e que reside em um dispositivo de armazenamento secundário.

⟶ Por meio de arquivos, dados são **persistidos,** podendo vir a ser acessados e processados no futuro, tanto pelo programa que os criou quanto por outros programas.

Tipos de Arquivo

Arquivos podem ser classificados em dois tipos: **arquivos de texto** e **arquivos binários**.

⟶ O programador deve escolher um desses tipos a cada novo arquivo criado.

⟶ A escolha terá impactos fundamentais na **manipulação do arquivo** (funções de leitura e escrita) e armazenagem.

Arquivos de texto

Um arquivo texto é uma sequência de caracteres

⟶ Se um arquivo texto for aberto num editor de texto convencional, o usuário poderá ler seu conteúdo e modificá-lo de uma forma humanamente interpretável.

Exemplo: se escrevermos em um arquivo texto o valor de uma variável **int** que guarda o número **43** teremos dois **caracteres** gravados: o ’4’ e , em seguida, o ‘3’

Arquivos binários

Um arquivo binário é uma sequência de bytes **sem tradução para caracteres**.

⟶ Por arquivos binários, variáveis na memória principal podem ter seus dados persistidos com correspondência de “um para um” (mais detalhes a seguir).

Exemplo

Como armazenar a estrutura abaixo em um arquivo?

```
struct pessoa {
      char nome [50];
      int idade;
      float salario;
};
```

Exemplo

Usando arquivo texto:

→ Pode-se guardar a estrutura num arquivo texto, escrevendo-se o nome da pessoa, sua idade e finalmente seu salário.

→ Sabendo-se o formato aplicado na escrita, o programador poderá criar um programa que lê os dados.

Exemplo

Usando arquivo binário:

⟶ **Toda a estrutura pode ser tratada como um único dado**
⟶ Os dados da variável podem ser diretamente armazenados no arquivo ou lidos do arquivo para a memória principal.

A estrutura ocupará 58 bytes (supondo arquitetura onde int e float ocupam 4 bytes cada e char ocupa 1 byte)

Arquivo texto X arquivo binário

Se **não** houver bons motivos para usar arquivos texto (como por exemplo, permitir que o usuário os altere facilmente), usar arquivos binários.

Arquivo texto

⟶ leitura /escrita de **um único campo da estrutura por vez**.

⟶ valores **sempre armazenados como string** (leitura do arquivo exigirá nova tradução de volta ao tipo original após ser lido (ex., de “43” para 43).

Arquivo binário

⟶ **persiste toda estrutura de uma só vez** e com os **tipos das variáveis originais**!

Manipulação de arquivos

Para serem utilizados nos programas, os arquivos são tratados como variáveis

→ Devem, portanto, ser declarados

Na linguagem C, o tipo FILE (biblioteca stdio.h) é usado para manipular arquivos.

→ um ponteiro para FILE deve ser declarado sempre que uma variável arquivo for utilizada no programa.

→ essa variável armazenará meta-informações sobre o arquivo.

FILE *meuArquivo;

Manipulação de arquivos

Antes de fazer qualquer operação sobre o arquivo, a função **fopen()** (abre um arquivo) deve ser chamada:

FILE *fopen (const char *parq, const char *modo)

```
//abrindo o arquivo para leitura
meuArquivo = fopen("meusDados.txt", "r");
```

Manipulação de arquivos

Depois de concluir as operações sobre o arquivo, a função **fclose()** (fecha um arquivo) deve ser chamada:

int fclose (FILE *parq)

//fechando o arquivo aberto para leitura
fclose(meuArquivo);

Modos de operação sobre arquivos texto

“r”: abre o arquivo para leitura, o arquivo deve existir ou um erro ocorre

“w”: cria um arquivo vazio para escrita, caso um arquivo com o mesmo nome exista o seu conteúdo é apagado

“a”: adiciona ao final de um arquivo (o arquivo é criado caso ele não exista)

Exemplo:

```
//adiciona (escrita) no final do arquivo
meuArquivo = fopen("meusDados.txt", "a");
```

Modos de operação combinados

“r+”: abre um arquivo para leitura e escrita (o arquivo deve existir ou um erro ocorre)

“w+”: cria um arquivo vazio para leitura e escrita (se um arquivo com o mesmo nome existe o conteúdo é apagado)

“a+”: abre um arquivo para leitura e adição (todas as operações de escrita são feitas no final do arquivo, o arquivo é criado caso não exista)

Exemplo:

```
//cria um arquivo para leitura e escrita:
meuArquivo = fopen("meusDados.txt", "w+");
```

Exemplo de inicialização de arquivo texto

```
FILE *pa ; // declaracao do ponteiro para arquivo
pa = fopen ("arquivo.txt" , "w");
if (pa == NULL) { // verifica erro na abertura
    printf ("Arquivo nao pode ser aberto.");
    return -1;
}
```

Escrita em arquivos texto formatados

A função **fprintf()** é similar a printf(), com o adicional de receber um **ponteiro para arquivo** como primeiro argumento:

```
int fprintf(FILE *parq, const char *formatacao, ...);
```

Exemplo:

```
int vet[5] = {1, 2, 3 , 4, 5};
FILE *meuArquivo = fopen("meusDados.txt", "w");
(...)
for(i=0; i<5; i++) {
    fprintf(meuArquivo, "%d ", vet[i]);
}
(...)
```

Leitura em arquivos texto formatados

A função **fscanf()** é similar a scanf(), com o adicional de receber um **ponteiro para arquivo** como primeiro argumento:

```
int fscanf(FILE *parq, const char *formatacao, ...);
```

Exemplo:

```
int vet[5]; //Declarado, mas não inicializado!
FILE *meuArquivo = fopen("meusDados.txt", "r");
(...)
for(i=0; i<5; i++) {
    fscanf(meuArquivo, "%d", &vet[i]);
}
(...)
```

Exemplo

1) Escreva um programa em C que crie e inicialize três variáveis (uma inteiro, outra double e outra cadeia de caracteres) e **salve em um arquivo texto** os valores dessas variáveis.

2) Agora escreva outro programa que abra e **leia o conteúdo do arquivo** de texto gerado e imprima na tela os valores lidos.

Exemplo de solução ex. 1:

```
include <stdio.h>
#include <string.h>
#define TAM_STR 50
int main () {
    int aux1 = 10;
    char str[TAM_STR];
    double aux2 = 547.1;
    FILE* arq;
    strcpy(str, "ola mundo!");
    if (!(arq = fopen ("meu_texto.txt", "w"))) {
        printf ("Erro na abertura de arquivo!\n");
        return 0;
    }
    fprintf (arq, "%d\n%s\n%lf\n", aux1, str, aux2);
    fclose(arq);
    return 0;}
```

Exemplo de solução ex. 2:

```
#include <stdio.h>
#define NOME_ARQUIVO "meu_texto.txt"
#define TAM_STR 50
int main () {
    int inteiro;
    double rac;
    char str[TAM_STR];
    FILE* arq;
    if (!(arq = fopen (NOME_ARQUIVO, "r"))) {
        printf ("Erro na abertura de arquivo!\n");
        return -1;
    }
    fscanf (arq, "%d %[^\n] %lf", &inteiro, str, &rac);
    fclose (arq);
    printf("%d, %s, %lf\n", inteiro, str, rac);
    return 0;}
```

Lendo e escrevendo arquivos binários

As funções **fread()** e **fwrite()** são usadas para leitura e escrita de dados em modo binário

→ fread e fwrite retornam o número de itens lidos ou escritos com sucesso, ou zero caso um erro ocorra.

Lendo arquivos binários

size t fread (void* buffer, size t num bytes, size t count, FILE* fp);

→ **buffer** é um apontador para uma região de memória que receberá o valor lido do arquivo
→ **num bytes** é o número de bytes do valor que será lido
→ **count** indica quantas variáveis do tamanho ***num bytes*** deverão ser lidas
→ **fp** é um apontador para o arquivo de onde serão lidos os valores
→ **fread** retorna o número de variáveis lidas com sucesso, ou zero caso um erro ocorra

Exemplo de uso de *fread()*

```
int main() {
	int inteiro; float real; char carac;
	if (!(arq = fopen ("arquivo_bin.dat", "rb"))) {
		return -1;
	}
	fread (&inteiro, sizeof(int), 1, arq);
	fread (&real, sizeof(float), 1, arq);
	fread (&carac, sizeof(char), 1, arq);
}
```

Escrevendo arquivos binários

size t fwrite (void* buffer, size t num bytes, size t count, FILE* fp);

→ **buffer** é um apontador para a variável que será escrita no arquivo
→ **num bytes** é o número de bytes o tipo da variável a ser escrita
→ **count** indica quantas variáveis do tamanho ***num bytes*** serão escritas
→ **fp** é um apontador para o arquivo onde serão escritos os valores
→ **fwrite** retorna o número de variáveis escritas com sucesso, ou zero caso um erro ocorra

Exemplo de uso de *fwrite()*

```
typedef struct {
    char nome [50];
    int idade; char sexo;
} tPessoa;
int main (){
    FILE* arq; tPessoa p;
    strcpy (p.nome, "Joao Silva");
    p.idade = 46; p.sexo = ‘M’;
    if (!(arq = fopen ("arquivo_bin.dat", "wb"))) {
        return -1;
    }
    fwrite (&p, sizeof(tPessoa), 1, arq);
    fclose (arq);
    return 0;
}
```

Exemplo de *fwrite() e fread()*

```
int inum =10; float fnum =2.5; FILE * pa;
if ((pa = fopen ("saida.dat", "wb+")) == NULL ) {
    puts ("erro fopen"); return -1; }
//escreve no arquivo
fwrite (&inum, sizeof(int), 1, pa);
fwrite (&fnum, sizeof(float), 1, pa);
//volta ao inicio
rewind (pa);
//le do arquivo
fread (&inum, sizeof (int), 1, pa);
fread (&fnum, sizeof (float), 1, pa);
printf ("%d, %f\n", inum, fnum);
fclose (pa);
```

Por hoje é isso!

Slides baseados no material de Silvana Rossetto.