Arquivos e Biblioteca

Roteiro

Roteiro

Arquivos

- Até agora, todos os programas feitos nesta disciplina lêem dados do teclado (entrada padrão) e escrevem na tela do monitor (saída padrão);
- As vezes não é suficiente para um programa usar somente a entrada e saída padrão;
- O armazenamento de dados em variáveis e arrays é temporário e de tamanho limitado;
- Arquivos são usados para armazenamento permanente de grandes quantidades de dados (e programas) em dispositivos de armazenamento secundário, como discos.

Arquivos

- Um arquivo (= file) é uma sequência de bytes que reside na memória secundária (ex., HD);
- Para que um programa possa manipular um arquivo, é preciso associá-lo a uma variável do tipo **FILE**;
- A variável do tipo **FILE** aponta para uma estrutura que contém informações de sistema sobre o arquivo;
- O tipo FILE é predefinido em stdio.h

```
1 FILE *fp;
2
```

Roteiro

Operações sobre arquivos

- Uma variável FILE estará associada a um arquivo;
- Arquivo de texto sem formatação: "texto puro", sem dados de formatação
- Associando/abrindo um arquivo:
  - Função fopen

```
1 FILE *fp1, *fp2;
2 fp1 = fopen("data.txt", "r");
3 fp2 = fopen("imagem.pgm", "w");
4
```

  - O primeiro argumento da função é o nome do arquivo;
  - O segundo argumento é modo que será aberto o arquivo:
    - "r": somente leitura (do início do arquivo);
    - "r+": leitura e escrita (do início do arquivo);
    - "w": escrita, apagando o conteúdo do arquivo ou criando (do início do arquivo);
    - "w+": leitura e escrita, apagando o conteúdo do arquivo ou criando (do início do arquivo);
    - "a": escrita, criando se não existir, posicionando no fim do arquivo;
    - "a+": leitura e escrita, criando se não existir, posicionando no fim do arquivo;

Operações sobre arquivos

- A função fopen devolve o endereço de um FILE (ou NULL, se não encontrar o arquivo especificado);

```
1 FILE *fp;
2 fp = fopen("data.txt", "r");
3 if(fp==NULL)
4     printf("Erro ao abrir o arquivo\n");
5
```

- Observação:
  - O teclado é o arquivo padrão de entrada (= standard input);
  - Ele está permanente aberto e é representado pela constante **stdin**;
  - A saída padrão também é um arquivo e está associado à constante **stdout**.

Roteiro

Processando arquivo de texto (caracteres)

- Depois que um arquivo de texto é aberto, existem 3 formas diferentes de ler ou escrever sequencialmente os dados:
  - um caracter por vez, usando as funções da biblioteca padrão fgetc() e fputc();
  - uma linha por vez, usando fgets() e fputs();
  - em um formato específico, usando fscanf() e fprintf();
- Vamos nos limitar ao uso do fscanf e fprintf;

Roteiro

Processando arquivo de texto (caracteres)

- A função fscanf() lê o conteúdo do arquivo representado por fp;
- É preciso informar:
  - Qual arquivo será lido;
  - Quais tipos de dados serão lidos através dos especificadores de formato: %d, %c, %f, %s;
  - Em quais variáveis os dados lidos serão gravados.

```
1 FILE *fp;
2 fp = fopen("data.txt", "r");
3 int i, r;
4 char s[11], f[100];
5 if(fp!=NULL)
6 {
7     r = fscanf(fp, "%d %s", &i, s);
8     r = fscanf(fp, "%99[^\n]", f);
9 }
```

- Valor retornado:
  - EOF se o final do arquivo foi atingido ou em caso de erro;
  - Em caso de sucesso: a quantidade de itens convertidos e atribuídos.

```
1 while(fscanf(fp, "%d %s", &i, s) != EOF)
2     printf("%d %s\n", i, s);
```

- fscanf (stdin, ...) equivale a scanf(...)

Roteiro

Processando arquivo de texto (caracteres)

- A função fprintf() escreve no arquivo conectado ao fp;
- É preciso informar:
  - Em qual arquivo será gravado;
  - O conteúdo que será gravado;
- Ou é preciso informar:
  - Em qual arquivo será gravado;
  - Quais tipos de dados serão através dos especificadores de formato: %d, %c, %f, %s;
  - Em quais variáveis estão os dados que serão gravados.

```
1 FILE *t;
2 t = fopen("teste", "w");
3 fprintf(t, "Ola mundo\n");
4
5 int a=10;
6 char b[]="lala";
7 fprintf(t, "%d %s\n", a, b);
```

- O valor de retorno é o número de caracteres escritos, ou negativo em caso de ocorrência de erros.
- fprintf(stdout, ...) equivale ao printf(...)

Roteiro

Operações sobre arquivos

- Fechando um arquivo:
  - Função fclose

```
1 FILE *fp;
2 fp = fopen("data.txt", "r");
3 ...
4 fclose(fp);
```

  - O protótipo da função fopen() é:

```
1 int fopen(FILE *);
```

  - Retorna 0 no sucesso e EOF no erro:

```
1 FILE *fp;
2 fp = fopen("data.txt", "r");
3 ...
4 if(fclose(fp)==EOF)
5     printf("Erro ao fechar o arquivo\n");
```

Exemplo

- Abrir um arquivo chamado “hino.txt”;
- Criar um novo arquivo chamado hino_copia.txt;
- Copiar o conteúdo do arquivo hino.txt para o hino_copia.txt;
- Linha por linha;

Exemplo

```
1  char linha[100];
2  FILE *origem, *destino;
3
4  origem = fopen("hino.txt", "r");
5  destino = fopen("hino_copia.txt", "w");
6
7  if( origem!=NULL && destino!=NULL )
8  {
9      while(fscanf(origem, " %[^\n]", linha) != EOF)
10     {
11         fprintf(destino, "%s\n", linha);
12     }
13 }
14 else
15 {
16     printf("Arquivo nao encontrado\n");
17 }
18 fclose(origem);
19 fclose(destino);
```

Exemplo

- Abrir um arquivo chamado “hino.txt”;
- Criar um novo arquivo chamado hino_copia.txt;
- Copiar o conteúdo do arquivo hino.txt para o hino_copia.txt;
- Caractere por caractere;

Exemplo

```c
char c;
FILE *origem, *destino;

origem = fopen("hino.txt", "r");
destino = fopen("hino_copia.txt", "w");

if( origem!=NULL && destino!=NULL )
{
    while(fscanf(origem, "%c", &c) != EOF)
    {
        fprintf(destino, "%c", c);
    }
}
else
{
    printf("Arquivo nao encontrado\n");
}
fclose(origem);
fclose(destino);
```

Exemplo

- Abrir uma imagem do tipo PGM;
- Criar um novo arquivo;
- Ler uma string e três inteiros (A, B, C) da imagem;
  - Copiar a string de 2 caracteres em uma linha da imagem cópia;
  - Copiar o primeiro e segundo números lidos em outra linha da imagem cópia;
  - Copiar o terceiro número lido em outra linha da imagem cópia;
- Ler AxB inteiros da imagem original e copiar para a imagem cópia;
- A cada A números lidos da imagem original e copiar o \n para a imagem cópia;

Exemplo

```
1  FILE *origem, *destino;
2  origem = fopen("brain.pgm", "r");
3  destino = fopen("brain_copia.pgm", "w");
4
5  char s[3];
6  int a, b, c, d;
7  int i, j;
8  if( origem!=NULL && destino!=NULL ) {
9
10     fscanf(origem, "%s %d %d %d", s, &a, &b, &c);
11     fprintf(destino, "%s\n", s);
12     fprintf(destino, "%d %d\n", a, b);
13     fprintf(destino, "%d\n", c);
14     i=0;
15     while(i<b) {
16         j=0;
17         while(j<a) {
18             fscanf(origem, "%d", &d);
19             fprintf(destino, "%d ", d);
20             j++;
21         }
22         fprintf(destino, "\n");
23         i++;
24     }
25 }
```

Roteiro

Bibliotecas - Visão geral

- **Definição**:
  - Conjunto de funções agrupadas;
  - Compartilhamento de funcionalidades;

- **Utilização**:
  - Inclusão do arquivo cabeçalho (.h): contém os protótipos de cada função;
  - Biblioteca padrão da linguagem C (também conhecida como libc): cada compilador C possui sua implementação da biblioteca padrão C;
  - Exemplos:
    - ★ math.h Funções matemáticas comuns em computação. Ex.: sqrt, sin, cos, tan, ceil
    - ★ stdio.h Manipulação de entrada/saída. Ex.: printf, scanf
    - ★ stdlib.h Diversas operações, incluindo conversão, geração de números pseudo-aleatórios, alocação de memória. Ex.: abs, rand, malloc
    - ★ string.h Tratamento de cadeia de caracteres.

Biblioteca padrão - string.h

Há funções para manipulação de string já definidas na biblioteca padrão C:

- **strlen**: tamanho de uma string;
- **strcmp**: comparar duas strings;
- **strcpy**: copiar uma string em outra;
- **man string**: outras funções

```c
#include <stdio.h>
#include <string.h>

int main() {
    char s1[11] = "alo";
    int i = strlen(s1); //3 caracteres
    char s2[11] = "ala";
    int i = strcmp(s1, s2); //0 se iguais
                            //ou a subtracao da
                            //  primeira diferenca
                            //'o'-'a' = 111-97 = 14
                            //se s1>s2 retorna positivo
                            //se s1<s2 retorna negativo

    char s3[11];
    strcpy(s3, s1); //copia s1 em s3
    return 0;
}
```

Bibliotecas - Funções de String

Possíveis implementações das funções:

```c
int strlen( char str[] ) {
    int comprimento = 0;
    while ( str[comprimento] != '\0' )
        ++comprimento;

    return comprimento;
}

int strcmp( char s1[], char s2[] ) {
    int i = 0;
    for (i = 0; x[i]!='\0' && x[i] == y[i]; ++i);

    return x[i] - y[i];
}

char *strcpy(char dest[], char src[]) {
    int i;
    //copia src[i] em dest[i] e se src[i] != 0 continua
    for (i = 0; (dest[i] = src[i]) != 0; ++i);

    return dest;
}
```

Biblioteca própria - Visão geral

1. Fazer o cabeçalho .h (arquivo com os protótipos das funções - ex. teste1.h);

```c
//se prototipos ainda nao foram definidos
#ifndef _TESTE1_H
#define _TESTE1_H //definir

#include <stdio.h>  //biblioteca padrao

int soma (int, int);
void soma_vetor (float [], float [], int);

#endif
```

2. Fazer o fonte .c (arquivo com a implementação das funções - ex. teste1.c);
3. Compilar o .c para a geração .o (código objeto):
   gcc -c teste1.c -o libteste1.o
4. Agora podemos utilizá-la em outros programas (ex. meuprograma.c):
5. Compilar o programa ligando com o código objeto gerado:
   gcc meuprograma.c -o meuprograma libteste1.o
6. Executar o programa:
   ./meuprograma

Biblioteca própria - Visão geral

1. Fazer o cabeçalho .h (arquivo com os protótipos das funções - ex. teste1.h);
2. Fazer o fonte .c (arquivo com a implementação das funções - ex. teste1.c);

```c
#include "teste1.h" //biblioteca propria

int soma (int a, int b) {
    return a+b;
}

void soma_vetor (float v1[], float v2[], int n) {
    int i;
    for(i=0; i<n; i++)
        v2[i] = v1[i]+v2[i];
}
```

3. Compilar o .c para a geração .o (código objeto):
   gcc -c teste1.c -o libteste1.o
4. Agora podemos utilizá-la em outros programas (ex. meuprograma.c):
5. Compilar o programa ligando com o código objeto gerado:
   gcc meuprograma.c -o meuprograma libteste1.o
6. Executar o programa:

Biblioteca própria - Visão geral

1. Fazer o cabeçalho .h (arquivo com os protótipos das funções - ex. teste1.h);
2. Fazer o fonte .c (arquivo com a implementação das funções - ex. teste1.c);
3. Compilar o .c para a geração .o (código objeto):
   gcc -c teste1.c -o libteste1.o
4. Agora podemos utilizá-la em outros programas (ex. meuprograma.c):
5. Compilar o programa ligando com o código objeto gerado:
   gcc meuprograma.c -o meuprograma libteste1.o
6. Executar o programa:
   ./meuprograma

Biblioteca própria - Visão geral

1. Fazer o cabeçalho .h (arquivo com os protótipos das funções - ex. teste1.h);
2. Fazer o fonte .c (arquivo com a implementação das funções - ex. teste1.c);
3. Compilar o .c para a geração .o (código objeto):
   gcc -c teste1.c -o libteste1.o
4. Agora podemos utilizá-la em outros programas (ex. meuprograma.c):

```c
#include "teste1.h"

int main() {
    float a[2]={1,2}, b[2]={3,2};
    int c;

    c = soma(1, 2);
    soma_vetor(a, b, 2);

    printf("%d %f %f\n", c, b[0], b[1]);

    return 0;
}

```

5. Compilar o programa ligando com o código objeto gerado:

Biblioteca própria - Visão geral

1. Fazer o cabeçalho .h (arquivo com os protótipos das funções - ex. teste1.h);
2. Fazer o fonte .c (arquivo com a implementação das funções - ex. teste1.c);
3. Compilar o .c para a geração .o (código objeto):
   gcc -c teste1.c -o libteste1.o
4. Agora podemos utilizá-la em outros programas (ex. meuprograma.c):
5. Compilar o programa ligando com o código objeto gerado:
   gcc meuprograma.c -o meuprograma libteste1.o
6. Executar o programa:
   ./meuprograma

Biblioteca própria - Visão geral

1. Fazer o cabeçalho .h (arquivo com os protótipos das funções - ex. teste1.h);
2. Fazer o fonte .c (arquivo com a implementação das funções - ex. teste1.c);
3. Compilar o .c para a geração .o (código objeto):
   gcc -c teste1.c -o libteste1.o
4. Agora podemos utilizá-la em outros programas (ex. meuprograma.c):
5. Compilar o programa ligando com o código objeto gerado:
   gcc meuprograma.c -o meuprograma libteste1.o
6. Executar o programa:
   ./meuprograma

Automatizar comandos: Make

- O utilitário para automatização de execuções (compilações, comandos)
- Pode-se usar make com qualquer linguagem de programação cujo compilador possa ser executado com um comando shell
- Os comandos ficam em um arquivo chamado Makefile
  - Insira todas as relações dos arquivos que compõe o seu programa (geração dos códigos objetos das suas bibliotecas, complicação dos seu programas, remoção de arquivos etc.)
  - Os comandos devem estar indentados por TABs
- Depois basta rodar, no terminal, o comando: **make**
- A ferramenta make irá executar os comandos contidos no Makefile

```
1  all: meuprograma
2
3  meuprograma: libteste1.o
4      gcc meuprograma.c -o meuprograma libteste1.o
5
6  libteste1.o:
7      gcc -c teste1.c -o libteste1.o
8
9  clean:
10     rm -rf *.o
```

Automatizar comandos: Make

Exemplo com passagem de valores:

```
1  all: makedouble clean rename
2
3  test: makepdf clean rename
4
5  makepdf:
6    pdflatex $(tex).tex
7    mv $(tex).pdf $(pdf).pdf
8
9  makedouble:
10   pdflatex $(tex).tex
11   pdflatex $(tex).tex
12
13 rename:
14 ifdef $(pdf)
15   mv $(tex).pdf $(pdf).pdf
16 endif
17
18 clean:
19   rm -f *.aux *.log *.nav *.out *.snm *.toc *.vrb
```

Roteiro

Escopo de variáveis - O que é?

- Conjunto de **regras** que determinam a **utilização de uma variável**;
- Alcance de uma variável;
- Podemos dividir as variáveis quanto ao escopo em três tipos: **variáveis locais, parâmetros formais e variáveis globais**.

Escopo de variáveis

- **Variáveis locais**
  - Declaradas dentro de um bloco ({ }), só valem dentro deste bloco;

```c
void funcao1() {
    int j=1;
    printf("%d", j);
}

void main() {
    int a=9, b; //pertencem ao bloco da main
    if (a>9) {
        int x; //pertence ao bloco do if
        x = a+1; //ok
    }
    printf("%d\n", x) //erro: x eh local do if

    funcao1();
    printf("%d\n", j) //erro: j eh local da funcao1
}
```

- Parâmetros formais
- Variáveis globais e externas
- Variáveis estáticas
- Baixe os códigos https://fga.rysh.com.br/eda1/codigos/variaveis.zip

Escopo de variáveis

- **Variáveis locais**
- **Parâmetros formais**
  - São os argumentos de uma função;
  - Passagens por valor: escopo local;
  - Passagens por referência: variável local porém pode acessar um escopo externo;

```
1  //v por referencia e j por valor
2  void alo(int v[], int j)
3  {
4      j=1;     //variavel local
5      v[0]=j; //variavel local v, acessa variável externa i
6  }
7
8  int main()
9  {
10     int j=0, i[2];
11
12     alo(i, j);
13
14     printf("%d %d", i[0], j); // 1 0
15 }
```

- Variáveis globais e externas
- Variáveis estáticas

Escopo de variáveis

- **Variáveis locais**
- **Parâmetros formais**
- **Variáveis globais e externas**
  - Declaradas fora de todas as funções (válidas a partir do ponto de declaração);
  - Compartilhada por todas as funções;
  - Existem durante toda a execução do programa, ocupando memória;
  - Devem ser manipuladas com atenção pois várias função podem utilizá-las;
  - Externas: acessíveis fora do arquivo

```
1  #include "variaveis.h"
2
3  int var1 = 1;
4  static int var2 = 2;
5
6  int retornaVar1(){
7      return var1;
8  }
9
10 int retornaVar2(){
11     return var2;
12 }
```

- Variáveis estáticas

Escopo de variáveis

- **Variáveis locais**
- **Parâmetros formais**
- **Variáveis globais e externas**
- **Variáveis estáticas**
  - Global: disponível somente no arquivo onde foi declarada
  - Local: disponível por toda execução do programa

```
1  #include "variaveis.h"
2  extern int var1;
3  int var2;
4  int main() {
5      printf("%d\n", var1);
6      printf("%d %d\n", retornaVar1(), retornaVar2());
7
8      var1 = 4;
9      printf("%d\n", var1);
10     printf("%d\n", retornaVar1());
11
12     var2 = 5;
13     printf("%d\n", var2);
14     printf("%d\n", retornaVar2());
15
16     return 0;
17 }
```

Escopo de variáveis

- **Variáveis locais**
- **Parâmetros formais**
- **Variáveis globais e externas**
- **Variáveis estáticas**
- Baixe os códigos https://fga.rysh.com.br/eda1/codigos/variaveis.zip
  - Compilar a biblioteca: gcc -c variaveis.c -o libvariaveis.o
  - Compilar o exemplo: gcc global.c libvariaveis.o
  - Executar: ./a.out