Revisão de C: Parte 1

Algoritmos e Estruturas de Dados 2
2017-1
Flavio Figueiredo (http://flaviovdf.github.io)

Boas práticas de C - Revisando AEDS1

Tópicos

- **Vetores e Strings**
- Passagem de parâmetros
- Entrada e saída
- Boas práticas
- Compilação e Debug

Alocação Estática de Memória

- Declaração de variáveis
- Espaço de memória suficiente é alocado
  - Espaço armazenado não é o mesmo para todos linguagens e/ou arquitetura
  - e.g., Java e Python podem alocar mais ou menos espaço

```
char c;     // 1 byte
short a;    // 2 bytes
int a;      // 4 bytes
long b;     // 8 bytes
float x;    // 4 bytes
double y;   // 8 bytes
```

Alocação Estática de Memória

- Ao fazer a alocação estática, apenas o espaço necessário na memória é reservado.

- O conteúdo de cada posição não é alterado
  - Uma variável apenas declarada pode conter qualquer coisa.

- **Inicializar as variáveis, atribuindo valores, antes do uso.**
  - **Inclusive vetores, matrizes e strings.**

```
int soma = 2;
```

- https://stackoverflow.com/questions/1262459/coding-standards-for-pure-c-not-c
- http://www.maultech.com/chrislott/resources/cstyle/indhill-cstyle.pdf

Vetores

- Aloca diversas variáveis
- Indexado por inteiros
- O valor entre chaves indica quantas vezes o espaço de 1 variável vai ser alocado
    - De forma simples, o número de elementos no vetor

```
int v[100];     // 100 * sizeof(int) = 400 bytes
long vl[100];   // 100 * sizeof(long) = 800 bytes
double z[100];  // 100 * sizeof(double) = 800 bytes
```

Vetores

- Indexados por números inteiros (referência)
- Tal referência indica a posição de memória onde buscar o valor
- Lembre-se de também iniciar os vetores

```
int num_elements = 1000;
float x[num_elements];

//Iniciando o vetor (boa prática)
int i = 0;
for (i = 0; i < 1000; i++) {
     x[i] = i * 3.0;
}

int valor = x[20];

// x[20] está na posição x+20*sizeof(float)
// Qual número está armazenado em valor?
```

Vetores

- **O compilador C não vai avisar se você está acessando uma região errada!**
- Um erro em tempo de execução vai ocorrer

```
float x[1000];
// ... código aqui inicializando etc...
float y = x[2000]; // não dá erro de compilação
```

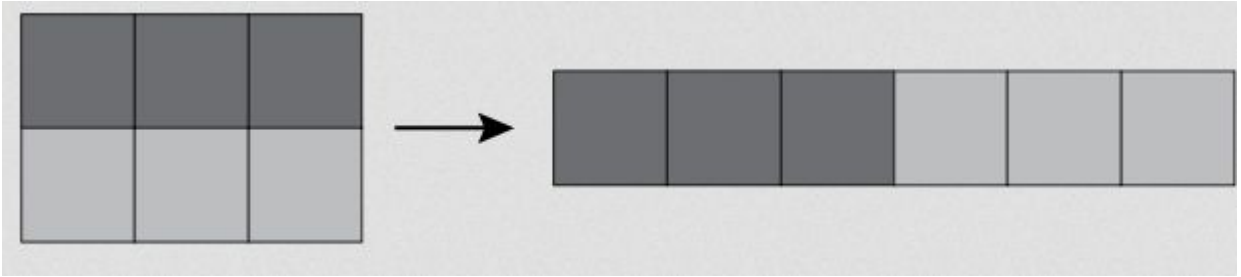
- Seu trabalho é garantir que tais erros não ocorram!
- Programe bem!
- Erro mais comum é o segmentation fault

Matrizes

- Vetores de mais de uma dimensão

```
int v[20][100];       // 20 * 100 * sizeof(int)
long vl[100][3];      // 100 * 3 * sizeof(long)
double z[100][100];   // 100 * 100 * sizeof(double)
```

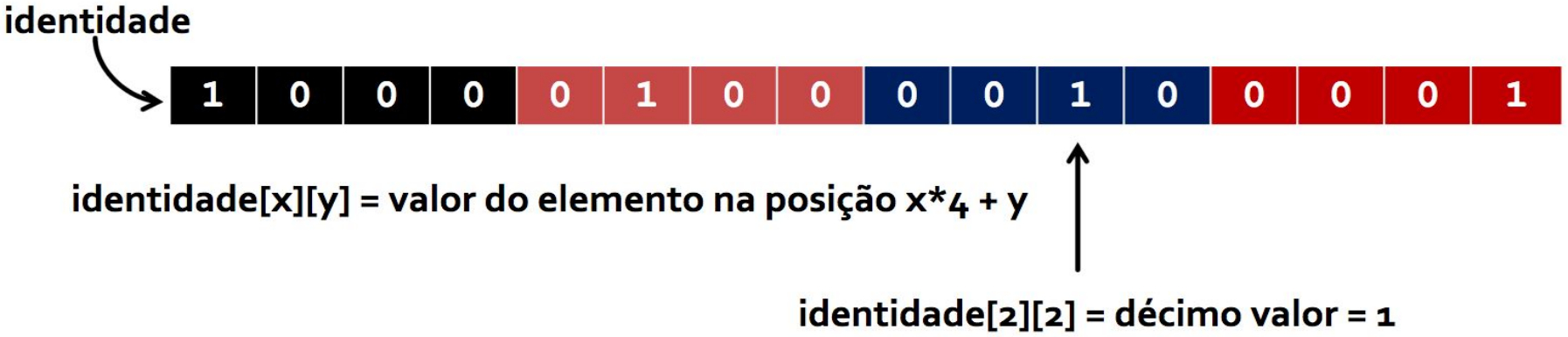
- Por baixo a alocação é linear
  - Matriz 2 por 3 abaixo, cada linha é um tom de cinza (`int M[2][3];`)
  - Lado esquerda representa a memória



```
M[1][2]; // posição: M + (1*3+2)*sizeof(int)
```

Matrizes

```
double identidade[4][4] = {{1, 0, 0, 0}, {0, 1, 0, 0},
                           {0, 0, 1, 0}, {0, 0, 0, 1}};
```



Strings

- Representa texto escrito “abcdefghijklmnopqrstuvxz”
- Em C utilizamos um vetor de caracteres

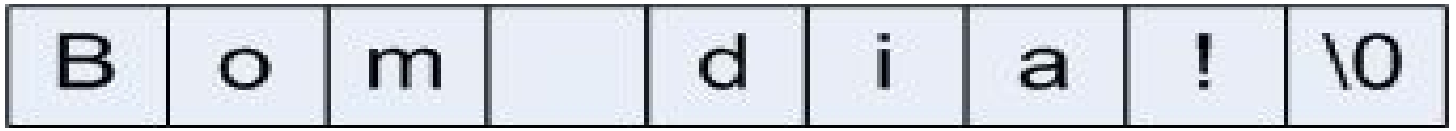
```
char nome[] = "Alexandre Silva";
//[A][l][e][x][a][n][d][e][ ][S][i][l][v][a][\0]
```

- Importante entender isto para saber como manipular texto
  - Atribuir e recuperar valores etc
- O caractere '\0' representa o fim da string (null em ASCII 0)
  - Útil quando é alocado mais espaço do que o tamanho da string
  - Sabemos onde termina o texto

Strings

- Para constantes utilizamos "

```
printf("%s", "Bom dia!");
```



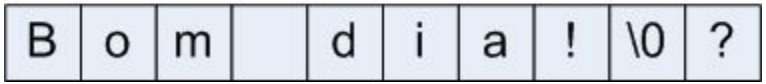
- Novamente note a presença do \0
- Pergunta? Qual a diferença de:

```
printf("%s\n", "Bom dia!");

printf("%s", "Bom dia!\n");
```

Strings

- Não é possível fazer atribuição direta para Strings
- Use strcopy ou strncopy
  - Diferença sutil entre as duas, em uma você pode passar o tamanho
  - No geral, pode utilizar strcopy, você já deve ter alocado o tamanho correto

```
char s[10];
strcpy(s, “Bom dia!”);
```



- Note o espaço no fim.

Strings

- As duas abordagens funcionam
- A segunda também pode ser utilizada para vetores de outros tipos

```
char[] nome = "Flavio";
char[] nome = {'F', 'l', 'a', 'v', 'i', 'o'};
```

- O nome da string representa o endereço de memória

```
char[] nome = 'Flavio';
printf("%s", nome + 3);
//Imprime vio
```

Funções de Strings

- strlen(st)
  - retorna o comprimento do string (com exceção do \0)
- strcat(s1, s2): concatena o s2 no s1
  - s1 tem que ter espaço suficiente alocado
- strcmp(s1, s2)
  - Comparação por ordem alfabética
  - retorna < 0 se s1 é menor que s2,
  - 0 se s1 é igual a s2,
  - e > 0 se s1 é maior que s2
  - **A comparação entre strings também tem que ser feita caractere a caractere, portanto não se pode usar s1==s2; isso só vai compara os endereços**

Exercício Poscomp 2009

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

int main (void) {
    char texto[] = "sem problemas";
    int i;
    for (i = 0; i < strlen(texto); i++)
    {
        if (texto[i] == ' ') break;
    }
    i++;
    for ( ; i < strlen(texto); i++)
          printf("%c", texto[i]);
    return 0;
}
```

- Qual vai ser a saída?

Strings

- strncat, strncmp e strncpy
  - Todas similares às respectivas anteriores
  - Especifica o número de caracteres

```
char str1[10];
char* str2 = "ABCDEFGHIJKLMNO";
strncpy(str1, str2, sizeof(str1));
printf("%s\n", str1);   // imprime "ABCDEFGHIJ"
```

Strings

- strtok: extrai tokens da string
- Vamos lembrar mais de AEDS1
- Explique o código ao lado
- Existem poucos trechos faltando (e.g., infile)

```
long int num; char linha[256]; char *p1 = NULL;
while (!feof(infile)) {
    fgets(linha, 256, infile);
    // delimitador: espaço ou fim de linha
    p1 = strtok(linha, " \n");
    while ((p1 != NULL) && (!feof(infile))) {
        num++; // Qual warning aconteceria aqui?
        fprintf(outfile, "%s\n", p1);
        p1 = strtok(NULL, " \n");
    }
}
printf("O arquivo tinha %ld palavras.\n", num);
fclose(infile);
```

Vetores de Strings

- Strings também são variáveis
- Podem ser declaradas dentro de vetores
- Dias da semana abaixo
- Nem todo dia tem 14 caracteres. Lembre-se do '\0'

```
char DiaSemana[7][14] =
    {"Domingo", "Segunda", "Terca", "Quarta", "Quinta", "Sexta", "Sabado"};
//...
printf("%s\n", DiaSemana[3]);
```

Vimos vetores e Strings
Continuamos na próxima aula
Devo passar o TP0

Trabalho Prático 0: Pontos Extra Revisão

- Computar 3 normas de matrizes
- Documentação na semana que vem
- Norma-1
  - Soma da coluna com a maior soma absoluta

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}|,$$

- Norma-Inf
  - Soma da linha com maior soma absoluta

$$\|A\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|,$$

- Norma Frobenius
  - Raiz quadrada da soma dos valores elevado ao quadrado

$$\|A\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

Tópicos

- Vetores e Strings
- **Passagem de parâmetros**
- Entrada e saída
- Boas práticas
- Compilação e Debug

Passagem de Parâmetros por Valor

- Por valor
- Uma cópia da variável é feita
- Qual a saída do código ao lado?

```c
#include <stdio.h>

int soma(int primeiro_valor, int segundo_valor) {
  int resultado = 0;
  primeiro_valor = 90;
  resultado = primeiro_valor + segundo_valor;
  return resultado;
}

int main(void) {
  int primeiro_valor = 7;
  int segundo_valor = 3;
  int soma_final = soma(primeiro_valor, segundo_valor);
  printf("Somando %d + %d\n", primeiro_valor, segundo_valor);
  printf("A soma foi %d\n", soma_final);
  return 0;
}
```

Passagem por Referência

- Aqui estamos passando o valor do ponteiro
- Ou seja, uma "referência"
- Usando
  - * para dereferencing
- Qual a saída?

```c
#include <stdio.h>

int sum(int *first_value, int second_value) {
  int result = 0;
  *first_value = *(first_value) + 3;
  result = *(first_value) + second_value;
  return result;
}

int main(void) {
  int first_value = 7;
  int second_value = 3;
  int final = sum(&first_value, second_value);
  printf("The sum of %d + %d was %d\n", \
    first_value, second_value, final);
  return 0;
}
```

Tópicos

- Vetores e Strings
- Passagem de parâmetros
- **Entrada e saída**
- Indentação
- Comentários
- Compilação e Debug

Entrada e Saída da Linha de Comando

- Executando:
  - ./codigo arg1 arg2
- Qual a saída?

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  //Note o tipo de argv
  //Ponteiro para array de chars
  printf("Foram passados %d parametros", argc);
  int i;
  for (i = 1; i < argc; i++)
    printf("Parametro %d foi %s\n", i, argv[i]);
  return 0;
}
```

Entrada e Saída da Linha de Comando

- Executando:
  - ./codigo arg1 arg2
- Qual a saída?
  - **argv[0]:** nome do programa
  - **argv[1]:** primeiro parâmetro
  - **argv[argc – 1]:** último parâmetro
  - **argv[argc]** é sempre NULL

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  //Note o tipo de argv
  //Ponteiro para array de chars
  printf("Foram passados %d parametros", argc);
  int i;
  for (i = 1; i < argc; i++)
    printf("Parametro %d foi %s\n", i, argv[i]);
  return 0;
}
```

GetOpt

- Serve quando se tem muitas opções na linha de comando
- Mais complicado inicialmente
- Uso opcional na disciplina

```
flaviovdf@chaplin:~$ gcc area.c -o area
flaviovdf@chaplin:~$ ./area
Usage: area -a num -l num
flaviovdf@chaplin:~$ ./area -a 2 -l 22
Area: 44
```

```c
#include <stdio.h>  //printf
#include <stdlib.h> //exit
#include <unistd.h> //getopt

void print_usage() {
  printf("Usage: area -a num -l num\n");
}

int main(int argc, char *argv[]) {
  int option = 0;
  int a = -1;
  int l = -1;
  int area = -1;

  //Duas opções a (altura) e l (largura).
  //: Indica que é obrigatório
  while ((option = getopt(argc, argv,"a:l:")) != -1) {
    switch (option) {
      case 'a' :
        //optarg é uma global que vem do <unistd.h>
        a = atoi(optarg);
        break;
      case 'l' :
        l = atoi(optarg);
        break;
      default: print_usage();
        exit(EXIT_FAILURE);
    }
  }
  area = a * l;
  printf("Area: %d\n", area);
  return 0;
}
```

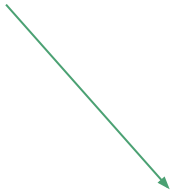
+ Funções de String

- Converter Strings para inteiros
  - int atoi(const char *string)
  - long atol(const char *string)
  - long long atoll(const char *string)
- Doubles
  - double atof(const char *string)
- Números no geral para Strings
  - sprintf(char *target, const char *format, ...)
  - sprintf(target, "%d", 2);

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

int main() {
  char str1[] = "124z3yu87";
  char str2[] = "-3.4211";
  char *str3 = "e24";
  printf("str1: %d\n", atoi(str1));
  printf("str2: %d\n", atoi(str2));
  printf("str3: %d\n", atoi(str3));

  printf("str1: %f\n", atof(str1));
  printf("str2: %.2f\n", atof(str2));
  printf("str3: %f\n", atof(str3));

  //Erro aqui. String sem '\0'!
  char target[4];
  sprintf(target, "%.2f", 2.099);

  sprintf(target, "%.1f", 2.099);

  return 0;
}
```

Entrada e Saída

- Precisamos da biblioteca stdio.h
  - #include <stdio.h>
- Sem ela não fazemos I/O (E/S)
- Maioria das linguagens vai precisar de uma biblioteca para E/S
  - São padrões e disponibilizadas junto com a linguagem



```c
#include <stdio.h>

void main(void) {
  int x = 99;
  int PI = 3.1415;
  int d = 2;
  char[] nome_aluno = "Flavio";
  printf("O valor de x eh %d", x);
  printf("Area: %f\n", PI*d*d/4);
  printf("Nome: %s", nomeAluno);
}
```

printf

- Primeiro argumento é uma string
- Esta string contém especificadores de formado ('%')
  - **%c** (char)
  - **%s** (string)
  - **%d** (int)
  - **%ld** (long int)
  - **%f** (float)
  - **%lf** (double)
  - **%e** (float notação científica)
  - **%g** (e ou f, ou seja, notação científica se necessário)

```c
#include <stdio.h>

void main(void) {
  int x = 99;
  int PI = 3.1415;
  int d = 2;
  char[] nome_aluno = "Flavio";
  printf("O valor de x eh %d", x);
  printf("Area: %f\n", PI*d*d/4);
  printf("Nome: %s", nomeAluno);
}
```

printf

%[opções][largura mínima][.precisão][tamanho]conversão

tamanho	
0	zeros à esquerda
#	alternativa
-	alinhar à esquerda
+	mostrar sinal positivo
	espaço para sinal positivo
'	agrupar milhares
I	digitos alternativos

tamanho	
hh	char
h	short
l	long
ll	long long
L	long double
z	size_t
t	ptrdiff_t
j	intmax_t

conversão	
c	char
d	int
u	unsigned int
x	int, hexadecimal
f	float
e	float, científico
g	float, e ou f
p	ponteiro
s	string
%	sinal percentual

printf

- Programa ascii_table.c
- Da para adivinhar a saída

```
032 = [ ]   033 = [!]   034 = ["]   035 = [#]
036 = [$]   037 = [%]   038 = [&]   039 = [']
040 = [(]   041 = [)]   042 = [*]   043 = [+]
044 = [,]   045 = [-]   046 = [.]   047 = [/]
048 = [0]   049 = [1]   050 = [2]   051 = [3]
052 = [4]   053 = [5]   054 = [6]   055 = [7]
056 = [8]   057 = [9]   058 = [:]   059 = [;]
060 = [<]   061 = [=]   062 = [>]   063 = [?]
064 = [@]   065 = [A]   066 = [B]   067 = [C]
068 = [D]   069 = [E]   070 = [F]   071 = [G]
072 = [H]   073 = [I]   074 = [J]   075 = [K]
076 = [L]   077 = [M]   078 = [N]   079 = [O]
080 = [P]   081 = [Q]   082 = [R]   083 = [S]
084 = [T]   085 = [U]   086 = [V]   087 = [W]
088 = [X]   089 = [Y]   090 = [Z]   091 = [[]
092 = [\]   093 = []]   094 = [^]   095 = [_]
096 = [`]   097 = [a]   098 = [b]   099 = [c]
100 = [d]   101 = [e]   102 = [f]   103 = [g]
104 = [h]   105 = [i]   106 = [j]   107 = [k]
108 = [l]   109 = [m]   110 = [n]   111 = [o]
112 = [p]   113 = [q]   114 = [r]   115 = [s]
116 = [t]   117 = [u]   118 = [v]   119 = [w]
120 = [x]   121 = [y]   122 = [z]   123 = [{]
124 = [|]   125 = [}]   126 = [~]   127 = [ ]
```

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

int main(int argc, char **argv)
{
    /* first printable char: */
    const unsigned char first= 32;
    /* last printable char: */
    const unsigned char last= 126;
    unsigned char c = first;
    int col;

    while(c <= last) {
       for(col = 0; col < 4; col++) {
          printf("%03hhd = [%c]   ", c, c);
          c++;
       }
       printf("\n");
    }
    printf("\n");
    return EXIT_SUCCESS;
}
```

Entrada com scanf

- Imagine como sendo o "oposto do printf"
- Lê dados da entrada padrão para variáveis
  - Exemplo lendo 2 vars de uma vez só
    - scanf("%f %d\n", &lado, &nlados);

```c
#include <stdio.h>

int main(void)
{
  int n;
  while (scanf("%d", &n))
    printf("%d\n", n);
  return 0;
}
```

```c
#include <stdio.h>

int main(void)
{
  char word[20];
  if(scanf("%19s", word) == 1)
    printf("%s\n", word);
  return 0;
}
```

Scanf mais avançado

- Especificadores de tamanho e filtro
  - %60s lê 60 caracteres
  - %[aeiou]s lê apenas vogais
  - %[^aeiou]s: ^ é uma negação, aqui não vamos ler vogais (parar nelas)
  - %60[^0123456789]s lê 60 caracteres até encontrar um número

```
char buffer[80];
scanf("%79s", buffer);
```

getchar/putchar(int c)/gets

- getchar()
  - Lê 1 caractere
- putchar
  - Imprime 1 caractere
- gets
  - Lê 1 linha (até \n')

```c
#include <stdio.h>

int main()
{
   char str[50];

   printf("Enter a string : ");
   gets(str);

   printf("You entered: %s", str);

   return 0;
}
```

Arquivos

- Arquivos não são tão diferentes da entrada e saída padrão
  - Entrada e Saída Padrão
    - stdin e stdout
    - scanf e printf lidam com as 2 respectivamente
    - [Geralmente] os 2 são o seu terminal
- Arquivos usam um handle
- fopen para criar
  - Abrir o arquivo

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

int main() {
        int r;
        char *fname_in = "entrada.txt";
        char *fname_out = "saida.txt";

        FILE *arquivo_in = fopen(fname_in, "r");
        if(!arquivo_in) {
                perror(NULL);
                exit(EXIT_FAILURE);
        }
        while(!feof(arquivo_in)) {
                char campo1[80];
                char campo2[80];
                r = fscanf(arquivo_in, "%s %s",
                                campo1, campo2);
                printf("leu %d campo(s):\n", r);
                printf("%s %s\n", campo1, campo2);
        }

        //Note o mode de escrita!
        FILE *arquivo_out = fopen(fname_out, "w");
        if(!arquivo_out) {
                perror(NULL);
        }
        r = fprintf(arquivo_out, "hello arquivo!\n");
        printf("escreveu %d campos\n", r);
        fclose(arquivo_in);
        fclose(arquivo_out);
        return EXIT_SUCCESS;
}
```

Arquivos Dicas

- Sempre verifique se o arquivo abriu corretamente
  - FILE *fopen(char *nome, char *modo)
  - Finalize seu programa em caso de erro
  - No erro o retorno é NULL do fopen
- Sempre verifique se não leu até o fim
  - int feof(FILE *file)
  - boolean like. False quando temos erro
- Sempre feche seus arquivos
  - fclose(FILE *file)

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

int main() {
        int r;
        char *fname_in = "entrada.txt";
        char *fname_out = "saida.txt";

        FILE *arquivo_in = fopen(fname_in, "r");
        if(!arquivo_in) {
                perror(NULL);
                exit(EXIT_FAILURE);
        }
        while(!feof(arquivo_in)) {
                char campo1[80];
                char campo2[80];
                r = fscanf(arquivo_in, "%s %s",
                                campo1, campo2);
                printf("leu %d campo(s):\n", r);
                printf("%s %s\n", campo1, campo2);
        }

        //Note o mode de escrita!
        FILE *arquivo_out = fopen(fname_out, "w");
        if(!arquivo_out) {
                perror(NULL);
        }
        r = fprintf(arquivo_out, "hello arquivo!\n");
        printf("escreveu %d campos\n", r);
        fclose(arquivo_in);
        fclose(arquivo_out);
        return EXIT_SUCCESS;
}
```

Modos de fopen

Mode	Meaning
“r”	Open a text file for reading
“w”	Create a text file for writing
“a”	Append to a text file
“rb”	Open a binary file for reading
“wb”	Create a binary file for writing
“ab”	Append to a binary file
“r+“	Open a text file for read/write
“w+“	Create a text file for read/write
“a+“	Open a text file for read/write
“rb+“	Open a binary file for read/write
“wb+“	Create a binary file for read/write
“ab+“	Open a binary file for read/write

Muitas das funções de E/S tem a versão f

- gets() ➜ fgets(arq, tamMax, string);
- getchar() ➜ fgetc(arq);
- putc(ch) ➜ fputc(arq, ch)
- printf ➜ fprintf(arq, string, valor)
- scanf ➜ fscanf(arq, string, endereço)

Um cat simples

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

int main() {
  FILE *infile;
  char ch;
  if ((infile = fopen("arquivo.txt", "r")) == NULL) {
     printf("Erro arquivo nao encontrado\n");
     exit(1);
  }
  while ((ch = fgetc(infile)) != EOF) {
     printf("%c", ch);
  }
  fclose(infile);
  return 0;
}
```

Caminhando em Arquivos

- Saber a posição atual de um arquivo
  long ftell(FILE *arquivo)

- Andar n-bytes para frente
  int fseek(FILE *arquivo, int base, int distância)
  - Base pode ser:
    - SEEK_SET (início do arquivo)
    - SEEK_CUR (posição inicial)
    - SEEK_END (final do arquivo)

- Opções úteis em arquivos binários
  - Uma imagem que foi escrita byte-a-byte

Tópicos

- Vetores e Strings
- Passagem de parâmetros
- Entrada e saída
- **Boas práticas**
- Compilação e Debug

Indentação

- Realça a estrutura lógica
- Torna o código mais legível
  - Exemplo de um código que compila
  - int main(void) { int i = 0; for (i = 0; i < 10; i++) if (i == 2) printf("Achei o 2"); else { printf(":("); }return 0;}
- Escolha um padrão e use
  - [Dica] Configure seu editor cedo para isto
  - [Dica] Escolha entre Tabs e Espaço
  - [Dica] Idente com 2 ou 4 espaços
    - Se usar Tab configure seu editor para que uma tab apareça como 2 ou 4 espaços
- **Os exemplos dos slides usaram diversos estilos**

Indentação: Estilos

K&R

```
static char * concat(char *s1, char *s2)
{
        while(x == y) {
                something();
                something_else();
        }
        final_thing();
}
```

Recomendado

Allman

```
static char *
concat(char *s1, char *s2)
{
        while(x == y)
        {
                something();
                something_else();
        }
        final_thing();
}
```

GNU

```
static char *
concat(char *s1, char *s2)
{
    while(x == y)
      {
        something();
        something_else();
      }
    final_thing();
}
```

Constantes

- Números mágicos são ruim
  - Um número mágico é aquele número solto no seu código
    - 3.1415
      - O que é isso?
    - PI
      - Bem melhor!
- Números soltos no código não dizem nada
- Melhor manutenção com constantes

```
#define PI 3.14159
#define TAMANHO_MAX_LINHA 256

char * le_linha(FILE *entrada) {
       char *linha = malloc(TAMANHO_MAX_LINHA);
       ...
       return linha;
}
```

Nomes

- Algumas variáveis merecem nomes significativos:
  MAX_VETOR, numClientes, listaAlunos

- Variáveis auxiliares em geral recebem nomes curtos: **i, j, aux, x**
  - Cuidado para não fazer confusão.
  - Não abusar: **i, ii, iii, aux1, aux2, aux3...**
  - Variáveis inteiras: **i, j, k**
  - Variáveis reais: **x, y, z**
  - Strings: **s1, s2**
  - Booleanas: nome do teste **(existe, valido, ocorre)**

Nomes: Dicas

- Constantes
  - Só maiúsculas (constantes: **PI, E, MAX**)

- Contadores e afins:
  - Só minúsculas (**i, num, conta**)

- Funções e Variáveis no Geral
  - CamelCase (**numMat, anguloEntrada**)
    ou
  - Underscore (**num_mat, angulo_entrada**)
  - **Escolha 1 estilo e sempre use ele!**

- Há quem prefira inserir comentários e usar nomes de variáveis em inglês, por ficar mais próximo da linguagem de programação

Comentários

- Daquelas coisas que em excesso é pior
- Comente os módulos
  - Vamos revisar eles ainda mesmo se não viram
- Comente funções
  - Comentário não é código!
- Decisões de código
  - Utilizamos tal biblioteca por isto...
- **Não exagere!**

```
/**
/* Computa a norma frobenius. Isto é, a soma de
/* todos os elementos ao quadrado.
/**
float frobenius(float **matrix)
{
...
}
```

Comentários Ruins

```
/**
/* Computa a norma frobenius. Para
/* isto, usamos 2
/* comandos for que itera em cima da
/* matriz de floats
/* e vamos agregando uma sola ao
/* quadrado...
/**
float frobenius(float **matrix)
{
...
}
```

Isso aqui é código, não comentário

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

int main() {
  FILE *infile; //um arquivo
  char ch;
  if ((infile = fopen("arquivo.txt", "r")) == NULL) {
     printf("Erro arquivo nao encontrado\n");
     exit(1);
  }
  while ((ch = fgetc(infile)) != EOF) {
     printf("%c", ch); //imprime na tela
  }
  fclose(infile);
  return 0;
}
```

Organização e Limpeza

- Procurar dar um aspecto organizado ao código, ajuda na compreensão.
- Entender o código fonte como um instrumento de comunicação.
- Comentar excessivamente código mal escrito não ajuda.
- Dar nomes adequados a variáveis ajuda bastante.

Tópicos

- Vetores e Strings
- Passagem de parâmetros
- Entrada e saída
- Boas práticas
- **Compilação e Debug**

Compilação

- Leia as mensagens de ERRO para entender o problema
- Usem -Wall para warnings

```
gcc -Wall -std=c99 codigo.c -o codigo
```

- Entreguem código sem erros e sem warnings
- Mensagens de erro são esquisitas
  - Olhe para a linha do erro
  - Leio o código perto da linha de erro

Debugger

- Serve para entender o código passo a passo
  - Watches para observar valores
  - Breakpoints para posicionar os passos

- Leia a documentação do code-blocks em debugging http://wiki.codeblocks.org

- GDB também funciona
  https://www.cs.cmu.edu/~gilpin/tutorial/

```
> gdb <path-to-program>
q | quit                          // quit and exit gdb
r | run                           // run the program
b | break <method|file:line>      // sets a breakpoint
n | next                          // move to the next line
c | continue                      // move to next breakpoint
d | disable                       // disable all breakpoints
l | list                          // list the source code
p | print <variable>              // print the variable
p | print <variable = value>      // set the variable
info locals                       // prints all local vars
```

Como testar seu código?

- Lembre-se dos casos comuns de erro
  - Limites de vetores
  - Acesso a um espaço de memória errado
  - Valores NULL nas variáveis
  - Contadores e valores de incremento
  - Lembre-se das dicas de arquivos

- Casos limite
  - Imagine uma função que trata uma matriz
  - Ela funciona quando a Matriz tem 1 dimensão zerada
  - E as 2?

- Caso base
  - Tudo está ok, meu código tem a solução correta?

Códigos com erro, execute os mesmos

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

void aloca(int *x, int n) {
  x = (int*) malloc(n*sizeof(int));
  if (x == NULL) {
     perror(NULL);
     exit(EXIT_FAILURE);
  }
  x[0] = 20;
}

int main() {
  int *a;
  aloca(a, 10);
  a[1] = 40;
}
```

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

int main(int argc, char **argv)
{
    int numbers[10];
    int i;
    for(i = 0; i <= 10; i++) {
      numbers[i] = 0;
    }
}
```

Próxima Aula

- Revisão de structs
- Revisão de alocação dinâmica
- **FIM DA REVISÃO!**
  - Aproveitem o pouco tempo para garantir que estão com AEDS1 bem fundamentado
  - Lembrem-sem to TP0
    - Simples
    - Revisão
    - Pontos Extra