Curso de C

PET-CoCE, UTFPR, DAINF
http://www.dainf.ct.utfpr.edu.br/petcoce

Contexto Histórico

- Necessidade de uma linguagem em que o programador não precisasse se preocupar com todas as operações da máquina
- BCPL(1967) -> B(1970) -> C(1972)
- ANSI C (1989) -> C90 -> C99

B e C

Ken Thompson

Dennis Ritchie

Características

- Compilada
- Estruturada
- Procedural
- Portabilidade

Estrutura de um programa em C

- Diretivas de pré-compilação
- main()
- printf(), scanf() e especificadores de formato

Especificadores de formato mais comuns:

- %c -> caracter
- %d -> inteiro
- %f -> float
- %s -> string
- %p -> endereço

Diferenças entre Java e C

- Declaração de Constantes (final -> define)
- Criação de vetores (int v[] = new int[10]; -> int v[10];)
- Declaração de Strings (String s; -> char s[];)
- Tipos de variáveis

Tabela: Todos os Tipos de dados definidos pelo Padrão ANSI C, seus tamanhos em bytes e suas faixa de valores.

Tipo	Tamanho em Bytes	Faixa Mínima
`char`	1	-127 a 127
`unsigned char`	1	0 a 255
`signed char`	1	-127 a 127
`int`	4	-2.147.483.648 a 2.147.483.647
`unsigned int`	4	0 a 4.294.967.295
`signed int`	4	-2.147.483.648 a 2.147.483.647
`short int`	2	-32.768 a 32.767
`unsigned short int`	2	0 a 65.535
`signed short int`	2	-32.768 a 32.767
`long int`	4	-2.147.483.648 a 2.147.483.647
`signed long int`	4	-2.147.483.648 a 2.147.483.647
`unsigned long int`	4	0 a 4.294.967.295
`float`	4	Seis digitos de precisão
`double`	8	Dez digitos de precisão
`long double`	10	Dez digitos de precisão

Retirado de :http://equipe.nce.ufrj.br/adriano/c/apostila/tipos.htm

Onde as variáveis são declaradas

- Variáveis locais : são variáveis automáticas, existem enquanto o bloco no qual estão definidas é executado.
- Variáveis globais : são reconhecidas por qualquer pedaço de código e guardam seus valores durante toda a execução do programa.

```
/* Esta função  esta errada, variáveis locais devem ser
declaradas no inicio do bloco no qual estão definidas */

void f (void) {

    int i;
    i = 10;
    int j; // isso provoca um erro
    j = 20;
}
```

```
/* Agora esta certo, define j dentro de seu próprio bloco de
código. */

void f (void) {

    int i;
    i = 10;

    if (i == 10)  {
        int j;
        j = 20;
        printf("%d", j);
    }
}
```

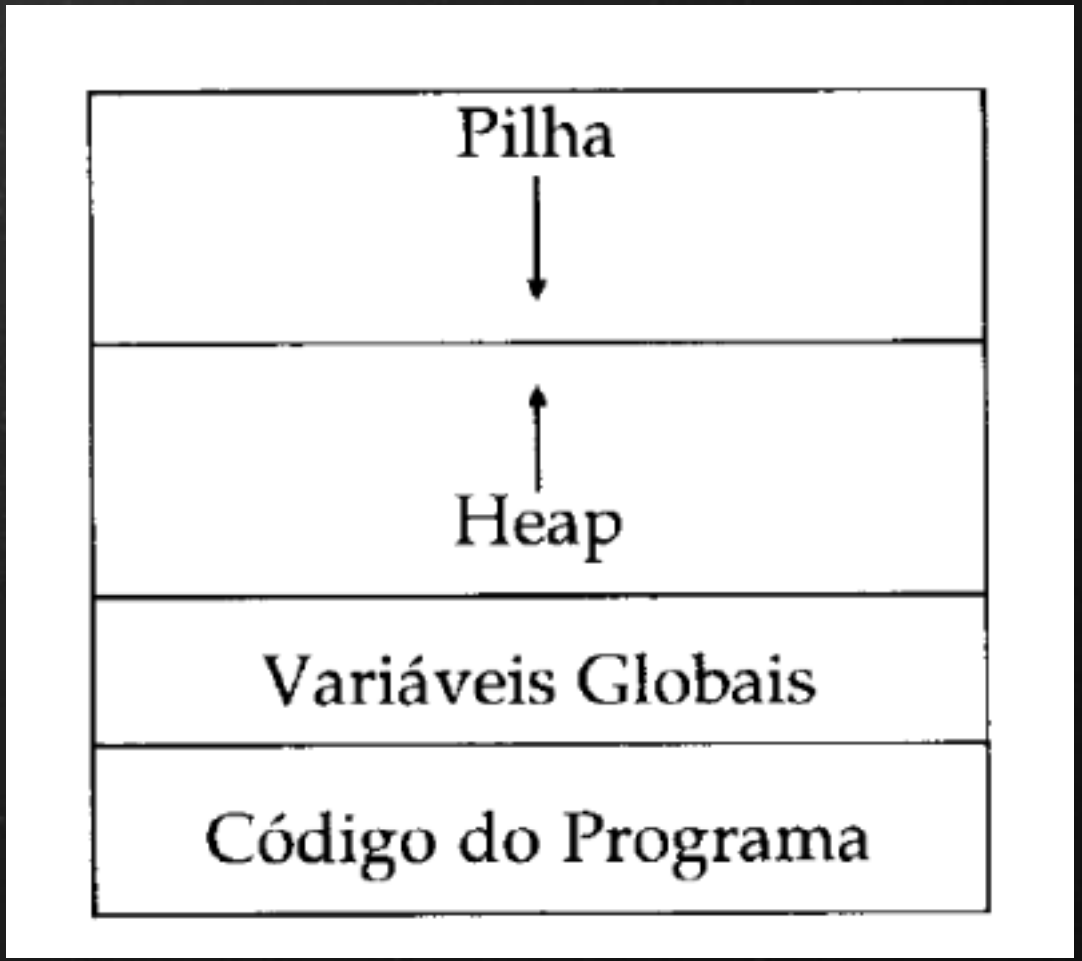
Tipos Estruturados (structs)

- Agrupar dados
- O operador de acesso é o "."
- Definição de "novos" tipos
- Aninhamento de estruturas

Funções

- Dividir grandes tarefas em tarefas menores
- Protótipo antes de ser chamada: void fat(int n);
- Passagem de parâmetros
- Retorno de valores
- void
- C trabalha com passagem por valor
- Variáveis Estáticas

Como o C enxerga a memória



Como o C enxerga a memória

- Um programa C compilado cria e usa 4 regiões de memória.

- Pilha :
  - Possui o endereço de retorno de chamadas de função.
  - parâmetros de funções
  - variáveis locais
- Heap :
  - Região de memória livre usada para se alocar memoria dinamicamente.

Exercício

Apresente todos os números divisíveis por 4 que sejam menores que 200.
Um número é divisível por 4 quando termina em 00 ou quando o número formado pelos dois últimos algarismos da direita for divisível por 4.

Exercícios

Preencha uma matriz (x,y) com números inteiros e solicite um número do usuário, pesquise se esse número existe na matriz. Se existir, imprima em qual posição ele se encontra (linha x coluna). Se não existir, imprima uma mensagem de erro.

Solicite um nome do usuário e escreva-o de trás para frente.

Exercício

Crie um programa para cadastrar as disciplinas que você irá cursar este semestre, pergunte ao usuário e em uma struct guarde o código, o nome, o professor, a sala e os horários da disciplina. Sujestão de menu:
(C)adastrar disciplina
(M)ostrar disciplinas cadastradas
(S)air

Ponteiros em C

Ponteiros: Motivação

- Fornecem os meios pelos quais as funções podem modificar seus argumentos
- São usados para suportar as rotinas de alocação dinâmica
- Pode aumentar a eficiência de certas rotinas

O que são ponteiros?

- Um ponteiro é um tipo especial de variável que contém o endereço de outra.

Operadores

& **é um operador que devolve o endereço na memória de seu operando**

***** **é um operador que devolve o valor da variável localizada no endereço que segue**

Cuidados

As variáveis de ponteiros devem sempre apontar para o tipo correto.

Exercício

1. Escreva uma função CALCULA que:
2.

. receba como parâmetros duas variáveis inteiras, X e Y;

. retorne em X a soma de X e Y;

. retorne em Y a subtração de X e Y.

Pergunta: a passagem dos parâmetros para a função deve ser por valor ou por referência?

Ponteiros como Parâmetros

- Passagem de parâmetros por referência é suportada somente pelo C++
- Em C, fazemos uma simulação utilizando ponteiros

Aritmética de Ponteiros

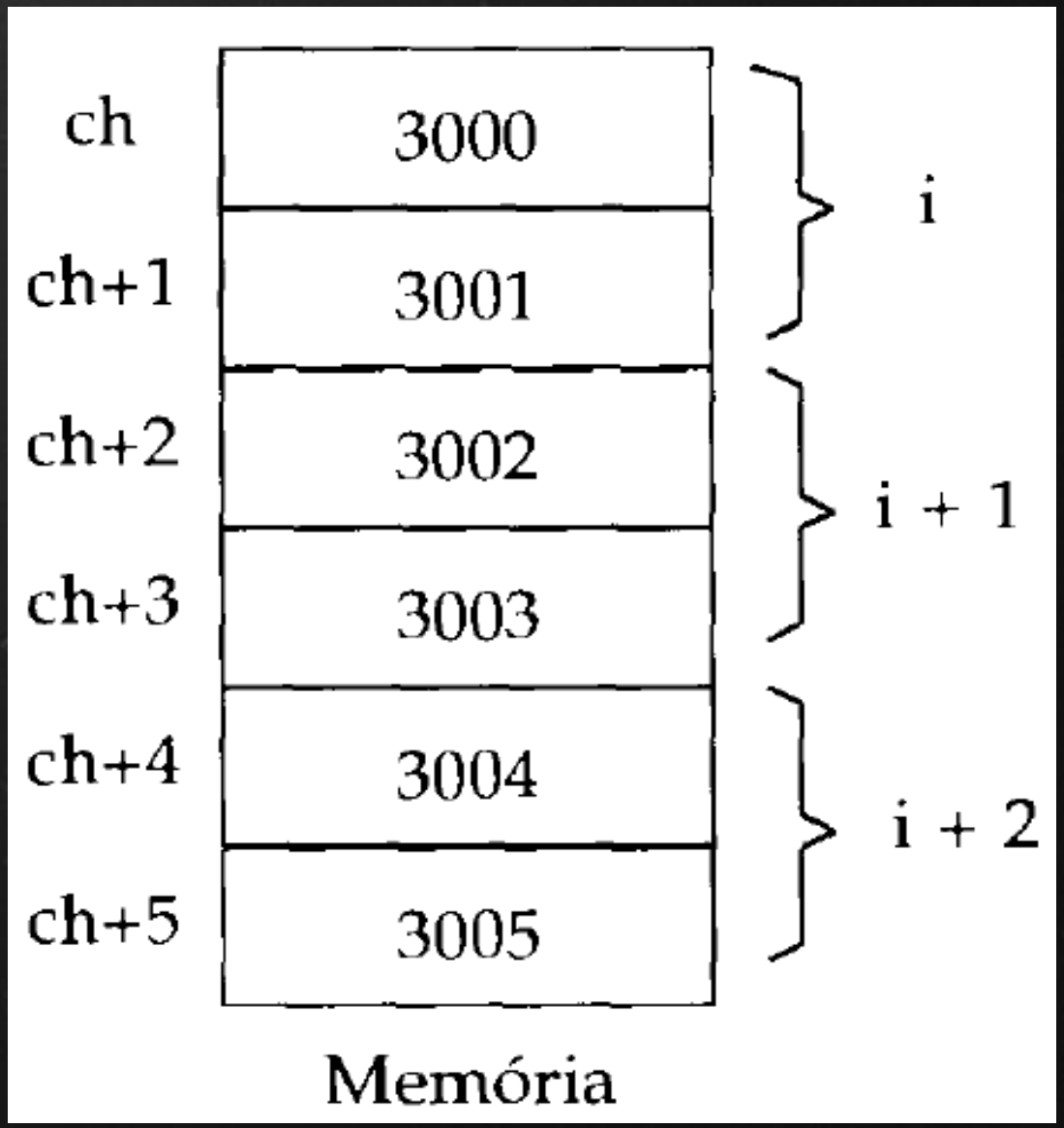
Duas operações: adição e subtração

Ex: (p1 é ponteiro para inteiro, o inteiro tem 2 bytes)

```
/* Neste ponto, p1 aponta para o endereço 2000 */
p1++;
/* Agora, p1 aponta para o endereço 2002  */
p1=p1-2;
/* p1 agora aponta parao endereço 1998 */
```

Aritmética de Ponteiros

```
char *ch = 3000;
int *i = 3000;
```



Comparação de Ponteiros

Exemplo:

```
if(p<q) printf("p aponta para uma posição de memória inferior que q\n");
```

Exercícios

Escreva uma função que receba 2 ponteiros para inteiro como parâmetros e faça a troca do valor destes inteiros. Protótipo da função:
void Troca (int *a,int *b);

Escreva um programa que declare uma matriz 100x100 de inteiros. Você deve inicializar a matriz com zeros usando ponteiros para endereçar seus elementos. Preencha depois a matriz com os números de 1 a 10000, também usando ponteiros.

Ponteiros e Matrizes

```
char str[80], *p1;
p1 = str;
```

/* ambos devolvem o quinto elemento */

```
str[4]
*(p1+4)
```

- Usar aritmética de ponteiros é mais rápido.

Matrizes de Ponteiros

Podemos criar matrizes de ponteiros:

```
int *x[10];
```

Para atribuir o endereço de uma variável int ao 3o elemento da matriz:

```
x[2] = &var;
```

Para encontrar o valor de var, escreve-se

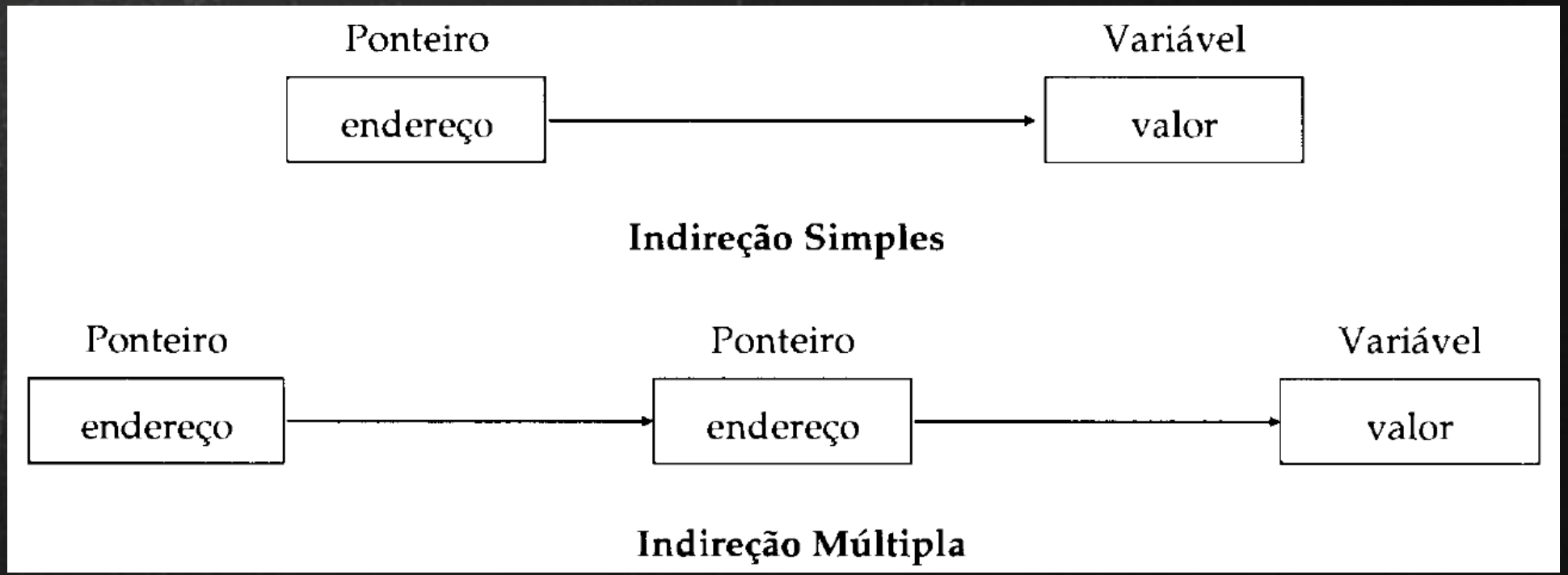
```
*x[2];
```

Exercício

Escrever um programa para ler uma frase qualquer do teclado e imprimir, esta mesma frase, um carácter por vez usando aritmética de ponteiros e indireção.

Indireção Múltipla

Indireção Múltipla: ponteiros apontando para ponteiros.



Indireção Múltipla

Declaração:

```
float **newbalance;
```

Exemplo:

```
#include <stdio.h>
void main(void)
{
  int x, *p, **q;
  x = 10;
  p = &x;
  q = &p;
  printf("%d", **q); /* print the value of x */
}
```

Alocação Dinâmica em C

- Como evitar o desperdício de memória quando alocamos vetores?
- Como criar um vetor do tamanho que será necessário ao usuário?
- Como devolver ao sistema regiões de memória que não são mais necessárias?

Alocação Dinâmica em C

Solução:

- Determinar o quanto de memória será necessário.
- Solicitar ao sistema a quantidade necessária de memória.
- Devolver ao sistema a região de memória após o seu uso.

Em C, temos duas funções especiais para resolver esse problema:

1. `malloc()`
2. `free()`

Função `malloc()`

```
<ponteiro> = malloc(<tamanho em bytes>);
```

- Recebe como parâmetro o número de **bytes** necessários (tamanho da memória).
- Retorna um **ponteiro** para a região de memória alocada dinamicamente pelo sistema.
- Se o sistema não conseguir alocar uma região com o tamanho especificado (por falta de espaço) a função **malloc**retorna **zero** ou **NULL**.

Função `free()`

free(<ponteiro para região de memória>**);**

- Recebe como parâmetro o ponteiro para região de memória alocada dinamicamente e devolve a região alocada ao sistema.
- Apenas regiões de memória alocadas com **malloc()** podem ser desalocadas com **free()**.
- A mesma região de memória só pode ser desalocada uma vez.

Exemplo de Alocação Dinâmica

```
#include<stdlib.h>
#include<stdio.h>
int main()
{
    int*vetor,n;
    printf("Entre com o tamanho do vetor: ");
    scanf("%d", &n);
    vetor= malloc(n* (sizeof(int)));
    if( vetor == 0)
    {
    printf("Erro: Memoriainsuficiente");
    return-1;
    }
    else...
```

Operador `sizeof()`

Para alocar um tipo-de-dado que ocupa vários bytes, é preciso recorrer ao operador sizeof, que diz quantos bytes o tipo especificado tem:

```
typedef struct {
   int dia, mes, ano;
} data;
data *d;
d = malloc (sizeof (data));
d->dia = 31; d->mes = 12; d->ano = 2008;
```

OBS: `sizeof()` não é função, é um operador.

Exercício: Verificação do tamanho

Compile e execute o seguinte programa:

```
typedef struct {
  int dia, mes, ano;
} data;

int main (void) {
  printf ("sizeof(data) = %d\n", sizeof(data));
  printf ("sizeof(data *) = %d\n", sizeof (data
*));
  return 0;
}
```

Falta de memória

Se a memória do computador já estiver toda ocupada, malloc não consegue alocar mais espaço e devolve NULL. Convém verificar essa possibilidade antes de prosseguir:

```
ptr = malloc (sizeof (data));
if (ptr == NULL) {
   printf ("Socorro! malloc devolveu NULL!\n");
   exit (EXIT_FAILURE);
}
```

Falta de memória

A codificação freqüente e repetida desse teste é cansativa para o programador e desvia a atenção do leitor. Por isso, vamos usar a seguinte versão alternativa de malloc:

```
void *mallocX (unsigned int nbytes)
{
   void *ptr;
   ptr = malloc (nbytes);
   if (ptr == NULL) {
      printf ("Socorro! malloc devolveu NULL!\n");
      exit (EXIT_FAILURE);
   }
   return ptr;
}
```

Alocação Dinâmica de Vetores

Eis como um vetor (= array) com n elementos inteiros pode ser alocado (e depois desalocado) durante a execução de um programa:

```
int *v;
int n, i;
scanf ("%d", &n);
v = mallocX (n * sizeof (int));
for (i = 0; i < n; i++)
   scanf ("%d", &v[i]);
for (i = n; i > 0; i--)
   printf ("%d ", v[i-1]);
free (v);
```

Exercício

Escreva um programa que recebe um número n seguido de 2n números, onde cada par de números denota o registro acadêmico de um aluno e a respectiva nota.

PS: Utilize um vetor de ponteiros de registros (estruturas) para armazenar os dados dos alunos.

PPS: O programa não deve limitar o número de alunos.

Alocação Dinâmica de Matrizes

- Matrizes bidimensionais são implementadas como vetores de vetores.
- Uma matriz com m linhas e n colunas é um vetor cada um de cujos m elementos é um vetor de n elementos.

```
int **A;
int i;
A = mallocX (m * sizeof (int *));
for (i = 0; i < m; i++)
  A[i] = mallocX (n * sizeof (int));
```

O elemento de A que está no cruzamento da linha i com a coluna j é denotado por A[i][j].

Alocação Dinâmica de Matrizes

- Matrizes bidimensionais são implementadas como vetores de vetores.
- Uma matriz com m linhas e n colunas é um vetor cada um de cujos m elementos é um vetor de n elementos.

```
int **A;
int i;
A = mallocX (m * sizeof (int *));
for (i = 0; i < m; i++)
  A[i] = mallocX (n * sizeof (int));
```

O elemento de A que está no cruzamento da linha i com a coluna j é denotado por A[i][j].

Realloc

Nos permite realocar um vetor dinâmicamente, preservando o conteúdo dos elementos.

```
v = (int*) realloc(v, m*sizeof(int));
```

Exercício

Escreva um programa que leia uma *string* que representa um número inteiro na base 10 e gere o valor inteiro correspondente. Faça isto sem usar as funções **atoi()** e **scanf()**.

Exercício

Escreva um programa que gere um vetor de três dimensões (X, Y e Z) em que cada posição guarda a soma de suas coordenadas. As dimensões da matriz deverão ser determinadas em tempo de execução e o programa deverá informar os valores gerados.

Exercício

Escreva um programa que simule uma pilha usando vetores. O programa deve implementar as seguintes operações na pilha:

1. Inserir.
2. Remover
3. Listar
4.
5. A pilha deve ser estática ou seja você deve determinar sua capacidade máxima
6.

Exercício

Faça um programa igual ao anterior mas que implemente a pilha sobre uma lista encadeada (pilha dinamica).

Funções de Strings

A biblioteca C padrão tem um conjunto rico e variado de funções para manipulação de strings e de caracteres

- A implementação exige o arquivo de cabeçalho STRING.H

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

Função strcmp

```
int strcmp(const char *str1, const char *str2)
```

A função strcmp() compara lexicograficamente duas strings e devolve um inteiro baseado no resultado, como mostrado a seguir:

VALOR	SIGNIFICADO
Menor que zero	str1 é menor que str2
Zero	str1 é igual a str2
Maior que zero	str1 é maior que str2

Função strcpy

```
char *strcpy(char *str1, const char *str2);
```

A função strcpy() copia o conteúdo de str2 em str1.

Função atoi

```
int atoi(char*);
```

A função atoi() converte uma string em um inteiro;

Funções matemáticas

O padrão C ANSI define 22 funções matemáticas que se encontram nas seguintes categorias:

- Funções trigonométricas
- Funções hiperbólicas
- Funções exponenciais e logarítmicas
- Miscelâneas

- Todas as funções matemáticas exigem o arquivo de cabeçalho MATH.H

```
#include <math.h>
```

Função para calcular o log

```
double log(double num);
```

A função log() devolve o logaritmo natural de num.

Função para calcular a raiz quadrada

```
double sqrt(double num);
```

A função sqrt() devolve a raiz quadrada de num. Se você chamar com um argumento negativo, ocorrerá um erro de domínio.

Arquivos

Podemos manipular arquivos de duas maneiras:

1. Modo texto: seqüencia de caracteres
2. Modo binário: seqüencia de bytes

Cursor: indica a posição de trabalho no arquivo

Funções para manipulação de arquivos

Abertura e fechamento de arquivos:

```
FILE* fp;
fp = fopen ("nomeDoArquivo", "modo");
fclose(fp);
```

Os caracteres interpretados em modo são:

r (read) - leitura
w (write) - escrita
a (append) - escrita ao final do existente
t (text) - modo texto
b (binary) - modo binário
+ (r+ ou w+) - ler e escrever ou escrever e ler

Exemplo

```
FILE* fp;
fp = fopen("entrada.txt", "rt");
if (fp == NULL)
    printf("Erro na abertura do arquivo!\n");

fclose(fp); /*fecha o arquvo*/
```

Leitura de arquivos em modo texto

```
/* semelhante ao scanf */
int fscanf (FILE* fp, "%formato", &variavel);

/* lê um caractere */
int fgetc(FILE* fp);

/* lê um número definido de caracteres */
char* fgets(char* s, int n, FILE* fp);
```

Escrita de arquivos em modo texto

```
/* semelhante ao printf */
int fprintf(FILE* fp, "%formato", variavel);

/* escreve um caractere */
int fputc(int c, FILE* fp);
```

Leitura caractere a caractere

```
int c;
int nlinhas = 0;
FILE *fp;

while ((c = fgetc(fp)) != EOF) {
    if (c == '\n')
        nlinhas++;
}
```

Leitura linha a linha

```
char linha[122];
FILE* fp;

while (fgets(linha,121,fp) != NULL) {
    ...
}

while (fscanf(fp, "%s", linha) == 1){
    ...
}
```

Exercício

Faça um programa que leia o conteúdo de um arquivo de texto e crie um outro arquivo com o mesmo conteúdo com todas as letras minúsculas convertidas em maiúsculas.

Dica: use a função toupper.

```
#include <ctype.h>
int c = 'a';
toupper(c);
```

Arquivos Binários - Motivação

- Variáveis int e float tem tamanhos fixos na memória
- Representação em texto precisa de um numero variável de dígitos
- Armazenamento em arquivos análogo ao utilizado em memória permite :
  - reduzir o tamanho do arquivo
  - permitir busca não sequencial

Leitura de arquivos em modo binário

int fread(void* p, int numBytes, int count, FILE* fp);

- p - Ponteiro para região de memória que receberá os dados do arquivo
- numBytes - Numero de bytes a ler do arquivo
- count - Determina quantos itens (cada um de comprimento numBytes) serão lidos
- retorno - numero de itens lidos

Escrita em arquivos de modo binário

```
int fwrite(void* p, int numBytes, int count, FILE* fp);
```

- p - Ponteiro para região de onde as informações serão escritas no arquivo
- numBytes - Numero de bytes a escrever no arquivo
- count - Determina quantos itens (cada um de comprimento numBytes) serão escritos
- retorno - numero de itens escritos

Usando fread() e fwrite()

```
FILE *fp;
double d = 12.23;

if((fp = fopen("test", "wb+")) == NULL)  {
    printf("arquivo nao pode ser aberto\n");
    exit(1);
}

fwrite(&d, sizeof(double), 1, fp);
rewind(fp);
fread(&d, sizeof(double), 1, fp);

printf("%f ", d);

fclose(fp);
```

- Uma das aplicações mais uteis para fread() e fwrite() consiste em ler ou escrever tipos de dados definidos pelo usuário (estruturas).

```
struct conta {
        float saldo;
        char nome[20];
} minhaConta;

// escreve o conteúdo de minhaConta em fp
fwrite(&minhaConta, sizeof(struct conta), 1, fp);
```

E/S com acesso aleatório

```
int fseek(FILE* fp, long numBytes, origem);
```

- numBytes é o numero de bytes a partir da origem que se tornará a nova posição corrente do cursor.
- origem é uma das seguintes constantes definidas em stdio.h :
  - SEEK_SET - Inicio do arquivo
  - SEEK_CUR - Posição atual
  - SEEK_END - Final do arquivo

As funções mais comuns do sistema de arquivos

Nome	Função
fopen()	Abre um arquivo
fclose()	Fecha um arquivo
putc()	Escreve um caractere em um arquivo
fputc()	O mesmo que putc()
getc()	Lê um caractere de um arquivo
fgetc()	O mesmo que getc()
fseek()	Posiciona o arquivo em um byte específico
fprintf()	É para um arquivo o que **printf()** é para o console
fscanf()	É para um arquivo o que **scanf()** é para o console
feof()	Devolve **verdadeiro** se o fim de arquivo for atingido
ferror()	Devolve **verdadeiro** se ocorreu um erro
rewind()	Recoloca o indicador de posição de arquivo no início do arquivo
remove()	Apaga um arquivo
fflush()	Descarrega um arquivo

Exercício

Elabore um programa que leia dois arquivos texto(salbruto.txt e desc.txt) contendo, respectivamente, os salários brutos e os descontos de até n funcionários. Para cada combinação de salário bruto e desconto criar uma estrutura funcionário com os campos nome, salário bruto, desconto e salário liquido (sal. bruto – descontos) e gravá-lo no arquivo binário funcionarios.bin.

Referências

- **Sites**
  - PUCRS - http://www.inf.pucrs.br/~pinho/LaproI/Historico/Historico.htm - Acesso em 11/02/2011
  - Wikipédia - http://pt.wikipedia.org/wiki/C_(linguagem_de_programa%C3%A7%C3%A3o) - Acesso em 11/02/2011
  - http://en.wikipedia.org/wiki/ANSI_C
  - http://en.wikipedia.org/wiki/C_(programming_language)#ANSI_C_and_ISO_C
  - http://www.dei.isep.ipp.pt/~abarros/docs/ANSI_C.pdf
  - Tutorial de Ponteiros e Arrays em C Ted Jensen (Tradução César A. K. Grossmann) http://cesarakg.freeshell.org/pointers.html#toc14
  - http://www.apostilando.com/download.php?cod=2686&categoria=C%20e%20C++
  - http://www.joinville.udesc.br/portal/professores/adriano/materiais/ListaRevis_oPonteiros.html
  - http://equipe.nce.ufrj.br/adriano/c/apostila/ponte.htm#exercicios

- **Livros**
  - SHEIDT, Herbert - C Completo e total.
  - CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas - Introdução a estrutura de dados.

- **Outros**
  - Slides de Estrutura de Dados I - Prof. André Vignatti - DAINF - UTFPR
  - Lista de exercícios sobre a linguagem C - Prof. Robson Ribeiro Linhares - DAINF - UTFPR