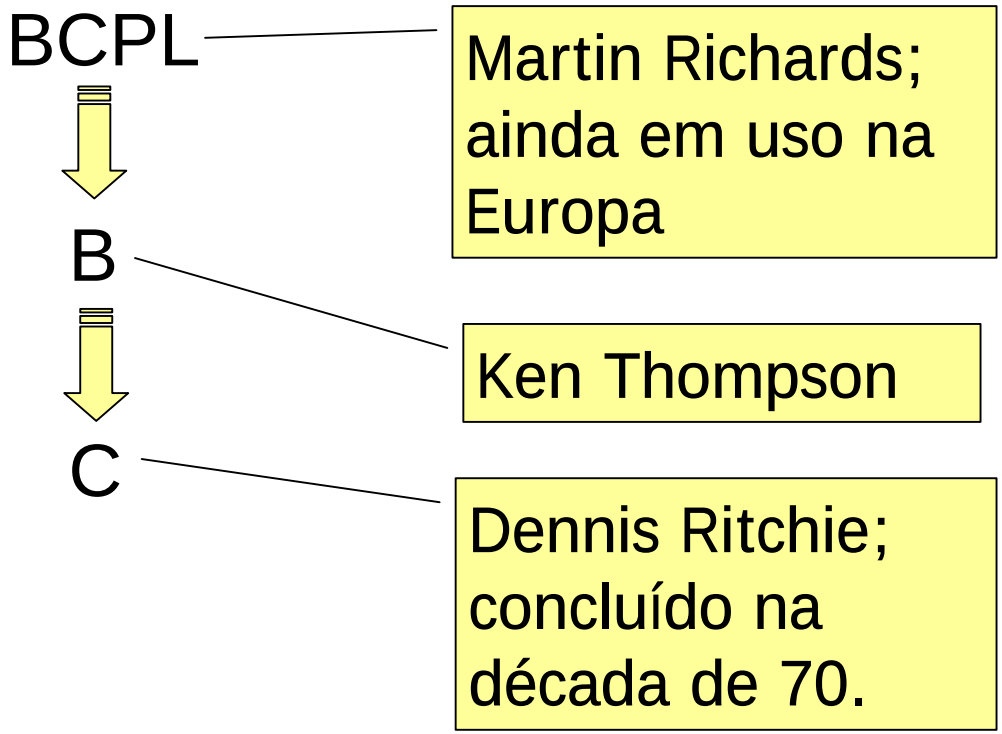
LINGUAGEM C

CONCEITOS BÁSICOS

HISTÓRICO

- Necessidade de escrever o Unix
- Evolução:



RECURSOS AVANÇADOS

- Hardware
  - acesso à porta serial;
  - acesso à memória de vídeo (aplicações gráficas);
  - acesso à memória RAM: manipulação de segmentos e overlays;
  - acesso à rede de comunicação;

RECURSOS AVANÇADOS

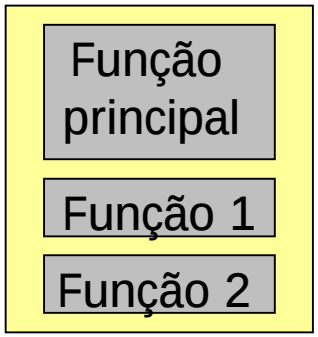
- Software
  - criação de processos em multitarefa;
  - comunicação inter-processos:
    - Semáforos;
    - Filas de mensagens;

APLICAÇÕES

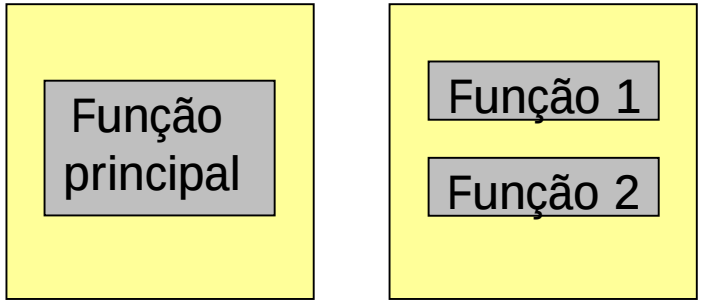
- Desenvolvimento de software básico;
- Complemento de aplicativos :
  - rotinas mais rápidas;
- Aplicações cliente-servidor;
- Rotinas de baixo nível para prover baixo tempo de resposta;

UM PROGRAMA EM C

- Uma função principal e zero ou mais funções auxiliares;
- Distribuição:



1 arquivo fonte



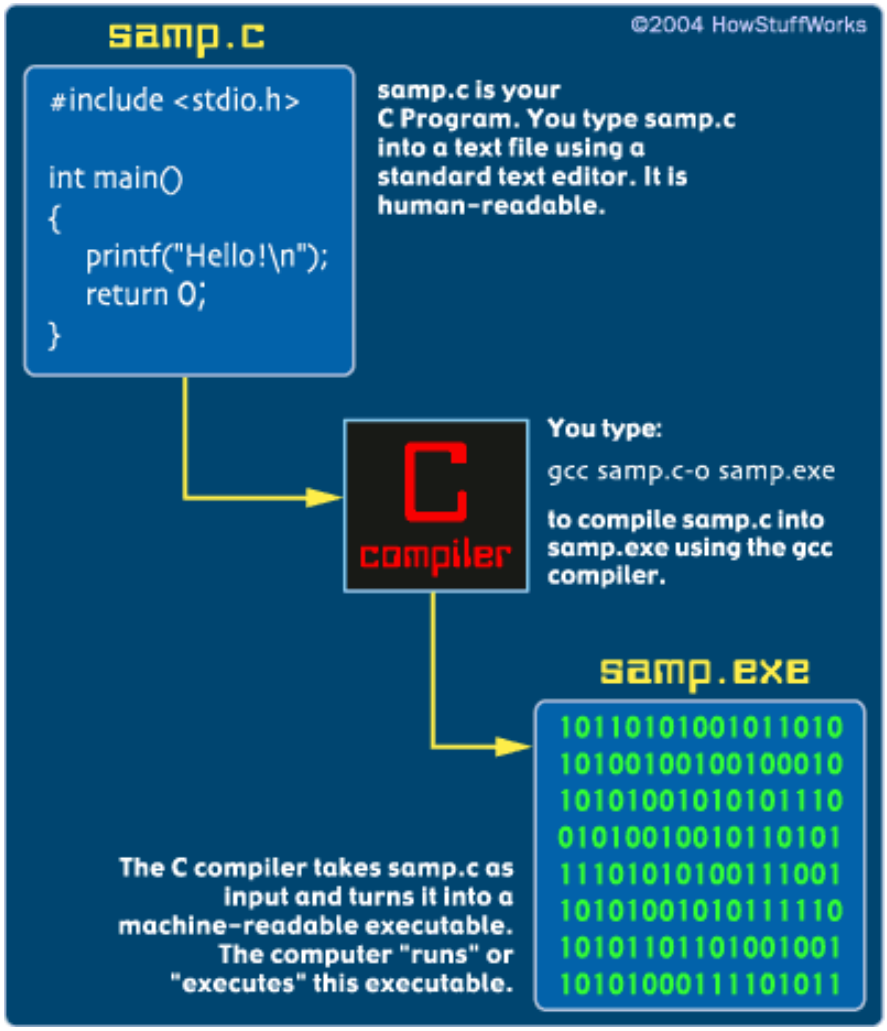
vários arquivos fonte

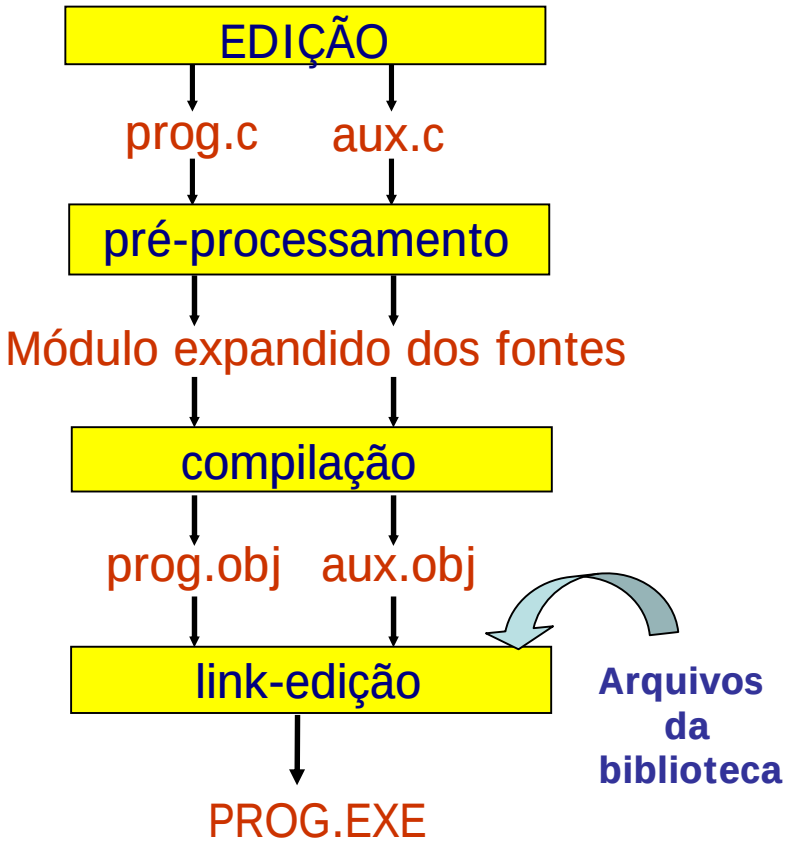
UM PROGRAMA EM C

- Tipos de arquivos:
  - fonte
    - contêm código;
    - extensão *c* ou *cpp*
  - cabeçalho
    - definições e declarações - extensão *h*
    - vários são oferecidos pelo compilador
    - o programador pode criar os seus
    - são incluídos no fonte através do comando

 # include <nome_do_arquivo.h>

Geração do Programa C





www.harpia.eng.br

rafaelaraújo

Estrutura do Programa C

```
/* COMENTÁRIOS */

definições de pré-processamento
declarações de funções;
declarações de variáveis globais e arrays;

main ( )
{
    declarações de variáveis locais e arrays;
    inicializações;

    /* COMENTÁRIOS */

    comandos;
    chamadas a funções;
}

    < código das funções >
```

Definição de constantes e inclusão de arquivos cabeçalho - .h

Todas as linhas de comando terminam em ;

Comandos de bloco são delimitados por {}

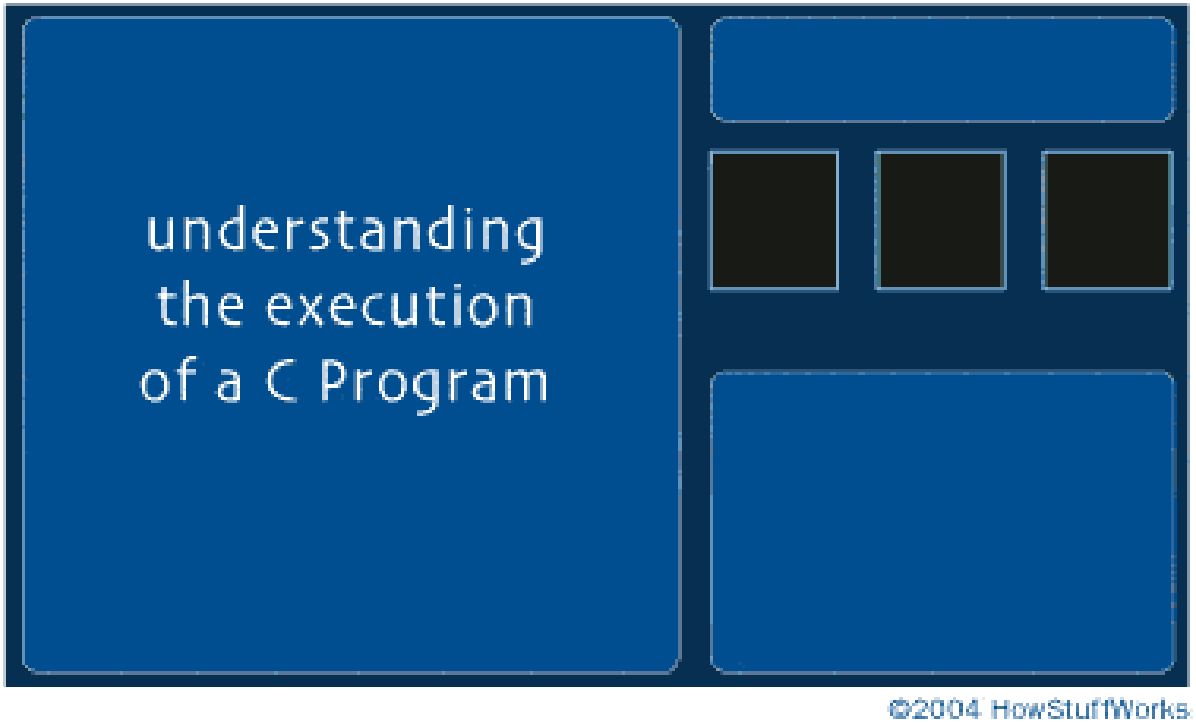
Exemplo 1 - *Hello World*

```
/*      Primeiro programa C
          arquivo - pgrad1.c          */

# include <stdio.h>

main ( )
{
   printf ("Hello World !");
}
```

Exemplo 2 – *How stuff works*



LINGUAGEM C

Introdução

VARIÁVEIS

» Forma geral para declaração (criação):

tipo lista de variáveis;

Que tipo de valor a(s) variável(is) vai(ão) guardar

Lista com o nome de cada variável que está sendo criada, sendo os nomes separados por vírgula

Variáveis

- Tipos Básicos
  - char
    - 1 caracter - 1 byte
  - int
    - inteiro - 2 a 4 bytes
  - float
    - real - 4 bytes, precisão de 6 dígitos
  - double
    - real - 8 bytes, precisão de 10 dígitos

Variáveis

» Modificadores de Tipo

- Podem ser aplicados aos tipos básicos para criar extensões a eles:
  - **long**
    - ≈ inteiro longo com sinal - >= int
  - **short**
    - ≈ inteiro curto com sinal - <= int (pouco usado)
  - **unsigned**
    - ≈ inteiro sem sinal
  - **signed**
    - ≈ inteiro com sinal - redundante

Variáveis

» Combinações válidas:
- long int
  - em geral 4 bytes
- long double
  - 16 bytes, precisão de 10 dígitos
- unsigned long
  - em geral 4 bytes
- unsigned char
  - 1 byte
- unsigned int
  - 2 bytes

Variáveis

- Nome:
  - tamanho depende do ambiente
  - maiúsculas, minúsculas, números e "_"
  - na prática:
    - minúsculas: variáveis, funções, comandos
    - maiúsculas: constantes
  - exemplos:

```
int x, y, z;
char opcao;
double Pi;
unsigned long infinito;
long NUM, num;
unsigned char _n1, _n2;
```

Variáveis

» Inicialização

- pode ser na própria declaração

```
int x = 0;
double Pi = 3.1415;
```

- pode-se inicializar mais de uma variável ao mesmo tempo

```
int x = 0, y = 10;
```

- pode misturar declaração e inicialização na mesma linha:

```
int x, y = 10, z;
```

Constantes

- Há duas formas de definir constantes:
  - usando definição de pré-processamento
    - *define*
  - com o modificador de tipo *const*

Constantes

» Com # define

```
# define PRIMEIRO    '1'              /* caracter */
# define VINTE        20              /* inteiro */
# define MES          ”Janeiro”       /* string */
# define TRINTA       036             /* octal */
# define TRINTA       0x1E            /* hexa */
# define PI           3.1415          /* real */
```

- Obrigatoriamente esse tipo de definição é feito no início do arquivo fonte ou cabeçalho, antes de qualquer comando ou declaração

Constantes

- Com const
  - Forma geral:

    const tipo nome = valor;
  - Exemplo:

    const int y = 2;
  - Esse tipo de definição pode ser feita
    - dentro e fora do main
    - dentro e fora de funções

Operadores

- Aritméticos
  - comuns: + - * / =
  - resto da divisão de x por y : x%y

    ```
    r = x % y;     /* se x é 7 e y é 2, r = ? */
    ```
  - atribuição:

    ```
    a = 2;
    b = a = 2;
    i = i + 2;  => i += 2;
    ```
  - a divisão de **int** por **int** é sempre **int**; se o resultado for fracionário, será truncado

Operadores

- Incremento e Decremento
  - i++ (pós-incremento):
    - utiliza variável e incrementa;
  - ++i (pré-incremento):
    - incrementa variável e utiliza;
  - i– (pós-decremento);
  - --i (pré-decremento).

Operadores

» Incremento e Decremento

- Exemplo:

```
int a, b = 10;
a = b++;
```

F valor de a? valor de b?

```
a = ++b;
```

F valor de a? valor de b?

- Pode-se também utilizar o recurso em contexto isolado:

```
a++;    /* incrementa de 1 a variável a */
```

E/S Formatada

- Saída: printf
  - copia dados com formatação para a saída padrão
  - sintaxe:

```
printf ("controle", lista de variáveis);
```

Texto +
Formatação de variável +
caracteres de controle

Todas as variáveis que serão enviadas para a saída, na mesma ordem em que aparecem seus formatos na cadeia de controle

E/S Formatada

- Formatos de variável
  - %d – inteiros;
  - %ld – inteiros longos;
  - %o – octal;
  - %x – hexadecimal;
  - %u – inteiro sem sinal;
  - %c – caracter;
  - %f – real, com 6 decimais;
  - %lf – real, com 10 dígitos;
  - %s – cadeia de caracteres;

E/S Formatada

- Caracteres de Controle
  - \n – próxima linha, 1ª coluna;
  - \t – tabulação - 8 colunas;
  - \a – sinal sonoro;
  - \b – *backspace.*

E/S Formatada - Exemplo

```
# include <stdio.h>
main ()
{
  int i = 65;
  printf ("Valor em decimal: %d\n", i);
  printf ("Caracter correspondente: %c\n", i);
  printf ("Valor em octal: %o\n", i);
  printf ("Valor em hexadecimal: %x\n", i);
}
```

E/S Formatada

» Detalhes importantes:

- Como limitar o número de decimais:

  "%.nf“

  – n é o número de decimais desejado

- Como alinhar inteiros pela direita:

  "%nd"

  – n é a largura do alinhamento

- Como alinhar reais pela direita:

  "%x.yf"

E/S Formatada

- Entrada: scanf
  - lê dados com formatação da entrada padrão
  - sintaxe:

    scanf ("controle", lista de variáveis);

Apenas formatação de variável

Todas as variáveis que serão lidas da entrada, na mesma ordem em que aparecem seus formatos na cadeia de controle, mas precedidas de &, seu endereço de memória

E/S Formatada - Exemplo

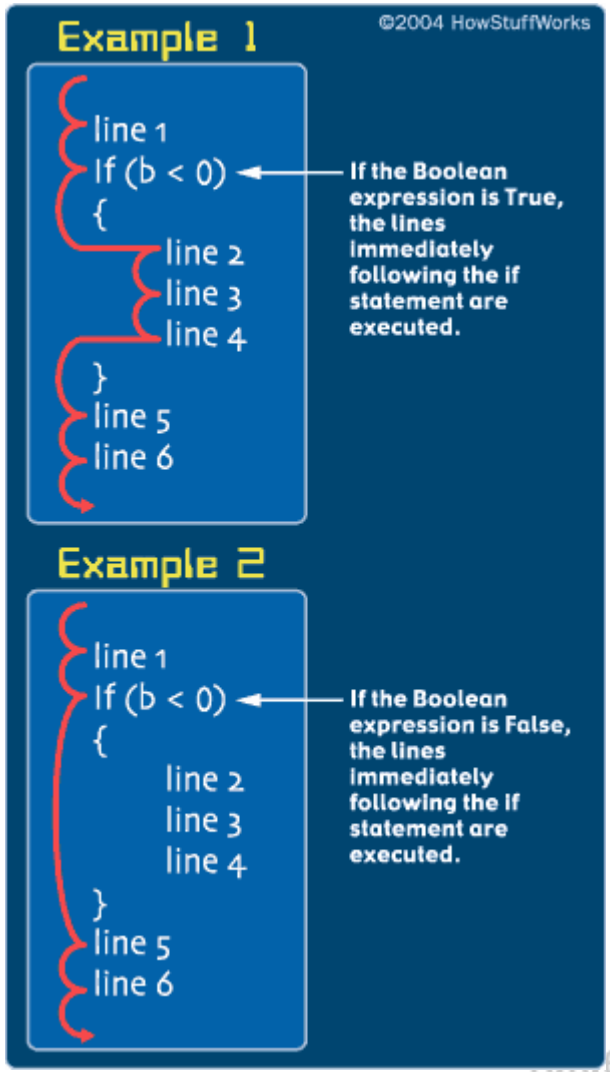
» Ler data no formato "dd/mm" e listar dia e mês separadamente

```
# include <stdio.h>
main ()
{
   int dia, mes;
   scanf ("%d/%d", &dia, &mes);
   printf ("\nDia: %d\nMes: %d\n", dia, mes);
}
```

Controle de fluxo – Condicional

» COMANDO: if ... else

```
if (expressão)
{
        comandos;
}
else
{
        comandos;
}
```



Operadores

- Relacionais

  > < <= >= == !=

  Cuidado!

  if (a = b)

  î faz a = b e verifica se a é TRUE

  if (a == b)

  î verifica se a é igual a b

Operadores

» Lógicos

AND - && OR - || NOT - !

Exemplo:

```
if (a == b  &&  a != 0)
    printf ("a diferente de zero");
```

Controle de fluxo - Exemplo

» Ler data (dd/mm) e listar dia e mês separados; mensagem de erro se dia e/ou mês inválidos.

```
# include <stdio.h>
main ()
{
   int dia, mes;
   scanf ("%d/%d", &dia, &mes);
   if ((dia < 1 || dia > 31) || (mês < 1 || mês > 12))
     printf ("\n\aERRO na digitacao\n");
   else
     printf ("\nDia: %d\nMes: %d", dia, mes);
}
```

Controle de fluxo – Condicional

» COMANDO: switch ... case

```
switch (argumento)
  {
    case const1:
      comandos;
      break;
    case const2:
      comandos;
      break;
    default:
      comandos;
  }
```

pode ser uma expressão algébrica, uma chamada a função, ou uma variável

um valor constante ou uma constante simbólica

se não existir, os case's seguintes serão executados de qualquer forma

Controle de fluxo - Repetição

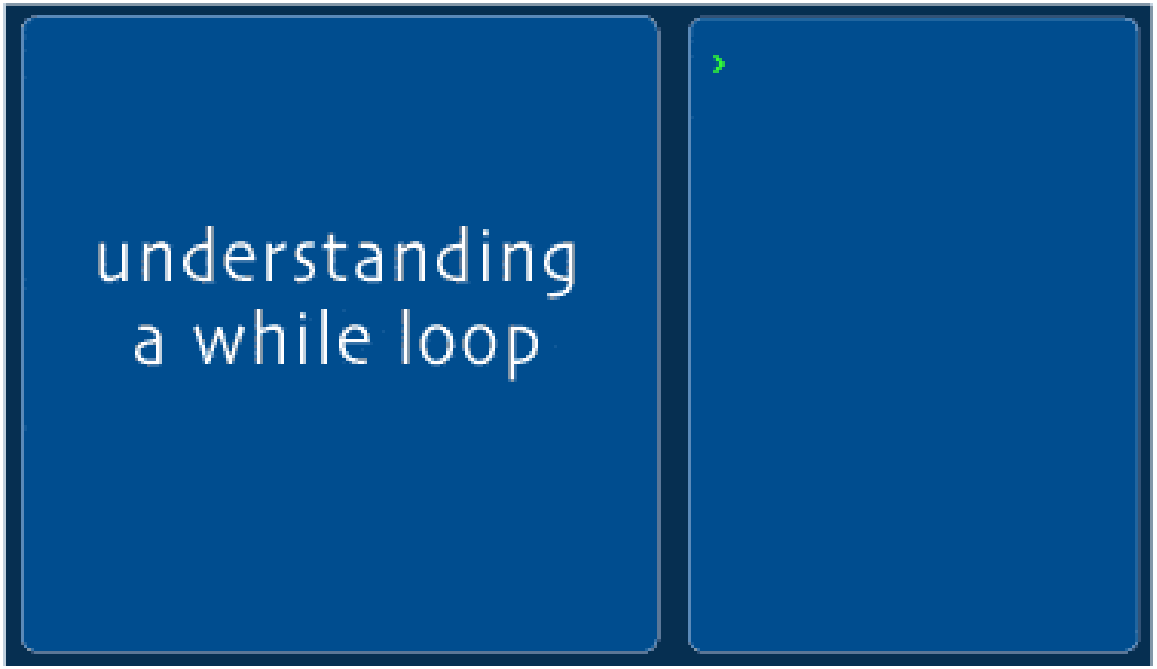
» COMANDO: while

```
while (condição)
{
   comandos;
}
```

Condição para que o loop continue.

CUIDADO COM OS LOOPS INFINITOS!

Controle de fluxo - Repetição



Controle de fluxo - Exemplo

» Ler inteiros até que um número negativo seja digitado e mostrar se cada um é par ou ímpar

```
main ()
{
   int num;
   scanf ("%d", &num);
   while (num > = 0)
   {
                if (num%2 == 0)
                        printf ("%d e' par ", num);
                else
                        printf ("%d e' impar ", num);
                scanf ("%d", &num);
    }
}
```

Controle de fluxo - Exemplo

- Programa para ler números reais, calcular a soma e listar o resultado.
- O usuário pode escolher entre inclusão ou listagem parcial da soma.
- Ao selecionar *final*, listar o resultado final.

```
main ()
{
   char op;
   float soma=0,
num;
   scanf ("%c",
&op);
   while (op != 's')
   {
      switch (op)
      {
         case 'i':
         scanf ("%f",
&num);
         soma +=
num;
         break;
         case 'l':
               printf ("%.2f", soma);
               break;
         default:
               printf ("\nOpcao Invalida\n");
      }
      scanf ("%c", &op);
   }
   printf ("\nSoma total: %.2f", soma);
}
```

Controle de fluxo - Repetição

» COMANDO: do/while

```
do
{
   comandos;
}
while (condição);
```

Garante que os comandos sejam executados ao menos uma vez.

Condição para que o loop continue.

Controle de fluxo - Exemplo

» Ler inteiros até que um número negativo seja digitado e mostrar se cada um é par ou ímpar

```
main ()
{
        int num;
        do
        {
                scanf ("%d", &num);
                if (num%2 == 0)
                        printf ("%d e' par ", num);
                else
                        printf ("%d e' impar ", num);
        }
        while (num > = 0);
}
```

Controle de fluxo

» COMANDO: for

```
for (inicialização; condição; incremento)
{
  comandos;
}
```

Valor inicial da variável que irá controlar o for

Condição para que o for continue; o mesmo que while

Incremento da variável de controle do for

Controle de fluxo - Exemplo

» Programa para calcular a soma dos números pares de 2 a 100

```
# include <stdio.h>
main ()
{
        int i, soma = 0;
        for (i = 2; i <= 100; i += 2)
         {
                soma += i;
        }
         printf ("Soma: %d\n", soma);
}
```

Exercícios – Introdução – 1ª Parte

- Programa para pedir cinco números e imprimir o quadrado de cada um;
- Programa para imprimir todos os numeros de acordo com o limite inferior e superior inseridos pelo usuário;
- Programa para ler a quantidade de números que serão digitados, em seguida ler os números e imprimir na tela o maior e o menor dentre os números digitados.
- Programa para ler 200 números inteiros e imprimir quantos são pares e quantos são impares;
- Programa para criar a tabuada de multiplicação que o usuário deseja imprimir na tela;

Exercícios – Introdução – 2ª Parte

- Chico tem 1,5 m e cresce 2 cm por ano, enquanto Juca tem 1,1 m e cresce 3 cm por ano. Desenvolver um programa para calcular e imprimir quantos anos serão necessários para que Juca seja maior que Chico;
- Uma das maneiras de se conseguir a raiz quadrada de um número é subtrair do número os ímpares consecutivos a partir de 1, até que o resultado da subtração seja menor ou igual a zero. O número de vezes que se conseguir fazer a subtração é a raiz quadrada exata (resultado 0) ou aproximada do número (resultado negativo). Criar um programa para calcular a raiz quadrada segundo esta metodologia.
- Na usina de Angra dos Reis, os técnicos analisam a perda de massa de um material radioativo. Sabendo-se que este perde 25% de sua massa a cada 30 segundos, criar um programa que imprima o tempo necessário para que a massa deste material se torne menor que 0,10 gramas. O programa pode calcular o tempo para várias massas.

LINGUAGEM C

Funções e Escopo de Variáveis

Funções

- Trecho de um programa independente com um objetivo determinado, simplificando o entendimento do programa e proporcionando menores chances de erro e complexidade;
- Justificativa:
  - Dividir e estruturar um programa em partes logicamente coerentes;
  - Facilidade em testar os trechos em separado;
  - Maior legibilidade de um programa;
  - O programador cria sua própria biblioteca de funções;
  - Evita que uma certa seqüência de comandos seja escrita repetidamente.

Funções

- Uma função é um bloco contendo início e fim, sendo identificada por um nome, pelo qual será referenciada em qualquer parte e em qualquer momento do programa;
- Podem estar em um ou mais arquivos;
- O uso de funções envolve dois passos:
  - Declaração:
    - antes do main ou no próprio main;
  - Definição:
    - normalmente após o main ou em outro arquivo;

Funções - Definição

- É o código da função e deve seguir a sintaxe:

```
tipo nome (declaração de parâmetros)
{
  declaração de variáveis;
  comandos;
}
```

- A linha com o nome da função é idêntica à declaração

Funções - Declaração

» Indica o formato da função:

tipo nome (lista de parâmetros);

- ***Tipo***: tipo de dado retornado pela função:
  - se a função não retorna nada î void;
- ***Parâmetros***: lista de tipos que serão passados como argumentos para a função:
  - se a função não recebe nada î void;

Funções - Definição

» Como retornar valor de uma função?

- comando *return*

```
return (expressao);  /* os ( ) são opcionais */
```

» Se a função não retorna valor:

```
return;
```

- o uso do return aqui é opcional.

Funções - Exemplo 1

» Cálculo da soma de dois números

```
int soma (int a, int  b);

main ()
{
   int n1, n2, res;

   scanf ("%d%d", &n1, &n2);
   res = soma (n1, n2);
   printf (”Soma de %d e %d: %d", n1, n2, res);
}
```

Funções - Exemplo 1

» Cálculo da soma de dois números

```
int soma (int a, int b)
{
   int x;

   x = a + b;
   return x;
}
```

Funções – Exemplo 2 – Arrays

» Ler array de inteiros e calcular média

```
void le_vet (int []);
float media (int []);

main ()
{
   int numeros [5];
   float res;
   le_vet (numeros);
   res = media (numeros);
   printf ("Media: %.2f", res);
}
```

Funções – Exemplo 2 – Arrays

```
void le_vet (int vet [])
{
   int i;
   for (i = 0; i < 5; i++)
     scanf ("%d", &vet [i]);
   return;
}
```

```
float media (int vet [])
{
   float med = 0;
   int i;
   for (i = 0; i < 5; i++)
         med += vet [i];
   med /= 5;
   return med;
}
```

Escopo de variáveis

- O C dispõe das seguintes classes de armazenamento de variáveis:
  - Automáticas;
  - globais*;
  - Externas;
  - Estáticas;
- Essas classes definem o escopo das variáveis.

Escopo de Variáveis

- Automáticas:
  - variáveis locais e parâmetros de funções;
  - válidas apenas na função em que são declaradas;
  - o valor é perdido quando a função termina;
- Globais:
  - válidas no arquivo fonte onde são criadas;
  - declaradas fora de qualquer função.

Escopo de Variáveis – Exemplo

```
void soma (int a, int b);
int res;
main () {
   int n1, n2;
   scanf ("%d%d", &n1, &n2);
   soma (n1, n2);
   printf ("Soma de %d e %d: %d", n1, n2, res);
}

void soma (int a, int b)
{
   res = a + b;
}
```

Escopo de Variáveis

» Externas

- válidas em todos os arquivos fonte compilados que farão parte do executável;
- devem ser declaradas em todos os fontes onde serão usadas:
  - no primeiro código fonte: tipo nome;
  - nos demais arquivos: extern tipo nome;
- exemplo:
  - programa de cálculo de média de um array.

Escopo de Variáveis – Exemplo

```
/* arquivo principal.c */

        void le_vet (void);
        float media (void);

        int numeros [5];
        float res;

        main ()
        {
           le_vet ();
           res = media ();
           printf ("Media: %.2f", res);
        }
```

Escopo de Variáveis – Exemplo

```
/* arquivo funcao.c */

extern int numeros [];
int i;

void le_vet (void)
{
   for (i = 0; i < 5; i++)
     scanf ("%d", &numeros [i]);
   return;
}
```

```
float media (void)
{
   float med = 0;
   for (i = 0; i < 5; i++)
     med += numeros [i];
   med /= 5;
   return med;
}
```

Escopo de Variáveis

» Estáticas

- declaração:

  static tipo nome;

- úteis quando se deseja utilizar o último valor de uma variável em uma próxima chamada à função:
  - não perdem o valor quando a função termina.
- exemplo:
  - série numérica.

Escopo de Variáveis – Exemplo

```
serie (void)
{
    static int num_serie = 10;

    num_serie = num_serie + 23;
    return (num_serie);
 }
```

- A cada chamada à função, num_serie resgatará o último valor armazenado.

Escopo de Variáveis

» Estáticas

- usadas também para inicializar arrays em funções

```
void inic_array (int impar [])
{
    static int impar [] = {1, 3, 5, 7, 9};
}
```

Funções – Exercícios

- Considerando que o compilador utilizado não contém a função de potenciação pow, escrever uma função para calcular “x” elevado a potência “n”;
- Escrever uma função para calcular o fatorial de um número dado pelo usuário;
- Escrever uma função que receba três números e coloque-os em ordem crescente;
- Escrever uma função menu para caixa de banco eletrônico (1-saldo; 2-depósito; 3-saque; 4-nova conta; 5-encerra conta; 6-sai do programa);

Automação dos protótipos

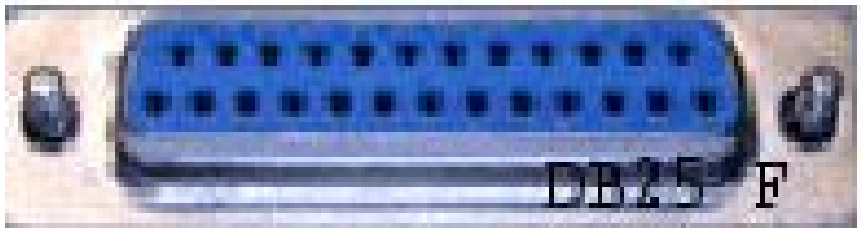
Linguagem C e Porta Paralela

www.harpia.eng.br

rafaelaraújo

Teoria de Porta Paralela

- Porta Paralela - interface de comunicação com periféricos (impressoras, scanners, câmeras de vídeo, unidade de disco removível e outros);
- Com algum conhecimento de programação e eletrônica, pode-se controlar um aparelho através da Porta Paralela do computador (conector DB25);



Teoria de Porta Paralela

- A porta paralela permite a alocação de 9 bits para entrada de dados e 12 bits para saída de dados.
- Com eletrônica básica faz-se o controle de dispositivos externos ou a leituras de sensores.
- Padronizada pela IEEE 1284 em 1994.

Teoria de Porta Paralela

- A porta paralela original possuía a capacidade para enviar bytes que representassem caracteres em ASCII para impressoras matriciais.

Barramentos I/O

- Envio ou recebimento de dados
  - Deve-se informar o endereço dos barramentos pelos quais a informação vai trafegar.
- PORTA PARALELA:
  - Endereço para o conjunto de 8 bits.
    - A informação (entrada ou saída) trafega sempre em 8 bits!
  - Valores utilizados pelo processador para identificar o caminho a ser utilizado:
    - Dados: 0x378 (hex) | 888 (dec)
    - Controle: 0x37A (hex) | 890 (dec)
    - Status: 0x379 (hex) | 889 (dec)

Barramentos de I/O

- Modos de Operação:
  - A porta paralela atualmente possui três modos de operações. São eles (IEEE 1284):
    - SPP (Standard Parallel Port) – bits de dados unidirecional;
    - EPP (Enhanced Parallel Port) – bits de dados bidirecional;
    - ECP (Extended Capabilities Port) – bits de dados bidirecional;
  - Estes modos de operação são configurados pelo BIOS Setup.
  - Diferenciam-se:
    - “Bidirecionabilidade”;
    - Velocidade de tranmissão;
    - Forma de transmissão;
    - Etc.

Barramentos de I/O

» Pinagem da porta paralela:

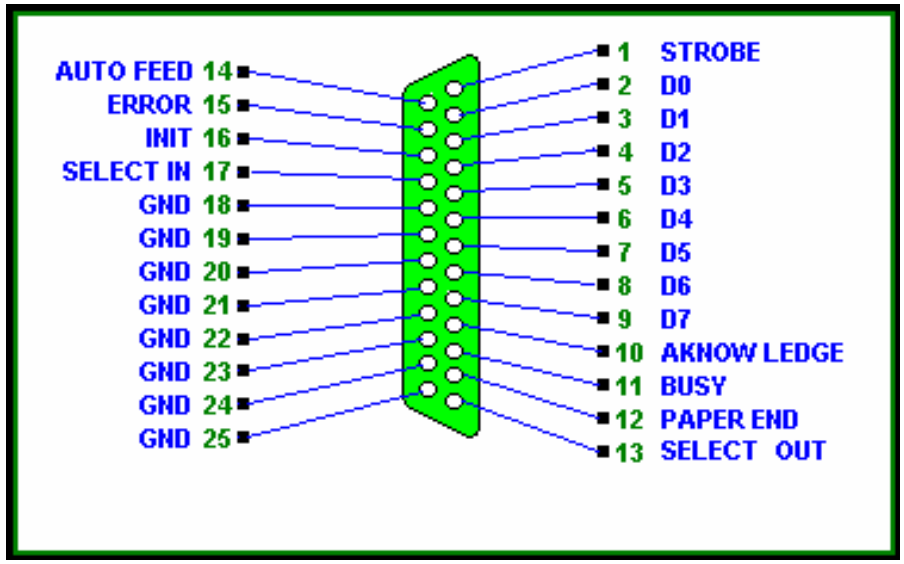
STATUS Endereço 379h		CONTROLE Endereço 37Ah		DADOS Endereço 378h	
D7	**11**	**D7**	**NC**	**D7**	**9**
D6	**10**	**D6**	**NC**	**D6**	**8**
D5	**12**	**D5**	**NC**	**D5**	**7**
D4	**13**	**D4**	**NC**	**D4**	**6**
D3	**15**	**D3**	**17**	**D3**	**5**
D2	**NC**	**D2**	**16**	**D2**	**4**
D1	**NC**	**D1**	**14**	**D1**	**3**
D0	**NC**	**D0**	**1**	**D0**	**2**

Padronização da Porta Paralela

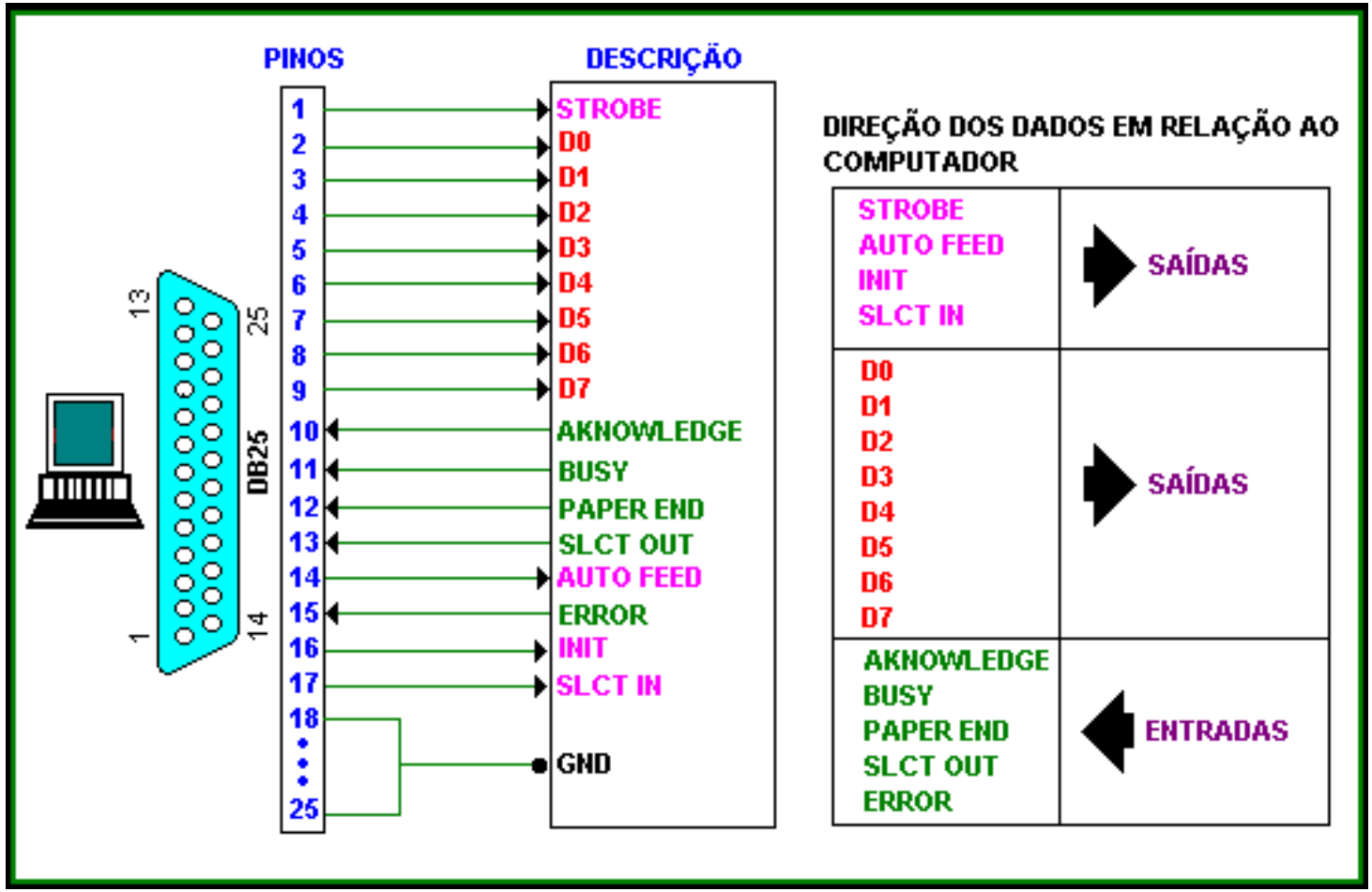
- Padronização IEEE 1284:
  - Define uma comunicação serial de dados:
    - Informação enviada em fila (FIFO);
    - Porta paralela pode ser considerada como uma sendo comunicação BYTE serial.
- Padronização:
  - Caracteres são enviados um a um como um conjunto de bits;
  - Conectores DB-25;
  - Recomendado para conexões curtas (<15 metros);
  - Os sinais variam de 3 a 15 volts positivos ou negativos;
    - +-5, +-10, +- 12 e +-15 volts
    - O nível lógico um é definido por ser voltagem negativa e o nível lógico zero é voltagem positiva.

Padronização da Porta Paralela

- Padronização:
  - Taxas de transmissão:
    - 300, 1200, 2400, 9600, 19200, etc (bits);
    - ambos os dispositivos devem estar configurados com a mesma velocidade;
  - Pinagem:

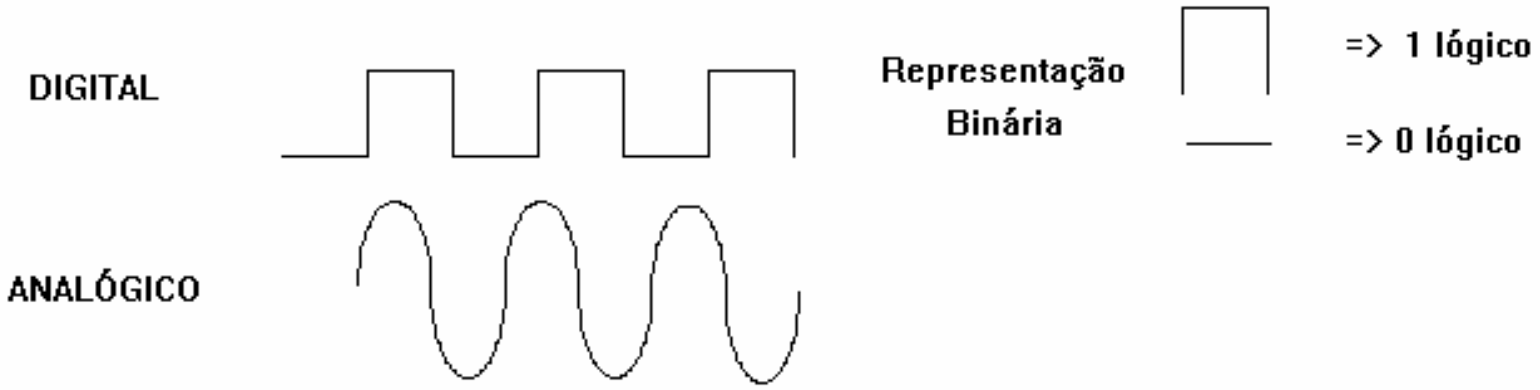


Conector DB25 – Pinagem



Formato do Sinal

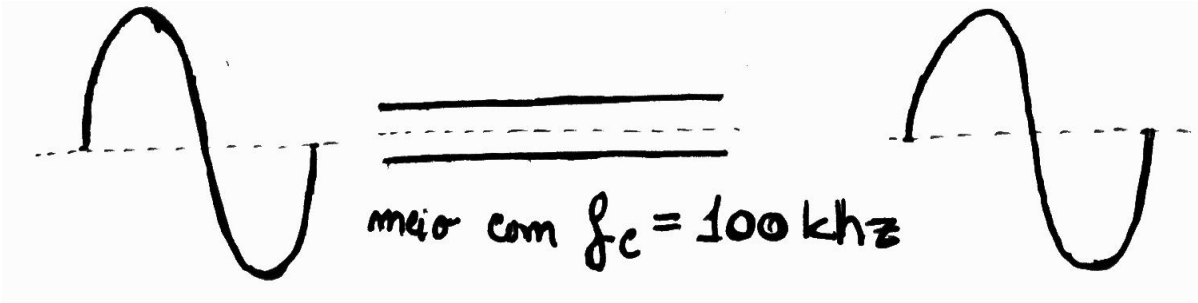
- Forma dos sinais:



- Sinal Analógico
  - Freqüência → ciclos/segundo [hz]
- Sinal Digital
  - Taxa de Transmissão → pulsos/segundo [bauds/s]
  - Se representa dois níveis lógicos → [bits/s]

Restrições à Passagem do Sinal

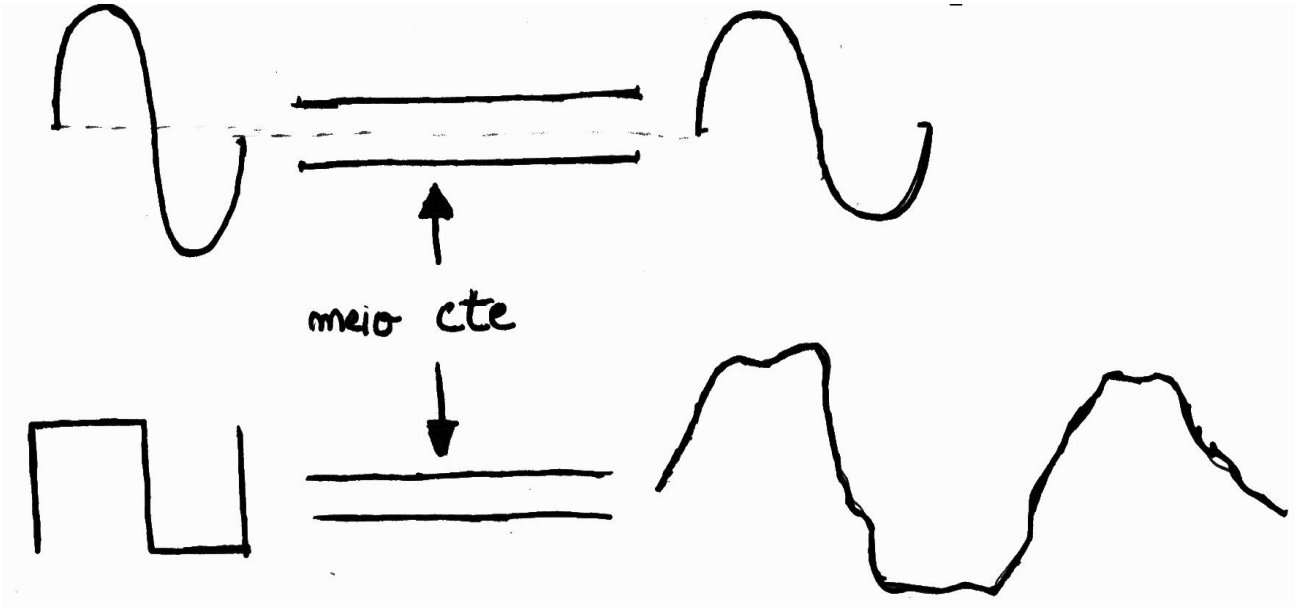
- Banda Passante e Freqüência de Corte (Filtro)
  - Todo meio de transmissão tem uma capacidade de passagem de “freqüências” estabelecida pela freqüência de corte
  - Assim um Sinal com F = 50 kHz:



- Sinal Analógico → Harmônico Fundamental
- Sinal Digital → ∞ Harmônicos
  - Fourrier decompôs matematicamente.

Restrição a passagem da informação

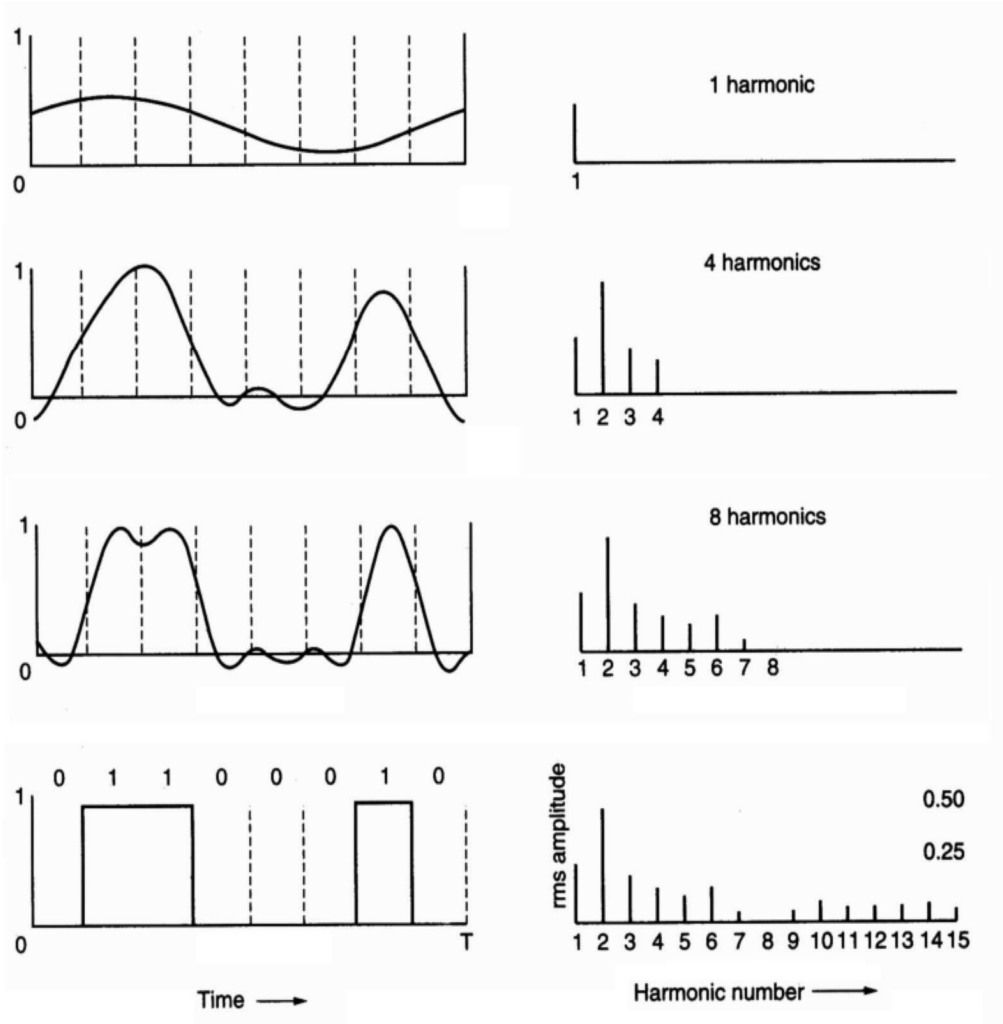
- Sinal Analógico: passa/não passa
  - pequena degeneração
- Sinal Digital tem perdas degenerativas



Restrição a passagem da informação

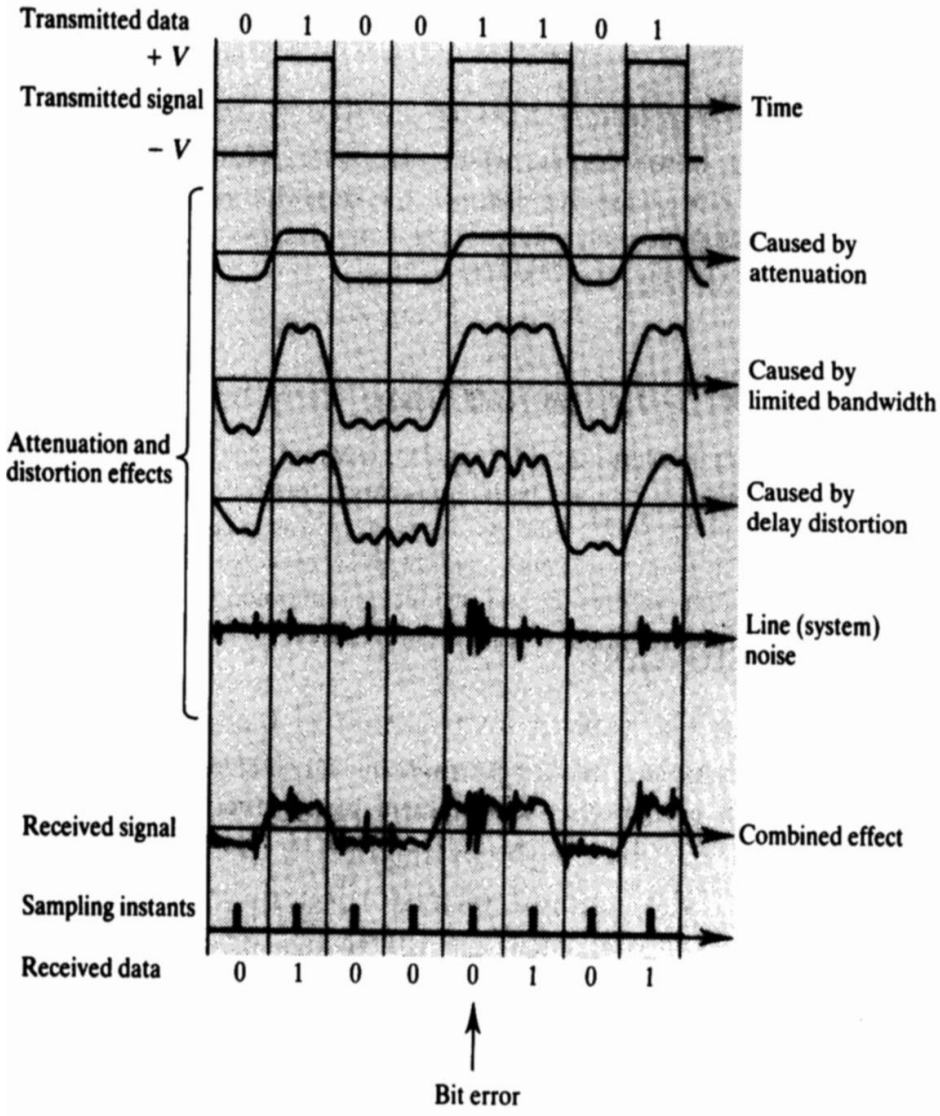
» Analisando o Espectro do Sinal Digital:

Bps	T (msec)	First harmonic (Hz)	# Harmonics sent
300	26.67	37.5	80
600	13.33	75	40
1200	6.67	150	20
2400	3.33	300	10
4800	1.67	600	5
9600	0.83	1200	2
19200	0.42	2400	1
38400	0.21	4800	0



Restrição a passagem da informação

- ***Conjunto das Restrições***:
- Atenuação (Atenuation)
- Filtro (Limited Bandwidth)
- Ruído (Noise)
- Atraso (Delay)
- Comparativo dos Atrasos
  - Satélite: 257 ms
  - Microondas: 3 µs/Km
  - Cabo Coaxial: 5 µs/Km
- Ruído Aleatório



www.harpia.eng.br

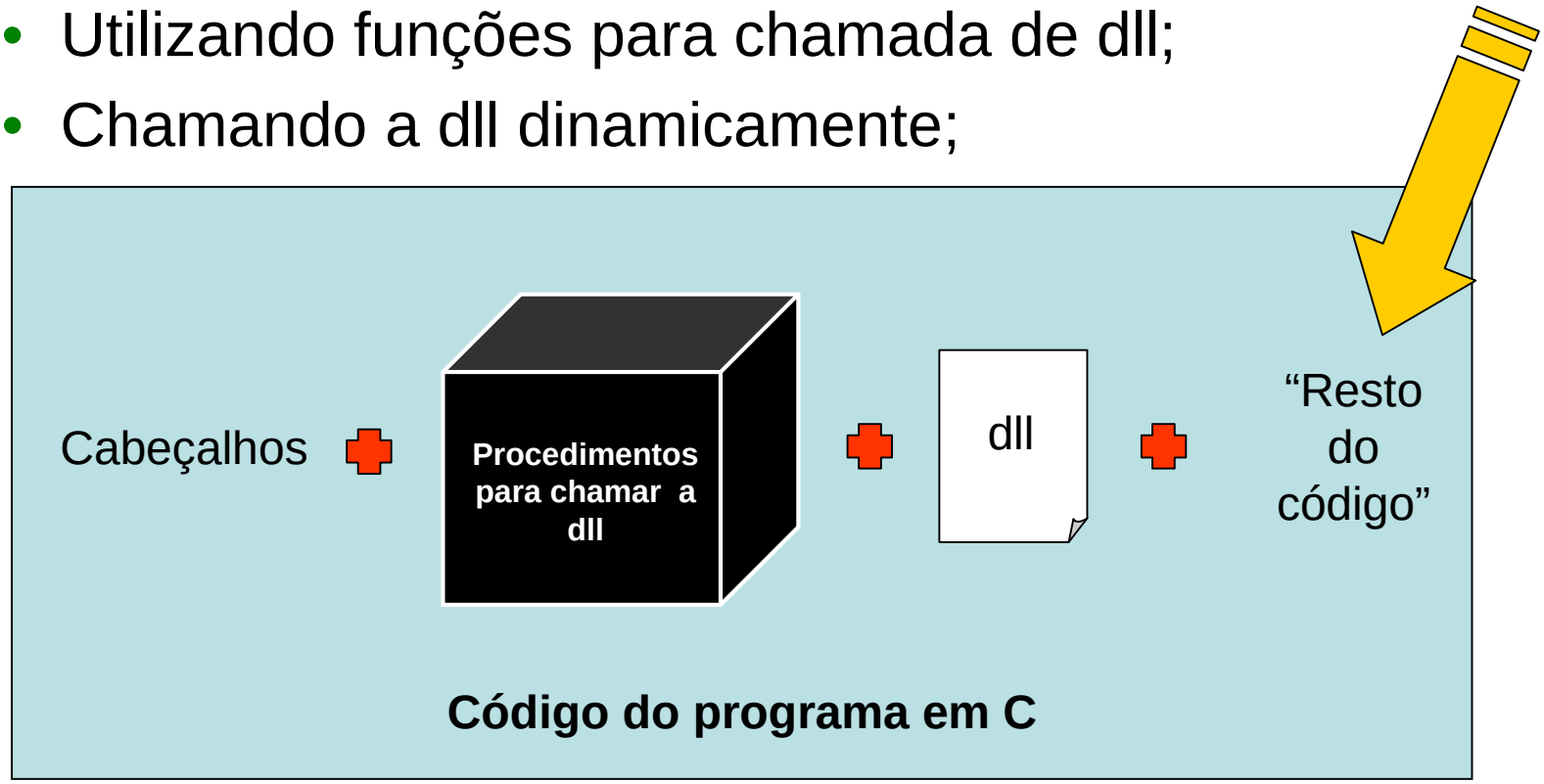
Acessando a porta paralela

- Sistema Operacional (“Maestro”);
  - HAL (*Hardware Abstraction Layer* – camada de abstração de hardware): permite o “*plug and play*”
  - Família Windows 9x , a maioria das linguagens de programação acessavam com facilidade a porta paralela através de funções nativas da própria linguagem ou via código assembler.
  - Windows NT/2000/XP não permitem o acesso direto a este tipo de porta. Nesse caso é necessário um driver de sistema.

Envio de Dados

» Como o compilador identifica as função de envio e recebimento de dados pela porta paralela?

- Utilizando funções para chamada de dll;
- Chamando a dll dinamicamente;



Acessando a porta paralela

- Chamada a DLL
  - Biblioteca será carregada dinamicamente durante a execução de um programa (quando se fizer necessária);
  - Carregar certas funcionalidades conforme a demanda;
  - Em alguns sistemas o programador precisa ser cauteloso para assegurar que a biblioteca seja carregada antes de ser chamada, enquanto outras também automatizam esse processo.
- A Inpout32.dll possui duas funcões: Out32 e Inp32.
  - Out32 escreve um valor (byte) num endereço de I/O
  - Inp32 lê um valor (byte) de um endereço de I/O.

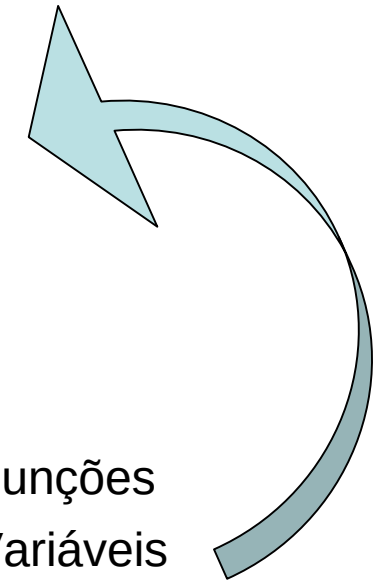
Chamada a dll (“caixa preta”)

```
#include <conio.h>
#include <process.h>
#include <windows.h>  //Necessário para: LoadLibrary(), GetProcAddress() e HINSTANCE.

//Declaração dos ponteiros para função.
typedef short _stdcall (*PtrInp)(short EndPorta);
typedef void _stdcall (*PtrOut)(short EndPorta, short valor);
HINSTANCE hLib; //Instância para a DLL inpout32.dll.
PtrInp inportB; //Instância para a função Imp32().
PtrOut outportB; //Instância para a função Out32().
int main(void)
{
	hLib = LoadLibrary("inpout32.dll"); //Carrega a DLL na memória.
	if(hLib == NULL) //Verifica se houve erro.
	{
			printf("Erro. O arquivo inpout32.dll não foi encontrado.\n");
			getch();
			return -1;
	}
	outportB = (PtrOut) GetProcAddress(hLib, "Out32"); //Obtém o endereço da função Out32 contida na DLL.
	if(outportB == NULL) //Verifica se houve erro.
	{
			printf("Erro. A função Out32 não foi encontrada.\n");
			getch();
			return -1;
	}
```

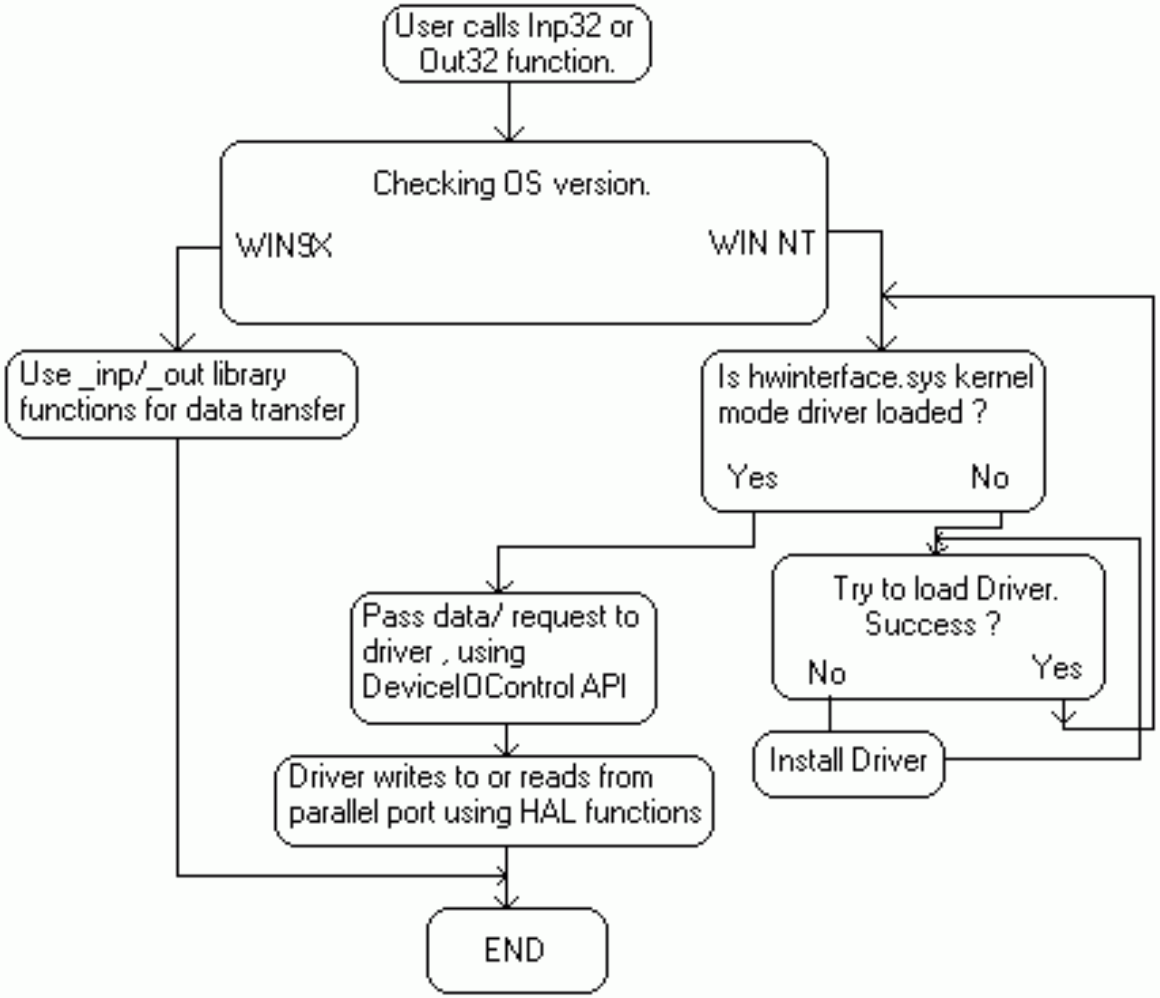
CÓDIGO!

```
	FreeLibrary(hLib); //Libera memória alocada pela DLL.
	return 0;
}
```



+ Funções
+ Variáveis
+ Cabeçalhos
+ implementações

Acessando a porta paralela



Envio de Dados

- O comando em C que atribui valores referente à porta desejada é o: outportB(ENDERECO, VALOR)
- Significado de tais parâmetros:
  - ENDERECO
    - São valores em hexadecimal que representa o endereçamento da porta desejada. Normalmente são:

Printer	Data Port	Status	Control
LPT1	0x378	0x379	0x37a
LPT2	0x278	0x279	0x37a
LPT3	-	-	-

Conector DB25 – Pinagem

» **Sobre os pinos de dados:** pinos de 2 à 9, normais, i.e., nível lógico alto (um) ativa. A letra D significa registrador de DADOS, e o número significa a posição do bit no byte.

Pinos de Dados								
Identificação	D7	D6	D5	D4	D3	D2	D1	D0
Pino	9	8	7	6	5	4	3	2
Posição	7	6	5	4	3	2	1	0
Byte	1	1	1	1	1	1	1	1

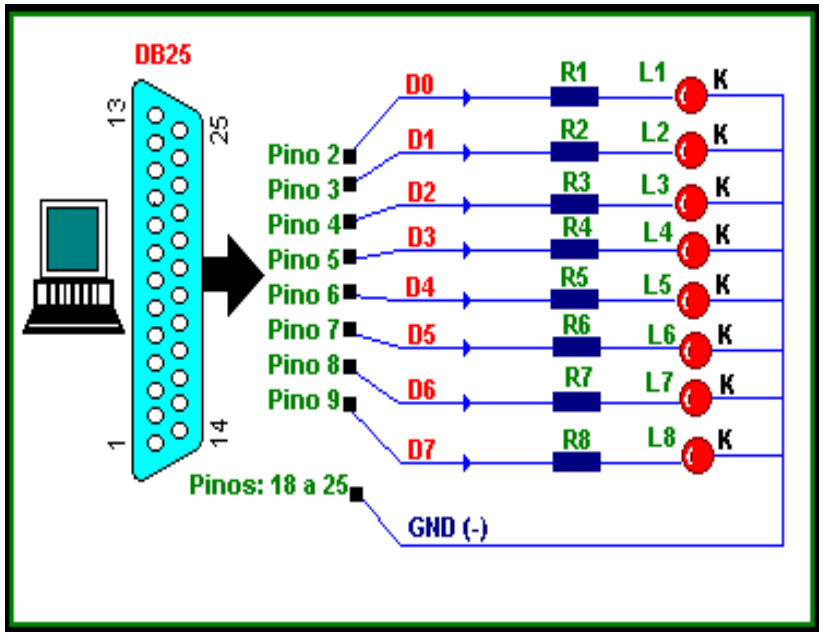
Envio de Dados

- outportB(ENDERECO, VALOR);
- VALOR:
  - É o valor decimal da representação binária relacionada bit a bit com a pinagem da do conector DB25.

Decimal	Binário	Pino/Fio ativo (5V)	Comentário
1	00000001	2 - D0	Cada bit do byte enviado à Porta Paralela está relacionado com um pino do DB25, e um fio do cabo paralelo, fisicamente. Ao enviar um byte, que o(s) bit(s) esteja(m) ligado(s) ou desligado(s), os LEDs acende(rão) ou apaga(rão) conforme os estados dos bits.
2	00000010	3 - D1	
4	00000100	4 - D2	
8	00001000	5 - D3	
16	00010000	6 - D4	
32	00100000	7 - D5	
64	01000000	8 - D6	
128	10000000	9 - D7	

Envio de Dados

» Então, para ativar o pino 3 – D1 dos oitos pinos da porta de dados de endereço 0x378 através de comando em C: outportB(0x378, 2);



» Estando o computador ligado ao circuito acima, este comando acenderia somente o LED L2.

Conector DB25 – Pinagem

» **Sobre os pinos de controle:** C0 (*Strob* – pino 1), C1 (*Auto Feed* – pino 14) e C3 (*Slct in* – pino 17) que são invertidos, i.e., nível lógico baixo (zero) ativa. Apenas o bit C2 (*Init* – pino 16) é normal, i.e., nível lógico alto (um) ativa. A letra C significa registrador de CONTROLE, e o número significa a posição do bit no byte.

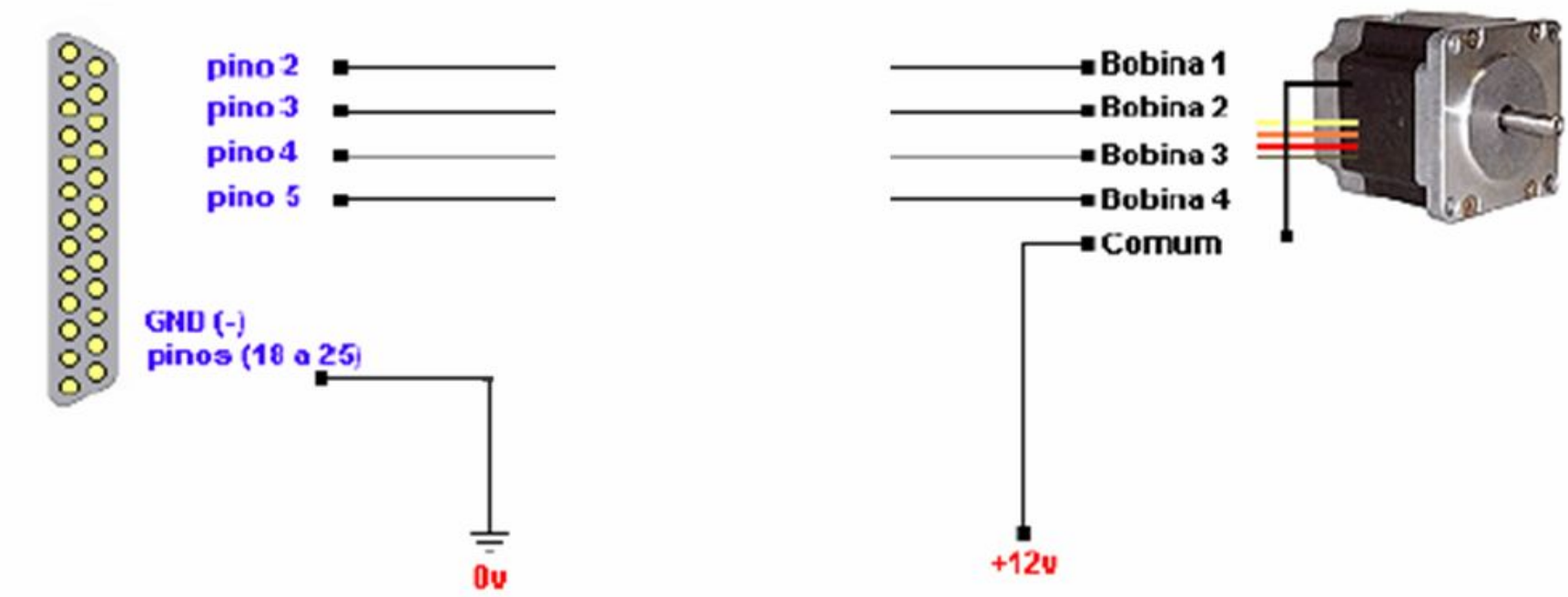
Pinos de Controle								
Identificação	Nenhum				C3	C2	C1	C0
Pino					17	16	14	1
Posição	7	6	5	4	3	2	1	0
Byte	0	0	0	0	1	1	1	1

Como acender os leds sequencialmente?

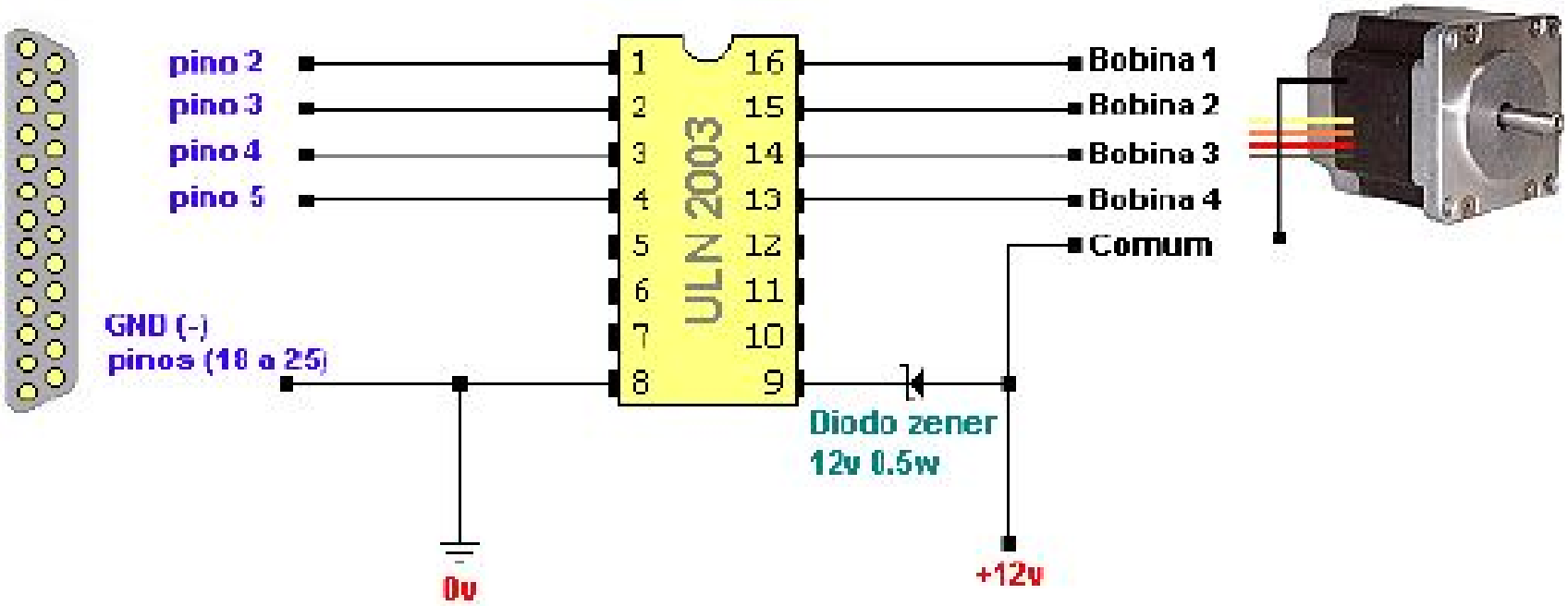
- LOOP!
  - for() / while() / do|while();
  - Depender da variável e da lógica utilizada;
  - Exemplo:

```
#define LPT1 0x378
int i;
unsigned char enviar=128; //10000000 em binário.
for (i=0; i<7; i++)
{
        outportB(LPT1,enviar);          //Envia para a Porta LPT1 - Acende o
                                        LED colocado na posição enviar.
        enviar = enviar >> 1;           //A cada passagem, o bit 1 é movido
                                        para a direita.
        sleep(100);                     //Tempo para o próximo LED acender.
}
```

Acionamento de motor de passo via porta paralela



Acionamento de motor de passo via porta paralela



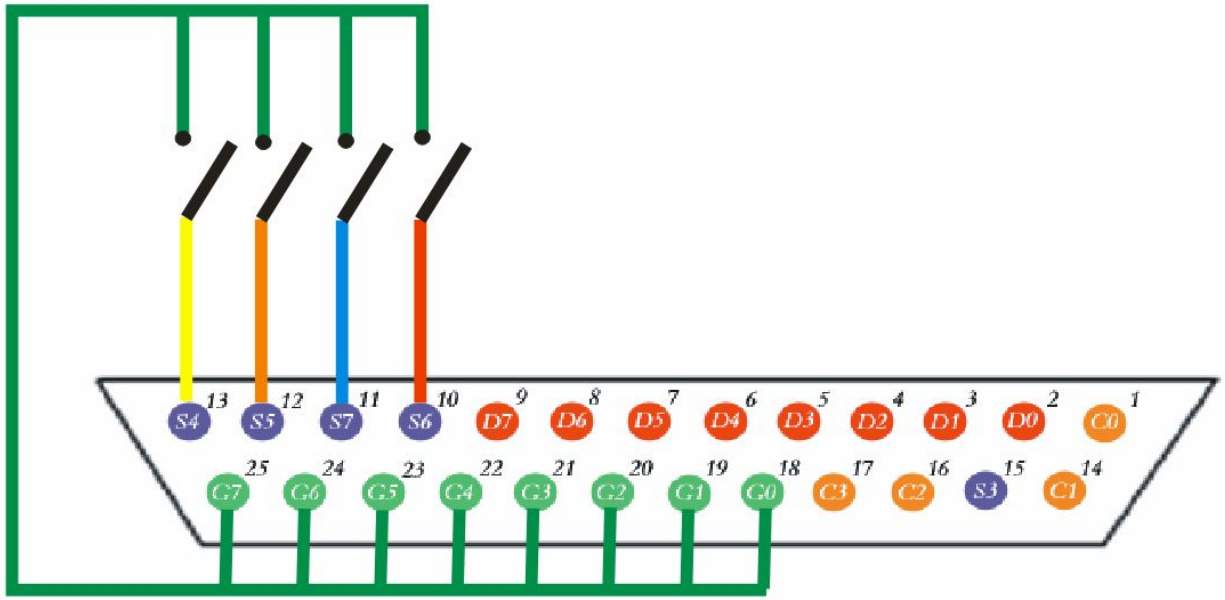
Recebimento de dados

- Variável = inportB(ENDERECO)
- Valor retornado:
  - É o byte retornado pela porta paralela, com representação binária relacionada bit a bit com a pinagem da do conector DB25.
- **Sobre os pinos de status:** S7 (*Busy* – pino 11) é invertido, i.e., nível lógico baixo (zero) ativa. Os outros bits, S6 (*Ack* – pino 10), S5 (*Paper end* – pino 12), S4 (*Slct out* – pino 13) e S3 (*Error* – pino 15) são normais, i.e., nível lógico alto (um) ativa. A letra S significa registrador de STATUS, e o número significa a posição do bit no byte.

Pinos de Status								
Identificação	S7	S6	S5	S4	S3	Nenhum		
Pino	11	10	12	13	15			
Posição	7	6	5	4	3	2	1	0
Byte	0	1	1	1	1	1	1	1

Leitura de sensores via porta paralela

- Pinos 18 a 25 GND da porta paralela
- Pinos 10 a 13 entrada de dados da porta paralela
  - **Ao fechar o curto da entrada com o terra o valor do pino é alterado**



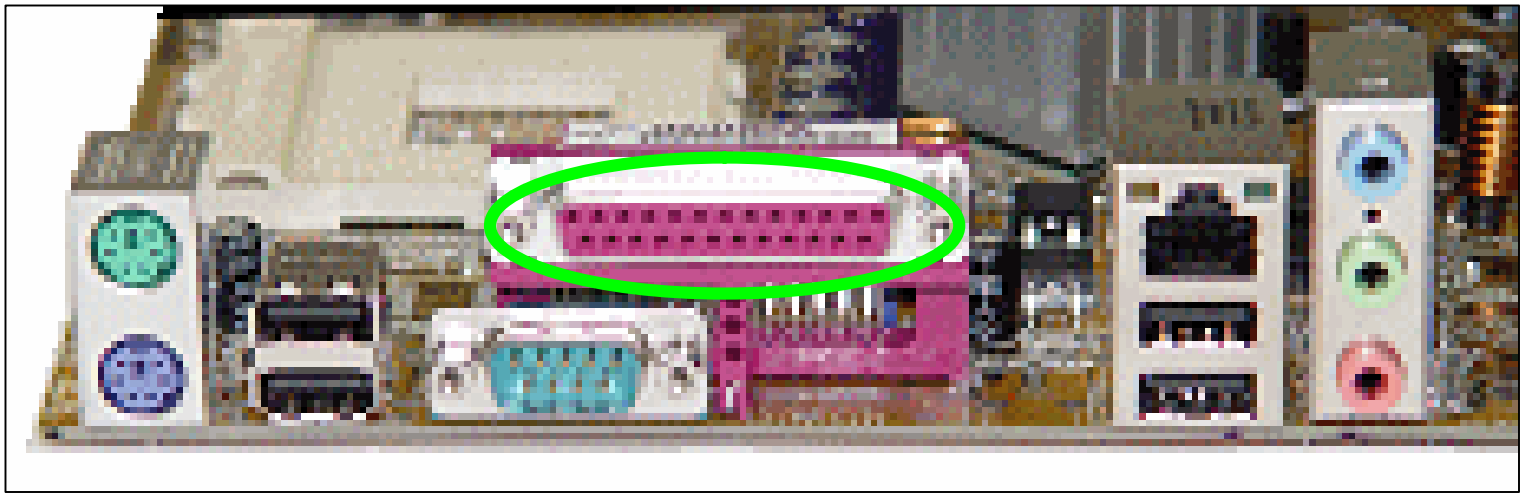
Como receber o estado de um sensor?

» Controle de fluxo:

```
int main(void)
{
        int Byte;           // Armazena o byte recebido da porta
                            paralela
        ...
        while ( ! kbhit() )         // executa enquanto nenhuma tecla for
                                    pressionada
        {
                Byte = inportB(0x379);  // lê um byte da porta paralela
                printf("%d\n", Byte); // mostra na tela o valor
                                      correspondente em inteiro.
                ...
        }
}
```

Cuidado !!!

» A Porta Paralela está ligada diretamente à placa mãe de seu computador. Muito cuidado ao conectar circuitos eletrônicos a essa porta, pois, uma descarga elétrica ou um componente com a polaridade invertida, poderá causar danos irreparáveis ao seu computador, seja coerente.



LINGUAGEM C

Arquivos

Arquivos

- O sistema de E/S do C permite manipulação de arquivos:
  - na representação binária interna;
  - no formato do tipo texto;

O arquivo é uma seqüência de caracteres onde cada linha é terminada com \n

O arquivo é formado de registros; entretanto não é mandatório seguir apenas um formato de registro

Arquivos

» Para efetuar operações em arquivos C:

- Criar descritor de arquivo:

```
FILE *fp;
```

- Abrir o arquivo indicando nome e modo no qual será utilizado:

```
fp = fopen ("nome", "modo");
```

- Inserir/Ler Registros:
- Fechar Arquivo:

```
fclose (fp);
```

Arquivos

- Modo indica, para arquivos texto:
  - abrir arquivo para leitura - r
  - criar arquivo para escrita - w
  - incluir registros no fim de arquivo - a
  - abrir arquivo para leitura/escrita - r+
  - criar arquivo para leitura/escrita - w+
  - incluir registros no fim e ler - a+

Arquivos - Exemplo 1

» Criando um arquivo texto

```
void main (void)
{
   FILE *arq;
    arq = fopen (“nome do arquivo”, "w");
    if (arq == NULL)
    {
         return;
    }
   fclose (arq);
}
```

Arquivos - Funções

- Inserir/Ler Registros - Arquivos Texto

  fputs (s, fp);

  - escreve a cadeia de caracteres s no arquivo fp

  fgets (s, tam, fp);

  - lê cadeia de tamanho tam de fp e armazena em s

  fprintf (fp, controle, variáveis);

  - como *printf*, só que imprime no arquivo *fp*

  fscanf (fp, controle, variáveis)

  - como *scanf*, só que lê de um arquivo *fp*

Arquivos - Funções

- Outras Funções

  fseek (fp, nbytes, origem);

  - reposiciona fp de *nbytes* a partir da origem;
  - origem é uma constante definida no stdio.h:
    - SEEK_CUR - posição atual;
    - SEEK_SET - início do arquivo;
    - SEEK_END - final do arquivo;

  rewind (fp);

  - reposiciona fp no início do arquivo;

  remove (nome);

  - apaga arquivo nome;

Arquivos - Exemplo 2

```
int main()
{
    int a,b,c,valor1,valor2,valor3;
    FILE *rafa;
    rafa=fopen("testando.txt","w");
    if(rafa==NULL)
    {
            printf("Nao foi possivel criar o arquivo \"%s\"\n", "testando.txt");
    }
    else
    {
            printf("Arquivo \"%s\" criado com sucesso\n", "testando.txt");
            printf("Digite valores para salvar no arquivo: ");
            scanf("%d%d%d", &valor1,&valor2,&valor3);
            fprintf(rafa,"%d\t%d\t%d",valor1,valor2,valor3);
    }
    fclose(rafa);
    rafa=fopen("testando.txt","r");
    fscanf(rafa, "%d%d%d", &a,&b,&c);
    printf("%d, %d e %d foram encontrados no arquivo\n", a,b,c);
    fclose(rafa);
}
```

Arquivos - Exemplo 3

```
int main(int argc, char *argv[])
{
	int cont=0;
	char caractere;
	FILE *ponteiro;
	ponteiro=fopen("contem_caracteres.txt","r");
	if(ponteiro==NULL)
	{
		printf("Erro ao abrir o arquivo.\n");
	}
	else
	{
		do
		{
			caractere=getc(ponteiro);
			if(caractere=='$')
			{
				cont++;
			}
		}
		while(caractere!=EOF);
	}
		printf("Foram encontrados %d caracteres $\n", cont);
}
```

LINGUAGEM C

Arrays

Arrays Unidimensionais

- Conjunto finito de elementos homogêneos
- Limites de um array:
  - inferior: primeiro elemento, índice 0
  - superior: número de elementos - 1
- Os limites não podem ser alterados durante a execução do programa
  - estruturas estáticas

Arrays Unidimensionais

» Uso:

- manter tabelas em memória
- quantidade conhecida de itens

» Operações possíveis

- extração de elemento
- armazenamento de elemento

Arrays Unidimensionais

» Declaração:

tipo nome_array [dim];

- Exemplo:

int num [15];

char cadeia [80];

» Extração/Armazenamento

x = num [7];

printf (“%d”, num [14]);

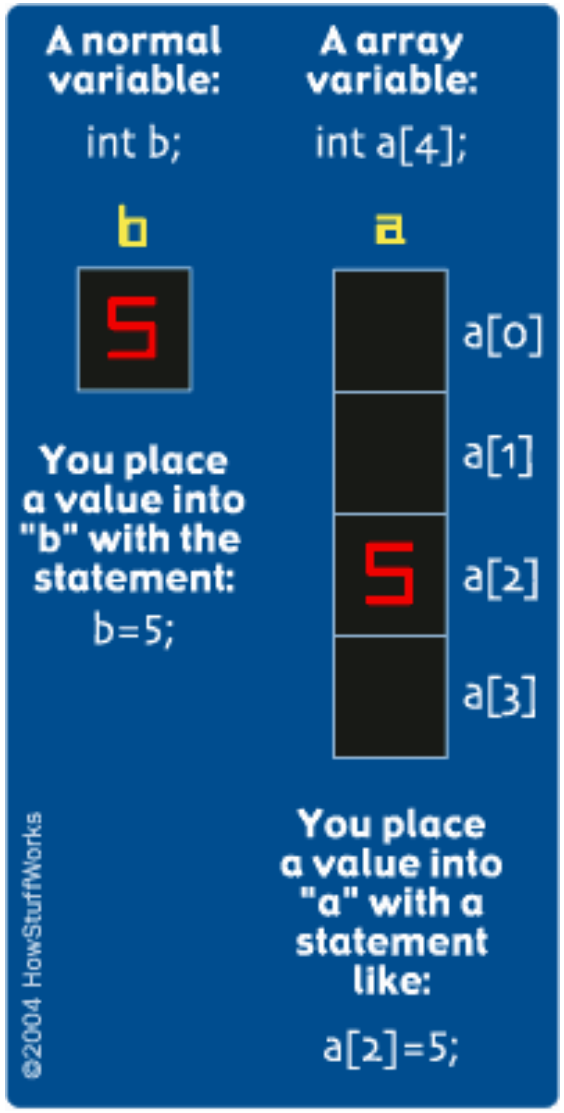
num[0] = 4;

Arrays Unidimensionais – Exemplo

» Ler números reais e armazenar em um vetor de até 10 elementos.

```
main ()
{
   float x [10], num;
   int i;
   for (i = 0; i < 10; i++)
   {
      scanf ("%f", &num);
      x [i] = num;
   }
 }
```

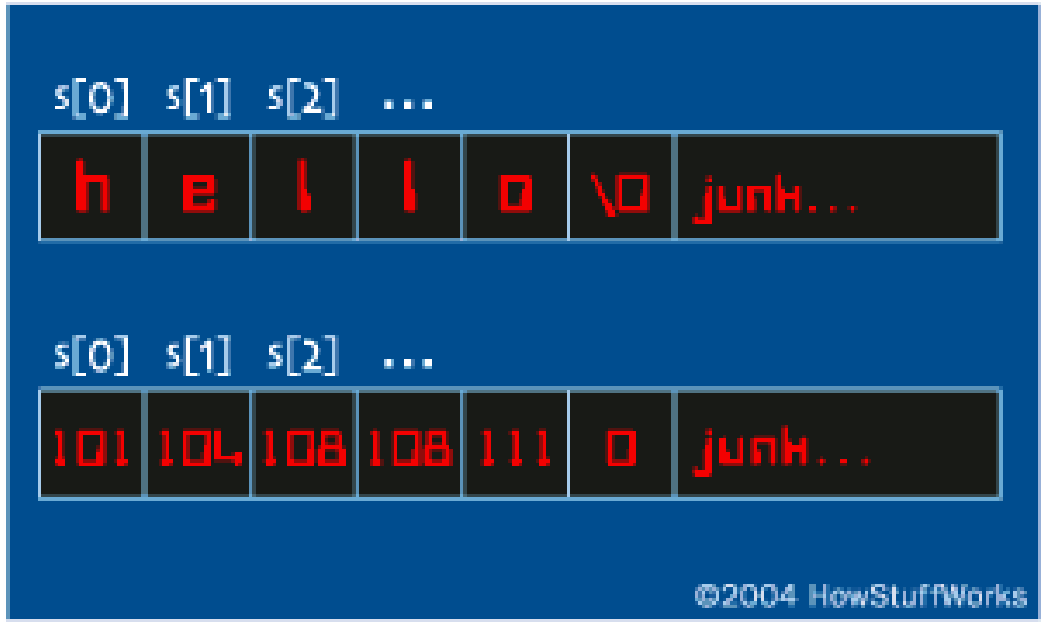
Arrays Unidimensionais – Exemplo



Arrays Unidimensionais

- Array de Caracteres – STRINGS
  - forma de operar com strings em C
  - terminado com o caracter NULL, que indica o fim da cadeia
  - o NULL conta como caracter no tamanho do array
  - em C, NULL = '\0'

Arrays Unidimensionais – Strings



Arrays Unidimensionais – Strings

- o C oferece funções especiais para arrays de caracteres:

```
gets (s); /* lê uma string da entrada padrão */
puts (s); /* escreve uma string na saída padrão */
strcpy (s1, s2); /* s1 = s2   */
strcat (s1, s2); /* s1 = s1 + s2 */
x = strlen (s); /* x recebe o tamanho de s */
k = strcmp (s1, s2); /* se s1 =  s2, k recebe 0 */
                     /* se s1 precede s2, k > 0 */
                     /* se s2 precede s1, k < 0 */
```

Arrays Unidimensionais – Strings

- Mais funções...

```
s1 = strchr (s2, ch); /* procura ch em s2 */
s1 = strstr (s2, sub); /* procura sub em s2 */
strupr (s); /* converte s para maiúscula  */
strlwr (s); /* converte s para minúscula */
strset (s, ch); /* preenche s com ch */
```

Arrays Unidimensionais – Strings

» Ler array e imprimir invertido

```
# include <string.h>
main ()
{
   char cad [30];
   int i, tam;
   gets (cad);
   tam = strlen (cad);
   for (i = tam-1; i >= 0; i--)
       putchar (cad [i]);
}
```

Arrays Unidimensionais

» Inicialização de Arrays

- Numéricos

  tipo nome_array [dim] = { i_0, i_1, i_2,..., i_n};

- Caracteres

  char nome_array [dim] = "$i_0 i_1 i_2 ... i_n$";

- Pode-se omitir a dimensão do array na inicialização, mas nunca na declaração

Arrays Unidimensionais – Exemplo

- Mostrar quantos dias tem determinado mês

```
int meses [] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
main ()
{
    int mes;
    scanf (“%d”, &mes);
    printf (“\nO mes %d possui %d dias”, mes, meses [mes-1]);
}
```

Arrays Unidimensionais – Exemplo

» Mostrar se determinada letra é vogal

```
# include <string.h>
char vogais [] = "aeiou";
main ()
{
    char ch;
    ch = getchar ();
    if (strchr (vogais, ch) != NULL)
      printf ("A letra digitada e' uma vogal");
    else
      printf ("A letra digitada nao e' uma vogal");
}
```

Arrays Bidimensionais

» Declaração:

tipo nome_array [n_linhas][n_colunas];

- Exemplo:

```
int num [15][10];
char nomes [30][40];
```

» Extração/Armazenamento

```
x = num [7][6];
printf (“%d”, num [3][2]);
num[0][3] = 4;
```

Arrays Bidimensionais

- Matriz de Caracteres
  - cada linha armazena um array de caracteres
  - o array também termina com '\0'
  - pode-se também armazenar caracteres independentes

Arrays Bidimensionais

» Inicialização de Matrizes

- Numéricas

  tipo nome_array [<nl>] [nc] = { {i_{00}, i_{01}, ..., i_{0n}}, ...
  {i_{n0}, i_{n1}, ..., i_{nn}} };

- Caracteres

  char nome_array [<nl>] [nc] = {"$i_{00}i_{01}...i_{0n}$", ...
  "$i_{n0}i_{n1}...i_{nn}$"};

- Pode-se omitir o número de linhas na inicialização, mas nunca na declaração

Arrays Bidimensionais – Exemplo

- Mostrar quantos dias tem determinado mês, considerando se o ano é bissexto

```
int meses [] [12] = {{31,28,31,30,31,30,31,31,30,31,30,31},
                     {31,29,31,30,31,30,31,31,30,31,30,31}};
main ()
{
    int mes, ano;
    scanf (“%d/%d”, &mes, &ano);
    if (ano%4 == 0)
      printf (“\nMes %d - %d dias”, mes, meses [mes-1][1]);
    else
      printf (“\nMes %d - %d dias”, mes, meses [mes-1][0]);
}
```

Arrays Bidimensionais – Exemplo

- Ler uma data e mostrar por extenso

```
char meses [][10] = {“Janeiro”, “Fevereiro”, “Marco”, ...};
main ()
{
        int dia, mes, ano;
        scanf (“%d/%d/%d”, &dia, &mes, &ano);
        printf (“\nData: %d de %s de %d”, dia, meses [mes-1], ano);
}
```

LINGUAGEM C

Ponteiros

Ponteiros

- Variáveis que armazenam um endereço de memória;
- Existem dois operadores:
  - Operador *****
    - usado para declarar o ponteiro e;
    - para retornar o conteúdo da variável apontada;
  - Operador **&**
    - retorna o endereço do operando.

Ponteiros

» Declaração de um ponteiro:

```
tipo *nome;
```

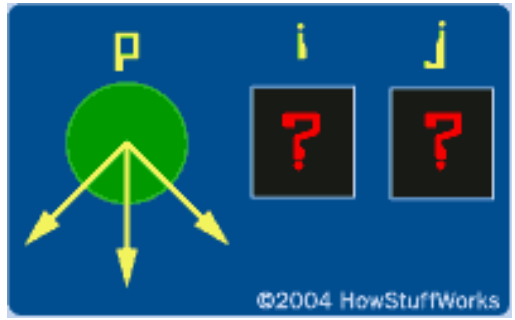
- *nome* é um ponteiro;
- *tipo* indica o tipo de conteúdo para onde *nome* aponta;

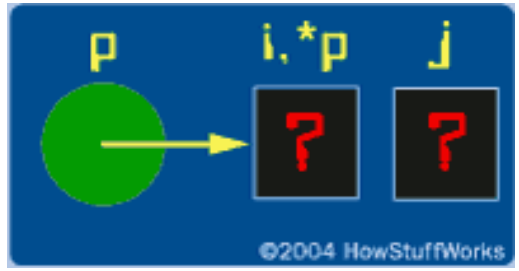
» Ponteiro tem duas partes:

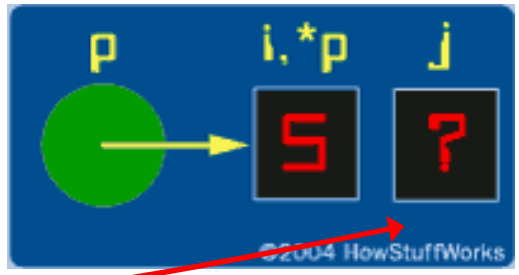
- O ponteiro guarda o endereço;
- O endereço aponta para um valor;

Ponteiros - Exemplo

```
#include <stdio.h>
    int main()
    {
        int i, j, *p;
        p = &i;
        *p = 5;
        j = i;
        printf("%d %d %d\n", i, j, *p);
        return 0;
    }
```







Ponteiros - Exemplo

» A diferença entre os operadores

```
main ()
{
   int num, *pont;
   scanf (“%d”, &num);
   pont = &num;
   printf (“\n%d”, pont);
   printf (“\n%d”, *pont);
   printf (“\n%d”, &pont);
 }
```

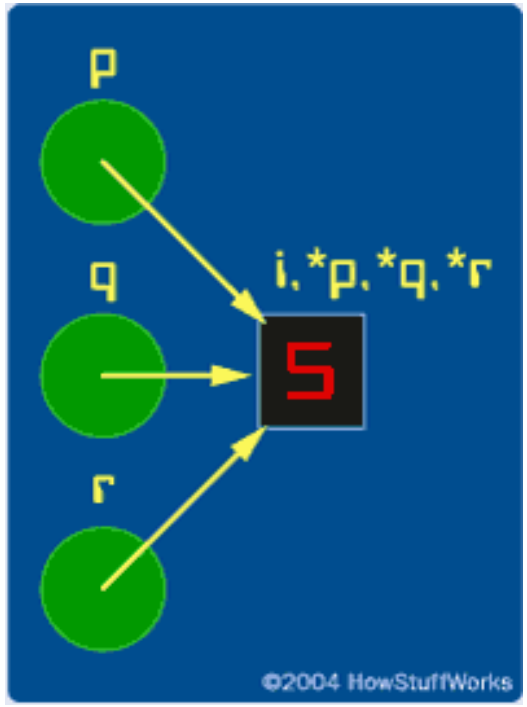
Ponteiros

- Diversos apontadores para o mesmo endereço:

```
int i;
int *p, *q, *r;

p = &i;
q = &i;
r = p; /*ponteiro para ponteiro*/
```

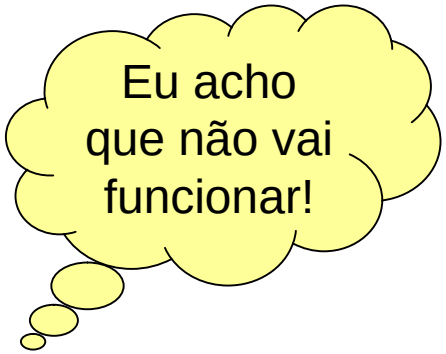
A variável i passa a ter 4 nomes: i, *p, *q, *r.



Ponteiros como parâmetros de função

» Função para permutar (*swap*) valores em variáveis:

```
void permuta(int i, int j)
{
    int t;
    t=i;
    i=j;
    j=t;
}
```



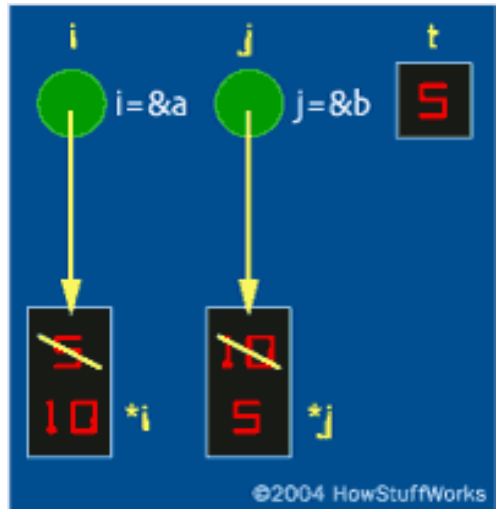

```
void main()
{
    int a,b;
    a=5;
    b=10;
    printf("%d %d\n", a, b);
    permuta(a,b);
    printf("%d %d\n", a, b);
}
```

Ponteiros como parâmetros de função

- Função para permutar (*swap*) valores em variáveis:

```
void permuta(int *i, int *j)
{
    int t;
    t = *i;
    *i = *j;
    *j = t;
}
```

```
void main()
{
    int a,b;
    a=5;
    b=10;
    printf("%d %d\n",a,b);
    permuta(&a,&b);
    printf("%d %d\n",a,b);
}
```



"*i" passa a ser outro nome para "a" e "*j" passa a ser outro nome para "b"

Ponteiros e arrays

» Ponteiros e arrays intimamente ligados:

```
#define MAX 10

int main()
{
    int a[MAX];
    int b[MAX];
    int i;
    for(i=0; i<MAX; i++)
        a[i]=i;
    b=a;
    return 0;
}
```

"a" e "b" apontam permanentemente para o primeiro elemento de seus respectivos arrays, ele mantêm os endereços de "a[0]" e "b[0]".

```
for (i=0; i<MAX; i++)
    b[i]=a[i];
```

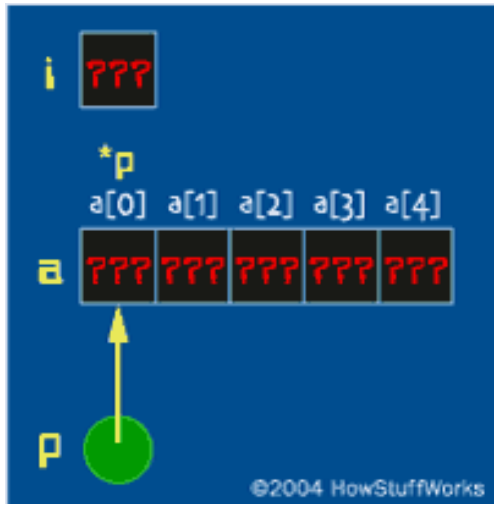
Ponteiros e arrays

» Ponteiros e arrays intimamente ligados:

```
#define MAX 10

void main()
{
    int a[MAX];
    int i;
    int *p;
    p = a; /* a é um ponteiro */
    for(i=0; i<MAX; i++)
        a[i]=i;
    printf("%d\n",*p);
}
```

Tecnicamente "a" aponta para o endereço do primeiro elemento do array.



Depois que p aponta para o primeiro elemento do array pode-se variar seu valor, ao contrário de "a" que é um ponteiro permanente e não pode ser mudada.

Ponteiros e arrays

- Substituindo for (i=0; i<MAX; i++) b[i]=a[i];

```
p=a;
q=b;
for (i=0; i<MAX; i++)
{
    *q = *p;
    q++;
    p++;
}
```

O compilador sabe que "p" aponta para um inteiro, essa instrução incrementará "p" a quantidade apropriada de bytes para movê-lo para o próximo elemento do array.

Ponteiros e arrays

» Passando array como parâmetro:

```
void imprime(int a[],int nia)
{
    int i;
     for (i=0; i<nia; i++)
        printf("%d\n",a[i]);
}
```

```
void imprime(int *p,int nia)
{
    int i;
     for (i=0; i<nia; i++)
        printf("%d\n",*p++);
}
```

Ponteiros – Exercícios

- Escreva uma função para calcular a média dos elementos de um vetor. Retornar a média.
- Escreva uma função que receba um vetor numérico unidimensional como parâmetro e verifique se há algum número negativo no vetor. Se houver, retornar 1; Se não houver, retornar 0.
- Dada uma matriz 4 x 4, fazer uma função para alterá-la multiplicando os elementos da diagonal principal por 3.
- Escrever uma função para receber uma frase e retornar quantas palavras a frase possui.

Ponteiros

- Alocação Dinâmica de Memória:
  - quando uma variável é definida, o compilador aloca o espaço necessário automaticamente;
  - quando um ponteiro é definido, compilador aloca espaço para armazenar o endereço, não um valor.
    - O programador tem que “pedir” explicitamente uma área para guardar os dados que deseja:
      - alocação dinâmica.

Ponteiros

- Podemos usar ponteiros para:
  - criar arrays dinamicamente;
  - criar estruturas de dados dinâmicas;
- Funções ANSI C para alocação/liberação de memória:
  - malloc
  - calloc
  - free

Ponteiros - Funções

- malloc:
  - aloca 1 bloco de armazenamento:
  - não limpa região alocada:

    p = malloc (nº de bytes).
- calloc
  - aloca vários blocos contíguos em memória;
  - limpa região alocada:

    p = calloc (nº de blocos, nº de bytes);

Ponteiros - Funções

» free

- libera área de memória alocada

  free (p)

as funções *malloc* e *calloc* retornam o endereço do 1º bloco alocado

– se não há memória suficiente, retornam *NULL*; logo, deve-se sempre testar o retorno

```
if ((p = malloc (10)) == NULL)
   /* tratamento do erro */
else
   ...
```

Ponteiros - Funções

- Se não se sabe quanto de memória alocar:
  - usar operador : *sizeof (tipo)*
  - exemplo:

    p = malloc (sizeof (int))
  - permite portabilidade
- Para usar as funções descritas deve-se incluir:
  - stdlib.h ou malloc.h

Ponteiros - Exemplo

» Ler array e listar invertido

```
main ()
{
   char *s;
   int t;
   if ((s = malloc (80 * sizeof (char)) == NULL)
   {
            return;
   }
   gets (s);
   for (t = strlen (s) - 1; t >= 0; t--)
   putchar (s [t]);
   free (s);
}
```

Parâmetros do “main”

- Pode-se criar um programa em C para receber um ou mais valores de diversos tipos;
- Esses valores são passados cada vez que o programa é executado:
  - exemplo: utilitário cp do Unix;

Parâmetros do “main”

» Forma geral:

void main (**int num, char *cad [ ]**)

contém o número de parâmetros que **foram** digitados

contém todos os parâmetros que **foram** digitados, inclusive o nome do programa

– sempre haverá esses dois elementos

Parâmetros do “main” - Exemplo

```
void main (int num, char *cad [])
{
  int i;
  if (num != 3)
  {
    printf ("Numero de argumentos invalido\n");
    return;
  }

  for (i = 0; i < num; i++)
    printf ("\n\t%s", cad [i]);
}
```

LINGUAGEM C

Tipos Estruturados

Tipos Estruturados

- No C há vários recursos para criação de novos tipos de dados:
  - dos quais veremos *estruturas;*
- Estruturas:
  - Agrupamento de variáveis de tipos diferentes, referenciado por um mesmo nome;
  - também chamadas de registros.

Tipos Estruturados

» Como utilizá-los no programa

- especificar a estrutura:
  - equivale a criar um novo tipo no programa;
  - pode ser feito em qualquer lugar; de preferência, no início do arquivo;
- declarar as variáveis que terão o formato da estrutura:
  - como a criação de qualquer variável;
  - possui as mesmas regras de escopo de qualquer variável.

Tipos Estruturados

» Criação da estrutura:

```
struct nome
{
  tipo1 elem1;
  tipo2 elem2;
   ...
};
```

Tipo1 pode ser outra estrutura;
elem1 é um campo da estrutura

» Declaração de variáveis:

```
struct nome var1, var2, ...;
```

Var1 e var2 terão os campos definidos pela struct

Tipos Estruturados - Exemplo

» Estrutura para armazenamento de endereço:

```
struct end
{
    char rua [30];
    int num;
    int apto;
    char bairro [20];
};
struct end Endereco;
```

Endereco tem os campos rua, num, apto, bairro

Tipos Estruturados

- Referência a elementos da estrutura:

  var. elem

  - var => nome da variável;
  - elem => elemento da estrutura;

- No exemplo anterior:

```
strcpy (Endereco.rua, “Wanderley Pinho”);
Endereco.num = 577;
Endereco.apto = 902;
strcpy (Endereco.bairro, “Itaigara”);
```

Tipos Estruturados

- Pode-se copiar o conteúdo de uma variável estrutura para outra de mesmo tipo:

```
struct end EnderecoNovo, EnderecoAntigo;
EnderecoNovo = le_dados ();
EnderecoAntigo = EnderecoNovo;
```

- Uma das maiores aplicações de estruturas é o uso de arrays:

```
struct end Enderecos [30];
```

Tipos Estruturados - Exemplo

» Ler dados e armazenar em um array de estruturas:

```
# define MAX 3

struct registro
{
    char nome [10];
    char dtlog [9];
    char hora [6];
    char dur [6];
} log [MAX];
```

Tipos Estruturados - Exemplo

```
struct registro le_reg (void);
void           lista_array (void);
main ()
{
    struct registro reg;
    int i;
    for (i = 0; i < MAX; i++)
    {
            reg = le_reg ();
            log [i] = reg;
    }
    lista_array ();
}
```

```
struct registro le_reg (void)
{
    struct registro temp;

    printf ("\nDigite login name,
    data, hora inicio, duracao\n");
    gets (temp.nome);
    gets (temp.dtlog);
    gets (temp.hora);
    gets (temp.dur);
    return temp;
}
```