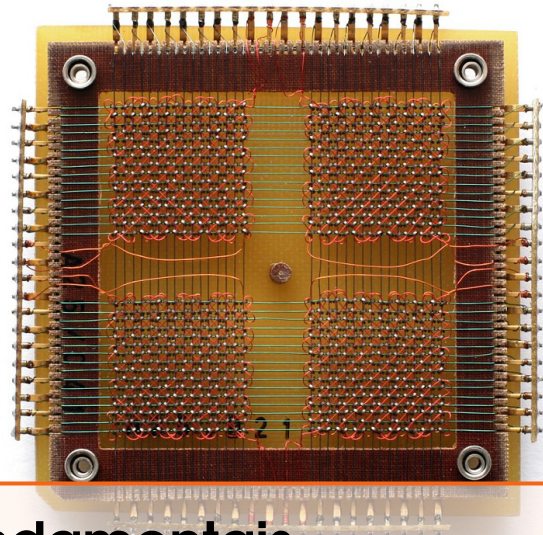


Sumário

1	Conceitos fundamentais	5
1.1	Linguagens de programação	5
1.2	Sistemas operacionais	5
1.3	O que são compiladores	5
1.4	Ambientes de desenvolvimento	5
2	Visão Geral da linguagem C	7
2.1	Origens da linguagem	7
2.2	Porque aprender C?	7
2.3	Onde C é usado?	7
2.4	Estrutura de um programa em C	7
2.5	Meu primeiro programa em C	7
2.6	Compilando meu programa	7
2.7	Executando	8
2.8	Entendendo meu programa	8
3	Variáveis, Tipos de Dados e Expressões Aritméticas	9
3.1	Tipo de dados básicos	9

3.2	Variáveis	9
3.3	Modificadores	9
3.4	Constantes	9
3.5	Expressões Aritméticas	9
3.5.1	Operação de resto (%)	9
3.6	Conversão de tipo de dados	9
4	Estruturas de Controle, Operadores Logicos e Relacionais	11
4.1	Operadores relacionais	11
4.2	Operadores lógicos	11
4.3	O comando if	11
4.3.1	O comando else	11
4.3.2	O comando if-else-if	11
4.3.3	Ifs aninhados	11
4.4	O comando switch	11
4.5	Operador ternário(?)	11
5	Estruturas de repetição (Loops)	13
5.1	Comando for	13
5.2	Comando while	13
5.3	Comando do-while	13
5.4	Controle de loops	13
5.4.1	O comando break	13
5.4.2	O comando continue	13
5.5	Loops aninhados	13
5.6	Loops infinitos	13
6	Arrays	15
6.1	Trabalhando com Arrays	16
6.1.1	Definindo Arrays	16
6.1.2	Declarando Arrays	16

6.1.3	Inicializando Arrays	16
6.1.4	Acessando Arrays	17
6.2	Array de Caracteres	17
6.3	Arrays na Vida Real?!	17
6.3.1	Imprimir Elementos	17
6.3.2	Somar Elementos	17
6.3.3	Inverter o Array	17
6.3.4	Ordenar o Array	17
6.4	Arrays como Parâmetros	17
6.5	Arrays Multidimensionais (Matrizes)	17
7	Ponteiros	19
7.1	Armazenamento Primário	19
7.1.1	Memória Principal	20
7.2	Usando Ponteiros	20
7.3	Usando Vetores	20
7.4	Vetores NÃO são Ponteiros	20



1. Conceitos fundamentais

Antes de começarmos a falar sobre C devemos nos perguntar: o que é o C?

C é uma linguagem de programação, cara!

Ok, ok. Você está certo. Mas entender o que é uma linguagem de programação é o primeiro passo. Se você já sabe o que é uma, não tenha pressa. Quem sabe você aprenda algo novo ou melhore seu conceito. Ou quem sabe contribua com este livro e melhore esta seção.

Bem, uma linguagem de programação é uma forma estruturada (e organizada) criada com a intenção de comunicar a uma máquina que ela deve realizar certas instruções e comandos. Esta máquina, não por acaso, é um computador (pelo menos no nosso caso).

A forma que estas instruções e comandos são ordenados irá alterar o que o computador entenderá e conseqüentemente o que ele executará. Este passo-a-passo é conhecido como *algoritmo*.

1.1 Linguagens de programação

Linguagens de programação.

1.2 Sistemas operacionais

Linux.

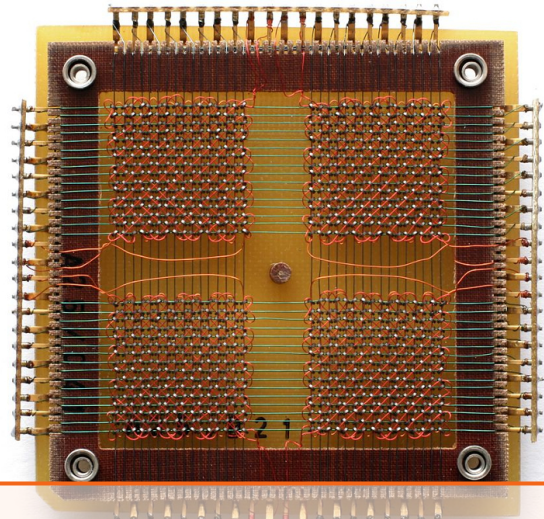
1.3 O que são compiladores

Compiladores.

1.4 Ambientes de desenvolvimento

Ambientes de desenvolvimento.

Origens da linguagem
Porque aprender C?
Onde C é usado?
Estrutura de um programa em C
Meu primeiro programa em C
Compilando meu programa
Executando
Entendendo meu programa



2. Visão Geral da linguagem C

blz

2.1 Origens da linguagem

blz

- oi

2.2 Porque aprender C?

blz

2.3 Onde C é usado?

blz

2.4 Estrutura de um programa em C

blz

2.5 Meu primeiro programa em C

blz

2.6 Compilando meu programa

blz

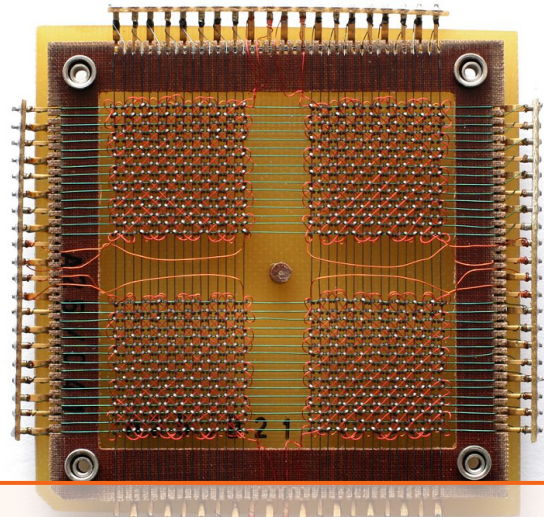
2.7 Executando

blz

2.8 Entendendo meu programa

blz

Tipo de dados básicos
Variáveis
Modificadores
Constantes
Expressões Aritméticas
 Operação de resto (%)
Conversão de tipo de dados



3. Variáveis, Tipos de Dados e Expressões Ar

Conceitos

- item

3.1 Tipo de dados básicos

3.2 Variáveis

3.3 Modificadores

3.4 Constantes

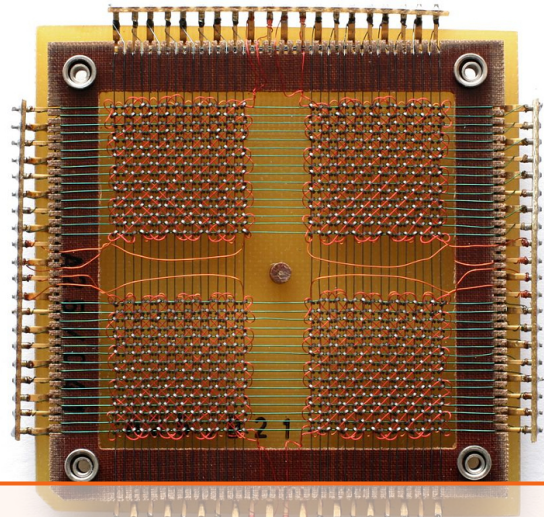
3.5 Expressões Aritméticas

3.5.1 Operação de resto (%)

3.6 Conversão de tipo de dados

Conversao.

Operadores relacionais
Operadores lógicos
O comando if
 O comando else
 O comando if-else-if
 ifs aninhados
O comando switch
Operador ternário(?)



4. Estruturas de Controle, Operadores Lógicos

Conceitos

- item

- 4.1 Operadores relacionais**
- 4.2 Operadores lógicos**
- 4.3 O comando if**
 - 4.3.1 O comando else**
 - 4.3.2 O comando if-else-if**
 - 4.3.3 ifs aninhados**
- 4.4 O comando switch**
- 4.5 Operador ternário(?)**

Conversao.

Comando for
Comando while
Comando do-while
Controle de loops
 ○ comando break
 ○ comando continue
Loops aninhados
Loops infinitos



5. Estruturas de repetição (Loops)

Conceitos

- item

5.1 Comando for

5.2 Comando while

5.3 Comando do-while

5.4 Controle de loops

5.4.1 O comando break

5.4.2 O comando continue

5.5 Loops aninhados

5.6 Loops infinitos

Conversao.

Trabalhando com Arrays

- Definindo Arrays
- Declarando Arrays
- Inicializando Arrays
- Acessando Arrays

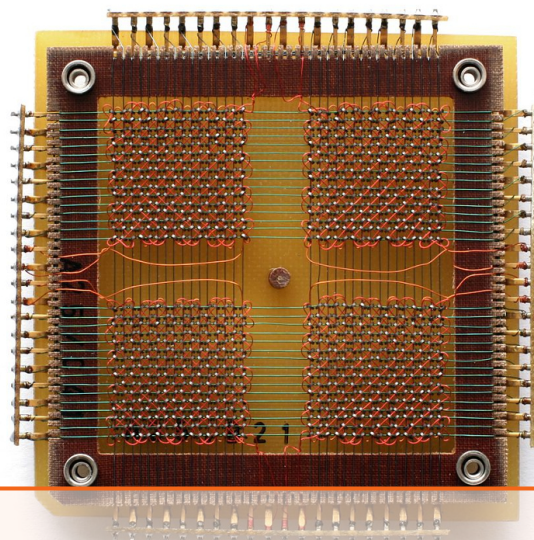
Array de Caracteres

Arrays na Vida Real?!

- Imprimir Elementos
- Somar Elementos
- Inverter o Array
- Ordenar o Array

Arrays como Parâmetros

Arrays Multidimensionais (Matrizes)



6. Arrays

Quem nunca precisou organizar dados semelhantes num único local? Eu sim, você não? Então quando você, por exemplo, escreve um texto, o que acha que está fazendo? “*Organizando ideias?*”. Sim, também! Contudo, o que você está organizando materialmente são caracteres. Vejamos:

Autor presunçoso!																
A	u	t	o	r		p	r	e	s	u	n	ç	o	s	o	!

A frase *Autor presunçoso!* é a organização de dezessete caracteres um atrás do outro. Isso mesmo, dezessete, pois o espaço também é um dado. Nunca se esqueça que, quando tratamos dados, até um “0” ocupa espaço na memória do computador. Engenheiros, acalmem-se, não estou falando de estado negativo num barramento de dados. *All right?*

Segue outra organização de dados. Desta vez de inteiros que serão o resultado do próximo sorteio da mega-sena.

9	20	11	7	43	31
---	----	----	---	----	----

Ambos são exemplos de **arrays** (também conhecidos como *vetores*) que, em C, são tipos de dados de alocação contígua derivados dos tipos simples. Em outras palavras, são estruturas de dados homogêneas, compostas de elementos de mesmo tipo, e posicionados em sequência na memória.

Obs.:

Em algumas fontes esparsas em idioma português, você, leitor, encontrará referências a arrays como sendo **arranjos**. Ressalte-se que um array, numa tradução mais fiel, é um **conjunto** de dados. Já um arranjo:



Figura 6.1: Array não é Arranjo

6.1 Trabalhando com Arrays

6.1.1 Definindo Arrays

Ao se definir um vetor é necessário determinar o respectivo tamanho.

```
1  int i[6];
2  char c[3];
```

O uso do operador de índice [] é que informa ao compilador que as variáveis *i* e *c* são do tipo vetor. Já o número entre os colchetes define a quantidade de elementos a serem alocados para o vetor em questão. No código acima, *i* é definido como um vetor de *int* com seis elementos e *c* um vetor de *char* com três elementos.

i[6]					
-57	3	791431480	0	32767	0
0	1	2	3	4	5

c[3]		
0	-10	49
0	1	2

Uma boa prática de programação é inicializar os vetores, assim como os tipos simples, pois, tanto em *i*[6] quanto em *c*[3], vê-se claramente que os elementos contêm valores aleatórios.

6.1.2 Declarando Arrays

Ao contrário do que se pensa comumente, vetores podem ser declarados sem especificação da quantidade de elementos.

```
1  extern int ext[];
```

6.1.3 Inicializando Arrays

```
1  static int stc[6];
```


6.1.4 Acessando Arrays

6.2 Array de Caracteres

6.3 Arrays na Vida Real?!

6.3.1 Imprimir Elementos

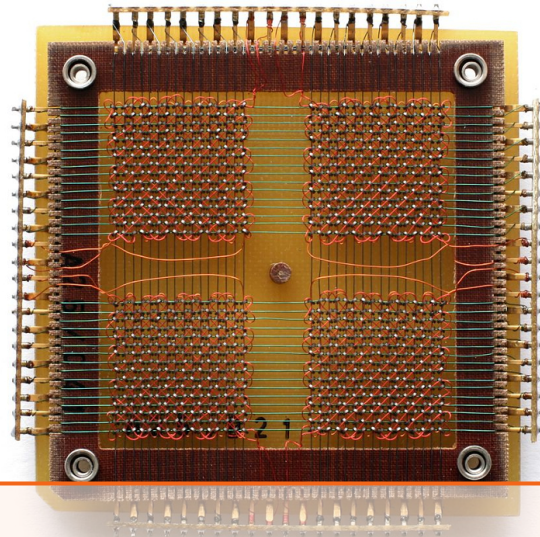
6.3.2 Somar Elementos

6.3.3 Inverter o Array

6.3.4 Ordenar o Array

6.4 Arrays como Parâmetros

6.5 Arrays Multidimensionais (Matrizes)



7. Ponteiros

7.1 Armazenamento Primário

Em um computador atual (2014), existem três tipos principais de armazenamento primário: **registradores do processador**; **cache do processador**, e; **memória principal**.

Registradores são pequenos locais de armazenamento, com tamanho estático, contidos no processador. Por fazerem parte do *ISA (Instruction Set Architecture)*, variam com a arquitetura (x86, x86_64, MIPS etc). Um exemplo de registradores de uso geral da arquitetura x86_64 é: *rax, rbx, rcx, rdx*.

A cache do processador é um contêiner de dados intermediário e de acesso aleatório com maior capacidade que os registradores. Ela se situa entre a memória principal e o próprio processador com o intuito de diminuir o tempo médio de acesso às informações; podendo ser subdividida em cache de instruções (lida apenas com a leitura da memória), cache de dados (trata da leitura e escrita da memória) e *TLB - Translation lookaside buffer* (com a finalidade de agilizar a tradução da memória virtual x física).

O termo “Memória Principal” é comumente usado como referência à memória RAM (*Random Access Memory*), uma vez que nesta reside a grande porção de dados utilizados antes e/ou após o processamento. Essa referência se dá, também, pela transparência que os registradores e a cache do processador tem em comparação à memória principal, tendo em vista a utilização direta dos dados contidos nela quando no desenvolvimento de software.

Há alguns detalhes importantes a serem ressaltados a respeito desses três repositórios primários de dados.

1. A memória principal (RAM), distintamente dos registradores e da cache, não se encontra no processador mas sim conectada a ele através do barramento de memória que, por sua vez, subdivide-se em dois: barramento de endereço e barramento de dados [Figura 7.1].
2. O gerenciamento da memória cache, mesmo que possível através de instruções como *INVD* e *WBINVD* na arquitetura x86, é e deve, por segurança, ser deixado ao encargo do próprio processador, salvo em casos realmente justificáveis.
3. Os dados de todos eles são obtidos por endereçamento, ou seja, por indexação, mas o que os torna diferentes, à visão do programador, é o fato de os registradores e a cache

só serem acessados dessa forma pelo microcódigo da arquitetura, enquanto a memória principal pode o ser por endereçamento via código de máquina (programa residente na própria memória).

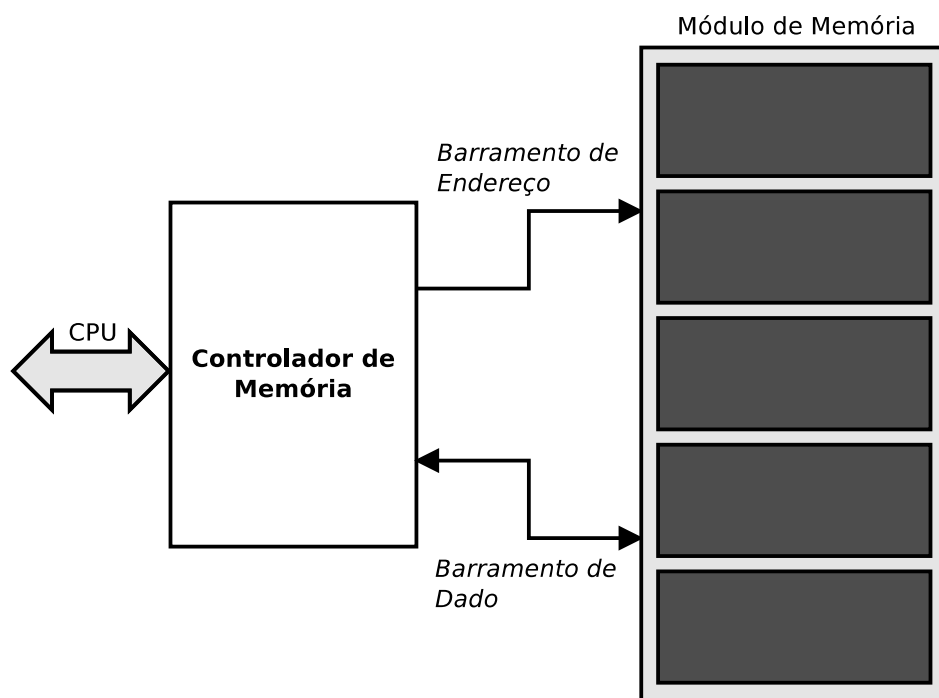


Figura 7.1: Barramentos de Endereço e de Dados

7.1.1 Memória Principal

Sobre a memória principal.

7.2 Usando Ponteiros

Como usar ponteiros.

7.3 Usando Vetores

Como usar vetores.

7.4 Vetores NÃO são Ponteiros

Vetores não são ponteiros.