

AED1 - Aula 07

Endereços, apontadores e estruturas

"Os conceitos de endereço e apontador são fundamentais em qualquer linguagem de programação, embora fiquem ocultos em algumas linguagens. Em C, esses conceitos são explícitos. Dominar o conceito de apontador exige algum esforço e uma boa dose de prática" - citação retirada do livro do Prof. Paulo Feofiloff.

"Para projetar algoritmos eficientes é importante manipular dados com eficiência. Estruturas de dados são maneiras de organizar dados para atender requisições com rapidez, mas cada estrutura de dados é eficiente em algumas situações e ineficiente em outras. Uma ótima maneira de entender os prós e contras de uma estrutura de dados é implementá-la. Para isso é necessário dominar conceitos como endereços, apontadores e estruturas, que nos permitem construir estruturas complexas a partir de tipos básicos de dados."

Endereços

A memória de um computador é uma sequência de bytes

- os quais são numerados sequencialmente.
- Numa analogia, podemos pensar na memória como um imenso vetor.
 - Assim, o número (ou índice) de um byte é seu endereço.

Cada objeto na memória ocupa um número de bytes consecutivos

- Exemplo:
 - `sizeof(char) = 1`
 - `sizeof(int) = 4`
 - `sizeof(long) = 4`
 - `sizeof(long long) = 8`
 - `sizeof(float) = 4`
 - `sizeof(double) = 8`
 - `sizeof(char *) = 4`
 - `sizeof(int *) = 4`
 - `sizeof(double *) = 4`
- Código:

```
printf("sizeof(char) = %d\n", sizeof(char));
printf("sizeof(int) = %d\n", sizeof(int));
printf("sizeof(long) = %d\n", sizeof(long));
printf("sizeof(long long) = %d\n", sizeof(long long));
printf("sizeof(float) = %d\n", sizeof(float));
```

```
printf("sizeof(double) = %d\n", sizeof(double));
printf("sizeof(char *) = %d\n", sizeof(char *));
printf("sizeof(int *) = %d\n", sizeof(int *));
printf("sizeof(double *) = %d\n", sizeof(double *));
```

Note que, usamos 4 bytes (32 bits) para endereçar uma posição da memória.

- Sabendo que cada byte tem seu próprio endereço,
 - qual o total de bytes que podemos endereçar com 32 bits?
 - $2^{32} \approx 4\text{GB}$
- Mas isso é menos do que a memória de muitos computadores modernos.
 - Como fazemos para usar toda a memória?
 - Gerenciamento do sistema operacional e endereços de 64 bits.

O endereço de uma variável corresponde ao endereço do seu primeiro byte.

- Exemplo, após declarar

```
char c;
int i;
struct
{
    int x, y;
} ponto;
int v[4];
```

- os endereços poderiam ser algo como:
 - endereço de c = 24515
 - endereço de i = 24516
 - endereço de ponto = 24520
 - endereço de ponto.x = 24520
 - endereço de ponto.y = 24524
 - “endereço” de v = 24528
 - endereço de v[0] = 24528
 - endereço de v[1] = 24532
 - endereço de v[2] = 24536
 - endereço de v[3] = 24540

Usamos o operador & para obter o endereço de uma variável

- Exemplo:
 - &i devolve o endereço de i.
- Código:

```
printf("endereço de c = %p\n", &c);
```

```

printf("endereço de i = %p\n", &i);
printf("endereço de ponto = %p\n", &ponto);
printf("endereço de ponto.x = %p\n", &ponto.x);
printf("endereço de ponto.y = %p\n", &ponto.y);
printf("endereço de v = %p\n", v);
printf("endereço de v[0] = %p\n", &v[0]);
printf("endereço de v[1] = %p\n", &v[1]);
printf("endereço de v[2] = %p\n", &v[2]);
printf("endereço de v[3] = %p\n", &v[3]);

```

O lugar mais comum de encontrar/usar o operador & é na função scanf

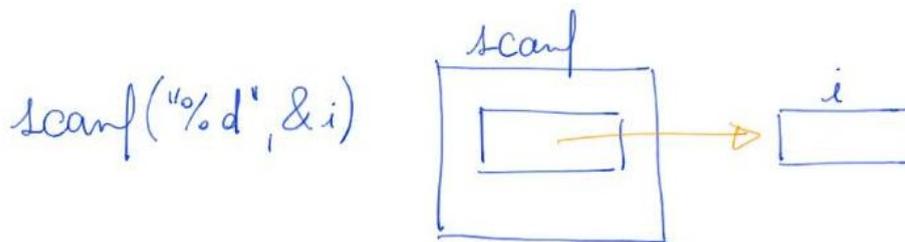
- Código:

```

int i;
scanf("%d", &i);
printf("endereço de i = %p\n", &i);
printf("conteúdo de i = %d\n", i);

```

- Por que precisamos usar &i?



Apontadores

São variáveis que armazenam endereços.

Declaramos um apontador p de um tipo de variável var colocando

- um * entre o nome do tipo e o nome da variável,
 - i.e., var * p;
- Usamos o valor especial NULL para indicar que
 - um apontador não endereça qualquer variável.
- Exemplos

```

char *p1;
int *p2, i;

p1 = NULL;

```

```
p2 = &i;
```

Se um apontador p armazena o endereço (&i) de uma variável i, dizemos que

- p aponta para i
- p é o endereço de i



- *p é o objeto apontado por p.
 - No exemplo da figura anterior, *p = i.
- Em geral, se p = &i então *p é igual a i.

Exemplo:

- Maneira complicada (que brinca com apontadores) de fazer c = a + b.

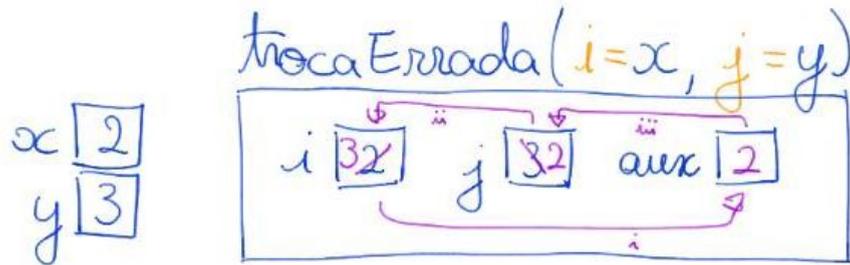
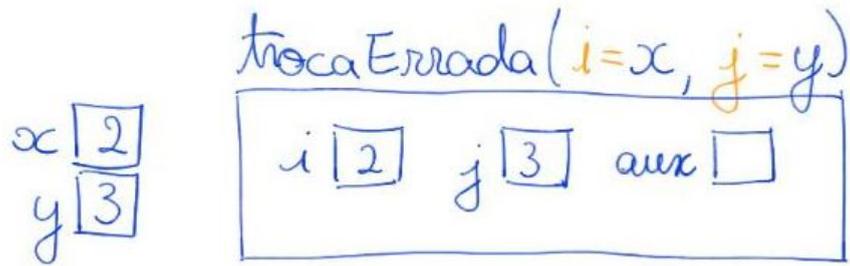
```
int a = 2;  
int b = 3;  
int c;  
int *p;  
int *q;  
p = &a;  
q = &b;  
c = *p + *q;
```

Uso de apontadores como parâmetros na função troca

```
void trocaErrada(int i, int j)  
{  
    int aux;  
    aux = i;  
    i = j;  
    j = aux;  
}
```

exemplo de uso

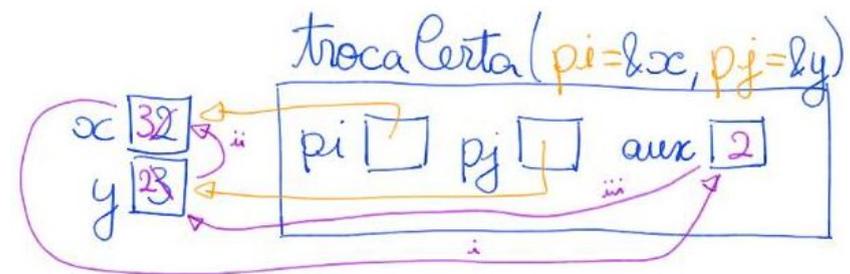
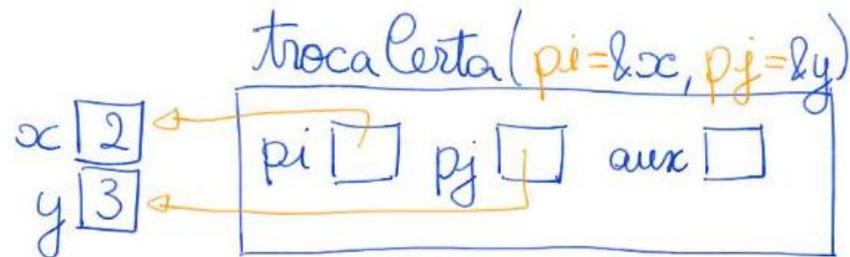
```
a = 1;    b = 2;  
printf("a = %d, b = %d\n", a, b);  
trocaErrada(a, b);  
printf("a = %d, b = %d\n", a, b);
```



```
void trocaCerta(int *i, int *j)
{
    int aux;
    aux = *i;
    *i = *j;
    *j = aux;
}
```

exemplo de uso

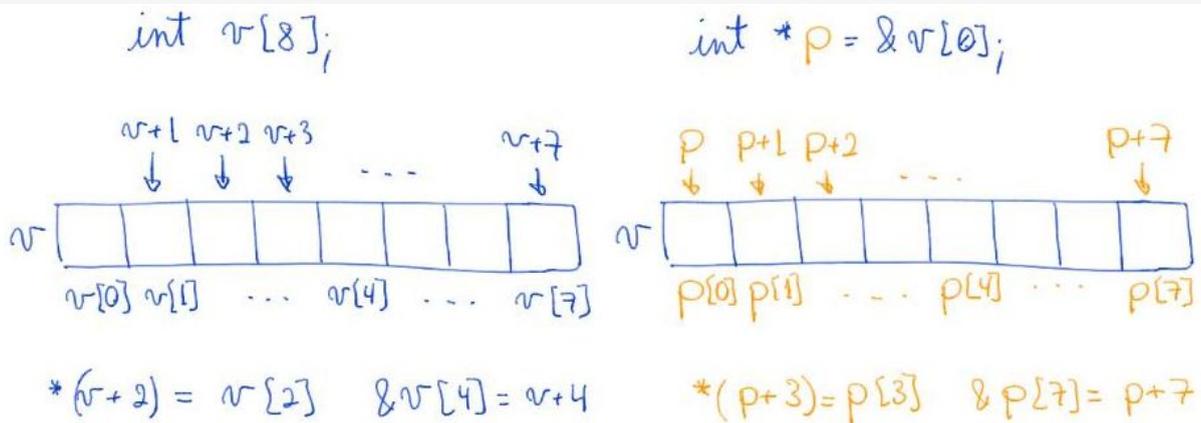
```
a = 1;   b = 2;
printf("a = %d, b = %d\n", a, b);
trocaCerta(&a, &b);
printf("a = %d, b = %d\n", a, b);
```



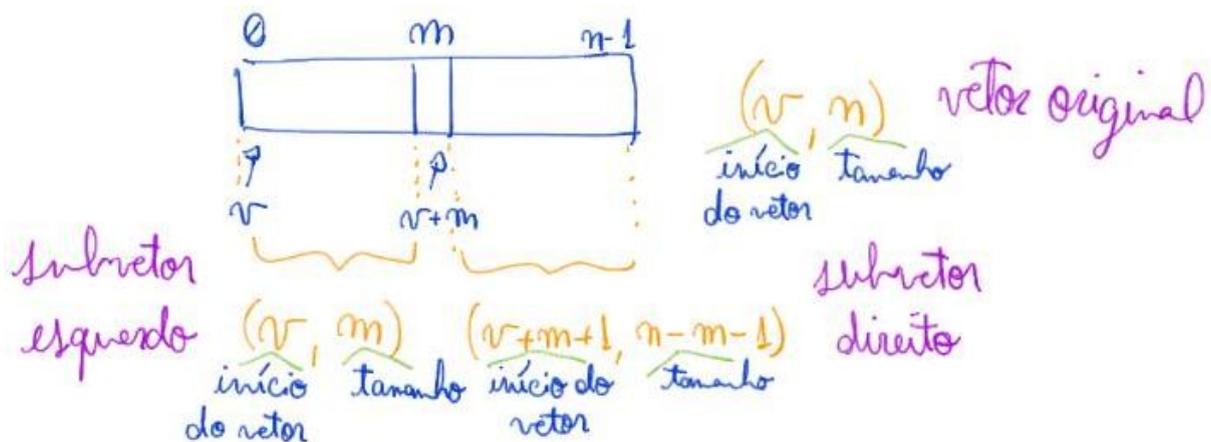
Vetores e aritmética de apontadores:

- Em C existe uma relação muito forte entre apontadores e vetores.
- O nome de um vetor é sinônimo do endereço da posição inicial do vetor.
 - Assim, $&v[0]$ é igual a v
 - e, de modo mais geral, $&v[i]$ é igual a $v + i$
 - ou, $v[i]$ é igual a $*(v + i)$

```
int v[8];
int *p;
p = &v[0];
```



- Ponteiros (p) e nomes de vetores (v) são equivalentes em quase tudo,
 - exceto que nomes de vetores alocados estaticamente são imutáveis.
 - Por isso, operações como $v++$ ou $v = v + 3$ são ilegais.
- Na figura a seguir remetemos à busca binária e mostramos
 - como definir subvetores usando aritmética de apontadores.

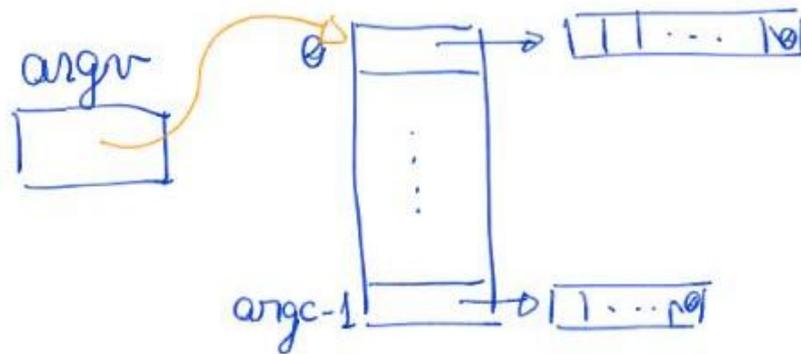


Como no exemplo das funções `trocaCerta` e `scanf`, é comum usarmos

- apontadores como parâmetros de função para devolver resultados.
 - Note que, muitas vezes usamos o comando `return` para isso,
 - mas ele só permite devolver um valor de um tipo simples,
 - embora esse valor possa ser um apontador.

Outro uso importante de apontadores como parâmetros de função é para passar

- vetores sem copiar seus conteúdos para a área de memória da função,
 - já que isso seria muito ineficiente.
- Um exemplo desse uso são os argumentos da função main
 - argc é um inteiro.
 - c vem de count.
 - Trata-se do número de argumentos na linha de comando.
 - argv[] é um vetor de strings.
 - Uma string é um vetor de caracteres terminado em '\0'.
 - Cada string contém um dos argumentos da linha de comando.
 - argv[0] é o próprio nome do programa chamado.



- Exemplo de chamada “.\entrada 10” em que
 - argc = 2
 - argv[0] = “.\entrada”
 - argv[1] = “10”
- Código:

```
int main(int argc, char *argv[])
{
    int n;
    if (argc != 2)
    {
        printf("Parametros incorretos. Ex.: .\entrada 15\n");
        return 0;
    }
    printf("%s\n", argv[0]);
    n = atoi(argv[1]);
    printf("%d\n", n);
    return 0;
}
```

Estruturas

Uma estrutura (struct) é uma coleção de variáveis,

- possivelmente de tipos diferentes.

Exemplo:

```
struct
{
    int dia;
    int mes;
    int ano;
} aniversario;
```

Usamos o operador . para acessar um campo de uma variável que é uma estrutura

- Exemplo:

```
aniversario.dia = 20;
aniversario.mes = 10;
aniversario.ano = 2010;
```

Normalmente damos um nome para as estruturas que declaramos.

- Assim fica fácil declarar diversas variáveis daquele tipo.
- Exemplo:

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

struct data aniversario;
struct data casamento;
```

Note que a declaração “struct nomeEscolhido” define um tipo.

- Para evitar repetir essa expressão em toda declaração de variável
 - usamos typedef para definir uma abreviatura.
- Exemplo:

```
typedef struct data Data;
```

```

struct data
{
    int dia;
    int mes;
    int ano;
};

```

- ou, numa forma equivalente

```

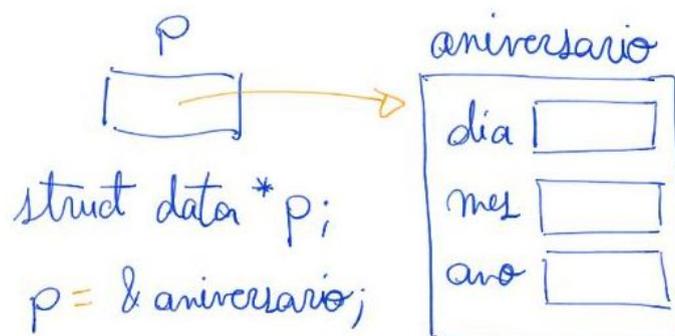
typedef struct data
{
    int dia;
    int mes;
    int ano;
} Data;

```

Estruturas e apontadores:

Quando um apontador endereça uma variável que é uma estrutura

- podemos acessar seus campos de duas formas equivalentes.
 - Sendo “Data * p = &aniversario”
 - temos “(*p).mes”
 - igual a “p->mes”
 - igual a “aniversario.mes”
- Exemplo:



- Código:

```

Data *p;
p = &aniversario;
(*p).dia = 10;
printf("%d\n", (*p).dia);
p->dia = 15;
printf("%d\n", p->dia);

```