Ponteiros

Linguagem C

Rafael Vargas Mesquita

Roteiro

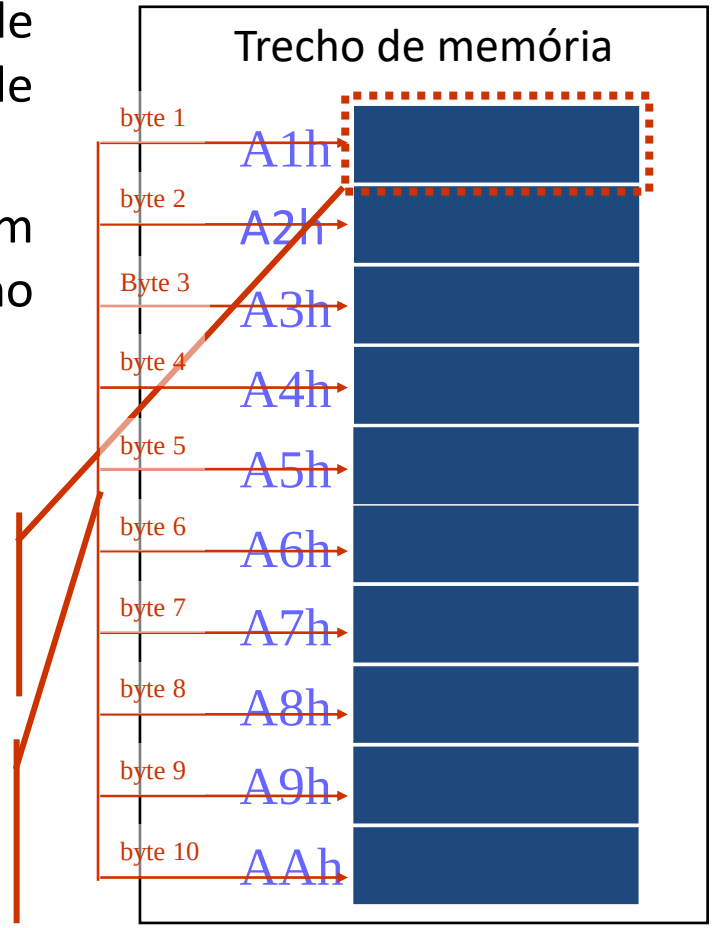
- Considerações sobre a memória
- Variáveis versus memória
- Ponteiros
- Aritmética de ponteiros

Considerações sobre a memória

- O computador possui uma grande capacidade de memória RAM, que o programador pode utilizar em seus programas.
- Por ser muito grande, vamos utilizar um pedaço, um trecho, de memória como exemplo.

A memória do computador é dividida em bytes. Cada pedaço azul representa 1 byte.

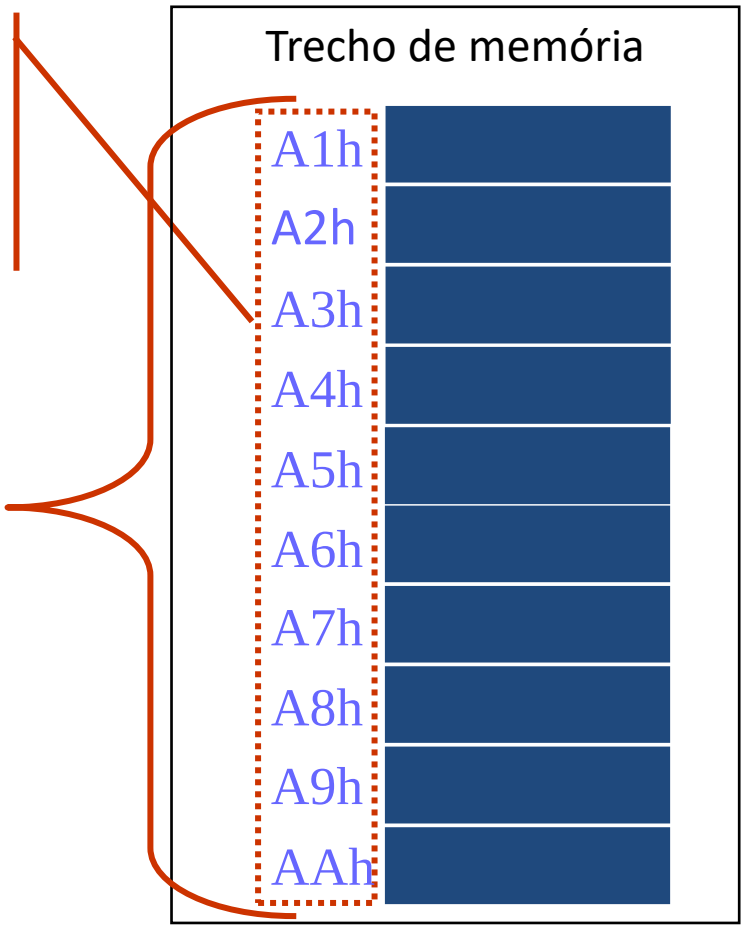
Portanto, neste trecho de memória, pego como exemplo, nós temos 10 bytes



Considerações sobre a memória

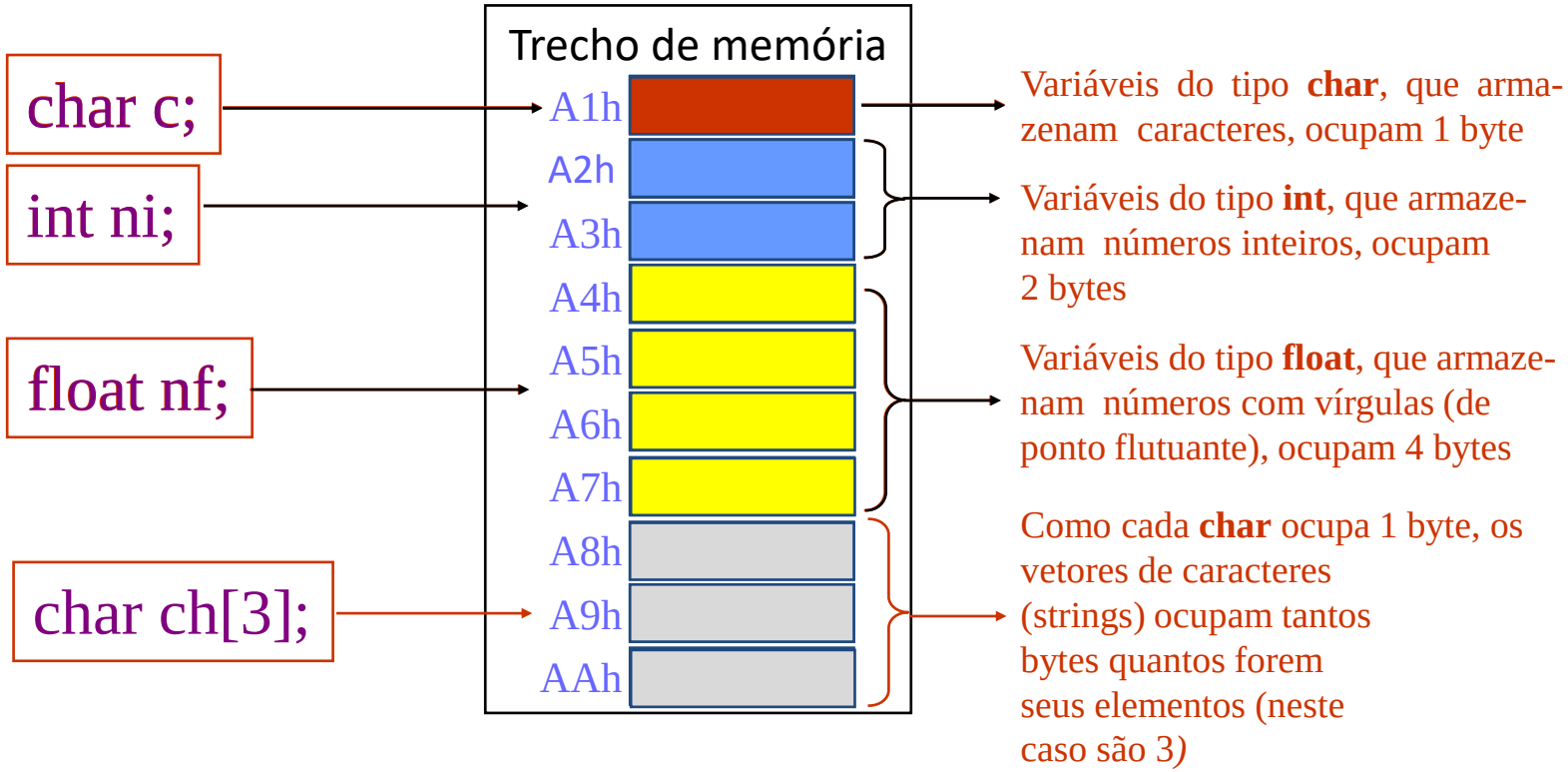
Esses números que aparecem ao lado de cada byte representam seu endereço (sua posição) de memória *(note que eles estão em hexadecimal)*

É nestes locais (bytes) que o computador vai armazenar o código e as variáveis dos programas que você fizer



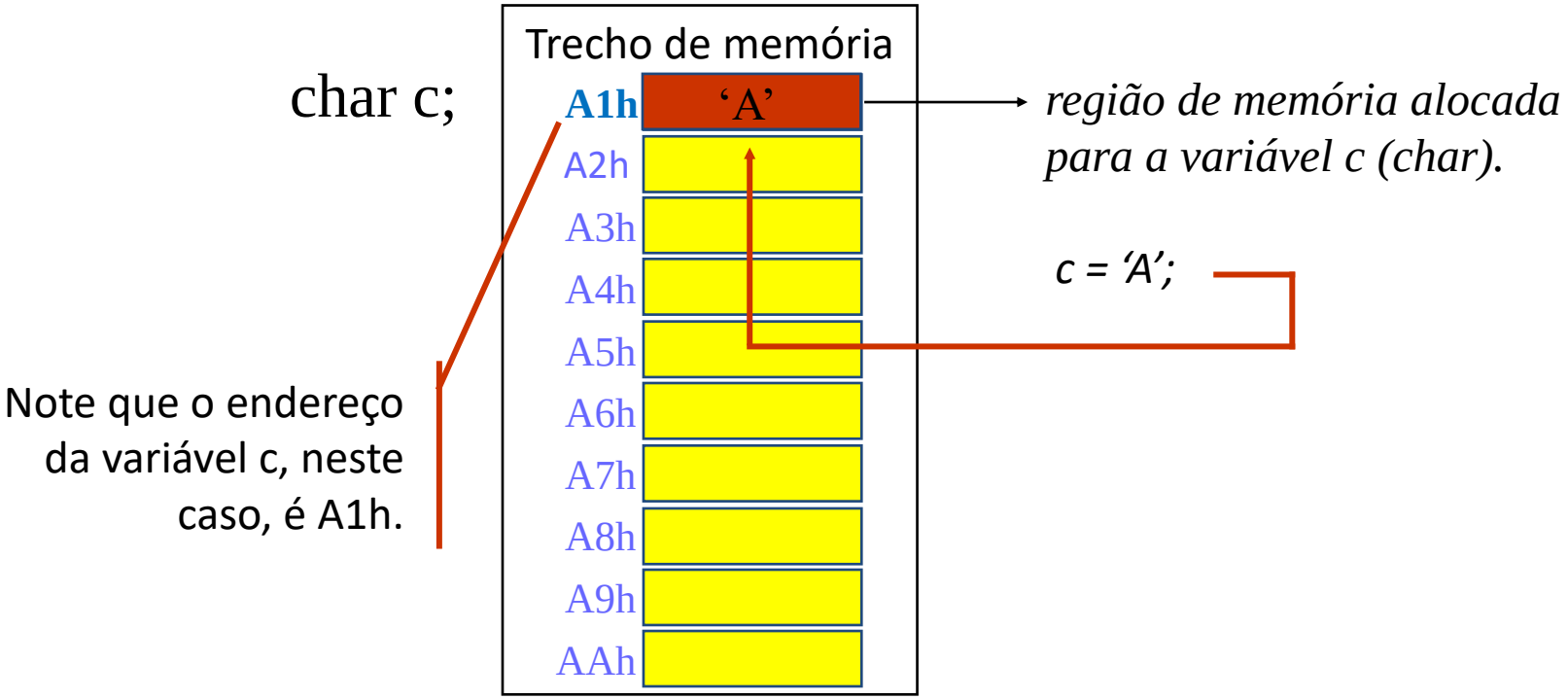
Variáveis versus memória

- Quando você declara uma variável, o compilador reserva uma região de memória para ela.
- Essa região de memória é reservada de acordo com o tamanho do tipo de dado para o qual a variável foi declarada.



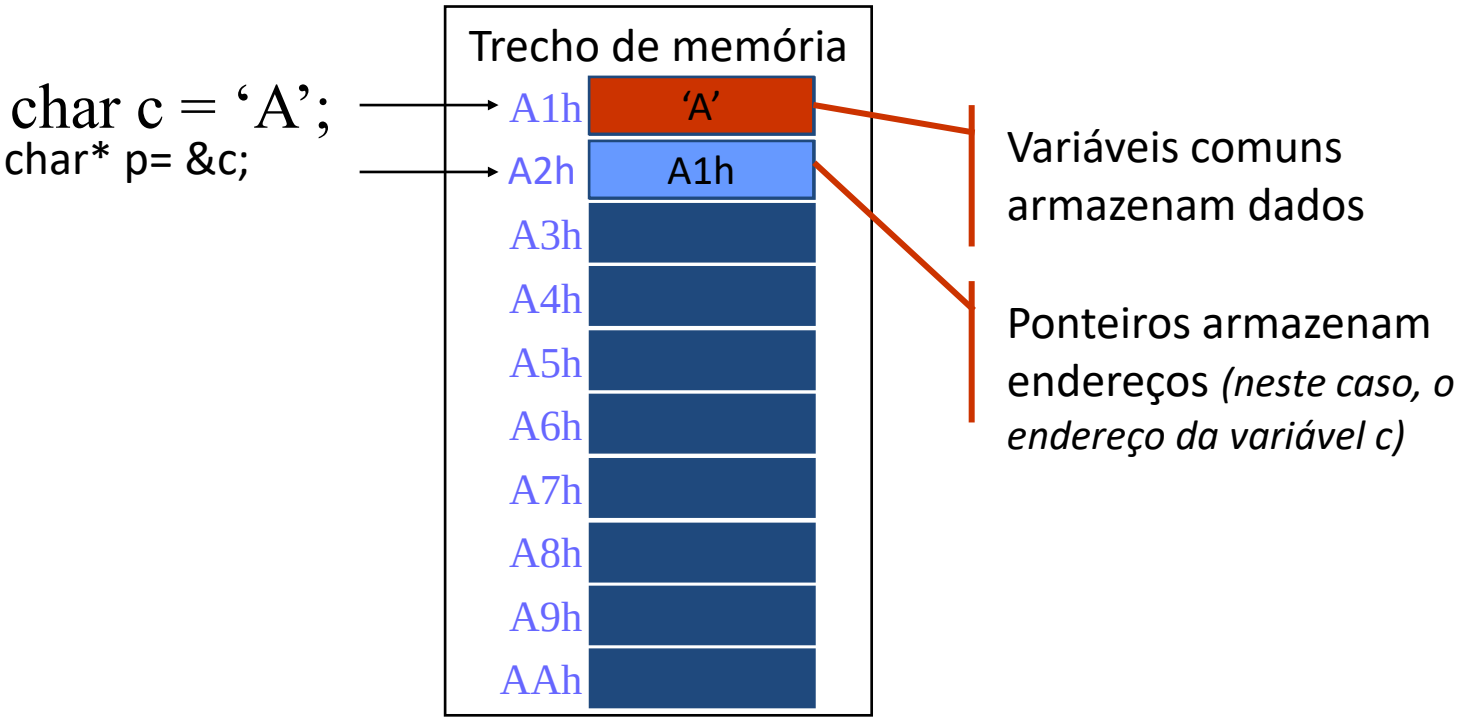
Variáveis versus memória

- Portanto, cada variável fica em um endereço de memória diferente;
- Quando fazemos uma atribuição a uma variável, o valor atribuído é colocado na região de memória que foi reservada para ela.



Ponteiros

- Sendo assim, nós podemos manipular o conteúdo de uma variável utilizando ela mesmo ou utilizando o seu endereço.
- A linguagem C oferece um tipo especial de variável chamado **ponteiro**. Os ponteiros são variáveis que armazenam endereços de outras variáveis.



Declarando ponteiros

- Os ponteiros são iguais às variáveis comuns. A única diferença está no fato dele armazenar um endereço, e não um dado ou valor.
- Para informar que você quer declarar um ponteiro, e não uma variável comum, coloque o símbolo de asterisco ao lado do tipo da variável:

```
char* ponteiro;
```

- O símbolo de asterisco é que vai indicar ao **C** que você quer um ponteiro e não uma variável comum. Fora esse detalhe, a declaração é idêntica a de uma variável comum.

Declarando ponteiros

- Para cada tipo de variável há um ponteiro específico.
- Isso significa que se você vai armazenar o endereço de uma variável do tipo int, deve criar um ponteiro para int (que é int*). Se for char, um ponteiro para char (char*).
  - char* pc;
- No exemplo acima, o C vai saber que você está criando uma variável chamada pc que vai armazenar endereços de variáveis do tipo char.
  - int* pn;
- No exemplo acima, o C vai saber que você está criando uma variável chamada pn que vai armazenar endereços de variáveis do tipo int.

Declarando ponteiros

- Alguns exemplos:

Tipo	Variável comum	Ponteiro p/o tipo
char	char letra;	char* pontLetra;
int	int numero;	int* pontNumero;
float	float num;	float* pontNum;

Utilizando ponteiros

- Utilizar um ponteiro é simples. Como qualquer variável, para colocar um valor dentro dela, basta atribuí-lo:

```
char* ponteiro = 10030;   // coloca o endereço 10030 no ponteiro
```

- Porém, os ponteiros são mais utilizados para armazenar o endereço de outras variáveis:

```
char c = 'A';             // coloca o valor 'A' na variável c
char* ponteiro = &c;      // coloca o endereço da variável c no ponteiro
```

- Note que o operador **&** (que retorna o endereço de uma variável) foi utilizado!

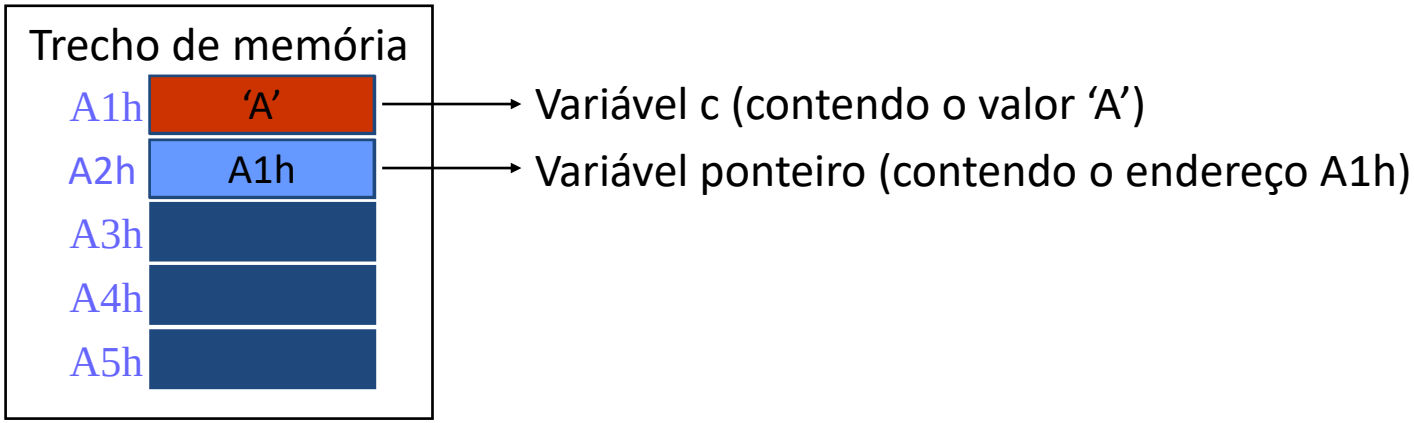
OBS: A princípio, você não pode atribuir endereços de variáveis que não sejam do mesmo tipo do ponteiro:

```
int a = 20;
char* pont = &a; // Errado, pois a não é do tipo char!
```

Utilizando ponteiros

- Se você mandar imprimir uma variável do tipo ponteiro, ela vai mostrar o endereço armazenado nela:

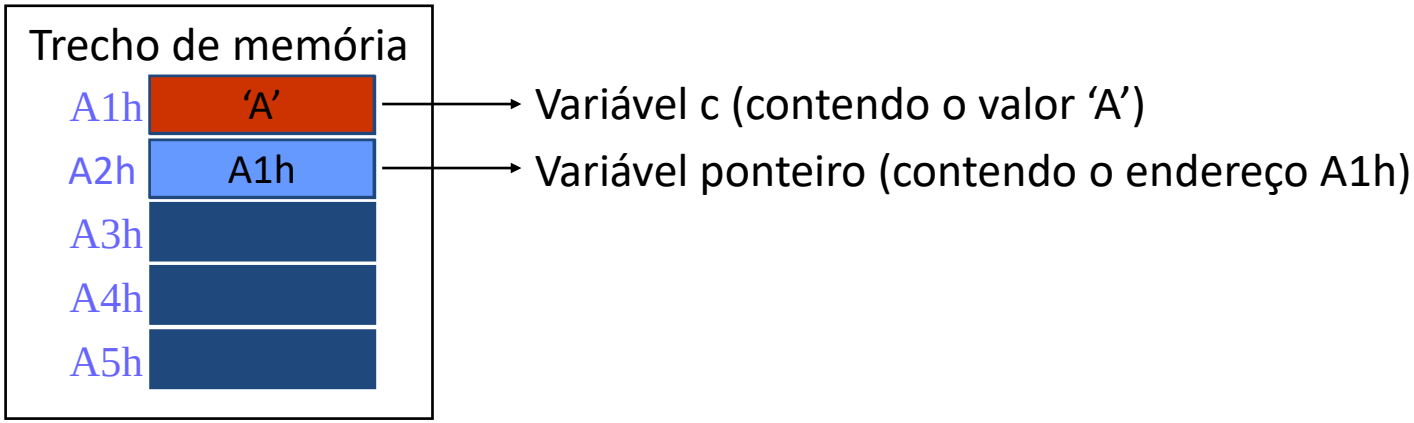
```
char c = 'A';                 // coloca o valor 'A' na variável c
char* ponteiro = &c;          // coloca o endereço da variável c no ponteiro
printf("%d", ponteiro);       // imprime o conteúdo do ponteiro, que é o
                              // endereço da variável c
```



Utilizando ponteiros

- Você pode utilizar o ponteiro para mostrar ou manipular o conteúdo do endereço (da variável) que ele aponta.
- Para fazer isso, você deve utilizar o operador * :

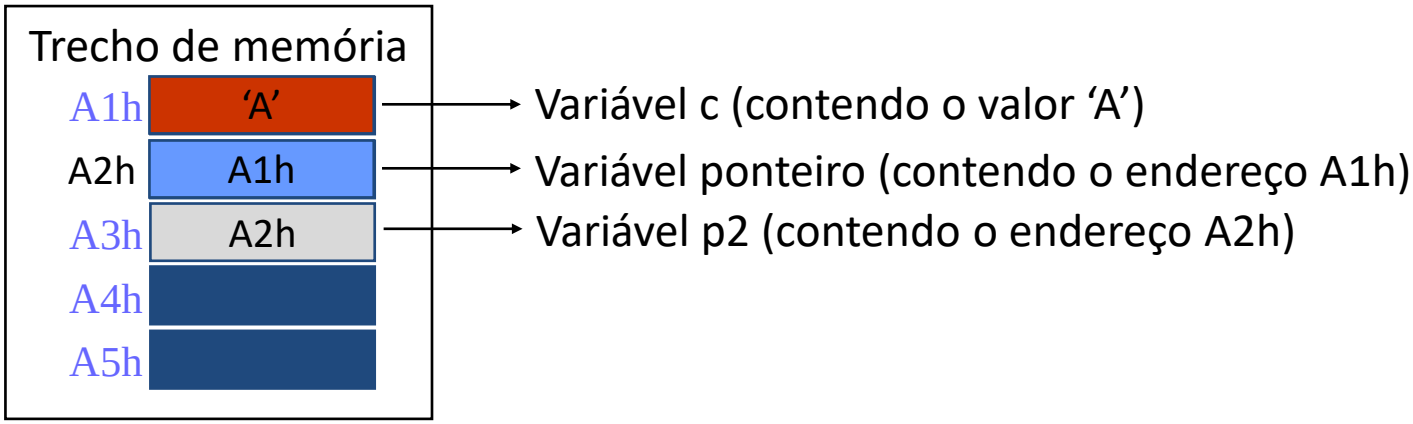
```
char c = 'A';                 // coloca o valor 'A' na variável c
char* ponteiro = &c;          // coloca o endereço da variável c no ponteiro
printf("%c", *ponteiro);      // imprime o conteúdo do endereço para o qual
                              // o ponteiro aponta ( que é 'A' )
```



Utilizando ponteiros

- Ponteiros também têm endereço. Logo, você também pode imprimir o seu endereço ou armazená-lo ou utilizá-lo em um outro ponteiro:

```
char c = 'A';                  // coloca o valor 'A' na variável c
char* ponteiro = &c;           // coloca o endereço da variável c no ponteiro
printf("%d", &ponteiro);       // imprime o endereço do ponteiro (A2h)
char** p2 = &ponteiro;         // cria um ponteiro de ponteiro, e coloca o
                               // endereço do ponteiro lá
```



Ponteiros para ponteiros

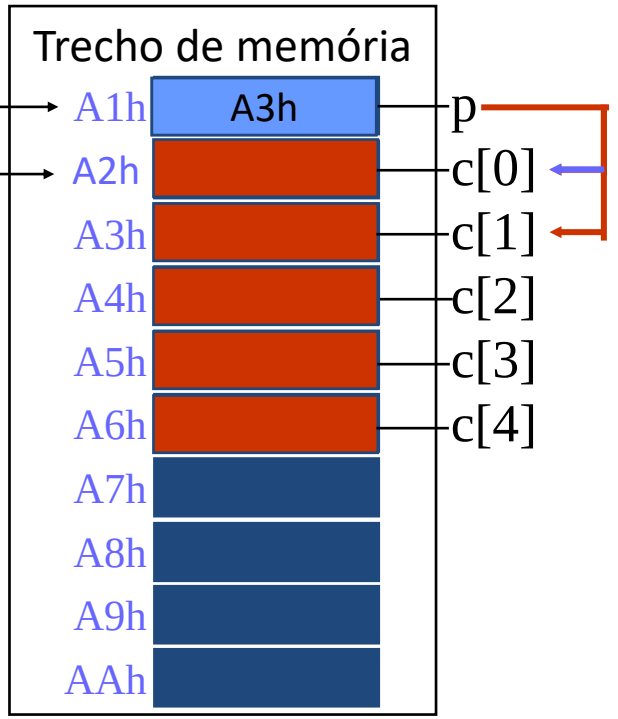
- C nos permite criar, também, ponteiros com níveis de apontamento. O que isso quer dizer? Simples. É possível criar um ponteiro que aponte para outro ponteiro, criando assim níveis, pois um ponteiro poderá apontar para outro ponteiro, que, por sua vez, aponta para outro ponteiro, que aponta para um ponteiro diferente e assim por diante.

```
int main(){
        int x, *p, **q;

        x = 12;
        p = &x;
        q = &p;

        printf("%d", **q);
}
```

Aritmética de ponteiros

- Assim como nas variáveis comuns, você pode realizar operações matemáticas com os endereços armazenados em ponteiros:

```
char* p;
char c[5];
p = &c[0];  // p recebe o endereço do 1º c
```



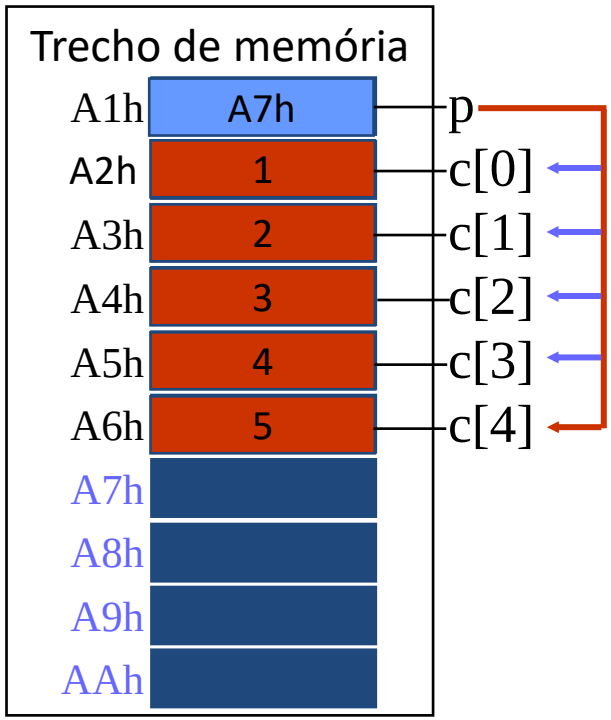
- Ao incrementar o p, a linguagem C identifica primeiramente qual é o tipo do ponteiro. Daí, incrementa o endereço que ele contém, de acordo com o tamanho do seu tipo.

```
p++;
```

Aritmética de ponteiros

- Deste modo, você pode fazer um programa que preenche o vetor sem ter que utilizar a variável do vetor (variável c):

```
void main() {
   char* p;
   char c[5];
   p = c; // o mesmo que p = &c[0]
   for(int cont=0;cont<5;cont++) {
      *p = cont+1; // coloca cont+1 no local apontado por p
       p++;  // passa para o endereço do próximo elemento
   }
 }
```

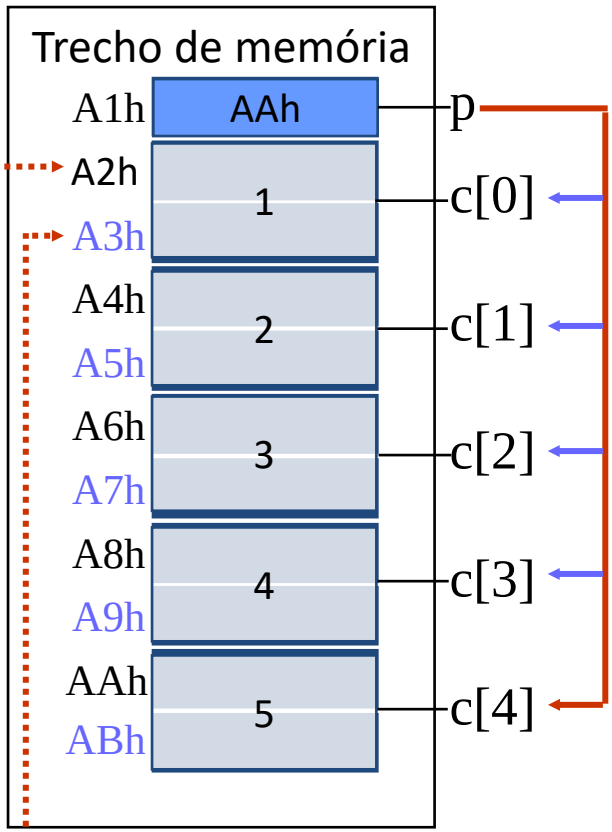


OBS: o nome do vetor (sem o índice) contém o endereço do primeiro elemento dele! *Por isso que no scanf("%s", ...) não vai o &*

Aritmética de ponteiros

- Se a variável c fosse um vetor de inteiros, o p seria incrementado de dois em dois bytes:

```
void main() {
  int* p;
  int c[5];
  p = c;  // o mesmo que p = &c[0]
  for(int cont=0;cont<5;cont++) {
     *p = cont+1; // coloca cont+1 no local apontado por p
      p++;  // passa para o endereço do próximo elemento
  }
}
```



- É por esse motivo que os ponteiros devem apontar para variáveis que sejam de um tipo igual ao seu. Caso contrário, ao serem incrementados ou decrementados, podem acabar apontando para o local errado:

```
char* p2 = &c[0];
p2++;
```

Resumo

- Operadores de ponteiros

Operador	Uso	Exemplo
*	Declaração de ponteiro	char* p;
&	Retorna o endereço	p = &variavel;
*	Utilização de conteúdo	char c = *p;
++	Incrementar o endereço *(aponta para o próximo elemento)*	p++;
--	Decrementar o endereço *(aponta para o elemento anterior)*	p--;

Nesta aula você...

- Aprendeu como as variáveis são armazenadas na memória do computador;
- Descobriu como declarar ponteiros para armazenar o endereço de outras variáveis;
- Aprendeu como um ponteiro pode ser utilizado para manipular diretamente o conteúdo da memória do computador (e, consequentemente, o conteúdo de outras variáveis).

Bibliografia

- SANTOS, Henrique José. Curso de Linguagem C da UFMG, apostila.
- FORBELLONE, André Luiz. Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados. São Paulo: MAKRON, 1993.