Algoritmos e Programação

Aula 14 - Ponteiros

Priscila Delabetha

Variáveis e endereços

- Uma variável é um espaço da memória principal reservado para armazenar dados.
- Variáveis possuem:
  - **Nome:**
    - Identificador usado para acessar o conteúdo.
  - **Tipo:**
    - Determina a capacidade de armazenamento.

      Ex: int, char, float, ...
  - **Endereço:**
    - Posição na memória principal.

Variáveis e endereços

- Toda e qualquer variável utilizada por um programa reside em um determinado endereço de memória.
- O acesso ao endereço de uma variável pode ser feito simbolicamente através de seu nome

Cada trecho da memória tem um endereço único.

Não existem dois bits, em uma máquina, que tenham o mesmo endereço de memória.

Endereços de memória são um tipo de dado tão importante, ou até mais, que qualquer outro tipo de dado.

E é isso que o um ponteiro é: um tipo de dado que serve para indicar, ou armazenar, um endereço de memória.

- Um ponteiro não é um inteiro, é um tipo que armazena o endereço em que o inteiro está.
- Um ponteiro não é um float ou double, ponteiro é um tipo de dado que armazena o endereço em que o float ou double está.
- Um ponteiro não é um char, ponteiro é um tipo de dado que pode armazenar o endereço em que um caractere está.

Não confunda ponteiro com números inteiros!

Ponteiros são um tipo de abstração, criado para facilitar o trabalho da computação em baixo nível, da computação que mexe diretamente com a memória de seu computador.

Variáveis e endereços

Declaração

```
int num = 0;
```

Produz →

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0

Variáveis e endereços

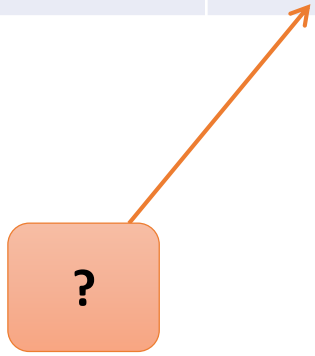
Declaração

```
int num = 0;
```

```
int num2;
```

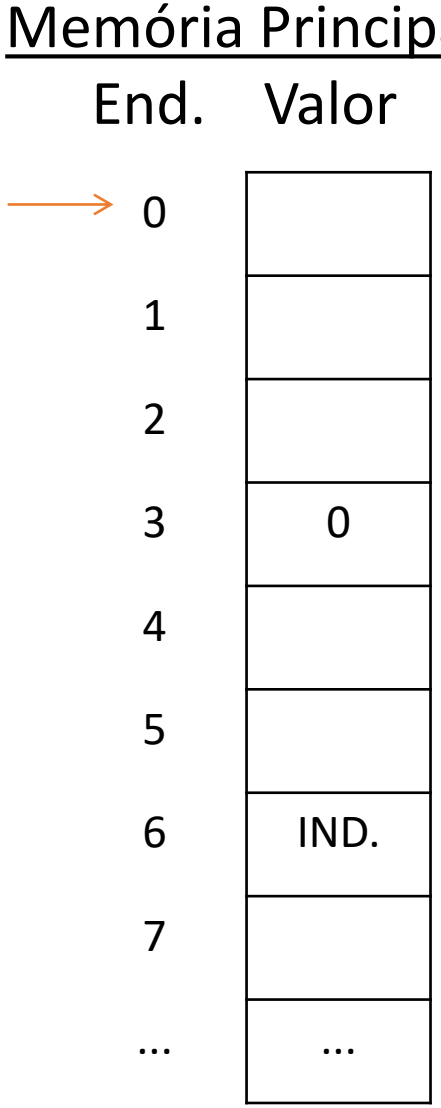
Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	indefinido



Variáveis e endereços

Memória Principal



Lista das variáveis existentes

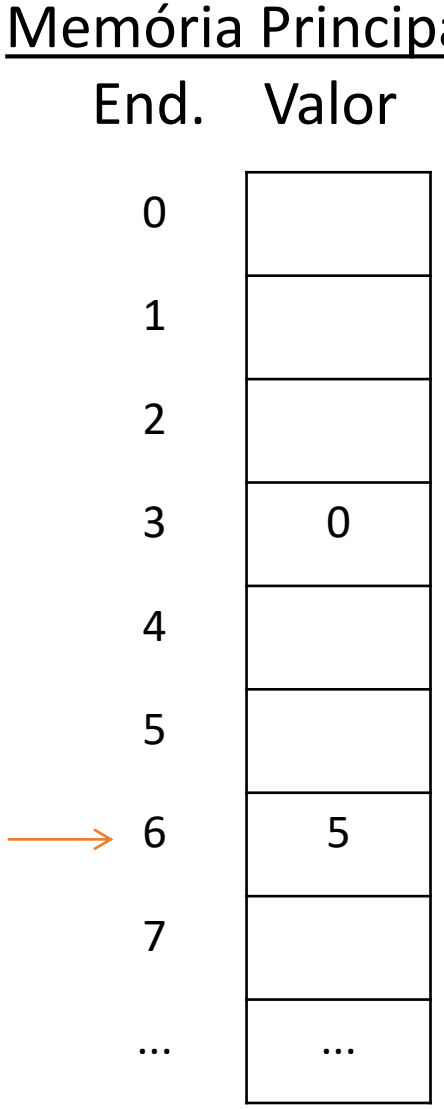
Nome	Tipo	Endereço	Valor
num	int	3	0
num2	Int	6	indefinido

O que acontece quando esse comando é executado?

```
num2 = 5;
```

Variáveis e endereços

Memória Principal



Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	5

num2 = 5;

ou

Podemos fazer isso manualmente?

endereço 6 = 5;

Sempre que declaramos uma variável e usamos ela, estamos trabalhando com seu valor.

```
int num=34;
int num2=1234;
char letrinha= 'c';
```

- Para saber o endereço da variável ‘num’, fazemos: &num
- Para saber o endereço da variável ‘num2’, fazemos: &num2
- Para saber o endereço da variável ‘letrinha’, fazemos: &letrinha

```
int main(void){
int numero1=1, numero2=2;
char letra1='a',
letra2='b';
printf("Valor numero1: %d\n", numero1);
printf("Endereco na memoria: %d\n\n", &numero1);
printf("Valor numero2: %d\n", numero2);
printf("Endereco na memoria: %d\n\n", &numero2);
printf("Valor letra1: %c\n", letra1);
printf("Endereco na memoria: %d\n\n", &letra1);
printf("Valor letra2: %c\n", letra2);
printf("Endereco na memoria: %d\n\n", &letra2);
```

```
numero1=2112;
numero2=666;
letra1='A';
letra2='B';
printf("Valor numero1: %d\n", numero1);
printf("Endereco na memoria: %d\n\n", &numero1);
printf("Valor numero2: %d\n", numero2);
printf("Endereco na memoria: %d\n\n", &numero2);
printf("Valor letra1: %c\n", letra1);
printf("Endereco na memoria: %d\n\n", &letra1);
printf("Valor letra2: %c\n", letra2);
printf("Endereco na memoria: %d\n\n", &letra2);


}
```

Exercício

Crie duas variáveis de cada tipo: char, int, float - e mostre o endereço delas usando **`&variavel`**

Ao declarar e ver os endereços, note que as variáveis do mesmo tipo, que foram declaradas juntas, estão em endereços de memória contínuos.

No caso das chars, elas estão ao lado da outra, pois só ocupam 1 byte cada.

No caso dos inteiros e floats, o espaço de endereço de uma variável pra outra é de 4 unidades, pois cada variável desta ocupa 4 bytes.

Nas variáveis do tipo double, seus endereços estão distantes em 8 unidades, bytes, um do outro.

Então...

Um ponteiro serve justamente para APONTAR para esse endereço que você viu!

Ele não guarda valores numéricos ou caracteres de texto. **GUARDA UM ENDEREÇO!**

Como declarar ponteiros?

Para declarar um ponteiro em C basta colocarmos um asterisco - * - antes do nome desse ponteiro.

Sintaxe:

tipo *nome;

Por exemplo:

```
int *aponta_inteiro;
float *aponta_float;
char *aponta_char;
```

Vamos usar!

1. Lá no seu código, declare um ponteiro para cada uma das variáveis que criou antes!
2. Declare os ponteiros conforme mostrado:

```
int *p_int, *p_int2;
char *p_char, *p_char2;
```

2. Agora use os ponteiros, apontando eles para algum endereço de variável. Assim:

```
p_int = &nome_da_variavel_inteira;
```

3. Onde você fazia a impressão dos endereços, troque agora pelos ponteiros:

```
printf("Endereco: %d", p_int);
```

É para isso que eles servem!

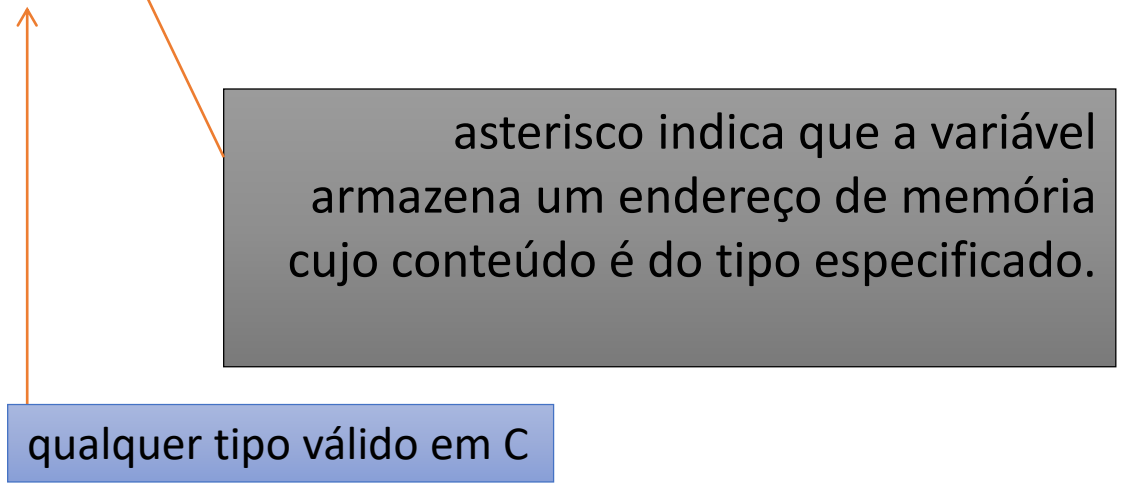
Você tem ponteiros, e já apontou eles para um endereço de memória!

Agora vamos entender todas as formar que os ponteiros podem ser chamados em seus programas!

Declaração de Ponteiros

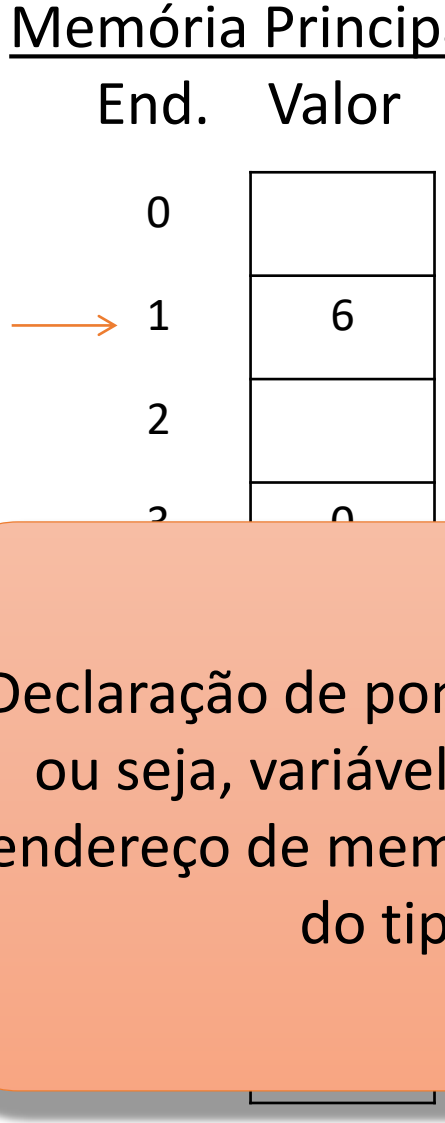
Sintaxe:

```
tipo *nome_da_variavel_ponteiro;
```



Variáveis e endereços

Memória Principal



Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	5
num3	*int	1	6

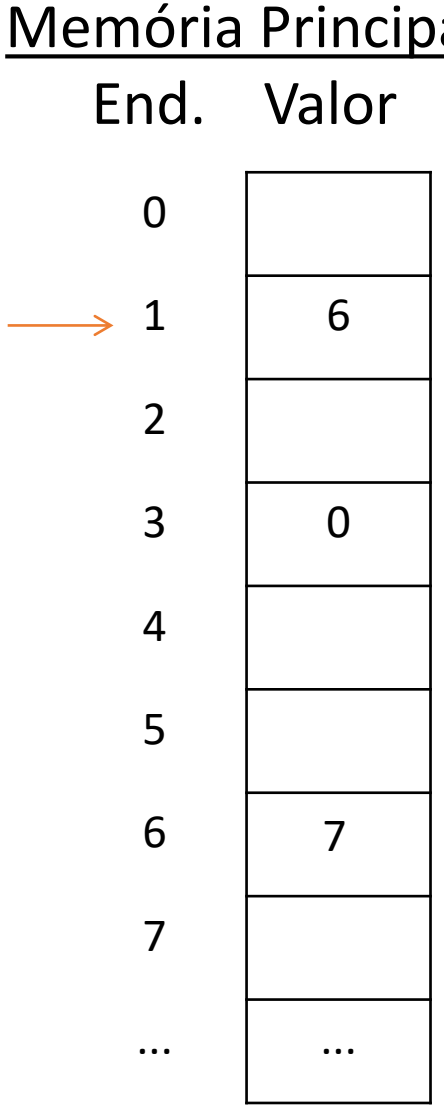
?

```
int *num3 = 6;
*num3 = 7;
```

Declaração de ponteiro do tipo int, ou seja, variável que armazena endereço de memória de variáveis do tipo int

Variáveis e endereços

Memória Principal



Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	5
num3	*int	1	6

Para onde vai o 7?

```
int *num3 = 6;
*num3 = 7;
```

Você ia pensar que era pra o 1 se não soubesse o que é um ponteiro!

Variáveis e endereços

Memória Principal

End.	Valor
0	
1	6
2	
3	0
4	
5	
6	7
7	
…	…

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	7
num3	*int	1	6

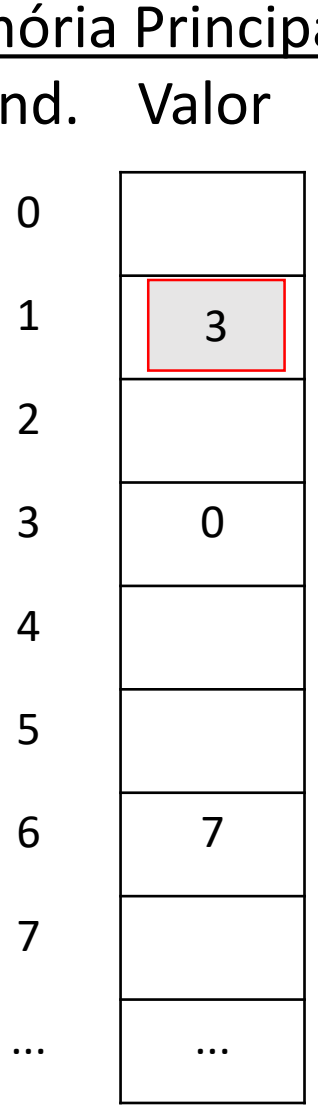
Mudamos o valor no endereço 6, o que muda o valor de num2

```
int *num3 = 6;
*num3 = 7;
```

Qual o valor de num2 ?

Variáveis e endereços

Memória Principal



Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
num	int	3	0
num2	int	6	7
num3	*int	1	3

```
int *num3 = 6;
num3 = &num;
```

Qual o valor atribuído a num3?

Operadores de Ponteiros

&

operador unário, que devolve o endereço de memória de seu operando

Exem
int
&nu

Não faz o mesmo quando utilizado na declaração de uma variável!!!
Na declaração o * diz a variável é do tipo ponteiro.

*****

operador unário que devolve o valor da variável localizada no endereço que o segue

Exemplo:

```
int *num;
*num;
```

Retorna o **valor** que se encontra no **endereço** que num **aponta**

Não confundir o *

Na declaração, o asterisco server para avisar ao compilador que aquela variável é um ponteiro!

Dentro do programa ele server para acessar o valor da variável para qual esse ponteiro está apontando!

Assim:

```
int *p_inteiro = &num1;
printf("\nEndereço apontado: ", p_inteiro);
printf("\nValor da variável que ele aponta:",
        *p_inteiro);
```

Faça isso! Realize a impressão do valor para qual os seus ponteiros estão indicando!

Valor do ponteiro **X** valor apontado pelo ponteiro

Para obtermos o valor da variável na qual o ponteiro aponta, devemos colocar um asterisco antes do ponteiro, assim, o ponteiro irá mostrar o valor da variável (a variável que ele aponta), e não mais seu endereço.

```
int * pont;
//atribuições
printf("%d", pont);
printf("%d", *pont);
```

Laboratório

Agora que você sabe como declarar, faça algumas experiências!

Além de imprimir o valor para o qual o ponteiro aponta, também mude ele!

Faça sempre as impressões da variável e do valor do ponteiro, para acompanhar o que está acontecendo!

- mudar valor da variável:

```
num1 = 12;
```

- mudar valor apontado pelo ponteiro:

```
*p_inteiro = 15;
```

- mudar o endereço para qual o ponteiro aponta:

```
p_inteiro = &variavel;
```

Atribuições com ponteiros

Como qualquer variável, um ponteiro pode ser usado no lado direito de um comando de atribuição para passar seu valor para um outro ponteiro.

Exemplo:

```
int x = 200, *p1, *p2;
p1=&x; // p1 aponta para x
p2= p1; // p2 recebe p1 e também passa a apontar para x
printf("x -%d, p1=%pe p2=%p\n", x, p1, p2); // endereço
printf("x -%d, p1=%de p2=%d\n",x,*p1,*p2); // conteúdo apontado
```

x – 200, p1 = 0022FF60 e p2 = 0022FF60
x – 200, p1 = 200 e p2 = 200

Exemplos

```
float *f; // f é um ponteiro para variáveis do tipo float
int *i; //i é um ponteiro para variáveis do tipo inteiro
char a, b, *p, c, *q; // podemos declarar junto com
                      // variáveis de mesmo tipo
```

- O asterisco é o mesmo da operação de multiplicação, mas não provoca confusão porque seu significado depende do contexto em que é usado!
- Quantos usos vocês já identificaram para o *?
- Multiplicação, declaração de variável do tipo ponteiro e retorno do valor na posição de memória a qual o ponteiro aponta

Exemplos

```
int count, q,*m;
count = 10;
m = &count; // m recebe endereço de memória da variável count
q = *m; // q recebe o valor armazenado no endereço apontado por m
```

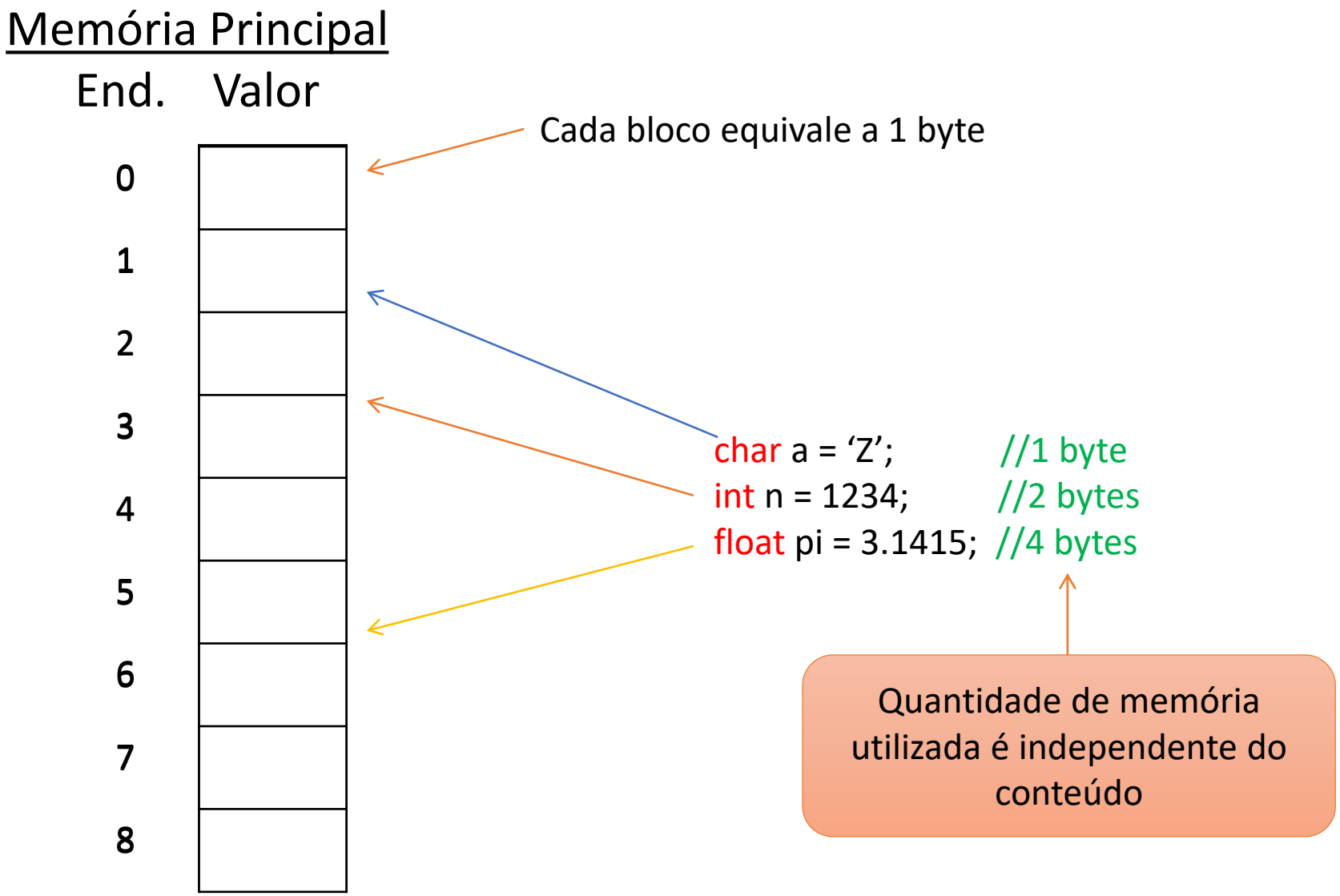
Por que os ponteiros têm que possuir um tipo?

- Diferentes tipos de dados necessitam de um número diferente de bytes de memória para armazenar seu conteúdo.
- A declaração do tipo de dado indica o número de bytes reservados para conteúdos:
  - char a = ‘Z’; //1 byte ?
  - int n = 1234; //2 bytes ?
  - float pi = 3.1415; //4 bytes ?

a função **sizeof();**

```c
#include <stdio.h>
int main() {
printf("Char: %d bytes\n", sizeof(char));
printf("Int: %d bytes\n", sizeof(int));
printf("Float: %d bytes\n", sizeof(float));
printf("Double: %d bytes\n", sizeof(double));
return 0;
}
```

Relação entre ponteiros e tipos de dados



Teste!

Vamos supor que queiramos declarar 10 mil inteiros em um vetor:

```
int vetorzao[10000];
```

E quantos bytes essa variável ocupa?

Teste com : **`sizeof(vetorzao);`**

Por ser um número muito grande de memória, há a possibilidade de máquinas ficarem lentas. A linguagem C é poderosa, você está manipulando e controlando cada pedacinho de sua máquina.

Não é à toa que C é a linguagem favorita dos hackers e a mais usada na criação de vírus.

Exercício

Dado o código à esquerda, preencha a tabela à direita com **todas as mudanças** de valor das variáveis até o final da execução do programa. Para representar o endereço de uma variável, utilize o símbolo & seguido do nome da variável. Ex.: &t (endereço da variável t).

```
void teste() {
       int t, j = 1, *p, *g;
       t = 5;
       p = &t;
       j = 3;
       g = &j;

       *p = (*g + t) * 2;
       *g = 4;

       g = p;

       for(j = 0; j < 2;j++) {
         *g = t*j + 1;
       }

       *p = t + j + *g;
}
```

t	j	p	g

A constante **NULL**

- É sempre bom inicializarmos os ponteiros, pois podem vir com lixo e você se esquecer de inicializar. Quando for usar, pensará que está usando o ponteiro de modo correto, mas estará usando o ponteiro com ele apontando para um lixo.
- Uma boa prática é apontar os ponteiros para a primeira posição de memória, que é conhecida como **NULL**. Sempre que terminar de usar um ponteiro, coloque ele pra apontar para a posição NULL.

```
int *ponteiro = NULL;
```

Exemplos

```
#include <stdio.h>
#include <stdlib.h>


int main()
{
  int count, q,*m;

  m=&count;// recebe endereço de count
  printf(”Informe count = “);
  scanf("%d", &count); // armazena no endereço de count!
  q =*m; // recebe valor apontado por m
  printf("m =%p\n", m); // %p para imprimir ponteiro (endereço apontado)
  printf("q =%d\n", q);
  printf("m aponta para %d\n\n",*m); // conteúdo apontado por m


  return 0;
}
```

```
Informe count = 20
m = 0022FF74
q = 20
m aponta para 20
```

Outra maneira de acessar a memória usando ponteiros

Memória Principal

End.	Valor
0	0
1	1
2	2
3	3
4	
5	0
6	

pNum[1] é o mesmo que que pNum + 1

Lista das variáveis existentes

Nome	Tipo	Endereço	Valor
vNum	int[3]	0	0
*pNum	*int	5	0

```
printf("%d", vNum[0]);
printf("%d",  pNum[0]);

printf("%d", vNum[1]);
printf("%d", pNum[1]);

printf("%d", vNum[2]);
printf("%d", pNum[2]);
```

0

1

2

Qual a diferença de pNum[1] para pNum++ ?

pNum[1] não altera o valor de pNum, já pNum++ altera

Aritmética de Ponteiros

Assim como outras variáveis, podemos incrementar e decrementar ponteiros.

Como os ponteiros são variáveis para armazenar endereços de memória, se fizermos operações de incremento e decremento em ponteiros, **é o endereço que irá mudar.**

Aritmética de Ponteiros

- Adição e Subtração de ponteiros:
  - Podemos somar ou subtrair inteiros de ponteiros.
  - O valor do ponteiro irá aumentar ou diminuir, dependendo do número de bytes que o **tipo** base **ocupa**, isto é, o endereço de memória apontado será deslocado em tantos bytes quanto forem os reservados para o tipo associado ao ponteiro.
- Supondo um ponteiro inteiro p1, apontando para o endereço de memória 2000:
  - p1++; // valor de p1 fica 2004*
  - p1--; // valor de p1 fica 1996
  - p1= p1 + 5; **?**

*Cada inteiro ocupa 4 bytes.

Exercícios

- Qual a diferença entre:
  - p++; (*p)++; *(p++);
  - onde p é um ponteiro para um inteiro

Exercícios

```
int main(){
  int y, *p, x;
  y = 0;
  p = &y;
  x = *p;
  x = 4;
  (*p)++;
  x--;
  (*p) += x;
  printf ("y = %d\n", y); //Qual o valor de y?
  return(0);
}
```

E as funções?

Passagem por referência

Para passarmos uma variável para uma função e fazer com que ela seja alterada, precisamos passar a referência dessa variável, em vez de seu valor.

Até agora...

Até o momento, vínhamos passando somente o valor das variáveis.

Conforme explicamos, quando passamos um valor, a função copia esse valor e trabalha com uma cópia dessa variável, e não na variável em si.

Por isso nossas variáveis nunca eram alteradas quando passadas para funções.

Passagem por Referência

Porém, é muito importante e útil que algumas funções alterem valores de variáveis.

Para fazer isso usamos a **Passagem por Referência**.

Significa que passamos um endereço.

Basta, em vez de passar o valor da variável, na passagem por referência, passar o endereço da variável para a função.

Para fazer isso, basta colocar o operador & antes do argumento que vamos enviar, e colocar um asterisco * no parâmetro da função, no seu cabeçalho de declaração, para dizer a função que ela deve esperar um endereço de memória, e não um valor.

Crie um programa que recebe um inteiro e dobra seu valor

```
#include<stdio.h>
void dobravalor(int a){
        a=a*2;
}

int main(void){
    int x=2;
    dobravalor(x);
}
```

Corrija a função para ela que altere o valor do inteiro passado para a função!

Exercício

Crie um programa em C que peça ao usuário o valor de uma variável ***x*** e depois de uma ***y***.

Crie uma função que inverta esses valores através do uso de ponteiros.

Semelhanças entre Ponteiros e Vetores

- O nome de um vetor corresponde a um **ponteiro constante**, isto é, ao ponteiro para o primeiro elemento.
- Todas as operações válidas para ponteiros podem ser executadas com vetores.
- Assim, as notações abaixo são equivalentes:

Com vetores	**Equivale a**	**Com ponteiros**
int v[100]		int *v
v[i] ou *(v+i)		*(v+i) ou v[i]
&v[i]		v+i

Semelhanças entre Ponteiros e Vetores

Com vetores	**Equivale a**	**Com ponteiros**
int v[100]		int *v
v[i] ou *(v+i)		*(v+i) ou v[i]
&v[i]		v+i

Declaração de vetor v de inteiros, com 100 elementos, indica que v é um ponteiro que aponta para o 1º elemento do vetor(v[0]), alocando a área necessária;

Declaração do ponteiro v aponta para a posição inicial, mas não aloca área para isso.

Diferença entre Ponteiros e Vetores

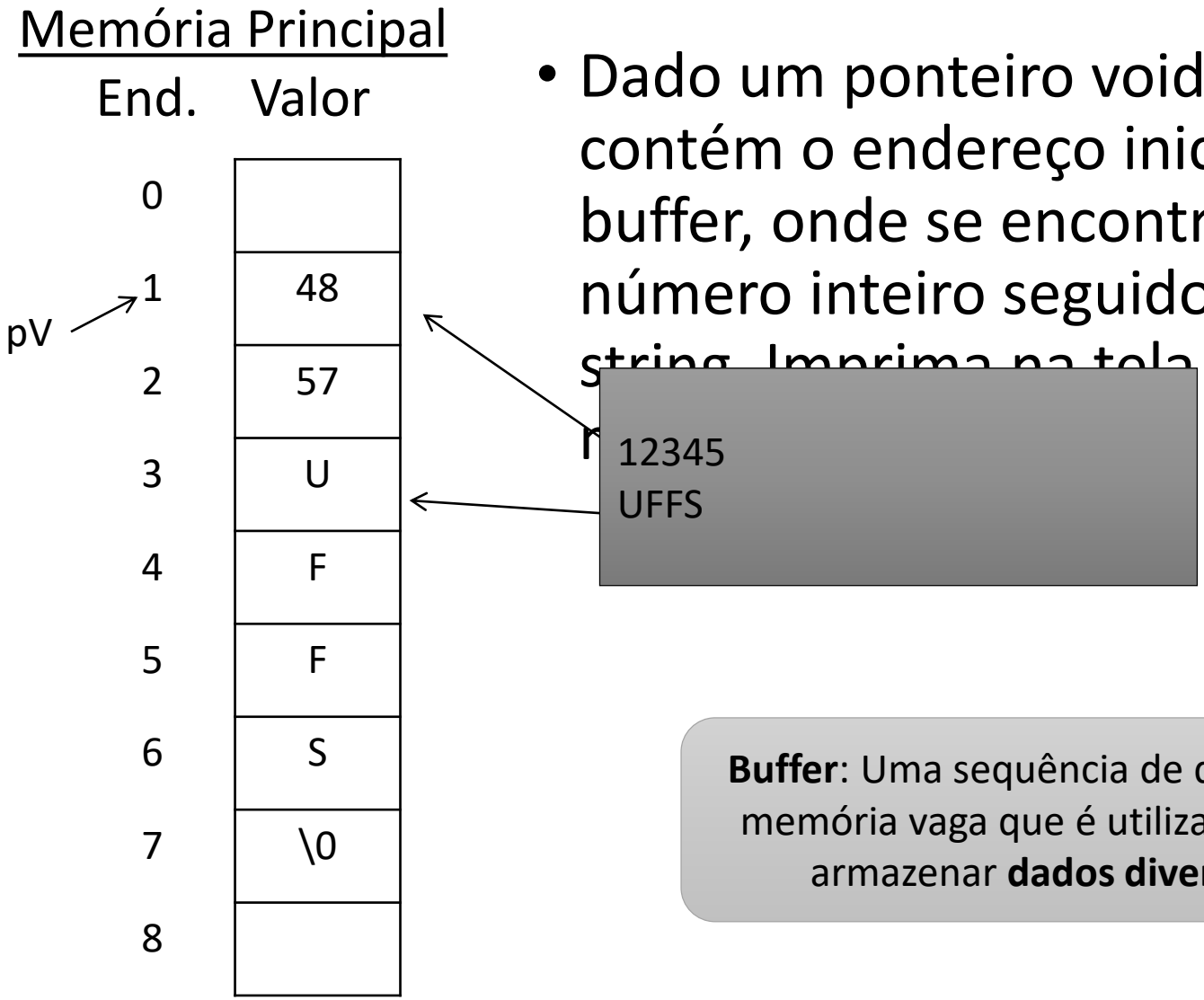
- Declaração de vetor:
  - O compilador **automaticamente reserva** um bloco de memória para que o vetor seja armazenado, do tamanho especificado na declaração (mas não controla se o uso é limitado à área reservada).
- Declaração de ponteiro:
  - o compilador aloca um ponteiro para apontar para a memória, **sem reservar** espaço de memória(usado para alocação dinâmica de memória).

Ponteiro para struct

```
struct Empregado {
  int cod;
  unsigned int salario;
};

int main(){
  Empregado EP, *pEP;
  EP.cod = 1;
  EP.salario = 1000;

  pEP = &EP;

  pEP->cod = 2;
  printf("%d", &pEP->salario);
  return(0);
}
```

Operador ->
É equivalente a usar:
(*pEP).cod
pEP->cod

Exercícios



- Dado um ponteiro void *pV que contém o endereço inicial de um buffer, onde se encontra um número inteiro seguido de uma string. Imprima na tela esse r

Buffer: Uma sequência de dados ou memória vaga que é utilizada para armazenar **dados diversos**

```
struct cliente{
    char nome[30];
    int codigo;
};

void cadastraCliente(struct cliente *cliente1){
      printf("\nDigite o codigo do seu cliente: ");
      scanf("%d", &cliente1->codigo);
      getchar();
      printf("\nDigite o nome do cliente: ");
      fgets(cliente1->nome, 30, stdin);
}

int main(){
      struct cliente c1;
      cadastraCliente(&c1);
}
```