UTFPR

Ciência da Computação

Algoritmos e Estrutura de Dados 1

Memória, Tipos Primitivos e Ponteiros

Organização dos Tipos Primitivos e Ponteiros na memória

Rafael Liberato

liberato@utfpr.edu.br

Agenda

- Memória
- Variáveis
  - Organização dos dados na memória
  - Tipos primitivos
  - Características
  - Representação simplificada na memória
- Ponteiros
  - Algumas características
  - Exemplo (Simulação)

Objetivos

Representar a organização dos dados de variáveis de tipos primitivos na memória 

Representar a organização dos dados de ponteiros na memória 

Manipular corretamente os dados na memória por meio de ponteiros 

Resultado Esperado

Manipular dados de tipos PRIMITIVOS em memória principal utilizando ponteiros.

- ❏ **1.** Você reproduz graficamente a organização de tipos primitivos na memória?
- ❏ **2**. Você reproduz graficamente a organização de ponteiros na memória?
- ❏ **3.** Você manipula dados por meio de ponteiros utilizando os operadores * e **&**?
- ❏ **4.** Você reproduz graficamente a organização de dados utilizando indireção múltipla.

Resultado Esperado

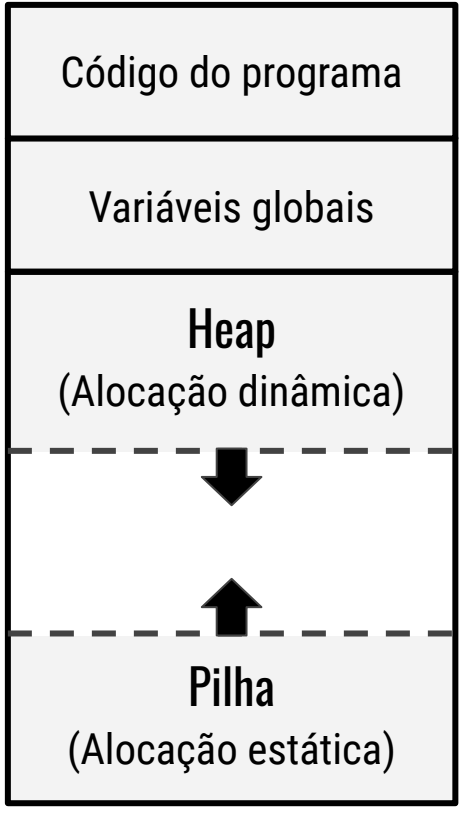
Manipular dados de tipos PRIMITIVOS em memória principal utilizando ponteiros.

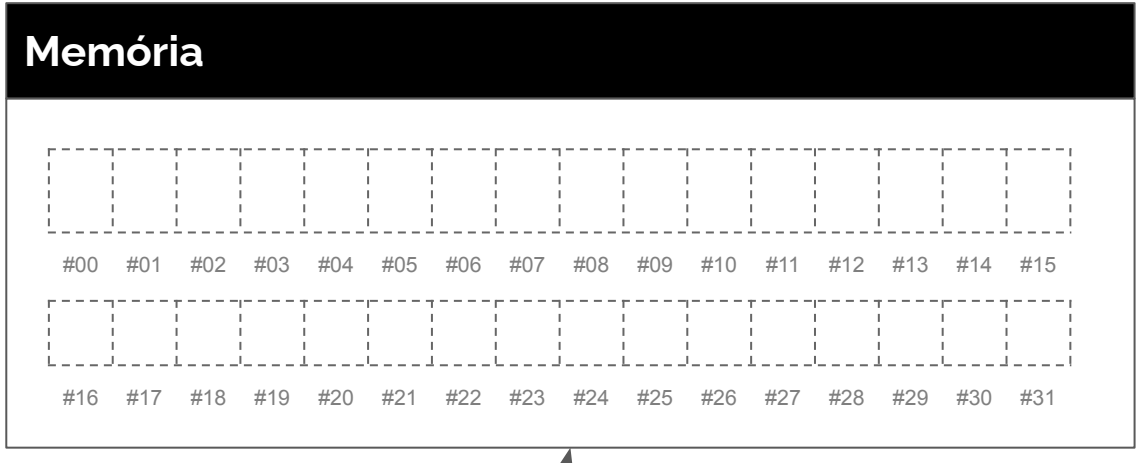
- **5.** Você manipula dados utilizando indireção múltipla.
- **6.** Você representa graficamente a organização de dados primitivos na memória a partir de um trecho de código
- **7.** Você escreve trechos de código que implementam a organização de dados primitivos na memória a partir de uma representação gráfica.

background

Memória

- Esquema didático da memória

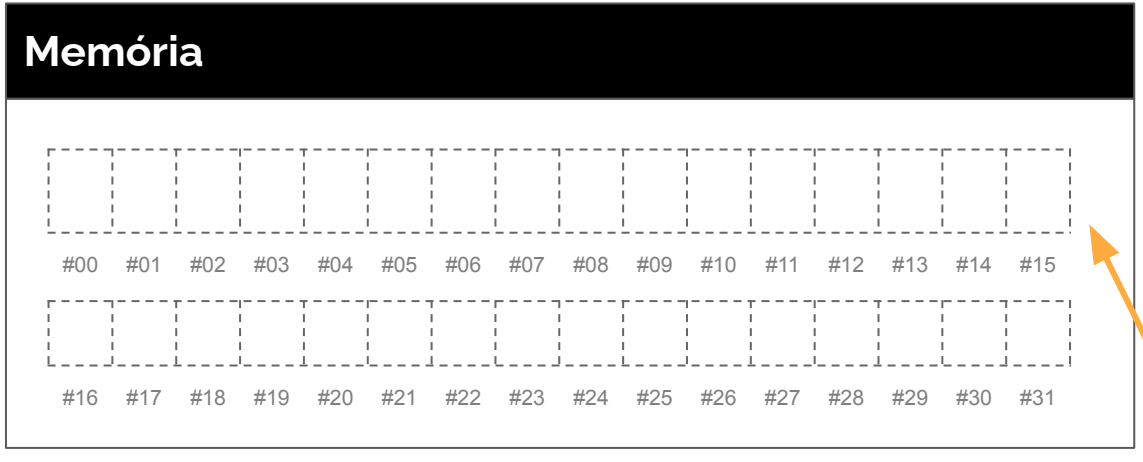




representação da memória que nós utilizaremos nesta aula

Memória

- Sequência de bytes
- Cada byte possui um endereço (tal como uma célula em uma planilha)
- Exemplo do endereçamento de um bloco de **32 bytes**



Por **questões didáticas**, ***utilizaremos o prefixo # seguido por um número inteiro para representar um endereço de memória.***

Portanto, **#15** é o endereço do byte que ocupa a décima sexta posição na memória.

Memória

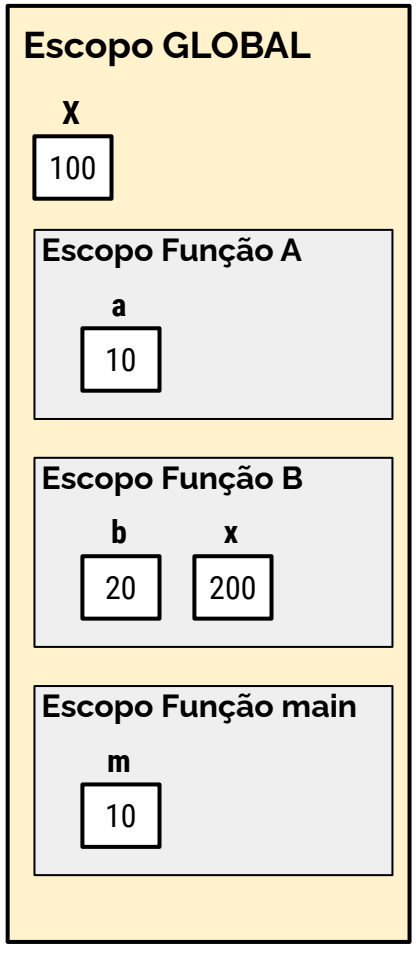
[Código no Repl.it]

Alocação Estática e escopo

- O tamanho do espaço é definido durante a codificação (**tempo de compilação**)
- **Não precisamos nos preocupar com o gerenciamento do espaço alocado**. A vida útil do espaço é definido pelo **escopo** em que a variável foi declarada
- Escopo **local** ou **global**
  - variáveis locais
  - variáveis globais

Código

```c
int x = 100;

void funcaoA(){
 int a = 10;
 printf("a: %d\n", a);
 printf("x: %d\n", x);
}

void funcaoB(){
 int b = 20;
 int x = 200;
 printf("b: %d\n", b);
 printf("x: %d\n", x);
}

int main(void) {
 funcaoA();
 funcaoB();

 int m = 10;
 printf("m: %d\n", m);
 printf("x: %d\n", x);
}
```



tipos primitivos

Tipos Primitivos

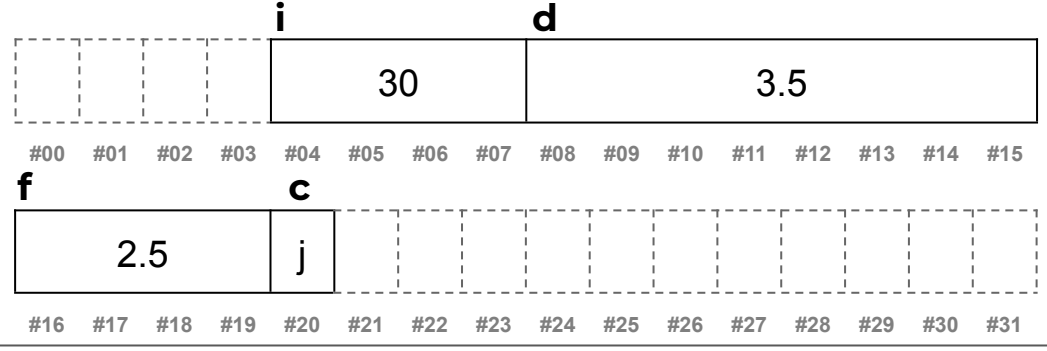
Organização na memória

Ocupa uma sequência de bytes consecutivos na memória de acordo com seu tipo de dado

Código

```
int    i = 30;
double d = 3.5;
float  f = 2.5;
char   c = 'j';
```

Memória



Tipos Primitivos

Organização na memória

Ocupa uma sequência de bytes consecutivos na memória de acordo com seu tipo de dado

Variável i

- alocada a partir do endereço **#04**
- ocupa **4 bytes** devido ao tipo **int** (bytes #04 #05 #06 e #07)
- armazena o valor **30**

Variável d

- alocada a partir do endereço **#08**
- ocupa **8 bytes** devido ao tipo **double** (bytes #08 #09 #10 e #11)
- armazena o valor **3.5**

Variável f

- alocada a partir do endereço **#16**
- ocupa **4 bytes** devido ao tipo **float** (bytes #16 #17 #18 e #19)
- armazena o valor **2.5**

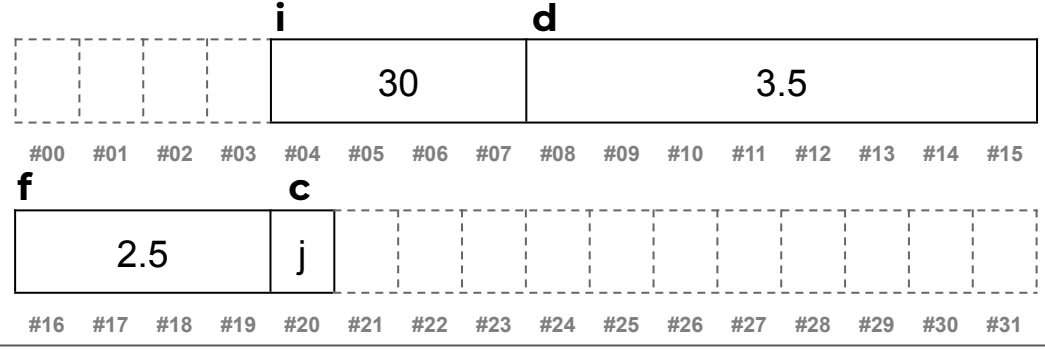
Variável c

- alocada a partir do endereço **#20**
- ocupa **1 byte** devido ao tipo **char**
- armazena o valor **j**

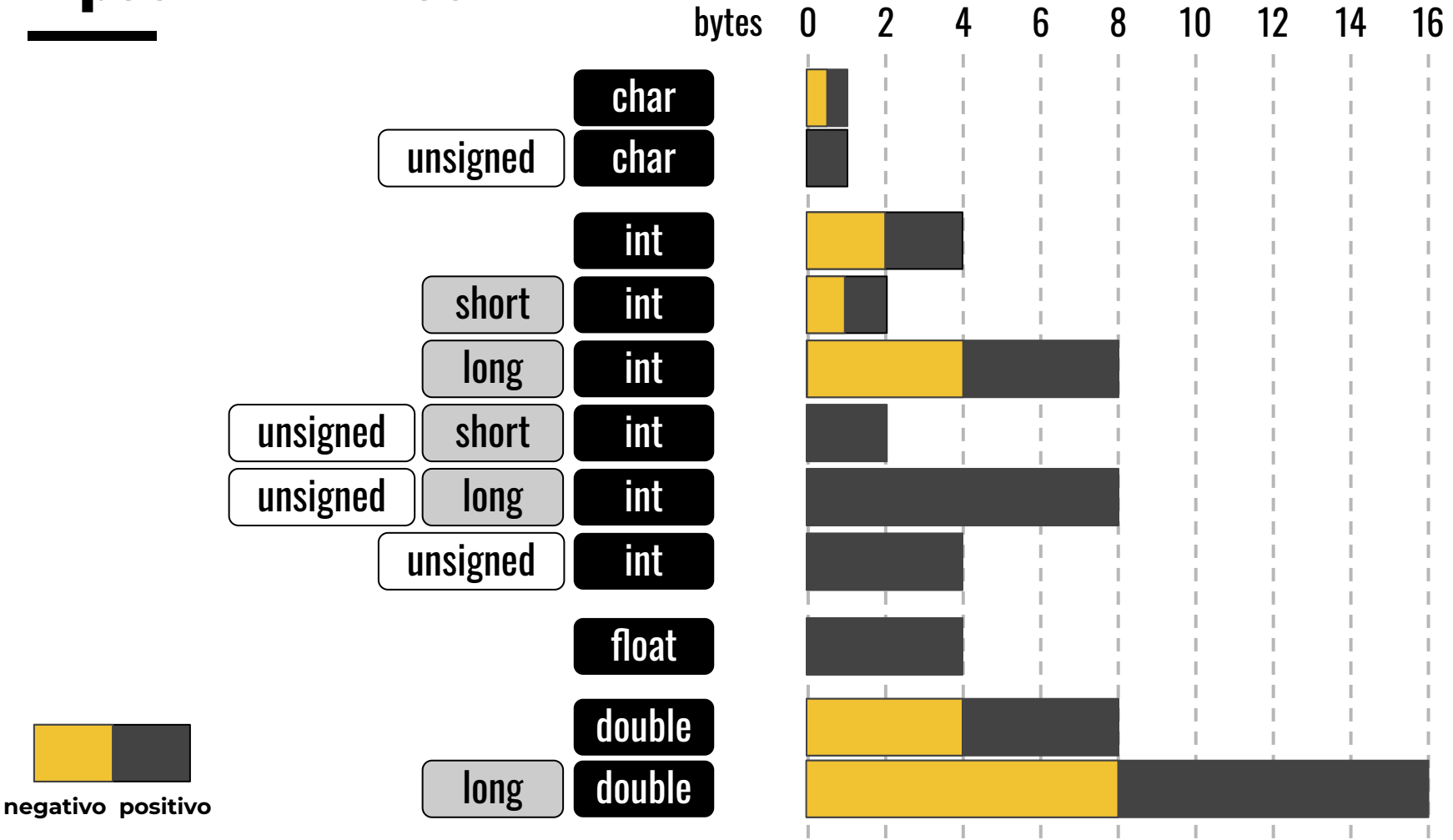
Código

```
int    i = 30;
double d = 3.5;
float  f = 2.5;
char   c = 'j';
```

Memória



Tipos Primitivos



Tipos Primitivos

Características

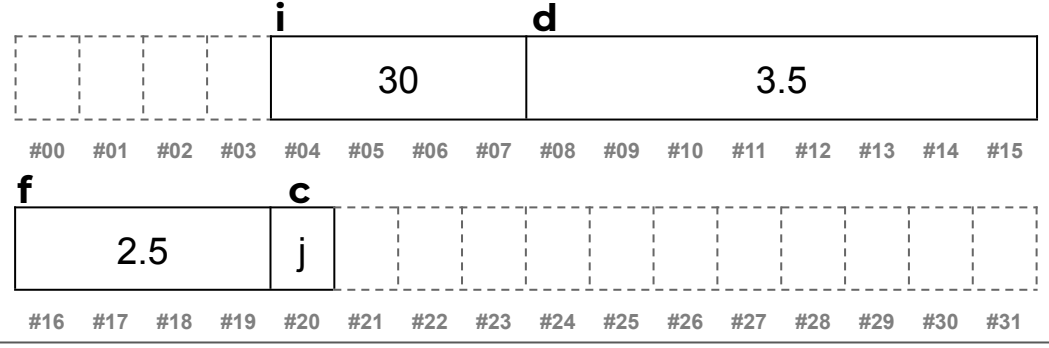
Na linguagem C, cada variável possui:

- Tipo
- Nome
- Valor
- Endereço na memória
  (**endereço do seu primeiro byte**)

Código

```
int    i = 30;
double d = 3.5;
float  f = 2.5;
char   c = 'j';
```

Memória



Tipos Primitivos

Características

Na linguagem C, cada variável possui:

- Tipo
- Nome
- Valor
- Endereço na memória
  (**endereço do seu primeiro byte**)

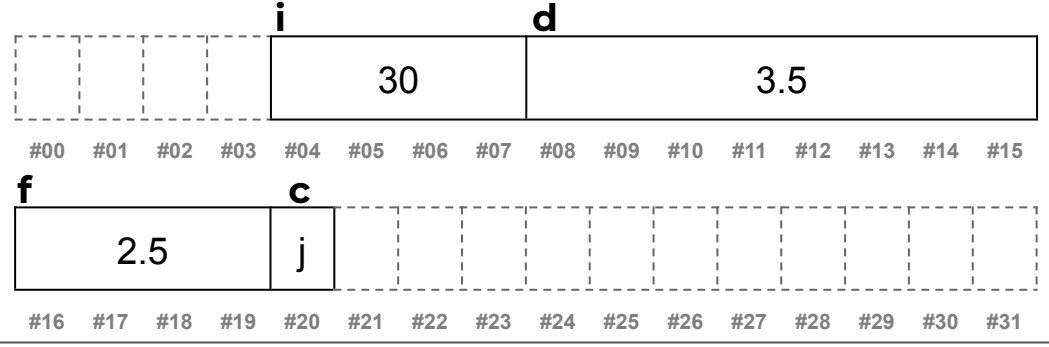
Para obter o endereço de uma variável utilize o operador & "endereço de"

Nome	Tipo	Valor	Endereço
`i`			
`d`			
`f`			
`c`			

Código

```
int    i = 30;
double d = 3.5;
float  f = 2.5;
char   c = 'j';
```

Memória



Tipos Primitivos

Características

Na linguagem C, cada variável possui:

- Tipo
- Nome
- Valor
- Endereço na memória
  (**endereço do seu primeiro byte**)

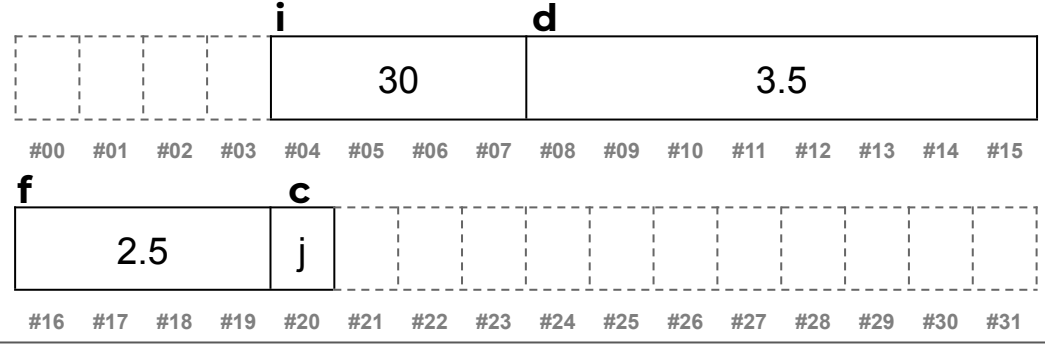
Para obter o endereço de uma variável utilize o operador & "endereço de"

Nome	Tipo	Valor	Endereço	
i	int	30	#04	← &i
d	double	3.5	#08	← &d
f	float	2.5	#16	← &f
c	char	j	#20	← &c

Código

```
int    i = 30;
double d = 3.5;
float  f = 2.5;
char   c = 'j';
```

Memória



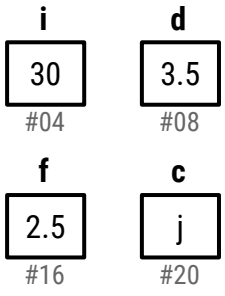
Tipos Primitivos

Representação simplificada na memória

Código

```
int    i = 30;
double d = 3.5;
float  f = 2.5;
char   c = 'j';
```

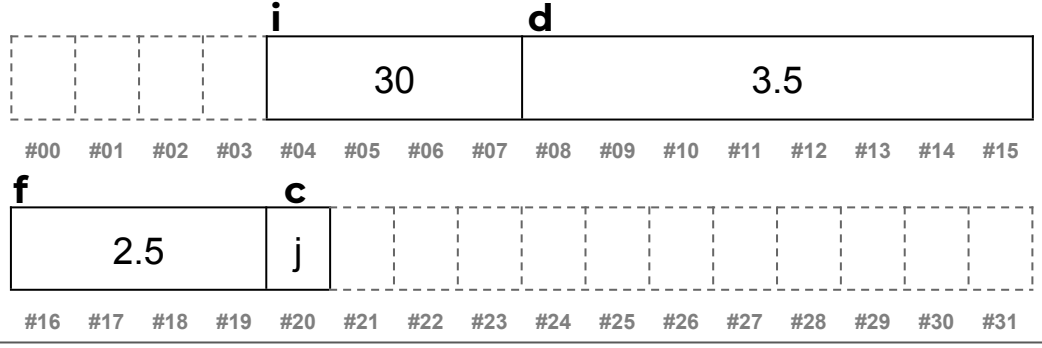
Representação simplificada



Nesta representação, a única informação omitida é o tamanho ocupado na memória.

Nome	Tipo	Valor	Endereço
i	int	30	#04
d	double	3.5	#08
f	float	2.5	#16
c	char	j	#20

Memória



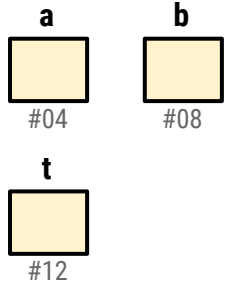
Pausa para praticar 1

Preencha as lacunas.

Código

```
int a = 20;
int b = 30;
int t = a;
a = b;
b = t;
```

Memória



Nome	Tipo	Valor	Endereço
a	int		
b	int		
t	int		

```
printf("%d", a);
```

```
printf("%d", b);
```

```
printf("%d", t);
```

```
printf("%p", &a);
```

```
printf("%p", &b);
```

```
printf("%p", &t);
```

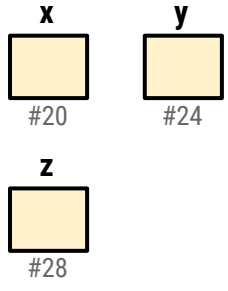
Pausa para praticar 2

Preencha as lacunas.

Código

```
int x = 10;
int y = x++;
int z = x;
x *= 2;
z = x++ + ++y;
```

Memória



Nome	Tipo	Valor	Endereço
x	int		
y	int		
z	int		

`printf("%d", x);` ____ `printf("%d", y);` ____ `printf("%d", z);` ____

`printf("%p", &x);` ____ `printf("%p", &y);` ____ `printf("%p", &z);` ____

ponteiros

Ponteiro

- O conceito de endereços e ponteiros são fundamentais em qualquer linguagem de programação
- Em outras linguagens esse conceito é oculto e representado de forma mais abstrata.
- Na linguagem C esses conceitos são explícitos

Dominar o conceito de ponteiros exige visualização de como os dados estão organizados na memória.

Ponteiro

É uma variável

Características

- Tipo
- Nome
- Valor
- Endereço

```
int qtde = 30;
int* p = &qtde;
```

- tipo: `int*`
- nome: `p`
- valor: `&qtde` (#04)
- endereço: `&p` (#08)

Indica ao compilador que você armazenará um endereço de memória

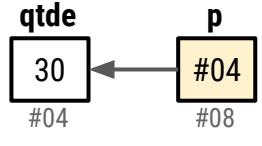
`<tipo> *<nome>`

e que nesse endereço, ele encontrará um valor desse tipo

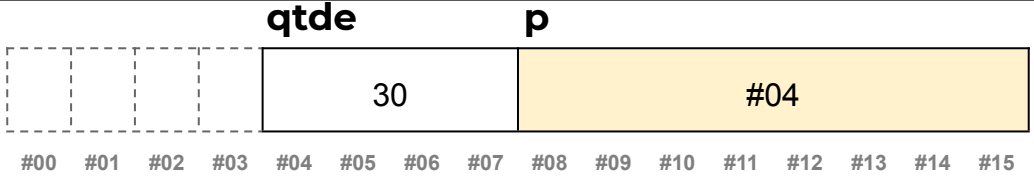
Código

```
int qtde = 30;
int *p = NULL;
p = &qtde;
```

Memória



Memória



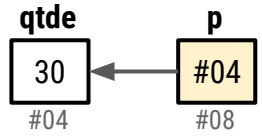
Ponteiro

Algumas características

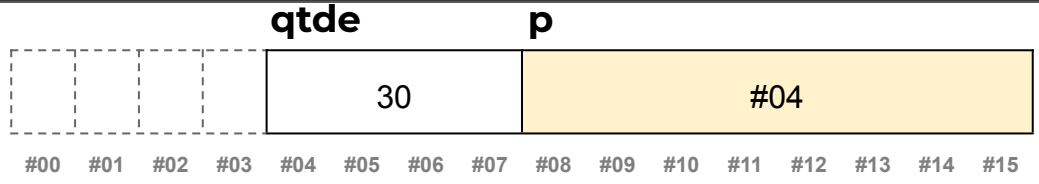
Código

```
int qtde = 30;
int *p = NULL;
p = &qtde;
```

Memória



Memória



- **TODO** ponteiro ocupa **8 bytes** na memória. Mesmo um ponteiro de char, por exemplo..
- Operador **&** devolve o endereço do **primeiro byte** em que a variável foi alocada na memória
- Operador ***** acessa a região indicada pelo ponteiro. Seja para **recuperar** o valor nela armazenada ou para **atribuir** um novo valor para esta região.
- A macro **NULL** está definida na biblioteca `stdlib.h` e seu valor é zero, indicando que a variável não referência nenhum endereço.

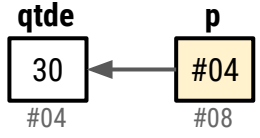
Ponteiro

Operações

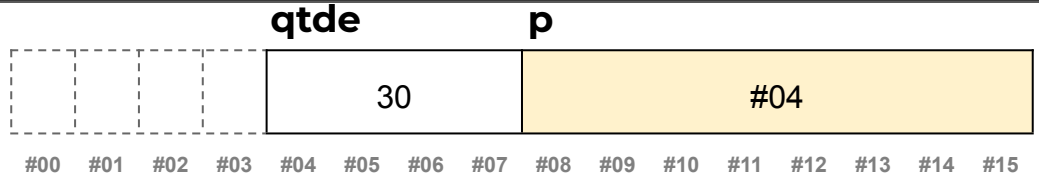
Código

```
int qtde = 30;
int *p = NULL;
p = &qtde;
```

Memória

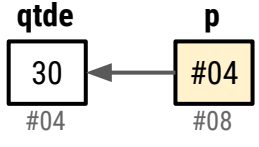


Memória



Analise as expressões abaixo de acordo com os endereços fictícios da representação das variáveis na memória.

Expressão	Valor	Tipo
qtde	30	int
&qtde	#04	int*
p	#04	int*
*p	30	int
&p	#08	int**



Qual efeito produzido por essa instrução?

```
*p = 50;
```

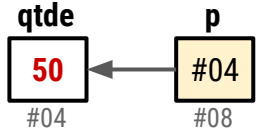
Ponteiro

Operações

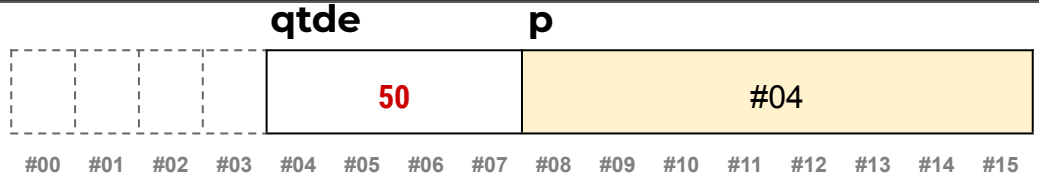
Código

```
int qtde = 30;
int *p = NULL;
p = &qtde;
```

Memória

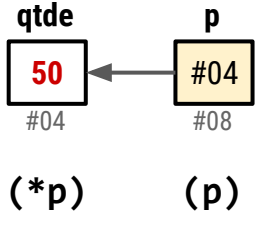


Memória



Analise as expressões abaixo de acordo com os endereços fictícios da representação das variáveis na memória.

Expressão	Valor	Tipo
qtde	50	int
&qtde	#04	int*
p	#04	int*
*p	50	int
&p	#08	int**



Qual efeito produzido por essa instrução?

```
*p = 50;
```

O espaço (**#04**) referenciado pelo ponteiro **p** recebe o valor 50.

Ponteiro

Exemplo

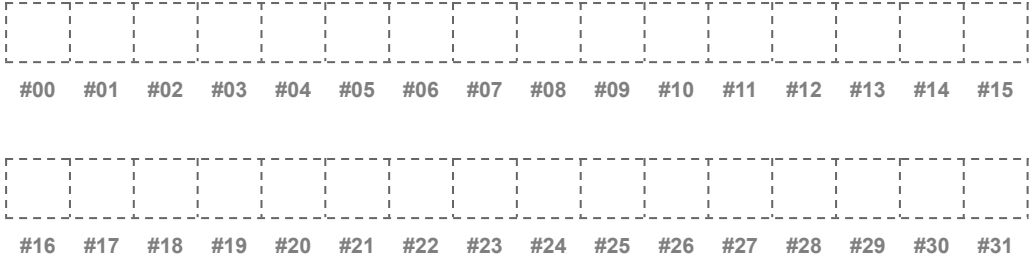
Acompanhe a simulação para auxiliar no entendimento.

Código

```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada

Memória



Ponteiro

Exemplo

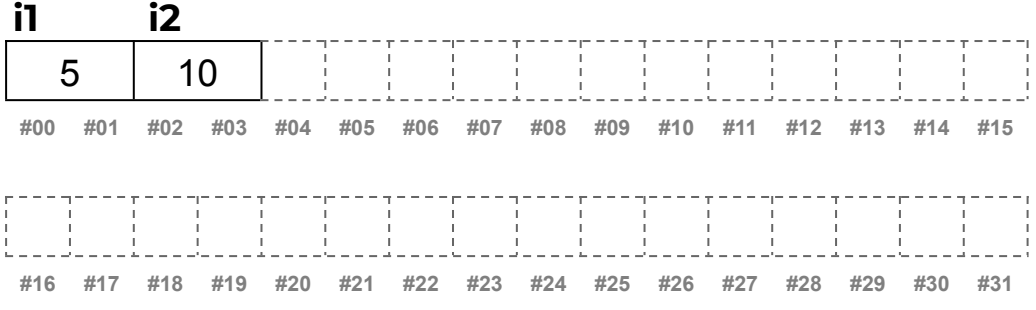
Acompanhe a simulação para auxiliar no entendimento.

Código

```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada

Memória



Ponteiro

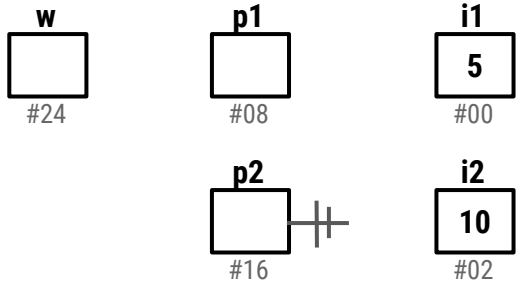
Exemplo

Acompanhe a simulação para auxiliar no entendimento.

Código

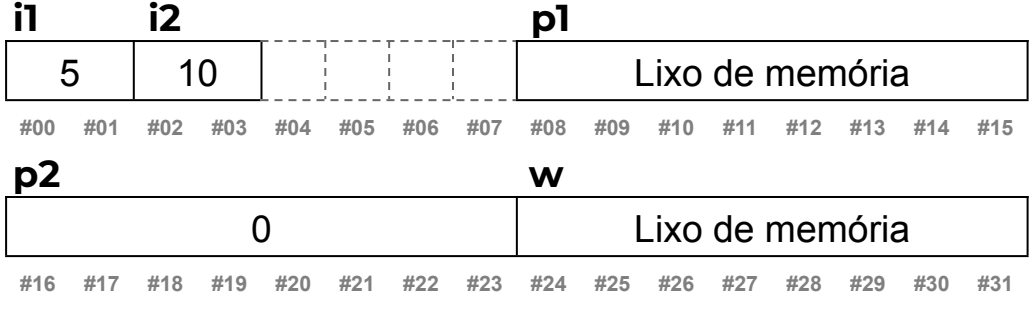
```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada



Símbolo que utilizaremos para representar o **NULL**

Memória



Ponteiro

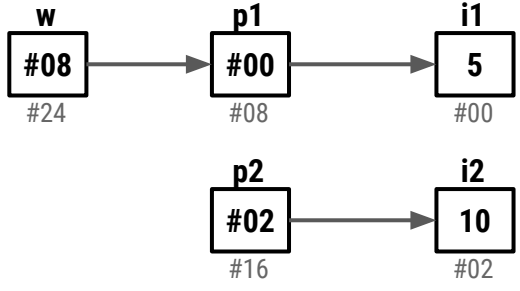
Exemplo

Acompanhe a simulação para auxiliar no entendimento.

Código

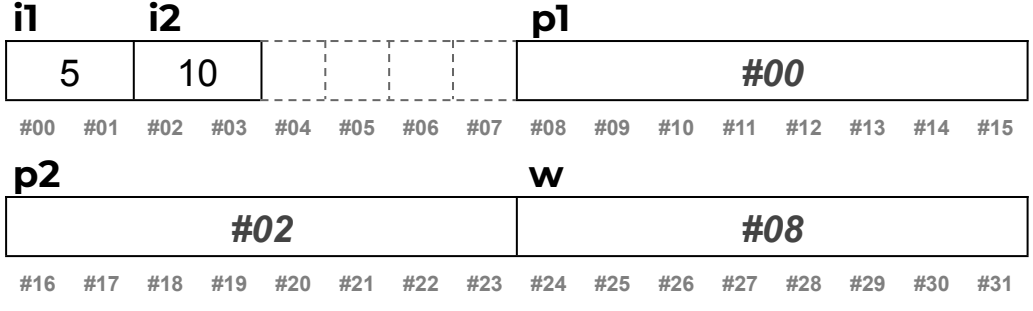
```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada




```
&i1 == #00
&i2 == #02
&p1 == #08
```

Memória



Ponteiro

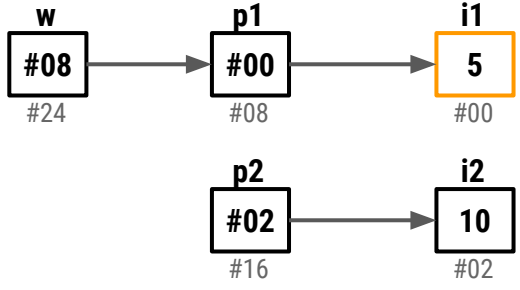
Exemplo

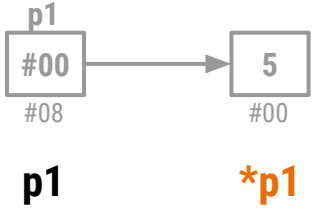
Acompanhe a simulação para auxiliar no entendimento.

Código

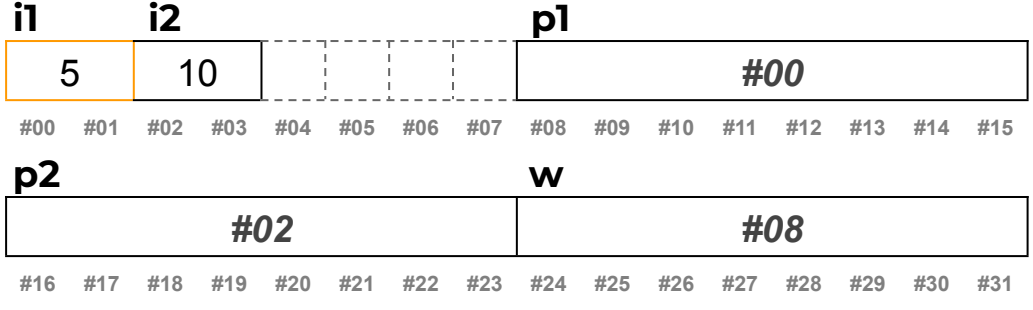
```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada





Memória



Ponteiro

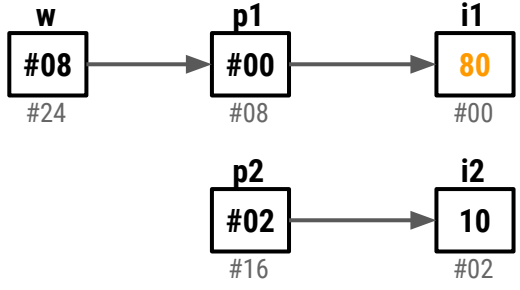
Exemplo

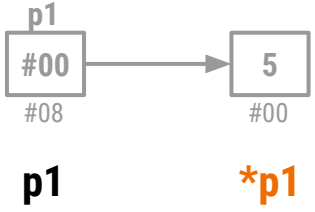
Acompanhe a simulação para auxiliar no entendimento.

Código

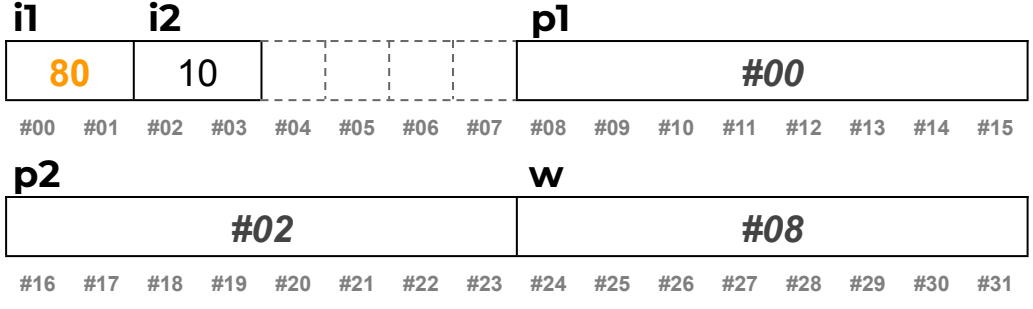
```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada





Memória



Ponteiro

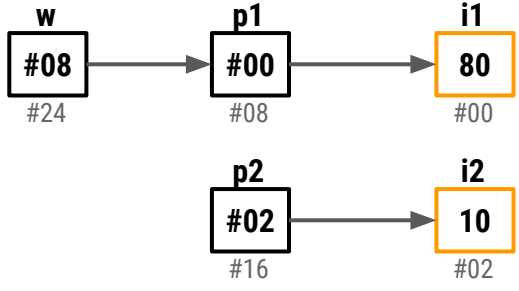
Exemplo

Acompanhe a simulação para auxiliar no entendimento.

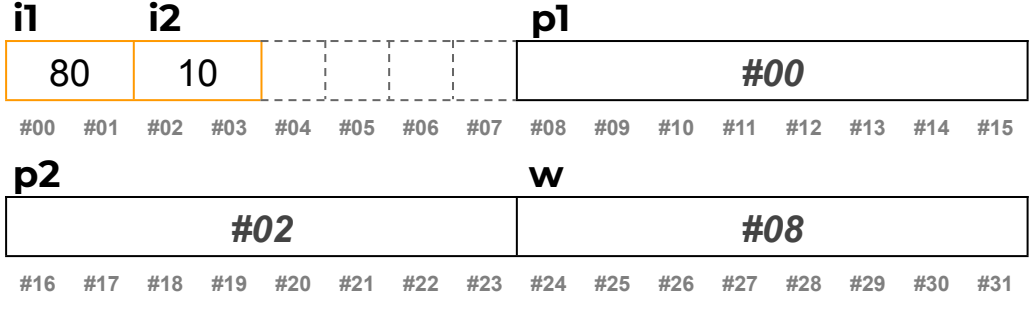
Código

```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada



Memória



Ponteiro

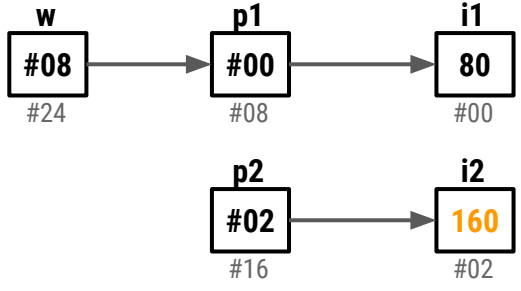
Exemplo

Acompanhe a simulação para auxiliar no entendimento.

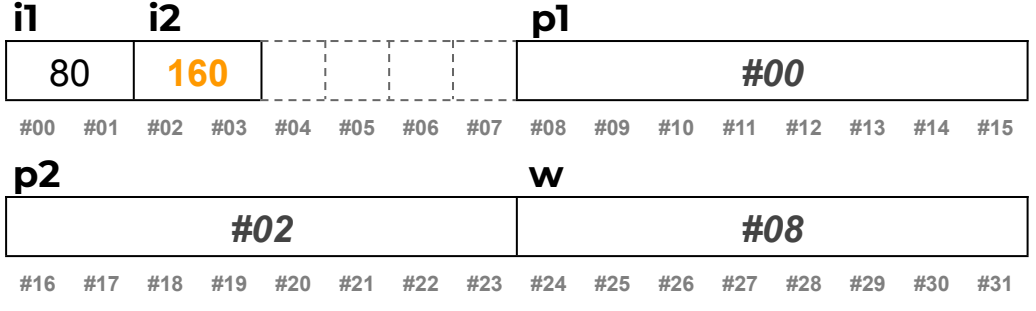
Código

```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada



Memória



Ponteiro

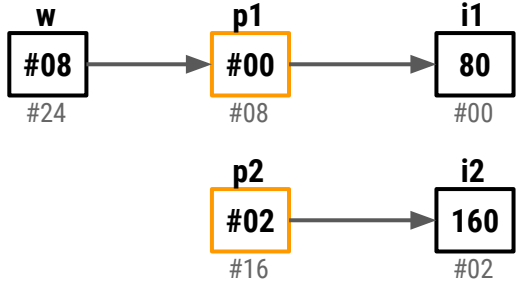
Exemplo

Acompanhe a simulação para auxiliar no entendimento.

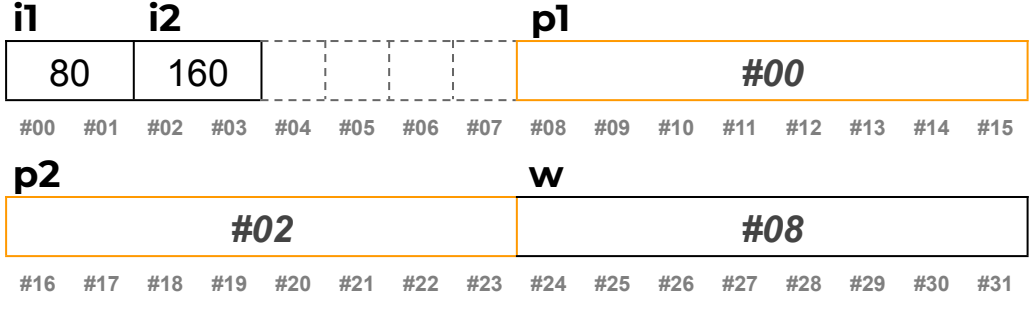
Código

```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada



Memória



Ponteiro

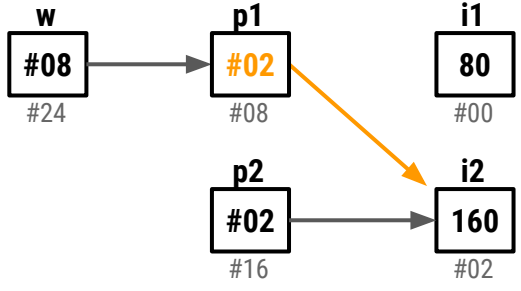
Exemplo

Acompanhe a simulação para auxiliar no entendimento.

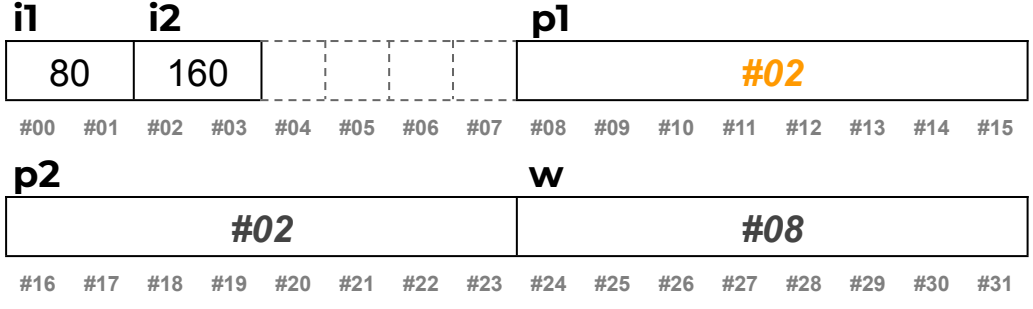
Código

```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada



Memória



Ponteiro

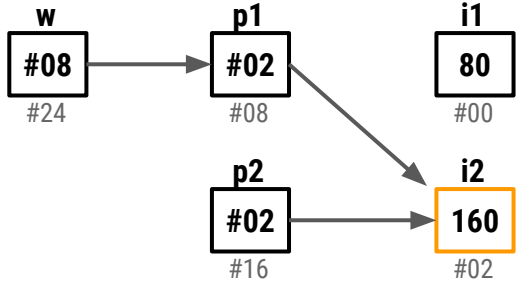
Exemplo

Acompanhe a simulação para auxiliar no entendimento.

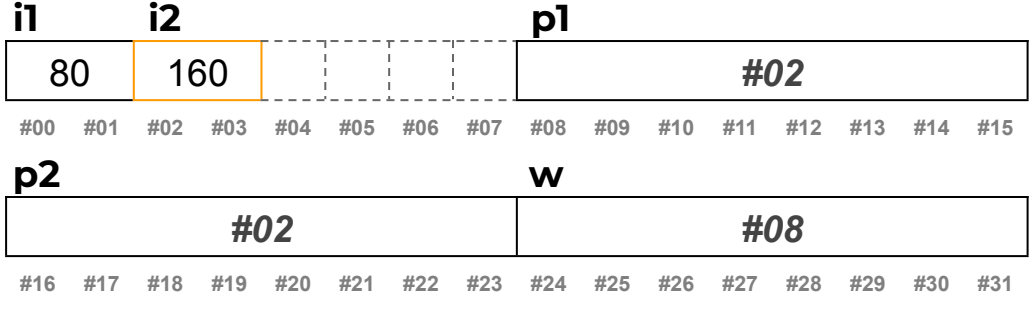
Código

```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada



Memória



Ponteiro

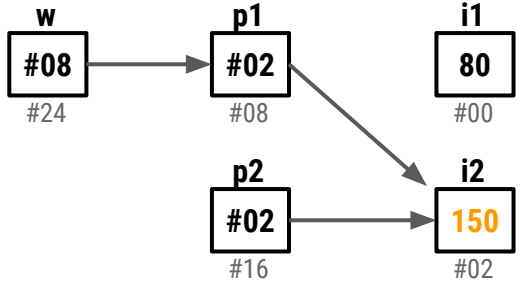
Exemplo

Acompanhe a simulação para auxiliar no entendimento.

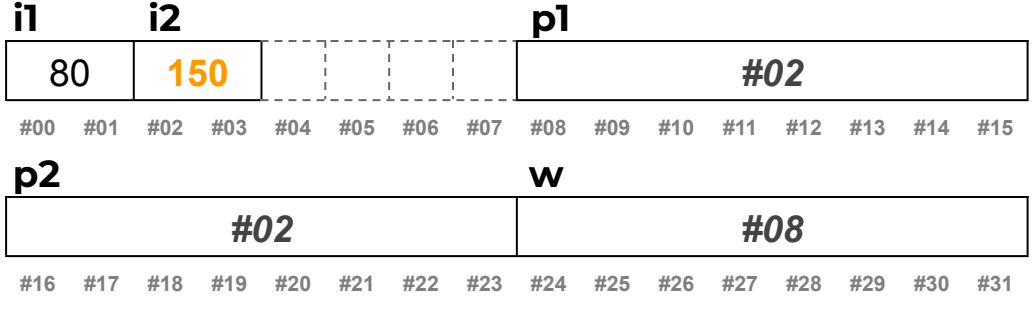
Código

```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada



Memória



Ponteiro

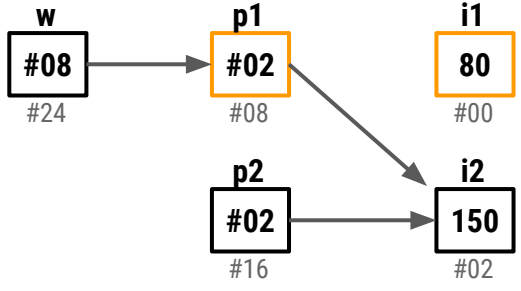
Exemplo

Acompanhe a simulação para auxiliar no entendimento.

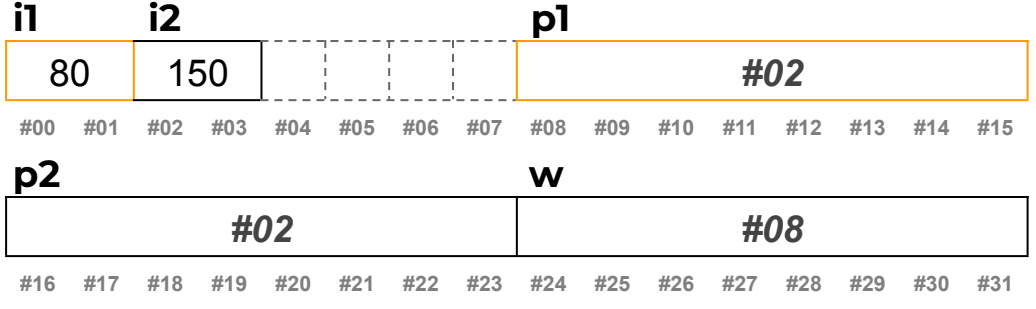
Código

```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada



Memória



Ponteiro

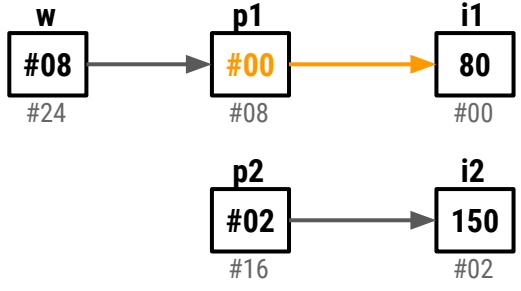
Exemplo

Acompanhe a simulação para auxiliar no entendimento.

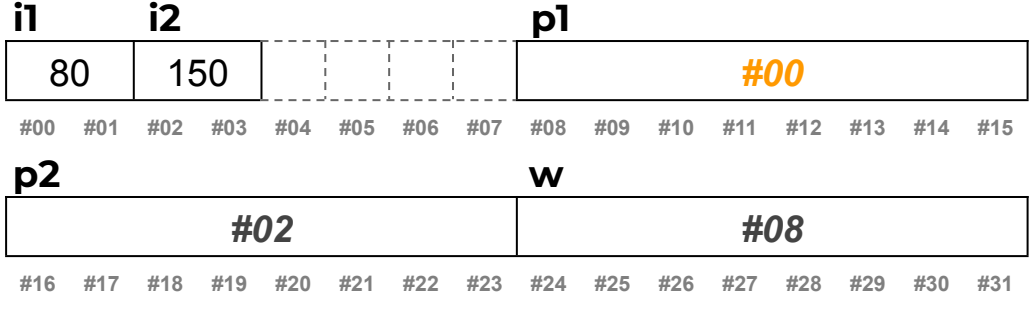
Código

```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada



Memória



Ponteiro

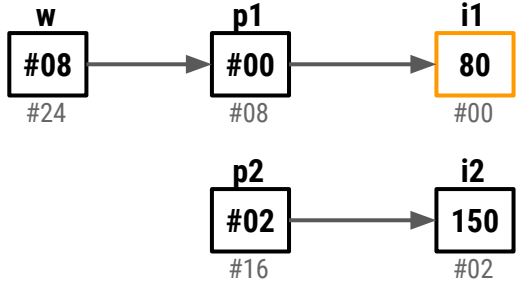
Exemplo

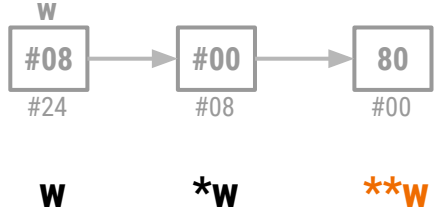
Acompanhe a simulação para auxiliar no entendimento.

Código

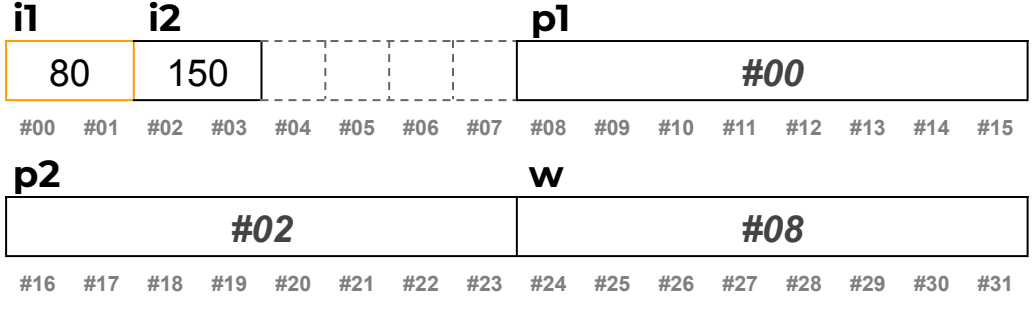
```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada





Memória



Ponteiro

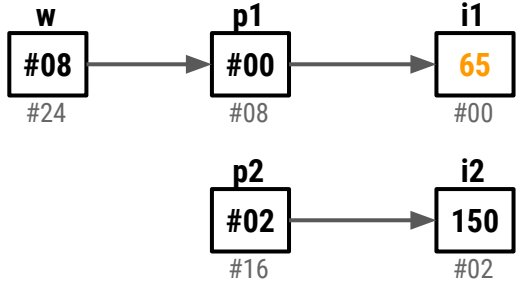
Exemplo

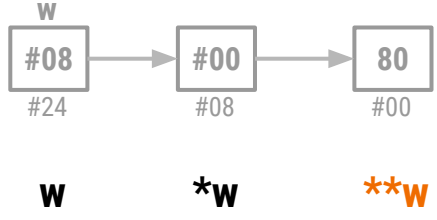
Acompanhe a simulação para auxiliar no entendimento.

Código

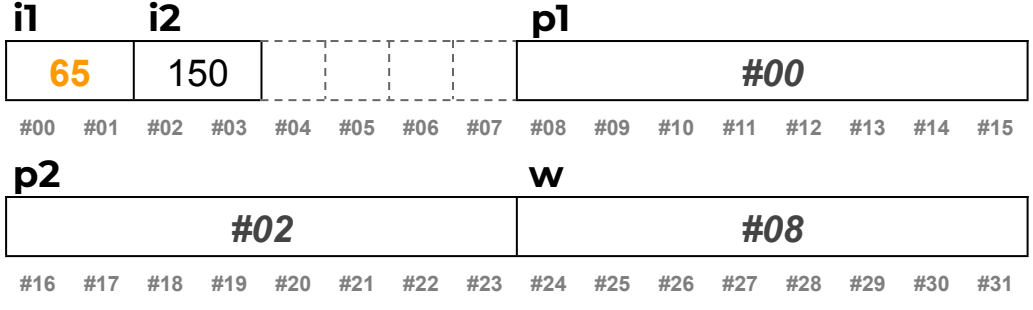
```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada





Memória



Ponteiro

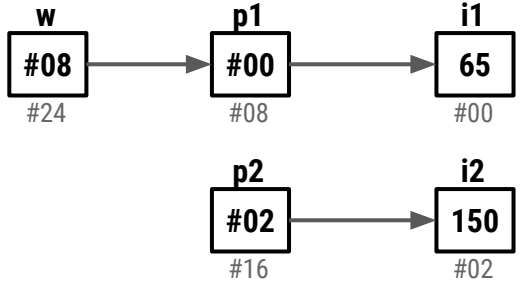
Exemplo

Acompanhe a simulação para auxiliar no entendimento.

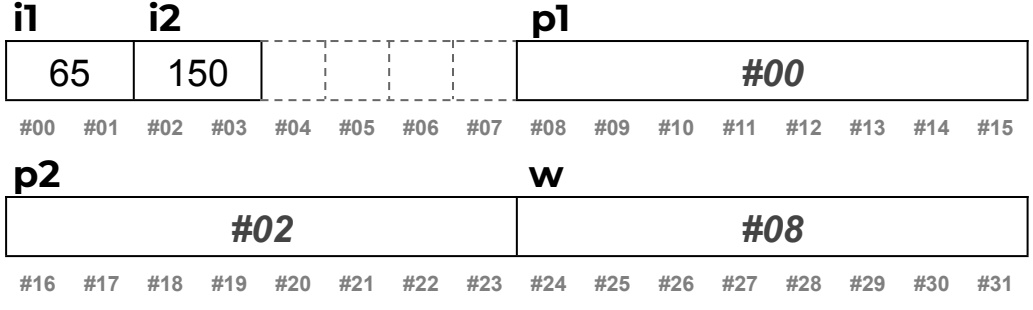
Código

```
short int i1 = 5;
short int i2 = 10;

short int *p1;
short int *p2 = NULL;
short int **w;

p1 = &i1;
p2 = &i2;
w  = &p1;

*p1 = 80;
i2 = *p1 * 2;
p1 = p2;
*p1 -= 10;
*w = &i1;
**w -= 15;
```

Representação simplificada



Memória



exercícios

Exercício 1

Faça o desenho correspondente ao código abaixo utilizando a **representação simplificada da memória**.

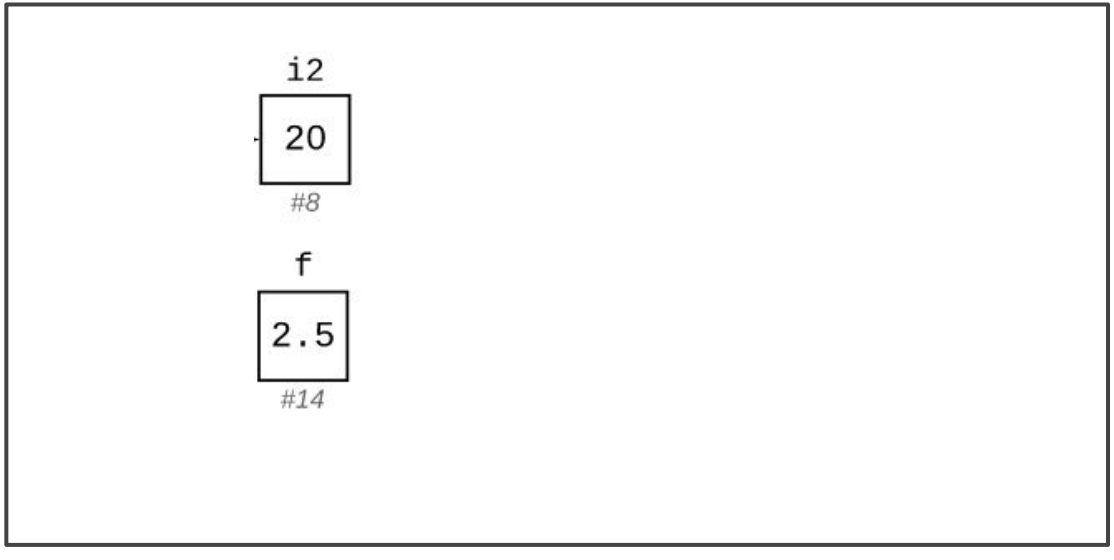
Código

```
int i2 = 20;
float f = 2.5;
int *p1 = &i2;
short int si = 30;
short int *psi = &si;
int **x = &p1;
```

Utilize os seguintes endereços de memória para as variáveis:

```
si   #12
psi  #48
p1   #32
x    #80
```

Desenho



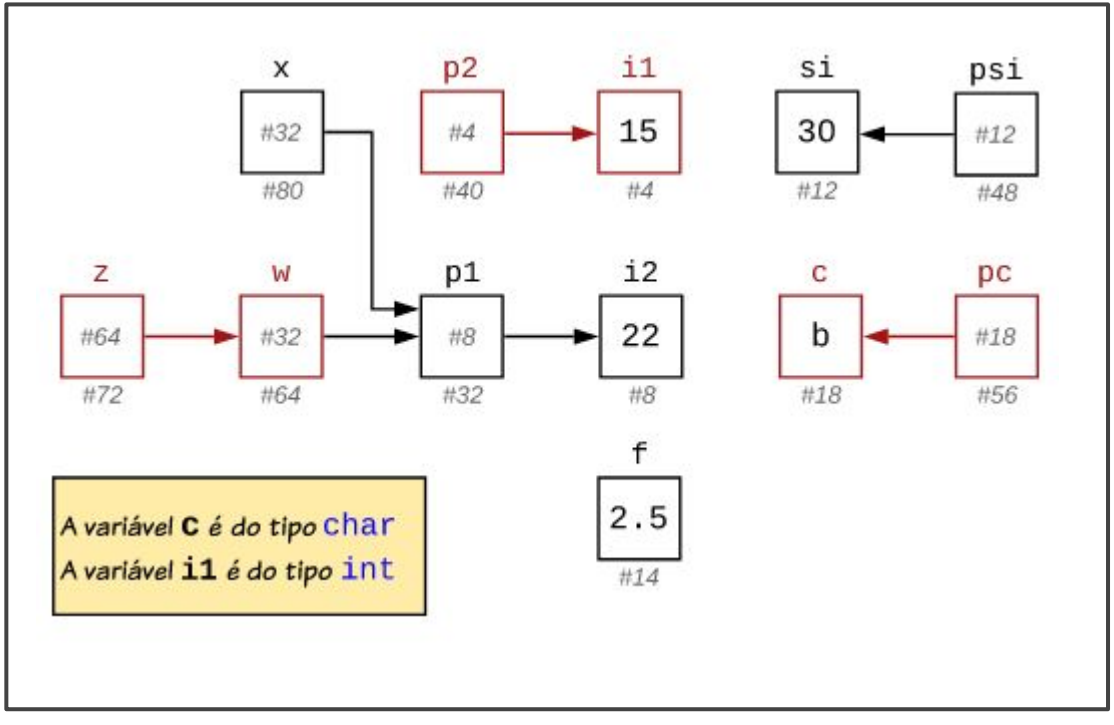
Exercício 2

A partir do resultado produzido pelo exercício anterior, **escreva o código** correspondente ao restante do desenho. (representado em vermelho)

Código

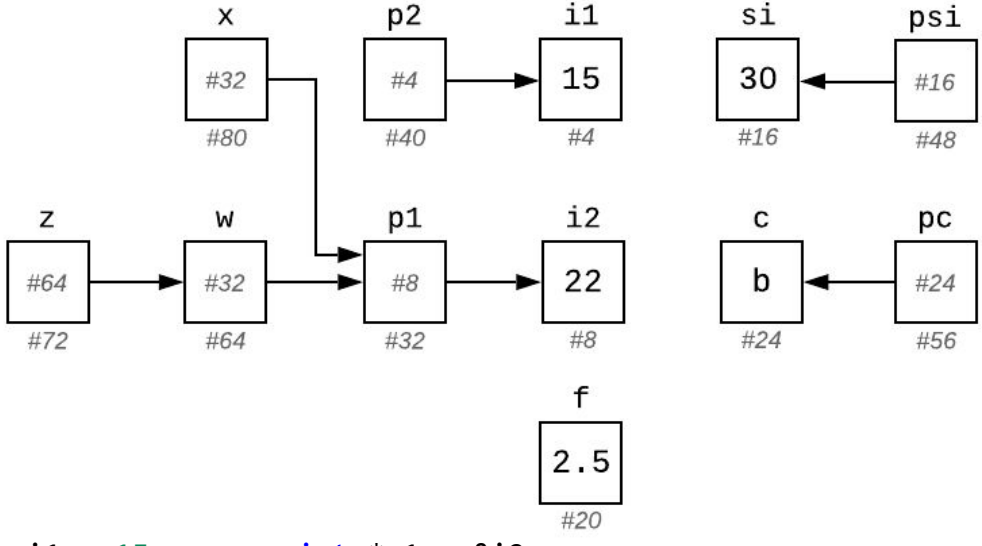
```
int i2 = 22;
float f = 2.5;
int *p1 = &i2;
short int si = 30;
short int *psi = &si;
int **x = &p1;
```

Escreva o restante do código abaixo



Exercício 3

Preencha a tabela de acordo com o código e sua representação na memória.




```
int i1 = 15;
int i2 = 22;
short int si = 30;
float f = 2.5;
char c = 'b';
```

```
int *p1 = &i2;
int *p2 = &i1;
short int *psi = &si;
char *pc = &c;
int **x = &p1;
int **w = &p1;
int ***z = &w;
```

parte 1

Expressão	Valor	Tipo
i1	15	int
i2	22	int
si	30	short int
f	2.5	float
c	b	char
&i1	#4	int*
&i2		
p2		
*p2		
&p2		
z		
*z		
**z		
***z		
&z		
w		
&w		
*w		


Fim