Fundamentos de Programação
Ponteiros

Dainf - UTFPR

Profa. Leyza B. Dorini
Prof. Bogdan T. Nassu

Ponteiros

Não é tão complicado quanto dizem por aí...

Antes de começar...

Onde você vai usar ponteiros?

- passagem de parâmetros por referência: desta forma, será possível fazer com que uma função consiga "retornar" mais de um valor;
- vetores;
- manipulação de estruturas de dados como pilha e fila;
- e muito mais....

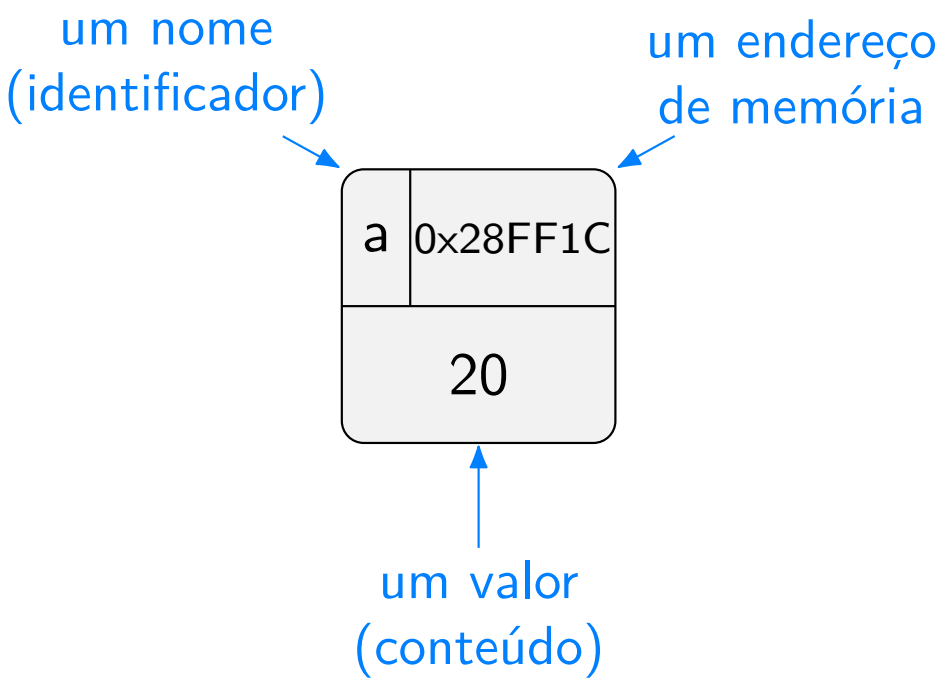
Neste material, veremos apenas como declarar, inicializar e manipular ponteiros.

Variáveis

Considere a seguinte declaração de uma variável:

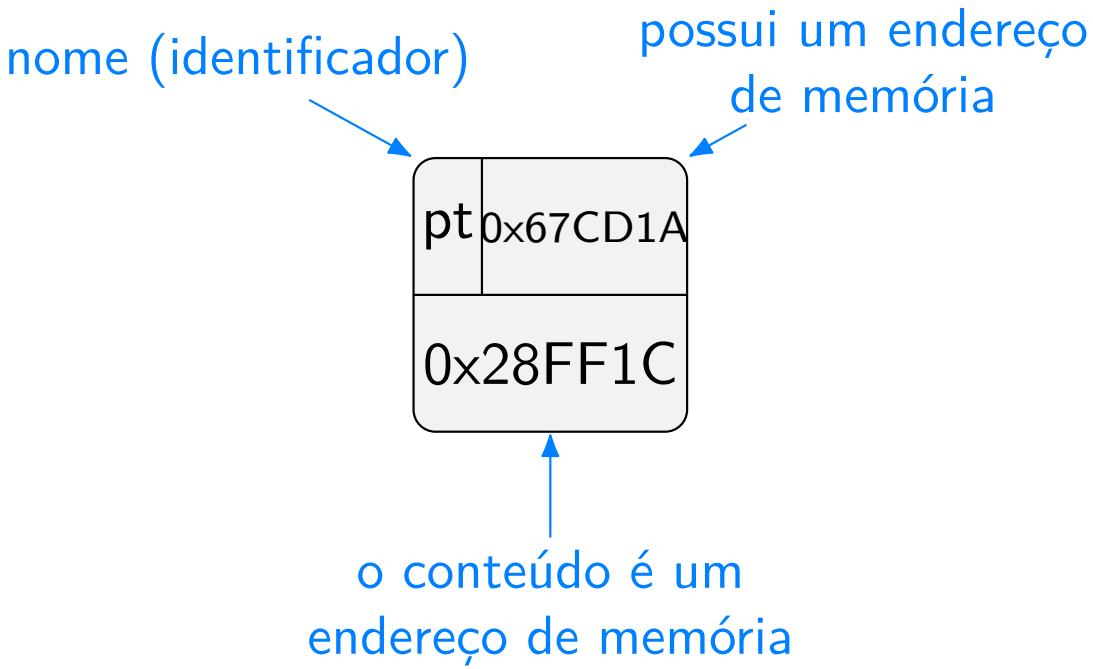
```
int a = 20;
```

Temos associados a ela os seguintes elementos:



Definição

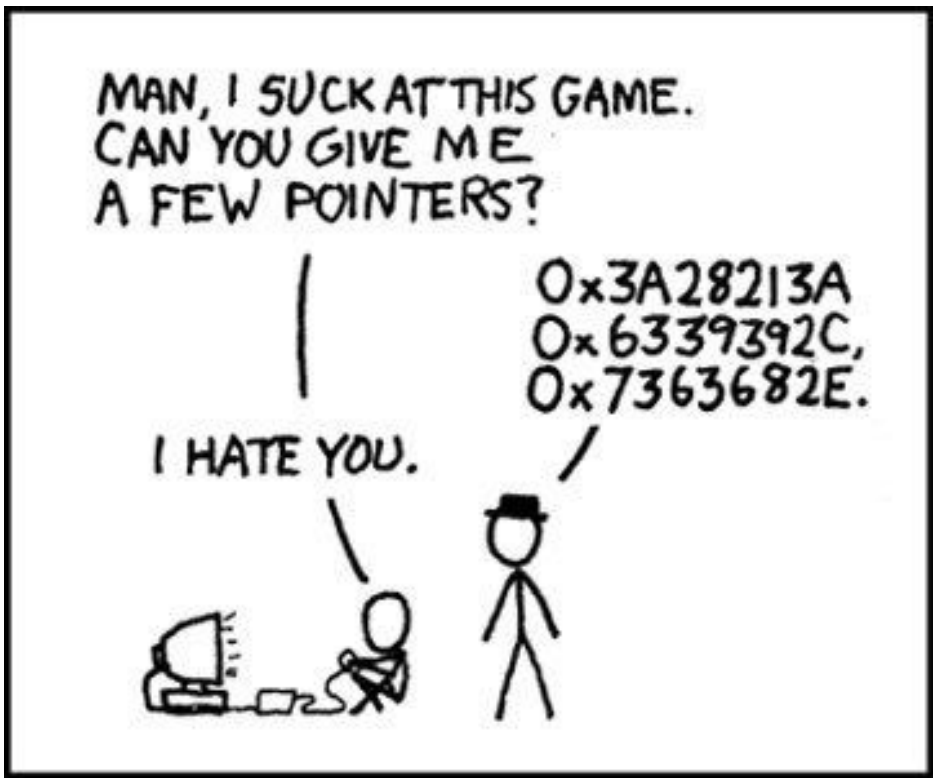
Um ponteiro nada mais é do que uma variável que **armazena um endereço de memória**.



Ponteiros

Lembre-se

Um ponteiro é uma variável que **armazena um endereço**.



Declaração de ponteiros

Declaração de ponteiros em C

Para declarar uma variável ponteiro (apontador), utilizamos o operador unário `*`:

Sintaxe

```
1 tipo *nome_do_ponteiro;
```

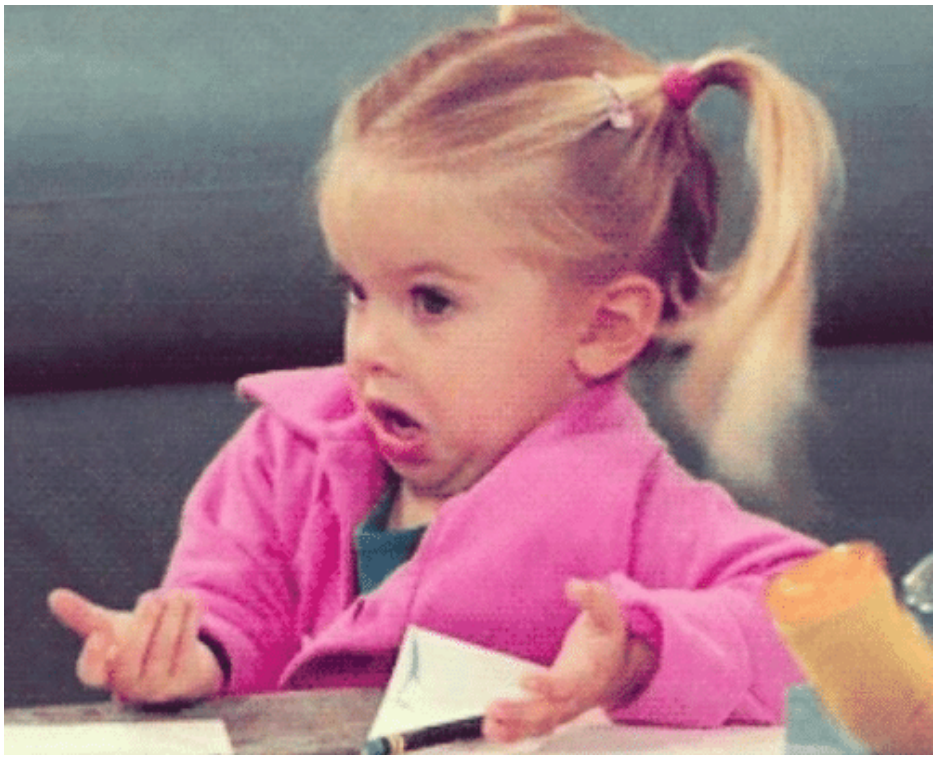
Exemplos:

```
int    *pt_int;
char   *pt_char;
float  *pt_float;
```

Qual o tipo de um ponteiro?

Como você já sabe, na linguagem C cada variável possui um tipo, o qual deve ser definido no momento da sua declaração. É ele quem determina o tipo de dado que pode ser armazenado.

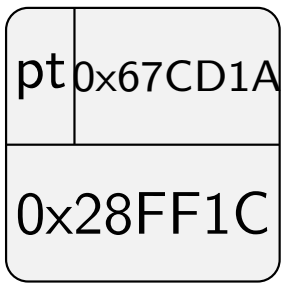
Se um ponteiro armazena um endereço, **qual o seu tipo**?



Qual o tipo de um ponteiro?

O tipo do ponteiro deve ser compatível com o tipo da variável que está no endereço que ele está armazenando. Vamos a um exemplo!

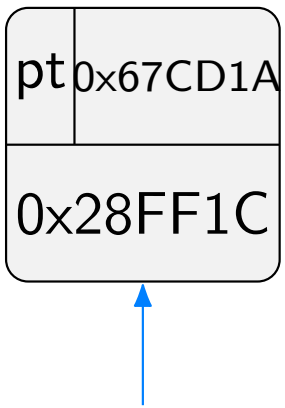
Qual o tipo do ponteiro abaixo?



Qual o tipo de um ponteiro?

O tipo do ponteiro deve ser compatível com o tipo da variável que está no endereço que ele está armazenando. Vamos a um exemplo!

Qual o tipo do ponteiro abaixo?

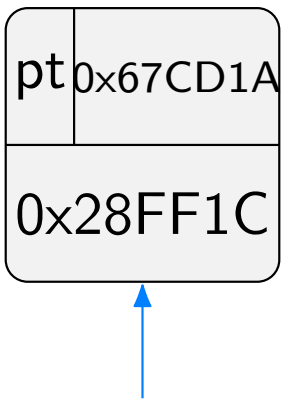


o tipo deste ponteiro
será igual ao tipo da variável
que está no endereço
`0x28ff1c`

por exemplo, se a variável no
endereço `0x28ff1c` for do tipo `int`,
o ponteiro também deverá ser
(o tipo seria `int *` —
depois veremos mais exemplos)

Qual o tamanho de um ponteiro?

O tamanho do ponteiro não tem relação com o tamanho do tipo apontado, ou seja, `sizeof(int*) != sizeof(int)`.



O tamanho do ponteiro é igual
ao tamanho do endereço na arquitetura
(normalmente, é o mesmo tamanho
de um `int`, 4 ou 8 bytes).

Declaração de ponteiros em C

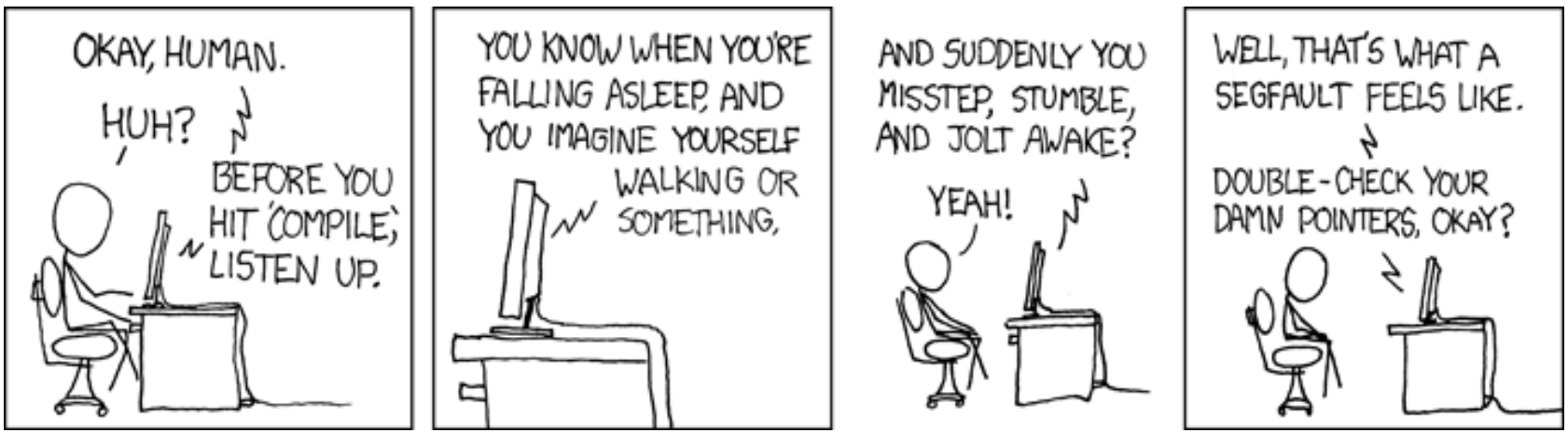
Cuidado ao declarar vários ponteiros em uma única linha. O operador * **deve preceder o nome de cada variável**. Por exemplo, se você quiser declarar três ponteiros do tipo `int`, a sintaxe correta é:

```
int *pt1, *pt2, *pt3;
```

Por exemplo, os comandos abaixo declaram um ponteiro para inteiro e dois inteiros:

```
int *p1, p2, p3;
\\ou
int int01, *pt1, int02;
\\ ou
int int01, int02, *pt1;
```

Inicialização de ponteiros



Inicializar corretamente ponteiros é essencial para evitar acessos indevidos à memória (*segmentation fault*).

Inicialização de ponteiros em C

Para inicializar uma variável do tipo ponteiro, precisamos atribuir a ela o endereço de uma variável! Para tal, usamos o operador *address-of* (&):

Exemplo

```
1  int main()
2  {
3      int a = 20;
4      printf("Valor de a = %d\n", a);
5      printf("Endereço de a = 0x%x\n", &a);
6      //%x imprime formato hexadecimal
7
8      return 0;
9  }
```

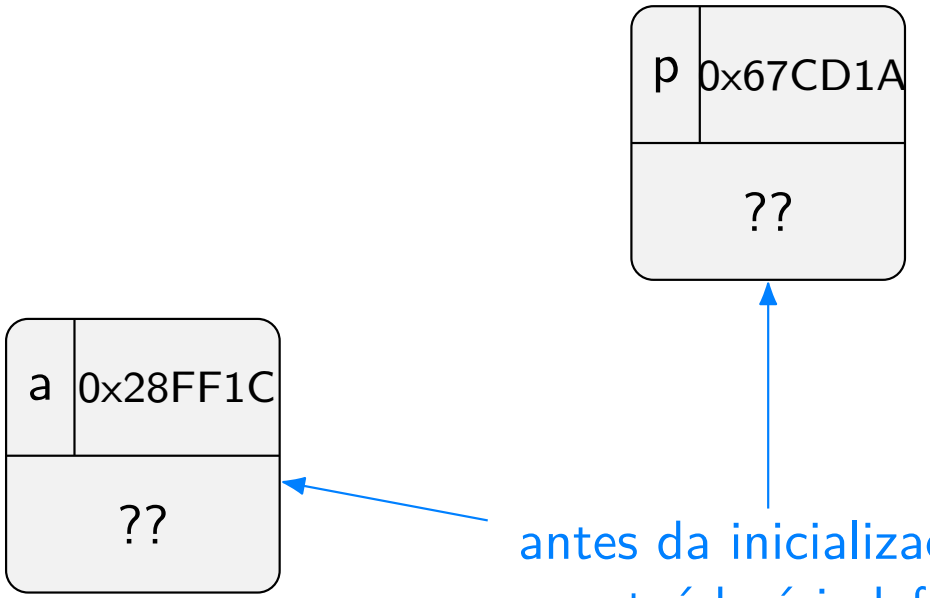
Saída

```
Valor de a = 20
Endereço de a = 0x28ff1c
```

Exemplo: inicialização de ponteiros em C

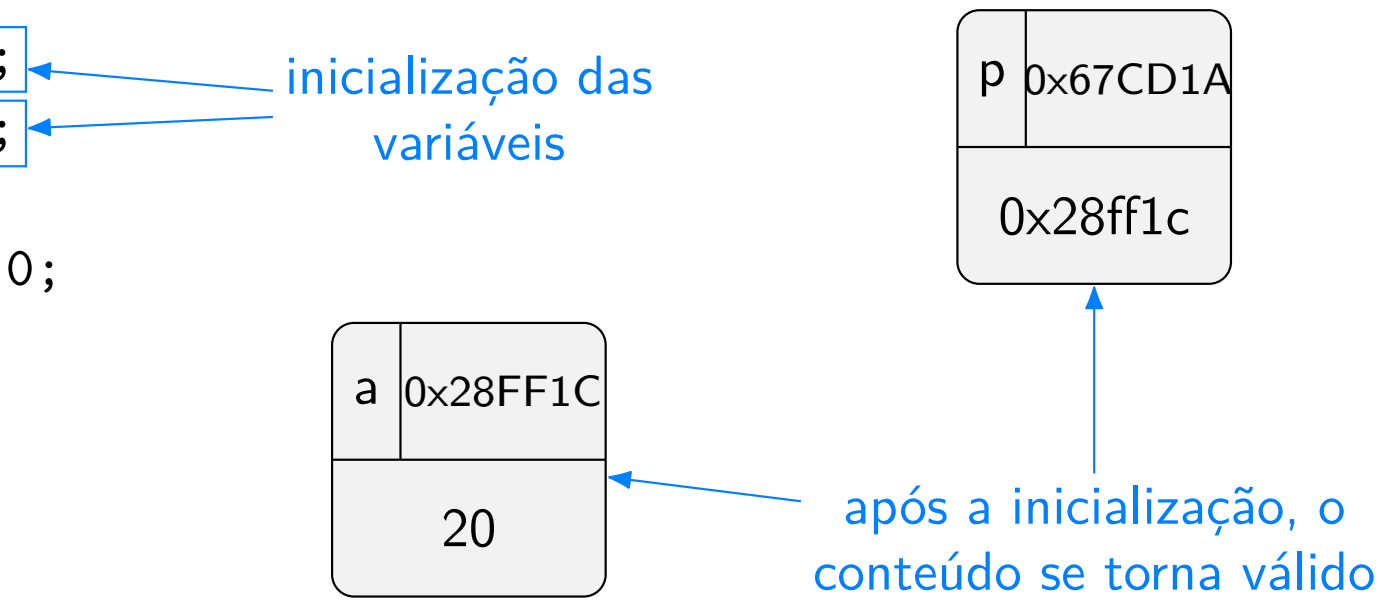
```
1  int main(){
2
3      int a;
4      int *p;
5
6
7
8
9      return 0;
10 }
```

declaração das variáveis a e p (o ponteiro é do tipo "ponteiro para int" porque vai apontar para uma variável desse tipo)



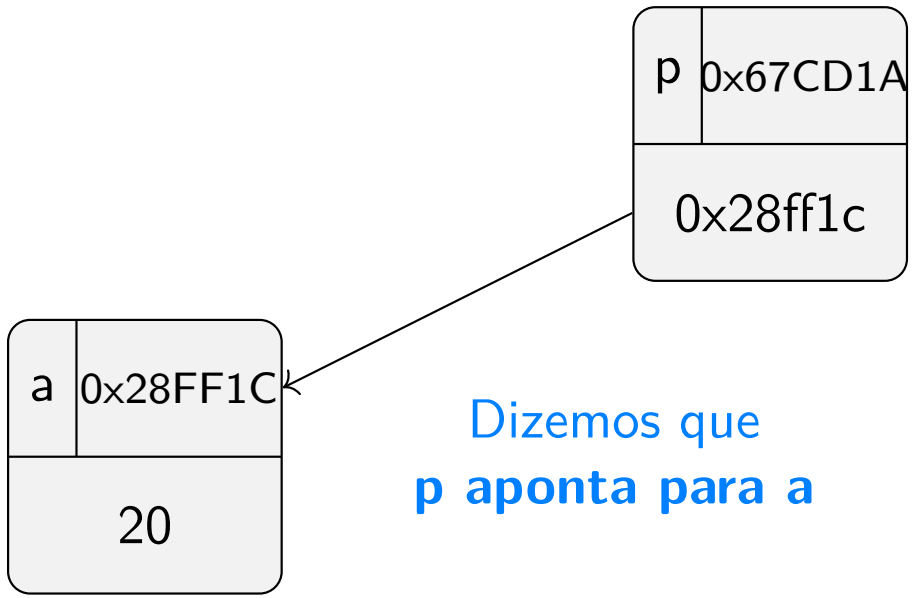
Exemplo: inicialização de ponteiros em C

```
1  int main(){
2
3      int a;
4      int *p;
5
6      a = 20;
7      p = &a;
8
9      return 0;
10 }
```



Exemplo: inicialização de ponteiros em C

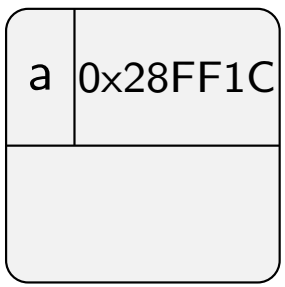
```
1  int main(){
2
3      int a;
4      int *p;
5
6      a = 20;
7      p = &a;
8
9      return 0;
10 }
```

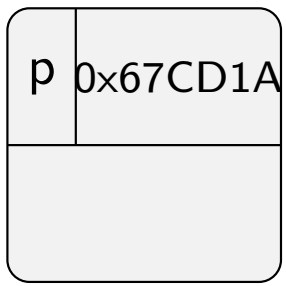


Checkpoint

Qual o erro do programa abaixo?

```
1  int main(){
2
3      int a;
4      int *p;
5
6      a = 20;
7      p = a;
8
9      return 0;
10 }
```



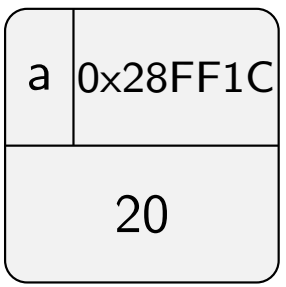


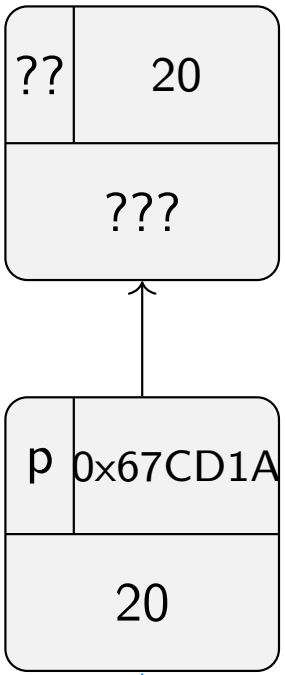
Checkpoint

Qual o erro do programa abaixo?

```
1  int main(){
2
3      int a;
4      int *p;
5
6      a = 20;
7      p = a;
8
9      return 0;
10 }
```

ERRO: atribui ao ponteiro p o **conteúdo da variável** a, ao invés do endereço...





...com isso, **p aponta** para o endereço 20 de memória. Como não temos controle sobre o que está armazenado nesta localização, ocorre um erro.

Inicialização de ponteiros em C

Quando um ponteiro não está associado a nenhum endereço válido é comum atribuir o valor `NULL` (definido na biblioteca `stdlib.h`). Isto é comumente usado em comparações com ponteiros para saber se um determinado ponteiro possui valor válido ou não. Exemplo:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int *pt = NULL, b=10;
6      pt = &b;
7      if (pt != NULL) {
8          printf("Numero : %d\n", *pt);
9          printf("Endereco: %p\n", pt);
10     }
11     return 0;
12 }
```

Dica: use `%p` para impressão de endereços.

Inicialização de ponteiros em C

Lembre-se

Nunca se esqueça de inicializar corretamente um ponteiro!

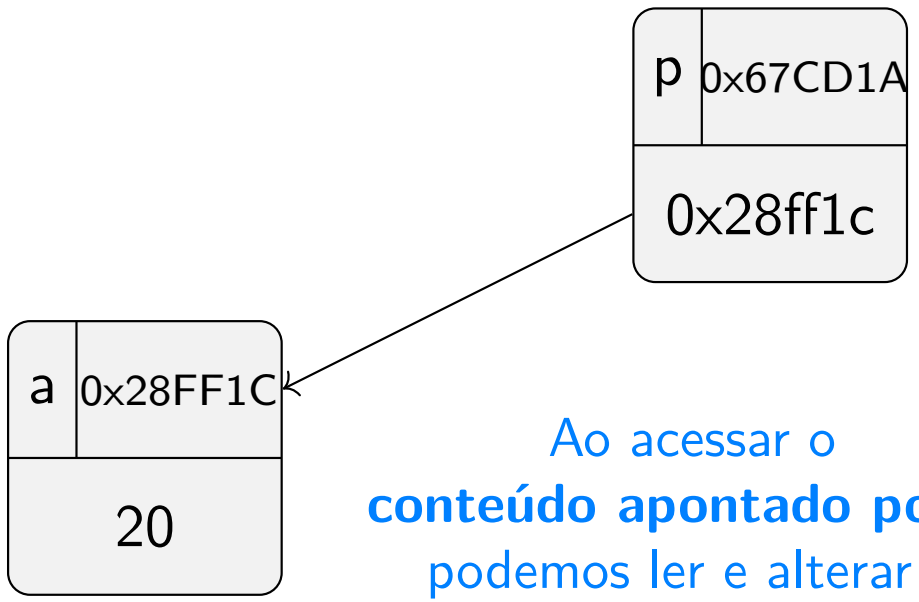
Mesmo que você atribua `NULL` na inicialização, ele **precisa** apontar para um endereço válido quando for acessado!

Acessando variáveis de forma indireta usando ponteiros

Acessando os valores das variáveis referenciadas

Utilizando ponteiros, podemos acessar de forma indireta o conteúdo das variáveis. No exemplo abaixo, podemos acessar o **conteúdo apontado pelo ponteiro** `p` **para manipular o valor da variável referenciada** `a`**.**

```
1        a = 20;
2        p = &a;
```

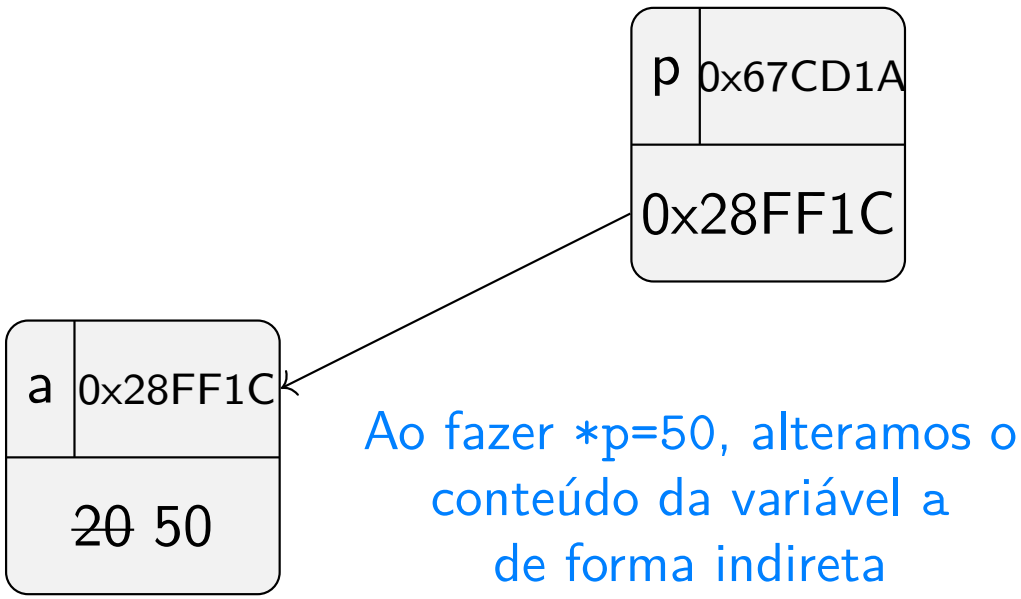


Ao acessar o **conteúdo apontado por** `p`, podemos ler e alterar o conteúdo da variável a

Exemplo: acessando variáveis referenciadas

Para acessarmos o valor de uma variável apontada por um endereço, também usamos o operador *

```
1  int main(){
2
3      int a;
4      int *p;
5
6      a = 20;
7      p = &a;
8
9      *p = 50;
10
11     return 0;
12 }
```



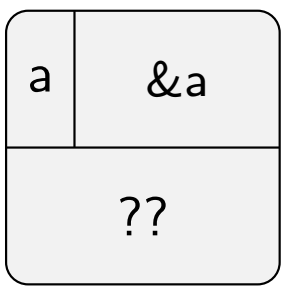
Uso do operador *

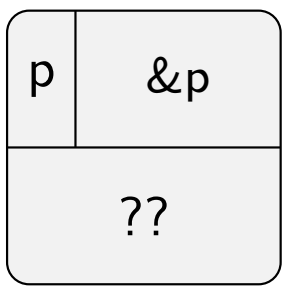
Exemplo: uso do operador *

```
1  int main(){
2
3      float *p, a;
4
5      a = 20.3;
6
7
8
9
10     return 0;
11 }
```

ao declarar um ponteiro,
é necessário usar o operador *
(só assim diferenciamos a declaração de um ponteiro para `float` daquela de um `float`, concorda?)

neste exemplo, foram declarados um ponteiro (`p`) e um `float` (`a`)



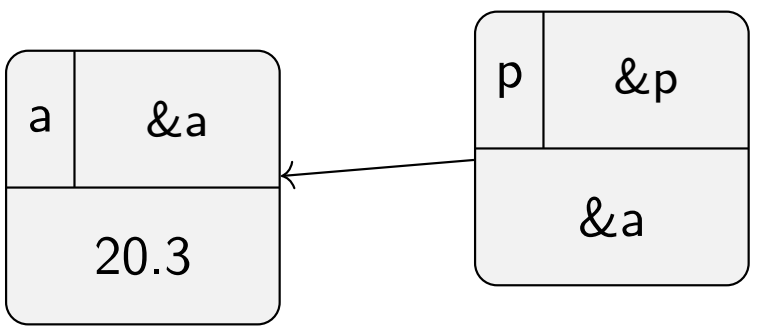


Exemplo: uso do operador *

```
1  int main(){
2
3      float *p, a;
4
5      a = 20.3;
6      p = &a;
7
8
9
10     return 0;
11 }
```

ao inicializar um ponteiro,
o operador * **não é utilizado**
(afinal, queremos armazenar o endereço
de a na variável p - e não no
endereço apontado por ela)

da mesma forma que fazemos
a=20.3 para armazenar 20.3
na variável a, fazemos p=&a para
armazenar o endereço no ponteiro



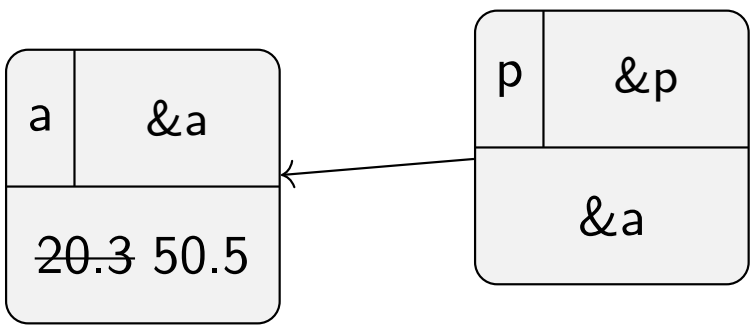
assim, p
passa a apontar
para a

Exemplo: uso do operador *

```
1  int main(){
2
3      float *p, a;
4
5      a = 20.3;
6      p = &a;
7
8      *p = 50.5;
9
10     return 0;
11 }
```

quando precisamos acessar o **endereço apontado pelo ponteiro**, o operador * é necessário

neste exemplo, o **endereço apontado** pelo ponteiro (ou seja, a variável a), recebe o valor 50.5



Uso do operador *

Em resumo

Ao manipular ponteiros, usamos o operador * quando:

- declaramos um ponteiro;
- precisamos acessar o conteúdo apontado pelo ponteiro (ou seja, o conteúdo da variável que está no endereço que o ponteiro aponta).

Agora é com você: sem olhar a resposta do slide anterior, faça um programa que declare uma variável do tipo `float` e um ponteiro que aponte para ela. Depois, altere o conteúdo armazenado de forma indireta (ou seja, usando o ponteiro).

Uso incorreto do operador *

É preciso atenção ao manipular ponteiros...

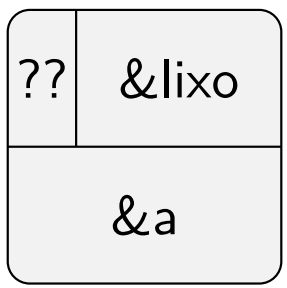
Exemplo 01: usando de forma incorreta o *

```
1  int main(){
2
3      int p, a; //erro
4
5      a = 20;
6      p = &a;
7
8      *p = 50;
9
10     return 0;
11 }
```

se você esquecer do operador *, vai declarar apenas uma variável deste tipo (e não um ponteiro)

neste exemplo, foram declarados dois inteiros: p e a. Portanto, não vai poder usar p como se fosse um ponteiro.

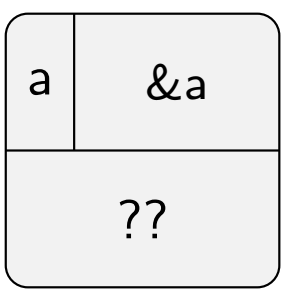
isso causa erro de compilação!

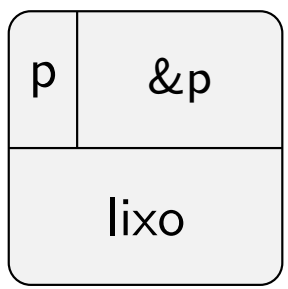


Exemplo 02: usando de forma incorreta o *

```
1  int main(){
2
3      int *p, a;
4
5      a = 20;
6      *p = &a;
7
8      *p = 50;
9
10     return 0;
11 }
```

se você usar * na inicialização, o que vai acontecer é que **o endereço de memória apontado por** p vai receber o endereço de a. O problema é que não temos controle sobre o endereço apontado por p

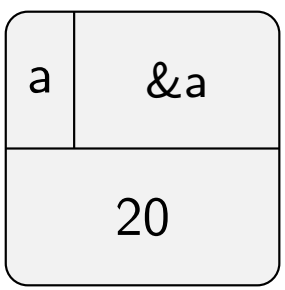


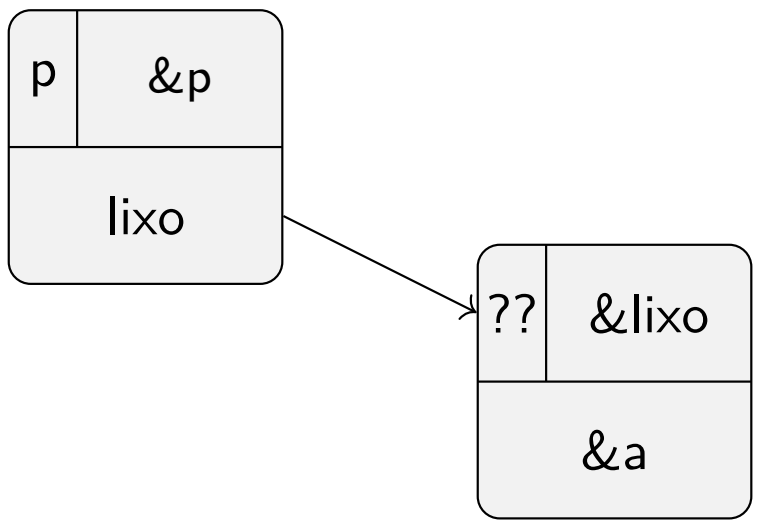


Exemplo 02: usando de forma incorreta o *

```
1  int main(){
2
3      int *p, a;
4
5      a = 20;
6      *p = &a;   //erro
7
8      *p = 50;
9
10     return 0;
11 }
```

o comando da linha 6 indica que algum local indefinido na memória deve receber o endereço de a (o que leva ao comportamento incorreto do programa)



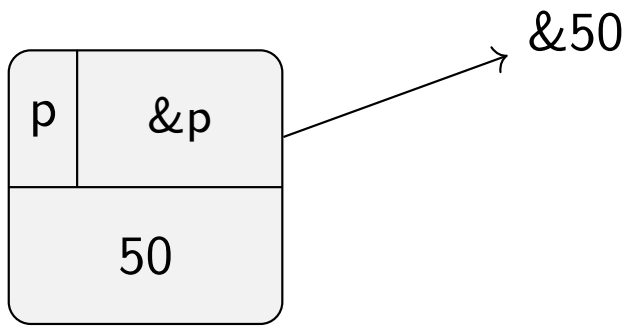


Exemplo 03: usando de forma incorreta o *

```
1  int main(){
2
3      int *p, a;
4
5      a = 20;
6      p = &a;
7
8      p = 50;  //erro
9
10     return 0;
11 }
```

quando queremos atribuir um conteúdo de forma indireta, **não podemos esquecer** o operador *

neste exemplo, ao atribuir p=50, fazemos com o ponteiro p passe a apontar para o endereço 50 de memória

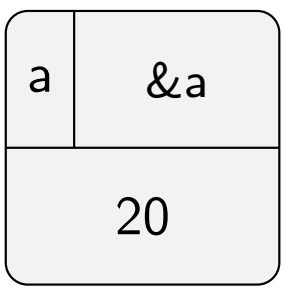


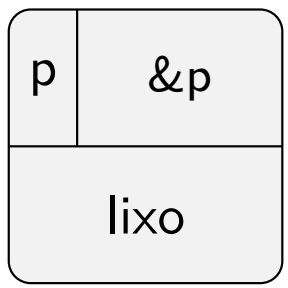
Exercício de fixação

Exercício de fixação 1

```
1  int main(){
2
3      int *p, a;
4
5      a = 20;
6      *p = &a;
7
8      *p = 5;
9      *p = 1;
10
11
12     return 0;
13 }
```

Quais as consequências deste erro?



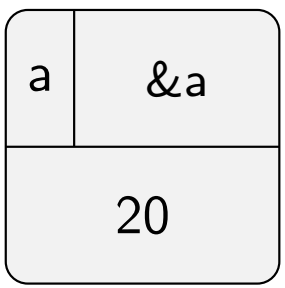


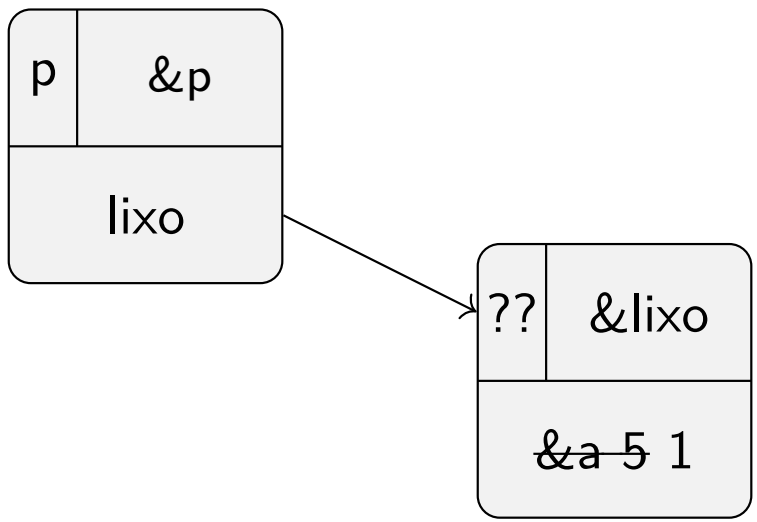
Exercício de fixação 1

```
1  int main(){
2
3      int *p, a;
4
5      a = 20;
6      *p = &a;
7
8      *p = 5;
9      *p = 1;
10
11
12     return 0;
13 }
```

como o ponteiro não está inicializado, aponta para um endereço qualquer

com isso, estas atribuições causam um comportamento incorreto do programa (veja abaixo)

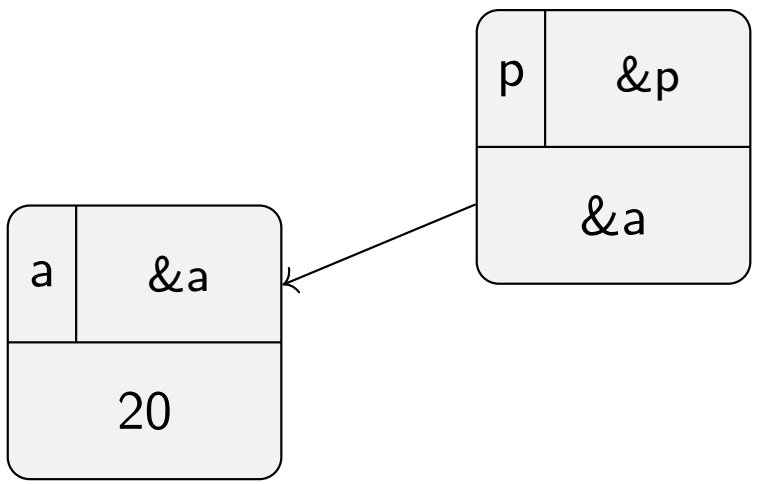




Exercício de fixação 2

```
1  int main(){
2
3      int *p, a;
4
5      a = 20;
6      p = &a;
7
8      p = 5;
9      *p = 1;
10
11
12     return 0;
13 }
```

Quais as consequências deste erro?

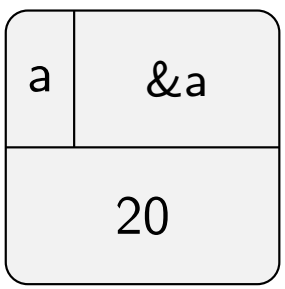


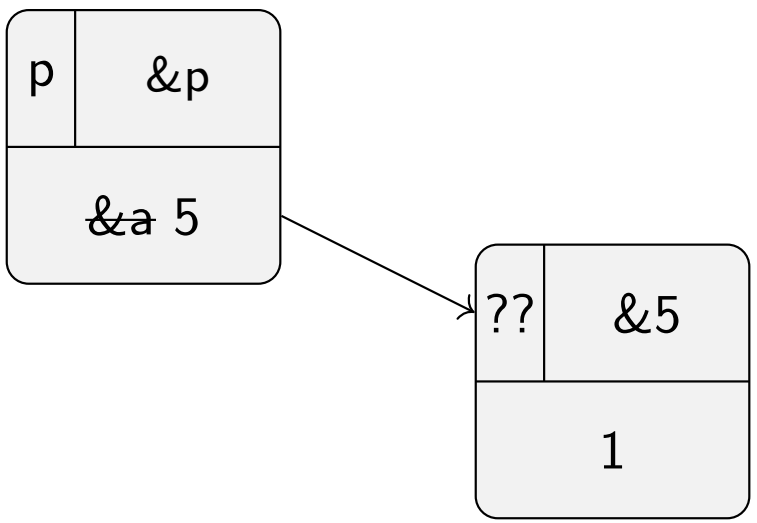
Exercício de fixação 2

```
1  int main(){
2
3      int *p, a;
4
5      a = 20;
6      p = &a;
7
8      p = 5;
9      *p = 1;
10
11
12     return 0;
13 }
```

como estamos acessando diretamente
o ponteiro (e não o endereço apontado),
o compilador vai entender que
o valor 5 é o endereço de memória
que o ponteiro deverá apontar

o comando da linha 9 fará com que
o valor 1 seja armazenado no endereço 5,
levando ao comportamento incorreto
do programa (veja abaixo)

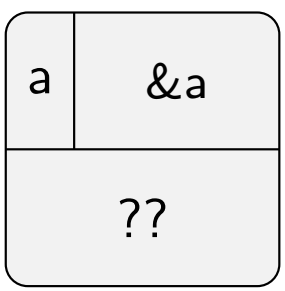


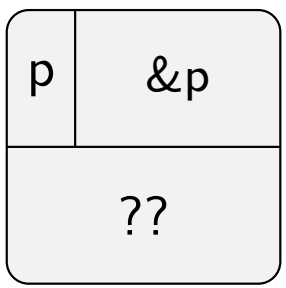


Exercício de fixação 3

```
1  int main(){
2
3      int p, a;
4
5      a = 20;
6      p = &a;
7
8      *p = 5;
9      *p = 1;
10
11
12     return 0;
13 }
```

Quais as consequências deste erro?



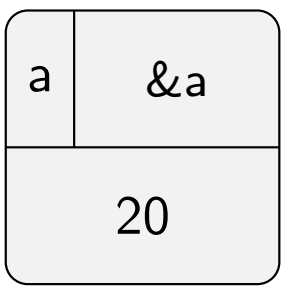


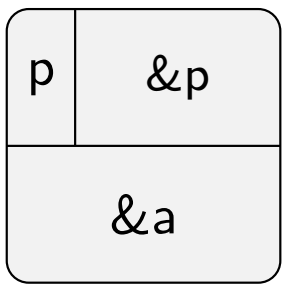
Exercício de fixação 3

```
1  int main(){
2
3      int p, a;
4
5      a = 20;
6      p = &a;
7
8      *p = 5;
9      *p = 1;
10
11
12     return 0;
13 }
```

nesta atribuição, a variável p receberá o endereço de a, mas vai interpretá-lo como um valor inteiro

como p não é ponteiro, não é possível acessar o endereço apontado. Ocorre um erro de compilação



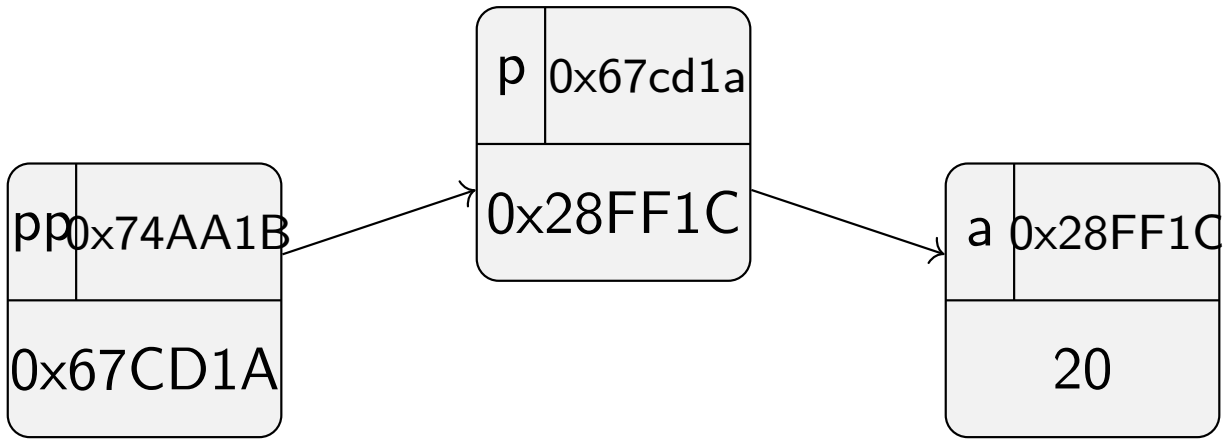


Ponteiro para ponteiro para...

Ponteiros para ponteiros

Podemos ter mais níveis de indireção (usando ponteiros para ponteiros para...).

```
1  int a, *p, **p;
2  a = 20;
3  p = &a;
4  pp = &p;
```



Por que usar ponteiros?

Neste ponto, você pode se perguntar qual a real utilidade dos ponteiros. Afinal, se temos acesso às variáveis, qual a motivação de acessá-las de forma indireta? Nas aulas seguintes você vai entender melhor a importância dos ponteiros ;-)

Exercícios de fixação

Você não precisa fazer estes exercícios... Entretanto, eles são interessantes para que você avalie se entendeu os conceitos básicos!

Exercício de fixação 1

Faça o teste de mesa para determinar o que o código abaixo imprime na tela:

```
1  int main() {
2
3    int b;
4    int *c;
5
6    b = 10;
7    c = &b;
8    *c = 11;
9    printf("b=%d", b);
10
11   return 0;
12 }
```

Exercício de fixação 1

Faça o teste de mesa para determinar o que o código abaixo imprime na tela:

```
1  int main() {
2
3    int b;
4    int *c;
5
6    b = 10;
7    c = &b;
8    *c = 11;
9    printf("b=%d", b);
10
11   return 0;
12 }
```

Resposta:

```
1  b=11
```

Exercício de fixação 2

Faça o teste de mesa para determinar o que o código abaixo imprime na tela:

```
1  int main() {
2
3      int num, q = 1;
4      int *p;
5
6      num = 100;
7      p = &num;
8      q = *p;
9      printf("q=%d", q);
10
11     return 0;
12 }
```

Exercício de fixação 2

Faça o teste de mesa para determinar o que o código abaixo imprime na tela:

```
1  int main() {
2
3      int num, q = 1;
4      int *p;
5
6      num = 100;
7      p = &num;
8      q = *p;
9      printf("q=%d", q);
10
11     return 0;
12 }
```

Resposta:

```
1  q=100
```

Exercício de fixação 3

Quais valores são impressos ao final destes programas?

```
1  int main()
2  {
3    int a,b,*c;
4    a = 4;
5    b = 3;
6    c = &a;
7    *c = *c + 1;
8    c = &b;
9    b = b+4;
10   printf("R1: %d %d %d",a,b,*c);
11
12     return 0;
13 }
14
```

```
int main()
{
   int a,b,*c,*d,*f;
   a = 4;
   b = 3;
   c = &a;
   d = &b;
   *c = *c / 2;
   f = c;
   c = d;
   d = f;
   printf("R2: %d%d",*c,*d);
   return 0;
 }
```

Exercício de fixação 3

Quais valores são impressos ao final destes programas?

```c
int main()
{
  int a,b,*c;
  a = 4;
  b = 3;
  c = &a;
  *c = *c + 1;
  c = &b;
  b = b+4;
  printf("R1: %d %d %d",a,b,*c);

    return 0;
}
```

```c
int main()
{
   int a,b,*c,*d,*f;
   a = 4;
   b = 3;
   c = &a;
   d = &b;
   *c = *c / 2;
   f = c;
   c = d;
   d = f;
   printf("R2: %d%d",*c,*d);
   return 0;
 }
```

Respostas:

```
R1: 5 7 7
```

```
R2: 3 2
```

Exercício de fixação 4

Qual o erro no programa abaixo?

```
1 int a, b;
2 int *c;
3 b = 10;
4 *c = 13;
```

Exercício de fixação 4

Qual o erro no programa abaixo?

```
1 int a, b;
2 int *c;
3 b = 10;
4 *c = 13;
```

Não se pode atribuir um valor ao endereço apontado pelo ponteiro, sem antes ter certeza de que tal endereço é válido! O correto seria, por exemplo:

```
1 int a, b;
2 int *c;
3 b = 10;
4 c = &a; //não esqueça da inicialização
5 *c = 13;
```

Exercício de fixação 5

No programa abaixo, ocorre um erro de compilação, pois o * é interpretado como operador de ponteiro sobre a variável c. Como corrigir o programa para que a variável a receba o resultado da multiplicação de b pelo conteúdo apontado por c?

```
1  #include <stdio.h>
2  int main() {
3      int b, a;
4      int *c;
5      b = 10;
6      c = &a;
7      *c = 11;
8      a = b * c;
9      printf("%d\n", a);
10     return 0;
11 }
```

Exercício de fixação 5

No programa abaixo, ocorre um erro de compilação, pois o `*` é interpretado como operador de ponteiro sobre a variável `c`. Como corrigir o programa para que a variável `a` receba o resultado da multiplicação de `b` pelo conteúdo apontado por `c`?

```
1  #include <stdio.h>
2  int main() {
3      int b, a;
4      int *c;
5      b = 10;
6      c = &a;
7      *c = 11;
8      a = b * c;
9      printf("%d\n", a);
10     return 0;
11 }
```

Precisamos fazer: `a = b * (*c);`

Exercício de fixação 6

Qual o erro no programa abaixo?

```
1  int main() {
2      double b, a;
3      int *c;
4      b = 10.89;
5      c = &b;
6      a = *c;
7      printf("%lf\n",a);
8      return 0;
9  }
```

Exercício de fixação 6

Qual o erro no programa abaixo?

```
1  int main() {
2      double b, a;
3      int *c;
4      b = 10.89;
5      c = &b;
6      a = *c;
7      printf("%lf\n",a);
8      return 0;
9  }
```

O tipo do ponteiro deve ser compatível com o tipo da variável apontada! Neste exemplo, além de o compilador alertar que a atribuição pode causar problemas, um valor diferente de `10.89` será impresso.