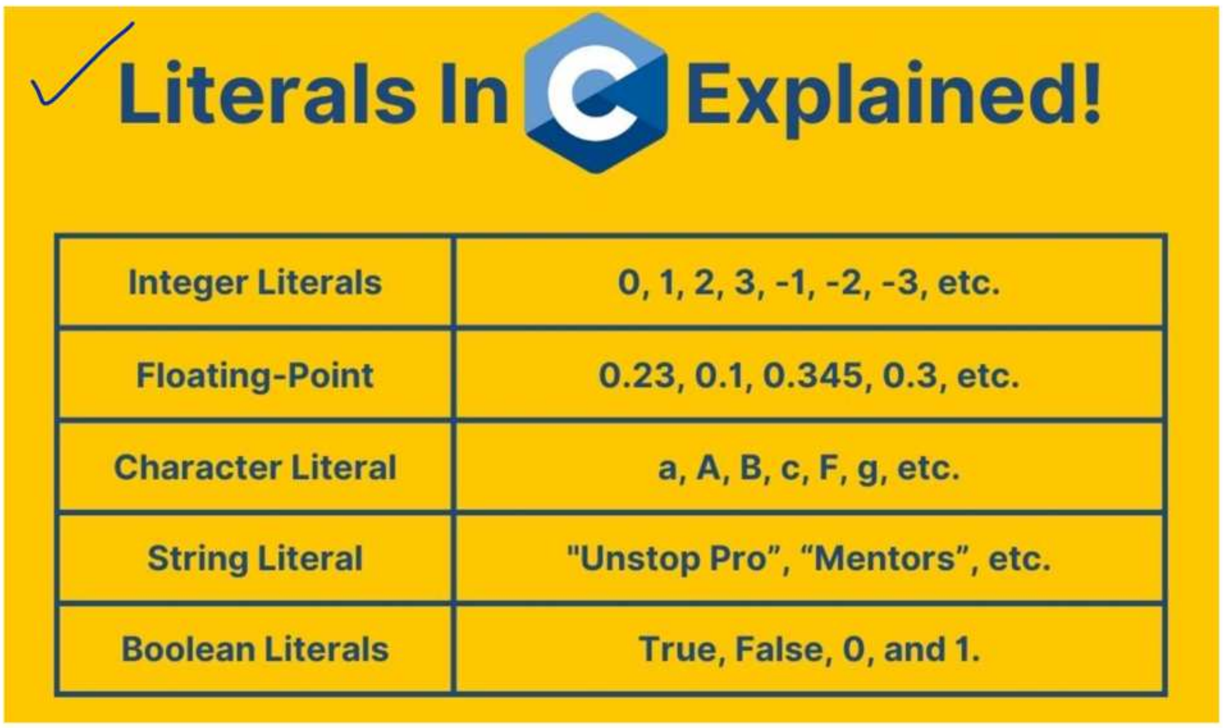
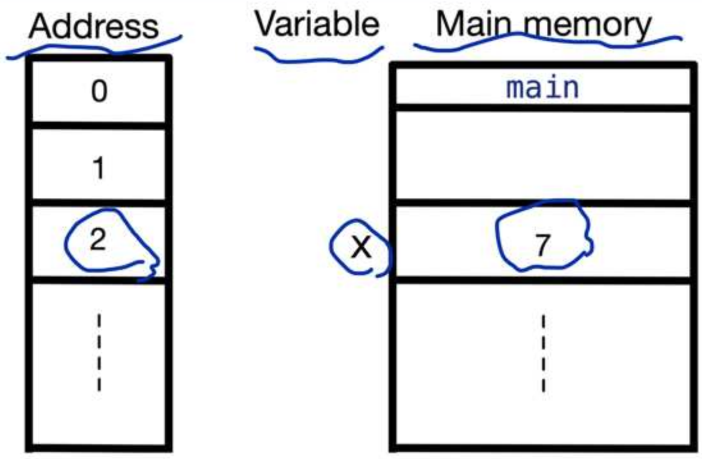
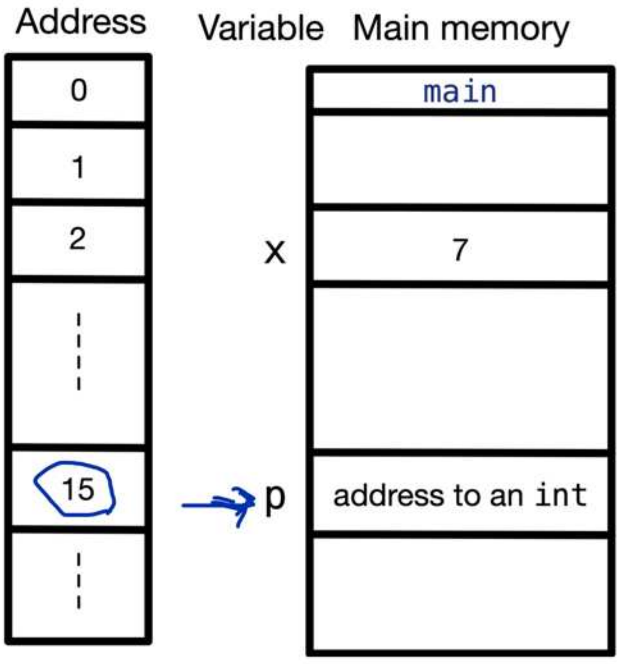


✓ # Literals In C Explained!

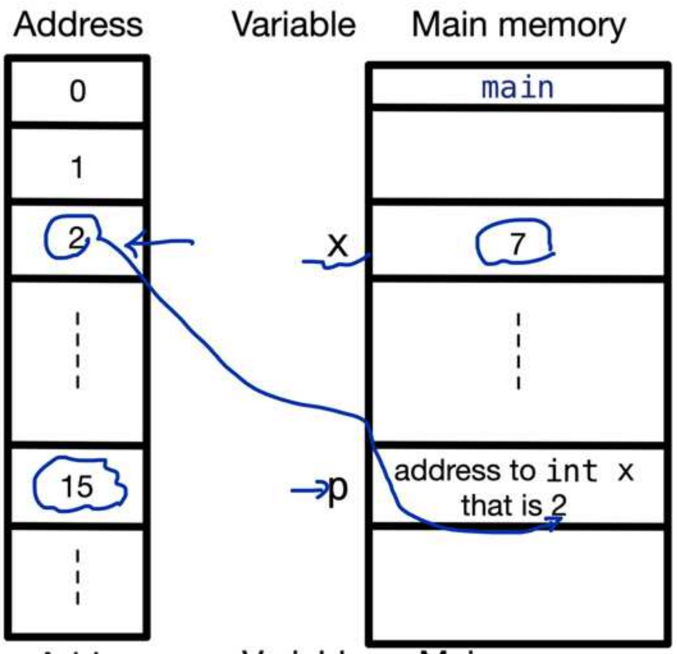
Integer Literals	0, 1, 2, 3, -1, -2, -3, etc.
Floating-Point	0.23, 0.1, 0.345, 0.3, etc.
Character Literal	a, A, B, c, F, g, etc.
String Literal	"Unstop Pro", "Mentors", etc.
Boolean Literals	True, False, 0, and 1.



```
#include <stdio.h>

int main(void) {

① int x = 7;   data type of x is int

  return 0;
}
```



```
#include <stdio.h>

int main(void) {

① int x = 7;   data type of x is int

② int *p;      data type of p is int*

  return 0;
}
```



```
#include <stdio.h>

int main(void) {
①  int x = 7;
②  int *p;
③  p = &x;
    return 0;
}
```

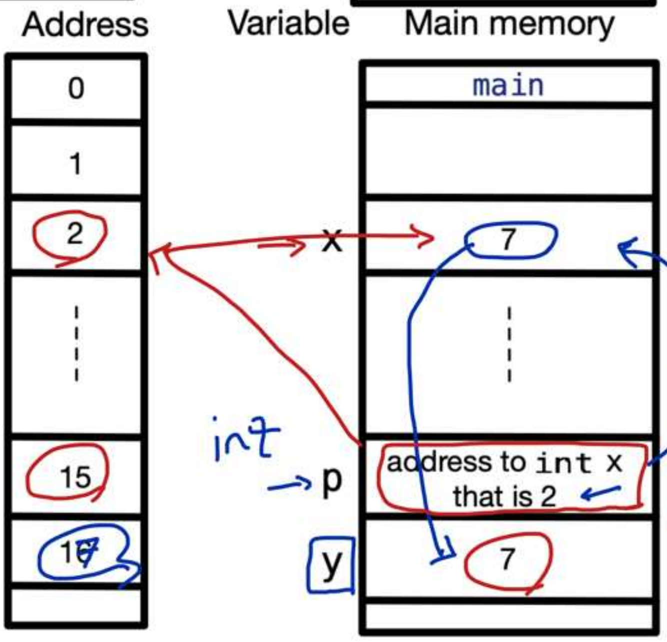
data type of x is int

data type of p is int*

p is assigned the address of x

p=&x

Reference operator: this means "address of"
Recall its usage in `scanf`



```
#include <stdio.h>

int main(void) {
①  int x = 7;
②  int *p;
③  p = &x;
④  int y;
⑤  y = *p;
    return 0;
}
```

data type of x is int

data type of p is int*

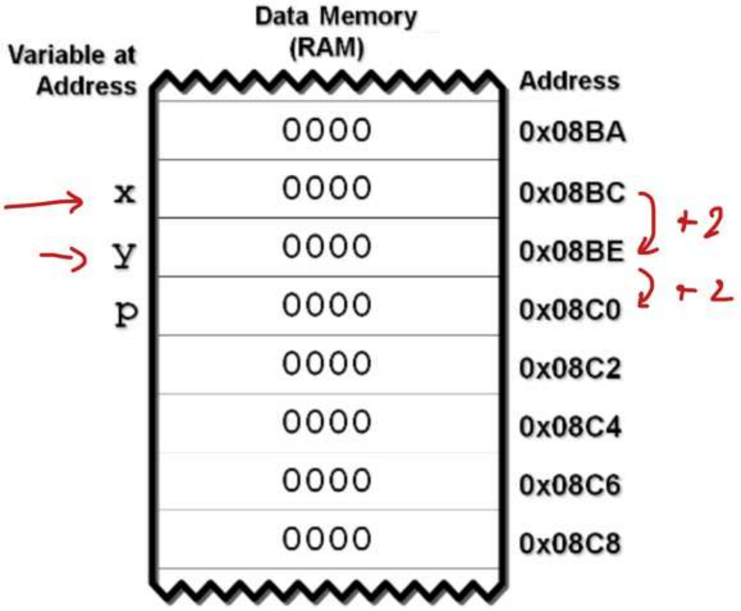
p is assigned the address of x

declare variable y

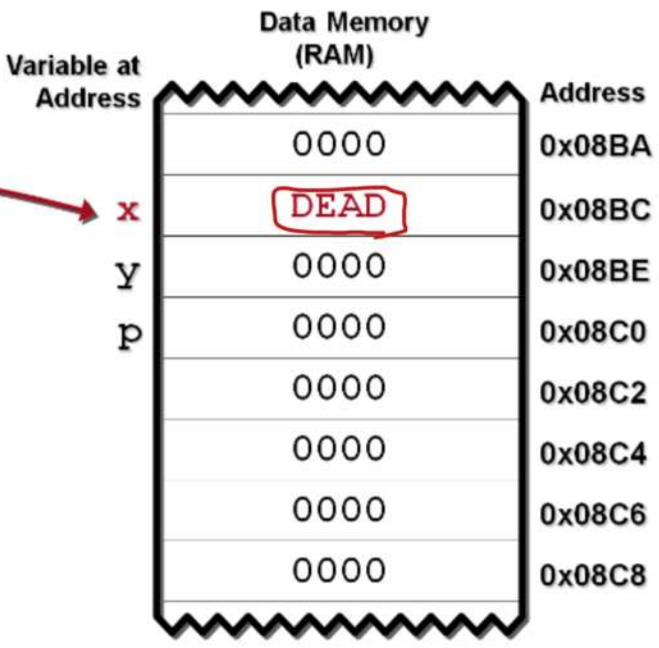
y is assigned the value stored at address p — x

Dereference operator: this means "value at" address p

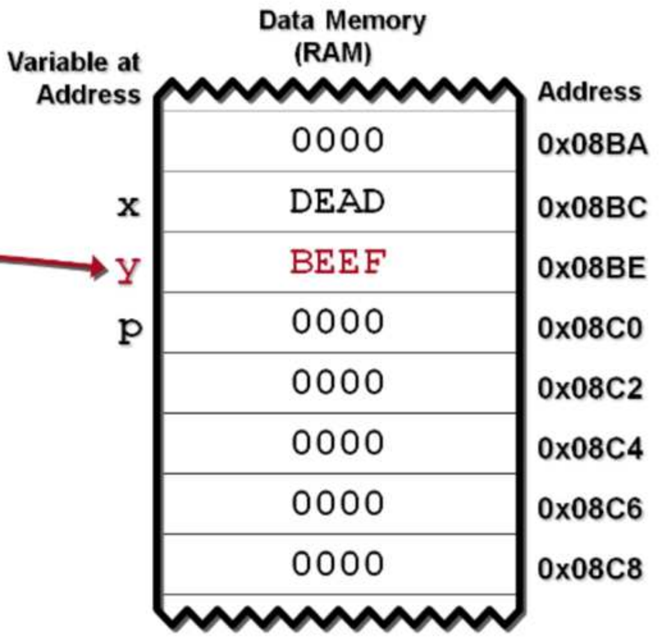
```
{
  int x, y;
  int *p;

  x = 0xDEAD;
  y = 0xBEEF;
  p = &x;

  *p = 0x0100;
   p = &y;
  *p = 0x0200;
}
```



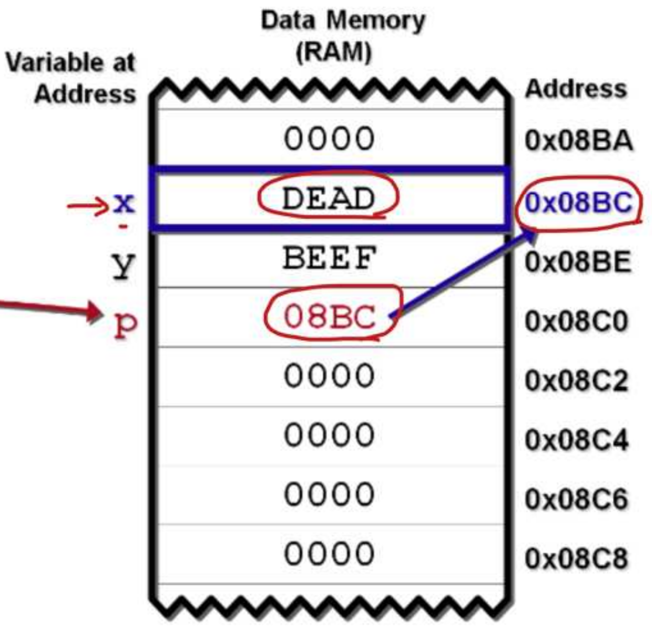
```
{
 int x, y;
 int *p;

 x = 0xDEAD;
 y = 0xBEEF;
 p = &x;

 *p = 0x0100;
  p = &y;
 *p = 0x0200;
}
```



```
{
 int x, y;
 int *p;

 x = 0xDEAD;
 y = 0xBEEF;
 p = &x;

 *p = 0x0100;
  p = &y;
 *p = 0x0200;
}
```



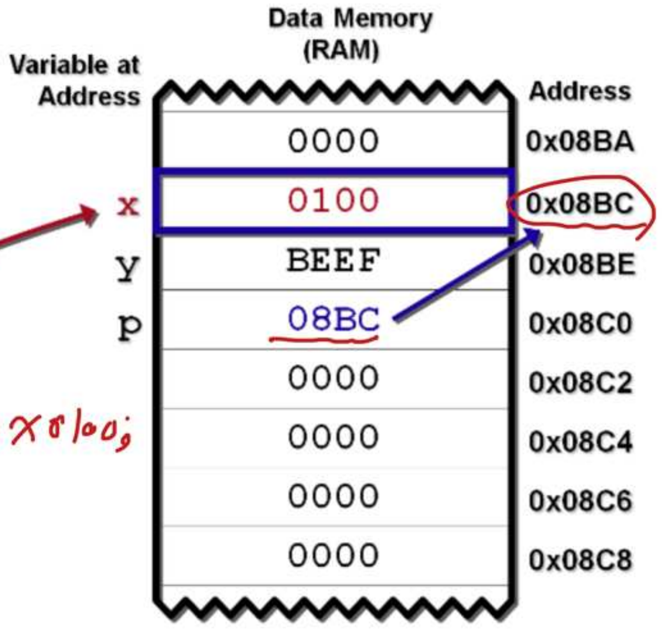
```
{
 int x, y;
 int *p;

 x = 0xDEAD;
 y = 0xBEEF;
 p = &x;

 *p = 0x0100;
  p = &y;
 *p = 0x0200;
}
```



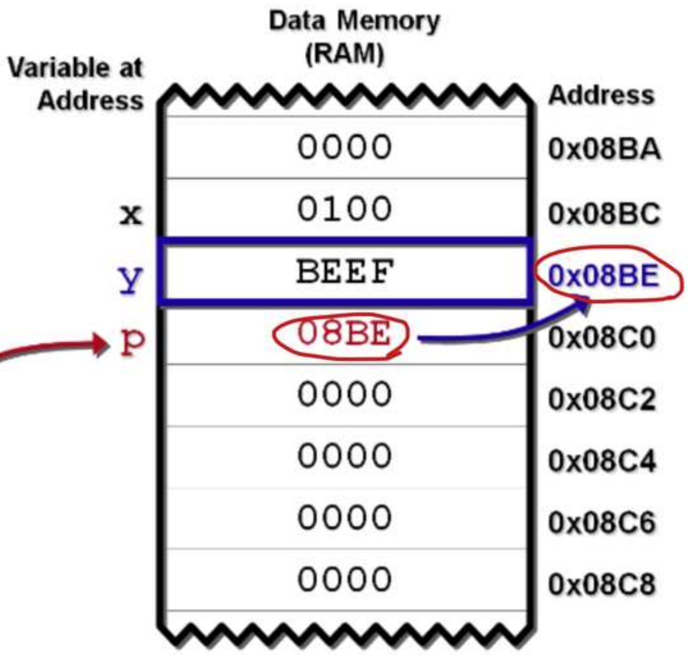
```
{
  int x, y;
  int *p;

  x = 0xDEAD;
  y = 0xBEEF;
  p = &x;

 *p = 0x0100;
  p = &y;
 *p = 0x0200;
}
```

x ←

x = 0x0100;



```
{
  int x, y;
  int *p;

  x = 0xDEAD;
  y = 0xBEEF;
  p = &x;

 *p = 0x0100;
  p = &y;
 *p = 0x0200;
}
```

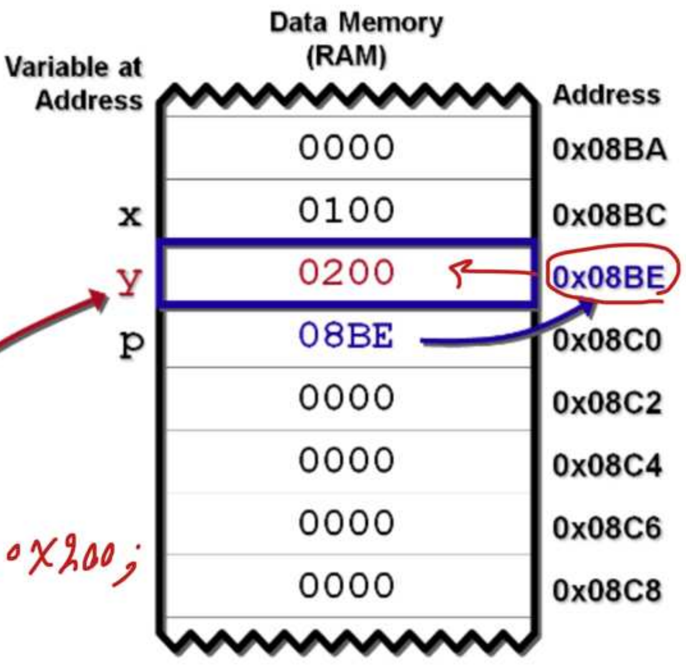


```
{
  int x, y;
  int *p;

  x = 0xDEAD;
  y = 0xBEEF;
  p = &x;

 *p = 0x0100;
  p = &y;
 *p = 0x0200;
}
```

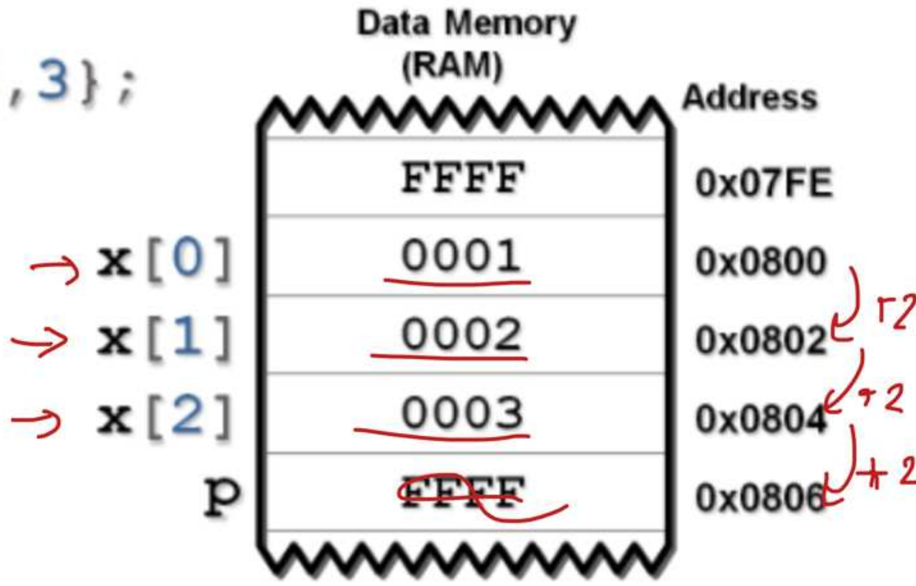
*

&y

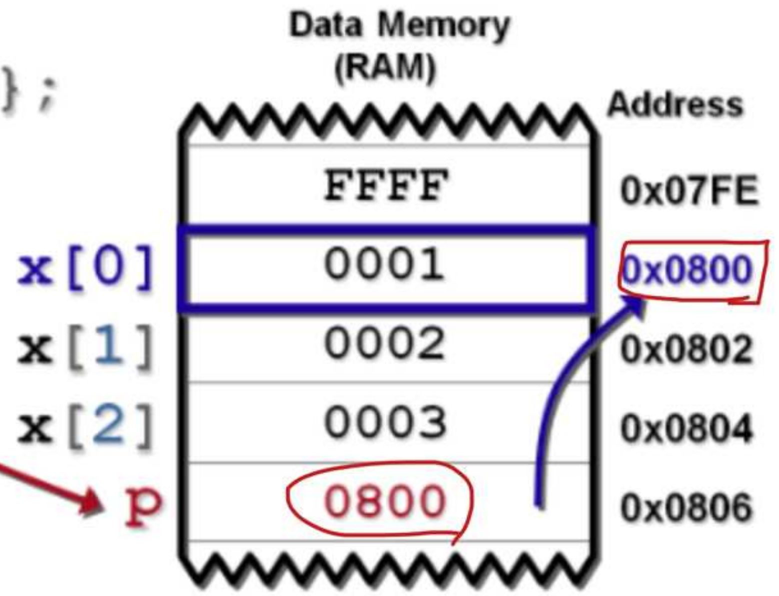
y = 0x200;



```
int x[3] = {1,2,3};
int *p;

p = &x;
p++;
```



```
int x[3] = {1,2,3};
int *p;

p = &x;
p++;
```



```
int x[3] = {1,2,3};
int *p;

p = &x;
~~p++;~~
```

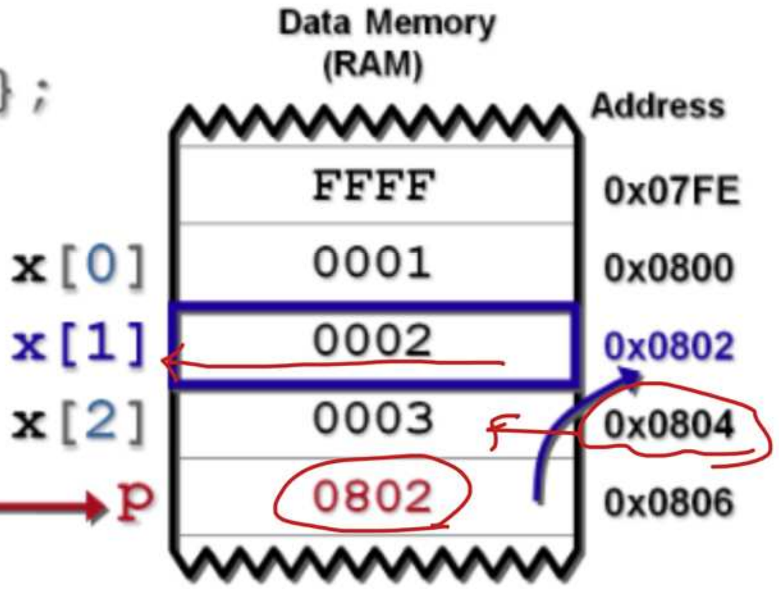
int

2Byte

$\boxed{P+4}$

↳p=p+2;

0x04



Pointers as input arguments

- By using pointers, a procedure **can access data that belongs to the caller**.
- **Example.** A procedure that swaps the value of two variables:

```
void swap(int a, int b) {
    int temp = a ;
    a = b ;
    b = temp ;
}

/* example usage */
int x = 10 ;
int y = 20 ;
swap(x,y) ;
/* x = 10, y = 20 */
```

Stack

a 2 + b 2 4byte

2 + 2 4byte

The function has no effect because calling it a *and* b *are copies of* x *and* y. x *and* y *remain unaffected.*

```
void swap(int *a, int *b) {
    int temp = *a ;
    *a = *b ;
    *b = temp ;
}

/* example usage */
int x = 10 ;
int y = 20 ;
swap(&x,&y) ;
/* x = 20, y = 10 */
```

4byte

4byte

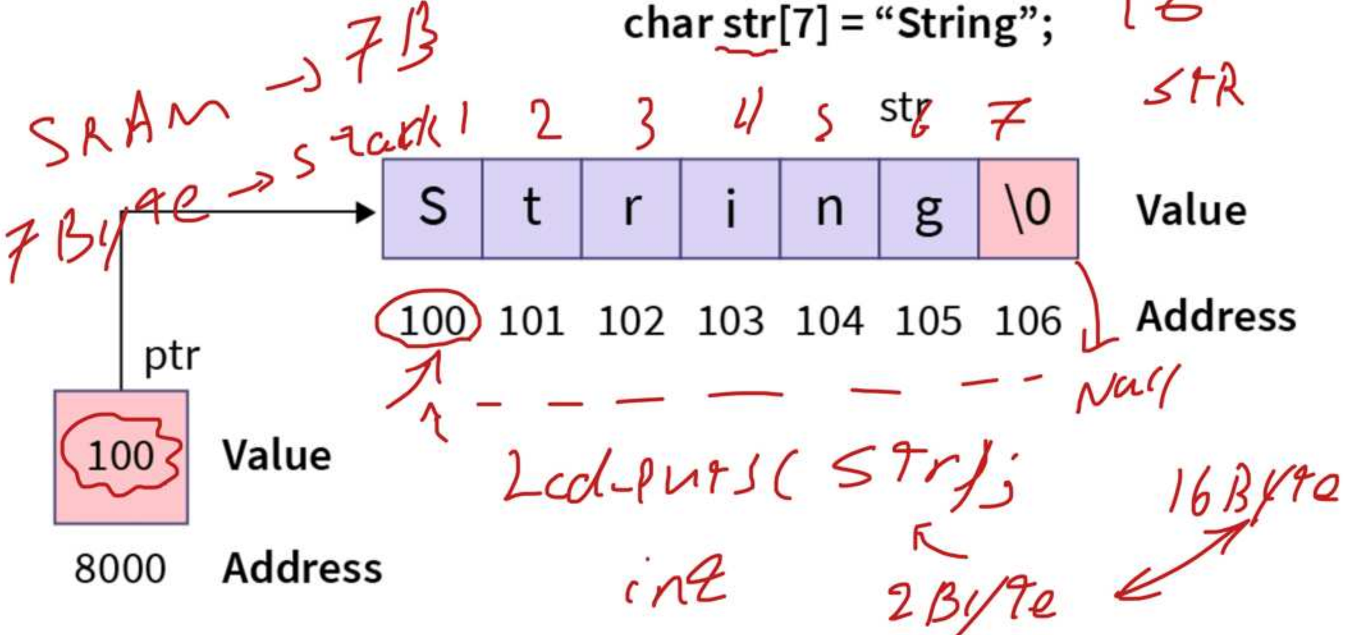
By passing pointers, the function can access the variables x and y in the caller and can swap them.

&str[0] ⟺ str

char str[7] = "String";

16

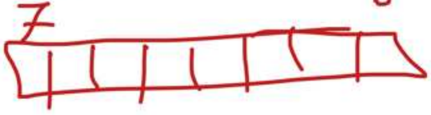
STR



ptr pointer points to starting address of pointer str that holds the string

```
void alcd_puts(char *str)
{
 while(*str != 0)
 {
  lcd_putchar(*str);
  str++;
 }
}
```

uint8_t a;



null

uint8_t a:3;

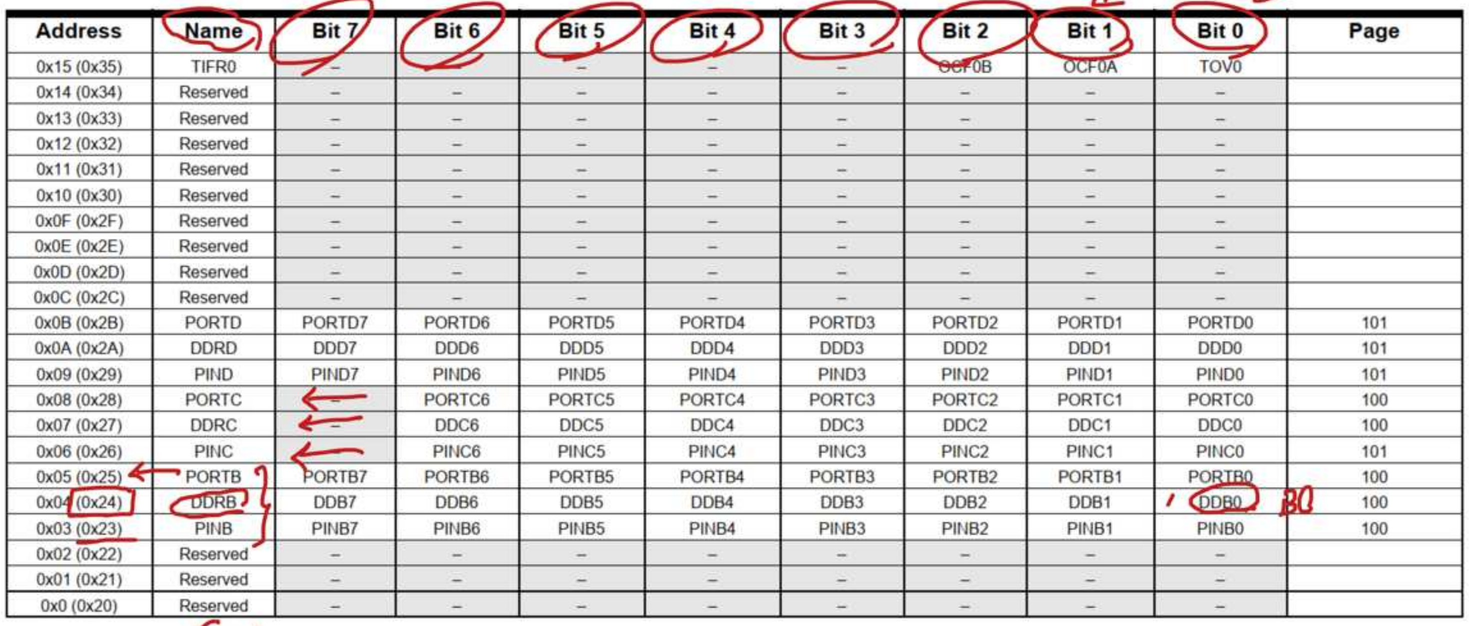
str

*str

struct

union

```
sprintf(lcd,"Value of A=%d", A);
alcd_puts(lcd);
```



Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	
0x14 (0x34)	Reserved	–	–	–	–	–	–	–	–	
0x13 (0x33)	Reserved	–	–	–	–	–	–	–	–	
0x12 (0x32)	Reserved	–	–	–	–	–	–	–	–	
0x11 (0x31)	Reserved	–	–	–	–	–	–	–	–	
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–	
0x0F (0x2F)	Reserved	–	–	–	–	–	–	–	–	
0x0E (0x2E)	Reserved	–	–	–	–	–	–	–	–	
0x0D (0x2D)	Reserved	–	–	–	–	–	–	–	–	
0x0C (0x2C)	Reserved	–	–	–	–	–	–	–	–	
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	101
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	101
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	101
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	100
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	100
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	101
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	100
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	100
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	100
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–	
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–	
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–	

B0

GPIO