

→ int main()

int num [5];

"hello";

}

character array +

{

{

"%d\n"

printf ("%s\n",

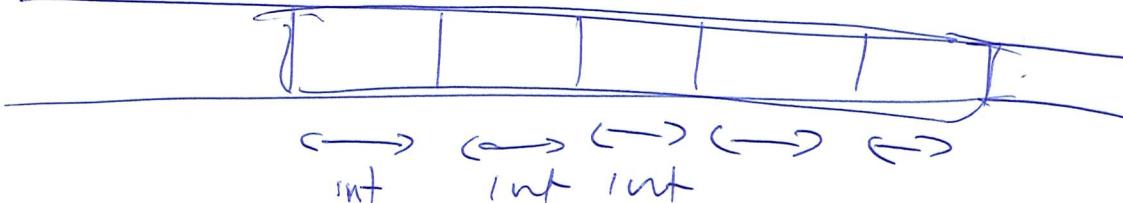
{

char rowing[] = "rowing";

}

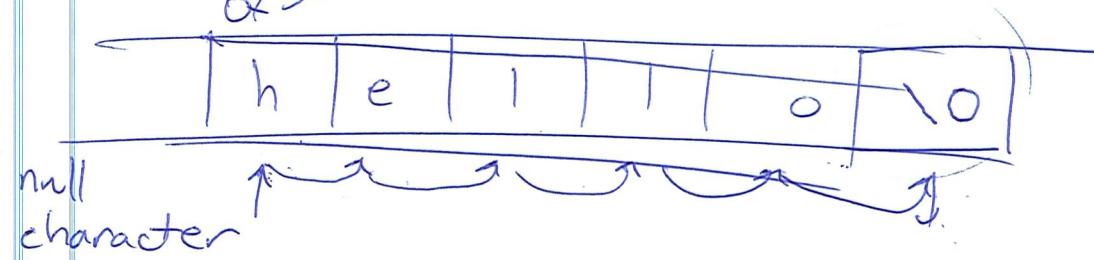
stack

0x1000



static/global

0x3000



0

char [] name = {'N', 'A', 'R', 'Y'};

stack 0x1000

| r | o | w | i | n | g | \0 |

7 bytes

char [] xc = name;

```

#include <stdio.h>
#include <string.h>

int main() {

    int numbers[5];

    char letters[7];

    char word_array[] = { 'P', 'O', 'I', 'N', 'T', 'E', 'R', 'S' };

    char *word = "POINTERS";
    letters[1] = word[3];
    numbers[4] = word[0];

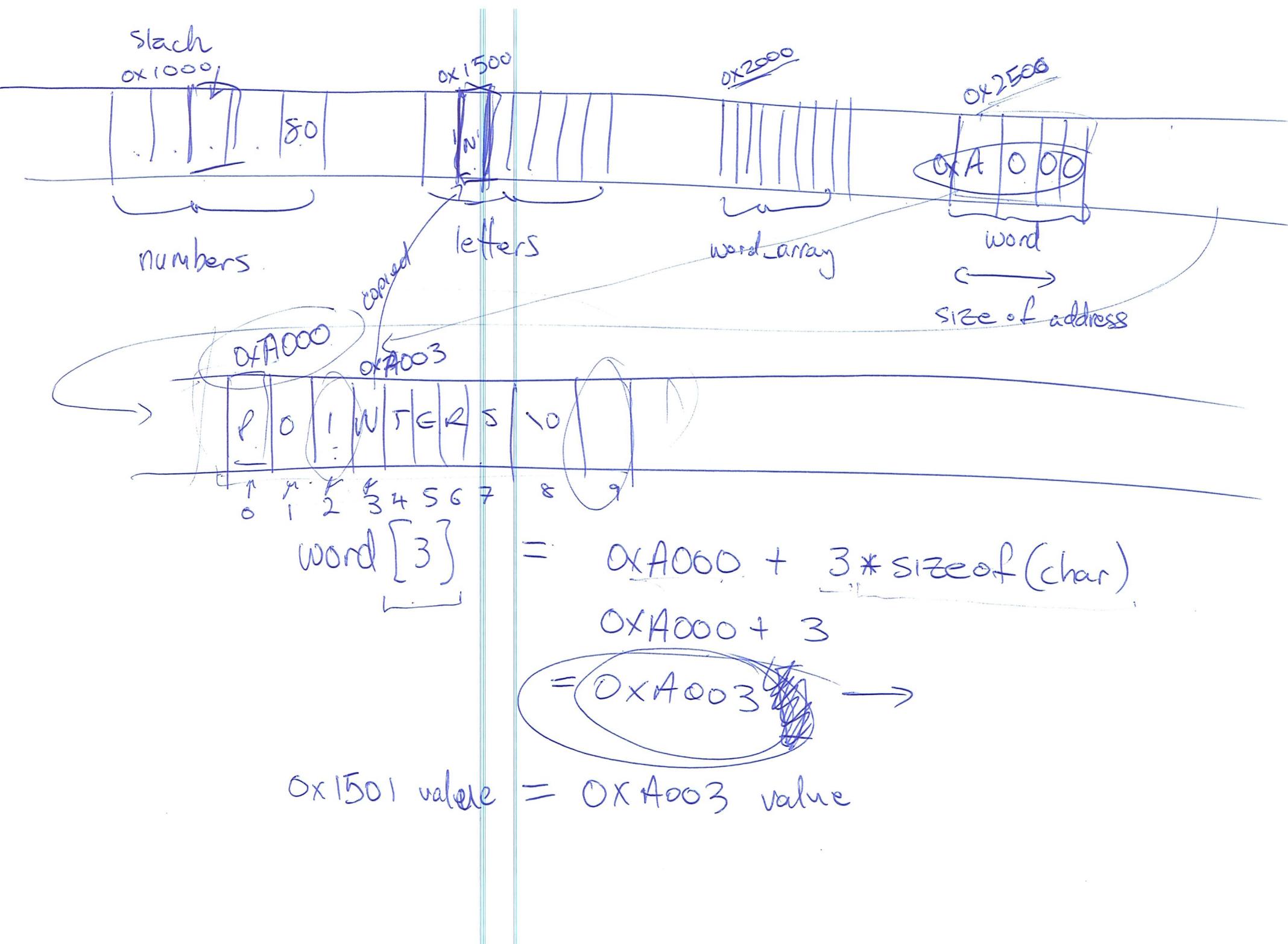
    printf("numbers[2] = %d\n", numbers[2]); → 68
    printf("numbers[4] = %d\n", numbers[4]); → 80
    printf("letters[1] = %d\n", letters[1]); → 78
    printf("word_array[2] = %d\n", word_array[2]); → 73
    printf("word[9] = %d\n", word[9]); → 65

    int i;
    for (i = 0; i < strlen(word); ++i)
        printf("%c = %d\n", word[i], word[i]);

    /*
    P = 80
    O = 79
    I = 73
    N = 78
    T = 84
    E = 69
    R = 82
    S = 83
    */

    return 0;
}

```



char msg[] = "hello!";

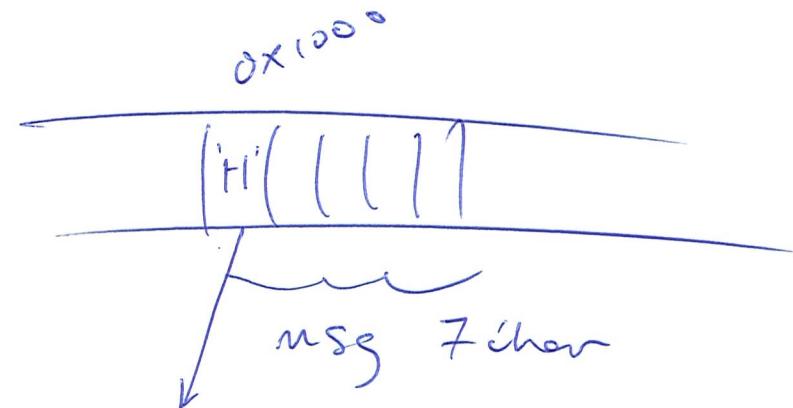


Figure A

char * str = ~~char msg[0];~~
 $\ast(0x1000) \rightarrow 'H'$ (73)
 $\ast(0x1001) \rightarrow 'E'$
 $\ast(0x1000 + 1) \rightarrow 'I'$
 $\ast(0x1000 + 1) \rightarrow 'E'$

Please note my mistake

boolean expressions

not (A and B)

not A

and

not B

msg + 0 * sizeof(char)

(0x1000)[0] \rightarrow 0

(0x1000 value of)

& (0x1000 value of)

0x1000

3 \neq $\frac{1}{2}$

'H'

⇒ Why this?

Consider

$$3 * 2 + 1$$

$$3 * (2+1)$$

11

9

$$(3*2) + 1$$

11

7

which one
do we
want?

E

```

#include <stdio.h>
#include <string.h>

int main() {
    int numbers[5];
    char letters[7];
    char word_array[] = { 'P', 'O', 'I', 'N', 'T', 'E', 'R', 'S' };
    char *word = "POINTERS";
    char word_array_2[] = "POINTERS";
    char *word_array_3[] = { "POINTERS" };
    char **word_array_4[] = { &word, word_array_3 };
    *** letters[1] = word[3];
    numbers[4] = word[0];

    printf("numbers address: \t%p\n", numbers);
    printf("letters address: \t%p\n", letters);
    printf("word_array address: \t%p\n", word_array);
    printf("word address: \t\t%p\n", word);
    printf("word_array_2 address: \t%p\n", word_array_2);
    printf("word_array_3 address: \t%p\n", word_array_3);
    printf("word_array_4 address: \t%p\n", word_array_4);

    // how to print string using word_array_3?
    printf("word_array_3: %s\n", word_array_3);
    // how to print both strings using word_array_4?
    printf("word_array_4: %s %s\n", word_array_4);
}

return 0;
}

```

Diagram illustrating pointer assignments:

- word**: A character pointer pointing to the memory location of the string "POINTERS". Address: **0x7500**.
- word_array_2**: A character array containing the string "POINTERS". Address: **0xA000**. It occupies 9 bytes of memory.
- word_array_3**: A character array containing the string "POINTERS". Address: **0xA000**. It occupies 9 bytes of memory.
- word_array_4**: A double pointer array containing two elements: the address of **word** and the address of **word_array_3**. Address: **0x2500**.
- letters**: A character array containing the character at index 3 of **word**, which is 'O'. Address: **0x7500**.
- numbers**: An integer array containing the value at index 0 of **word**, which is 80. Address: **0x7500**.

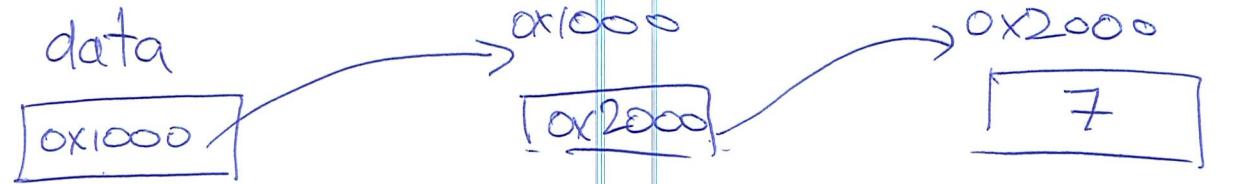
Annotations:

- 9 bytes copied content**: Referring to the size of the string "POINTERS".
- 1x 64 bits**: Referring to the size of the pointer variable.

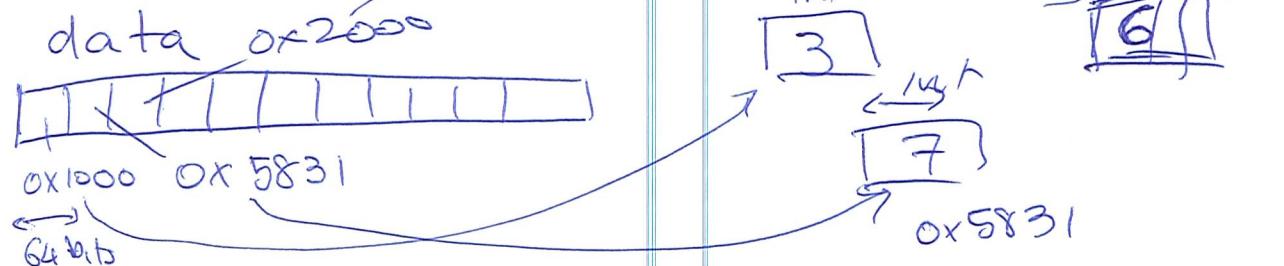
int **data

$\ast(\ast(0x1000))$

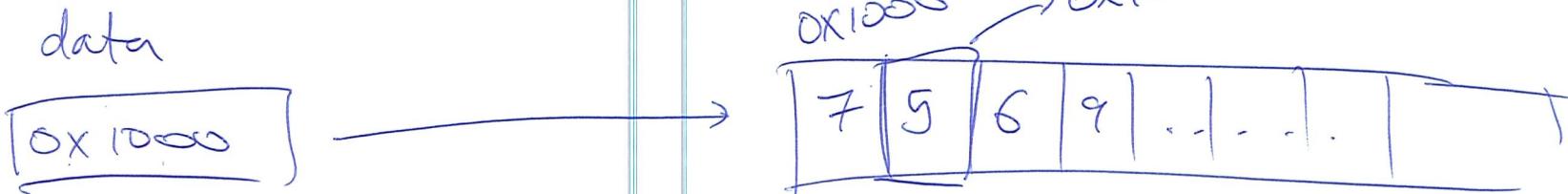
①



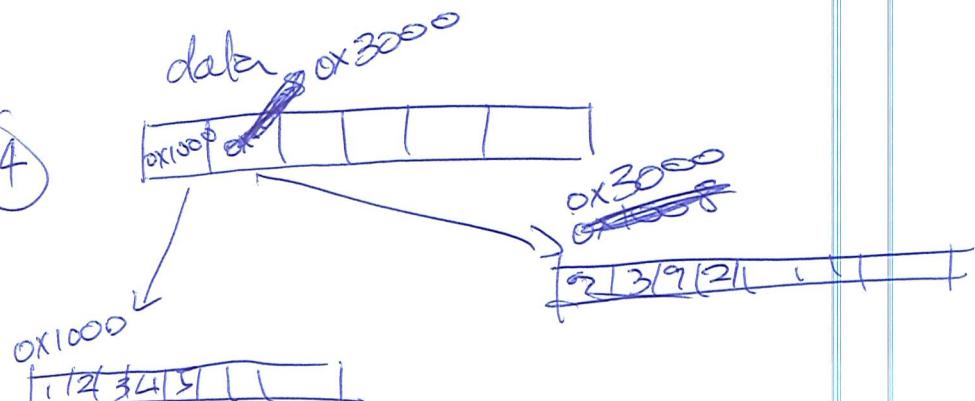
②



③



④



$(\ast \text{data})[1]$
 $\ast((\ast \text{data}) + 1)$

$\# \text{data} == 0x1000$
 $0x1000[1]$

$(\ast \text{data})$

$\ast(0x2000)$

7

equivalent

⇒ Why this?

Consider

$$3 * 2 + 1$$

$$3 * (2+1)$$

11

9

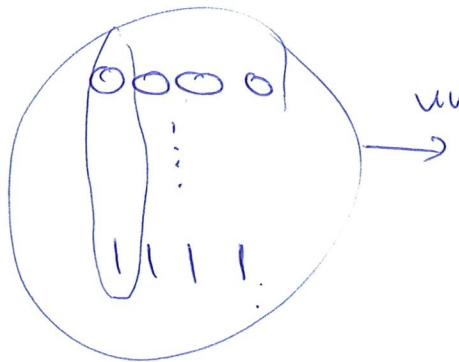
$$(3 * 2) + 1$$

11

7

which one
do we
want?

E



unsigned

0

↓

15

signed

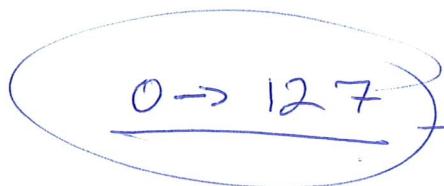
-8

1 ...

7

0 ...

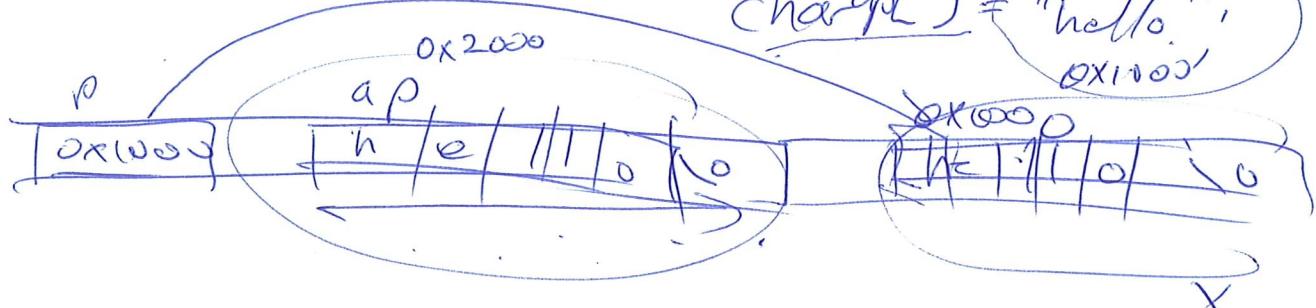
0000*



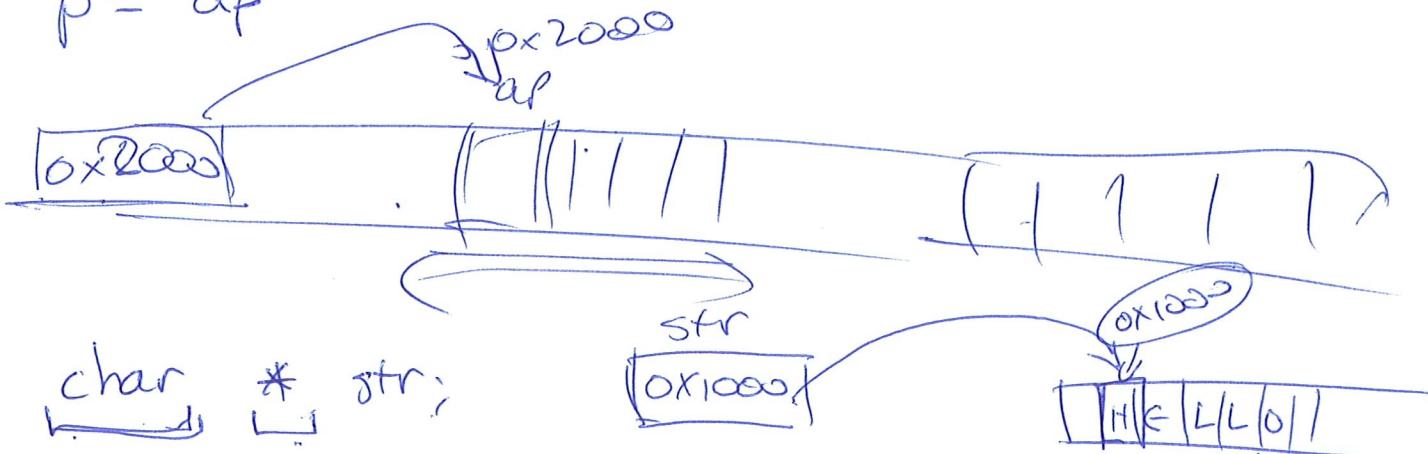
extended ASCII 0 - 255

`char * p = "hello";`

`char * p = { "h" ... },`
`char p[] = "hello.",`
`0x1000`



`p = ap`



< stdint.h > `(uint32_t *) *str get sizeof(char) bytes`
`uint32_t *istr = "hello"; from address 0x1000`
`*istr → gets 4 bytes`

~~static~~ ~~char~~ ~~hello~~
print (" %s \n", istr) → "hello"

~~uint32_t~~ tmp = *istr;
~~char msg[] = *~~
~~i = 0;~~
for (~~i~~; i < 4; i++)
 print("%c", ((char*)tmp)[i]);
printf("\n");

$$\frac{A=65}{a=97}$$

printf ("%op\n")