The C Programming Language: Part II

Lecture 3

1107186 – Estruturas de Dados

Prof. Christian Azambuja Pagot
CI / UFPB

Universidade Federal da Paraíba
Centro de Informática

Strings in C

- C **does not** have a type **string**.
- Strings are implemented as **arrays** of **characters** terminated with binary zero (**\0**).
- **Example:**

C code excerpt:

```
char *name;
```

Strings in C (cont.)

- **How to fill a string?**

1 **C code excerpt:**

```
char *name;
...
name[0] = 'C';
name[1] = 'h';
name[2] = 'r';
name[3] = 'i';
name[4] = 's';
name[5] = '\0';
```

2 **C code excerpt:**

```
char *name;
...
*(name)   = 'C';
*(name+1) = 'h';
*(name+2) = 'r';
*(name+3) = 'i';
*(name+4) = 's';
*(name+5) = '\0';
```

3 **C code excerpt:**

```
char name[6] = {'C','h','r','i','s','\0'};
```

4 **C code excerpt:**

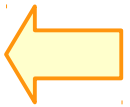
```
char name[6] = “Chris”;
```

Strings in C (cont.)

- **How to copy a string?**

C code excerpt:

```
char *name1 = “Chris”;
char *name2;
...
name2 = name1;
```



It just **copies** the **address** of the **first char** of the string **pointed by name1** to the pointer **name2!**

Strings in C (cont.)

- **How to copy a string?**

C code excerpt:

```
void my_strcpy(char *destination,
               char *source)
{
    char *p = destination;

    while (*source != '\0')
    {
        *p++ = *source++;
    }

    *p = '\0';
}
```

C code excerpt:

```
char *name1 = “Chris”;
char *name2;
...
my_strcpy(name2,name1);
```

Suggested Exercises

- Read about the following C string functions, and, through inverse engineering, implement your own versions of those functions:
  - `strlen()`;
  - `strcat()`;
  - `strchr()`;

2D Arrays in C

- Creating 2D arrays with array notation:

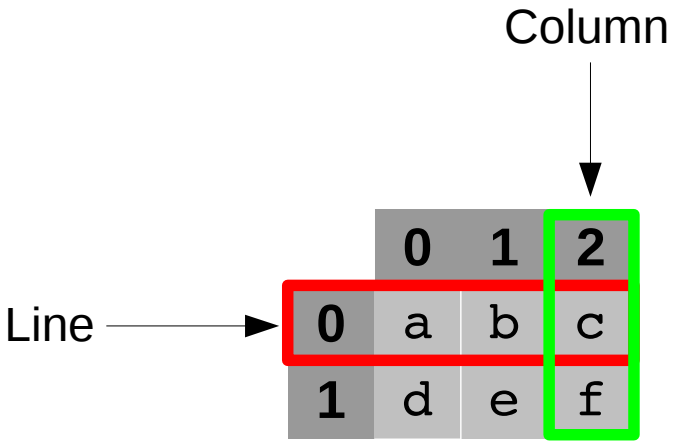
C code excerpt:

```
char x[2][3];
```

- Writing to array elements:

C code excerpt:

```
x[0][0] = 'a';
x[0][1] = 'b';
x[0][2] = 'c';
x[1][0] = 'd';
x[1][1] = 'e';
x[1][2] = 'f';
```

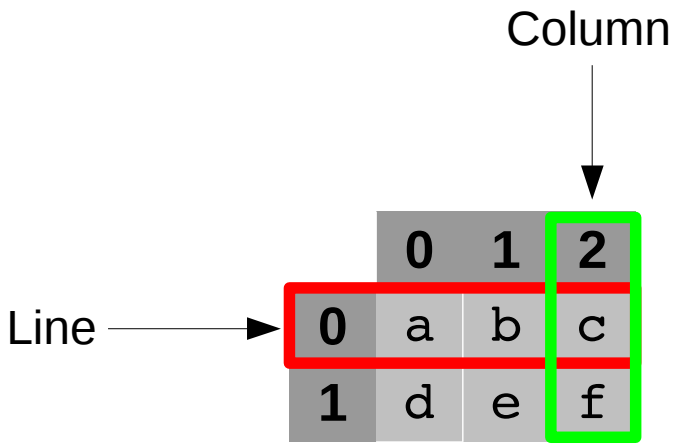


2D Arrays in C

- Writing to array elements (cont.):

C code excerpt:

```
char x[2][3];
...
x[0] = {'a','b','c'};
x[1] = {'d','e','f'};
```



C code excerpt:

```
char x[2][3];
...
x[0] = “abc”;
x[1] = “def”;
```

C strings imply an **extra char** for the “**\0**”!
In the above case, `x` should be declared as:

```
char x[2][4];
```

2D Arrays in C

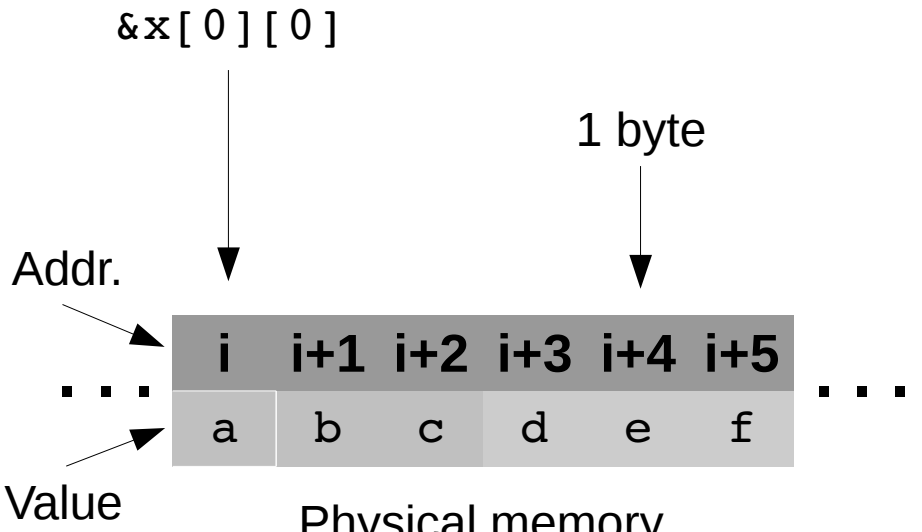
- Actual array storage pattern in memory:

C code excerpt:

```
char x[2][3];
...
x[0] = {'a','b','c'};
x[1] = {'d','e','f'};
```

	0	1	2
0	a	b	c
1	d	e	f

Logical array arrangement.



Physical memory.

All *n*-dimensional arrays are linearized in physical memory!

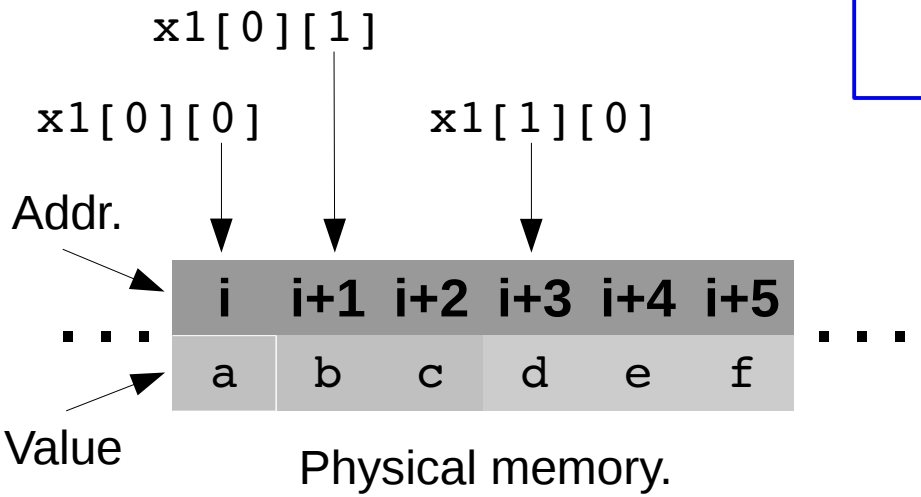
Simulating 2D Array with Pointers

C code excerpt (array):

```
char x1[2][3];
```

3 bytes offset.

1 byte offset.



C code excerpt (pointer-based simulation):

```
char (*x2)[3];

int row, col;
…
*(*(x2 + row) + col) = 'a';
```

`x2` is a pointer to an array (with 3 elements) of **`char`**.

C code example...
(ptr2.c)

Pointers and Structures

- Suppose the following structure:

C code excerpt:

```
struct person {
    char name[20];
    int age;
    float weight;
};

struct person a;

...

printf("%s", a.name);
```

How to **print** the **name** field?

C code excerpt:

```
struct person *b;

...

printf("%s", (*b).name);

printf("%s", b->name);
```

How to **print** the **name** field?

The **same** as '**(*b).**'

Function Parameters

- **Parameters** are passed by **value** in C.

C code excerpt:

```
void function1(int a)
{
    a = a + 1;
};
...
int age = 15;

function1(age);

printf("%i", age);
```

15

C code excerpt:

```
void function1(int *a)
{
    *a = *a + 1;
};
...
int age = 15;

function1(&age);

printf("%i", age);
```

Parameter **a** is still a **copy!!!**

16