Computer Architecture

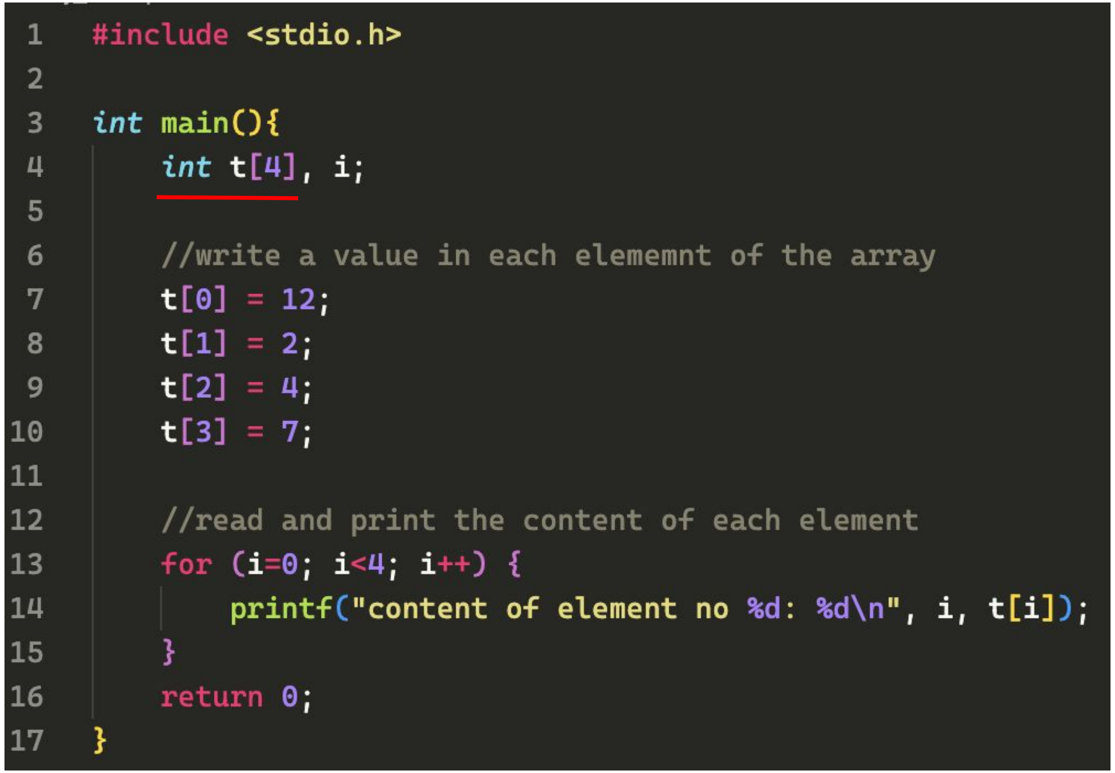
Programming in C

Arrays and strings

Agenda

- Array
- String

Declaring an array



```c
#include <stdio.h>

int main(){
    int t[4], i;

    //write a value in each elememnt of the array
    t[0] = 12;
    t[1] = 2;
    t[2] = 4;
    t[3] = 7;

    //read and print the content of each element
    for (i=0; i<4; i++) {
        printf("content of element no %d: %d\n", i, t[i]);
    }
    return 0;
}
```

`int t[4]` ⇒ allocates a piece of memory sufficiently large to contain four integers

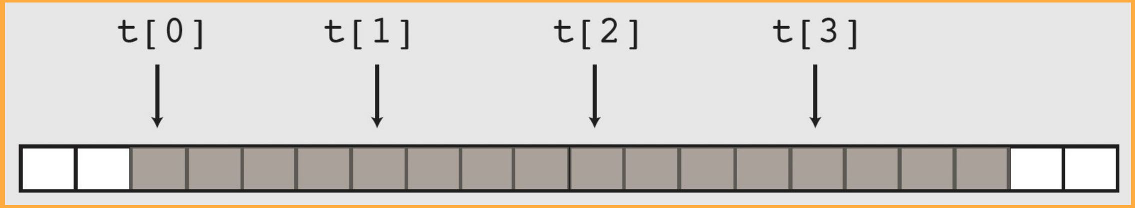
- 4 x 4 bytes = 16 bytes = 4 x 32 bits = 128 bits

We can access the elements in an array by its index:

- e.g., `t[i]`

Declaring an array

- The four elements of the array of integers are accessed in memory in the following way:

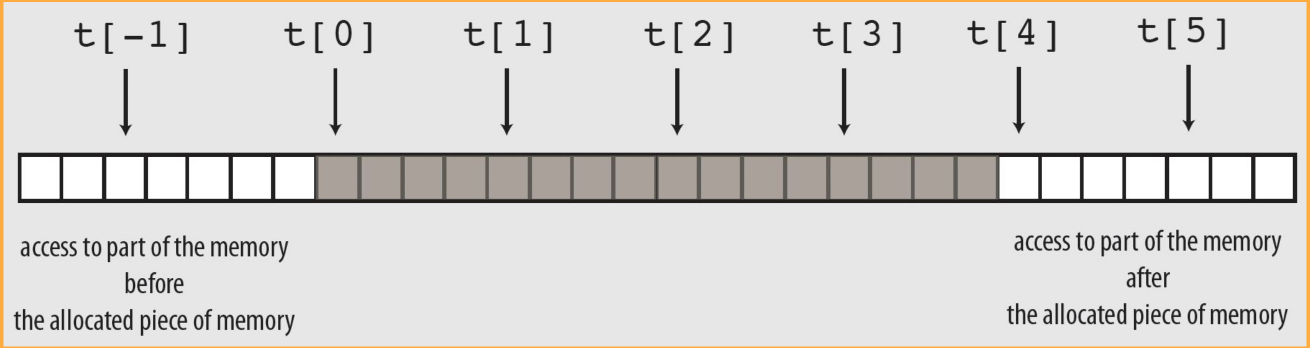


- Each `t[i]` for `i = 0, 1, 2, 3` is then treated as a variable of type integer.

Note: the first element of the array is indexed by 0 rather than 1.

Arrays

- **Warning:** *no verification* is made that you are accessing an element allocated in memory during the declaration of the array.


t[-1]
t[0]
t[1]
t[2]
t[3]
t[4]
t[5]
access to part of the memory before the allocated piece of memory
access to part of the memory after the allocated piece of memory

So, if you defined a `int t[4]` but call `t[5]` (or `t[-1]`) in your code, the compiler will **not** raise errors for it.

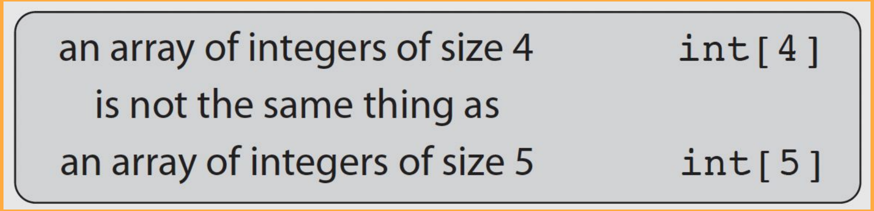
- It may lead the program to:
  - read or write another variable of the program
  - or have access to some protected part of the memory ⇒ this case will crash the program and the OS will raise a signal: `Segmentation fault (core dumped)`

Immediate initialization of an array

- `int t[4]` ⇒ the values of the four integer variables are not determined
- `int t[] = {12, 2, 4, 7}` ⇒ the number 4 of elements of the array is deduced
- `int t[4] = {12, 2}` ⇒ the array contains 4 elements with
    - `t[0]` initialized to 12
    - `t[1]` initialized to 2
    - `t[2]` initialized to 0
    - `t[3]` initialized to 0
- `int t[100] = {}` ⇒ the array contains 100 elements, all initialized to 0.

Arrays: an usual kind of <variable> …

- It is **not** possible to re-assign the value of an array:
  - `t = …` //cannot be compiled
- To compare two arrays `t`, and `u`, you cannot use `t==u;` You have to compare them element by element. (e.g., `t[i]==u[i]`)
- It is impossible to alter the size of an array after declaring it; such kind of changes mean altering its type:



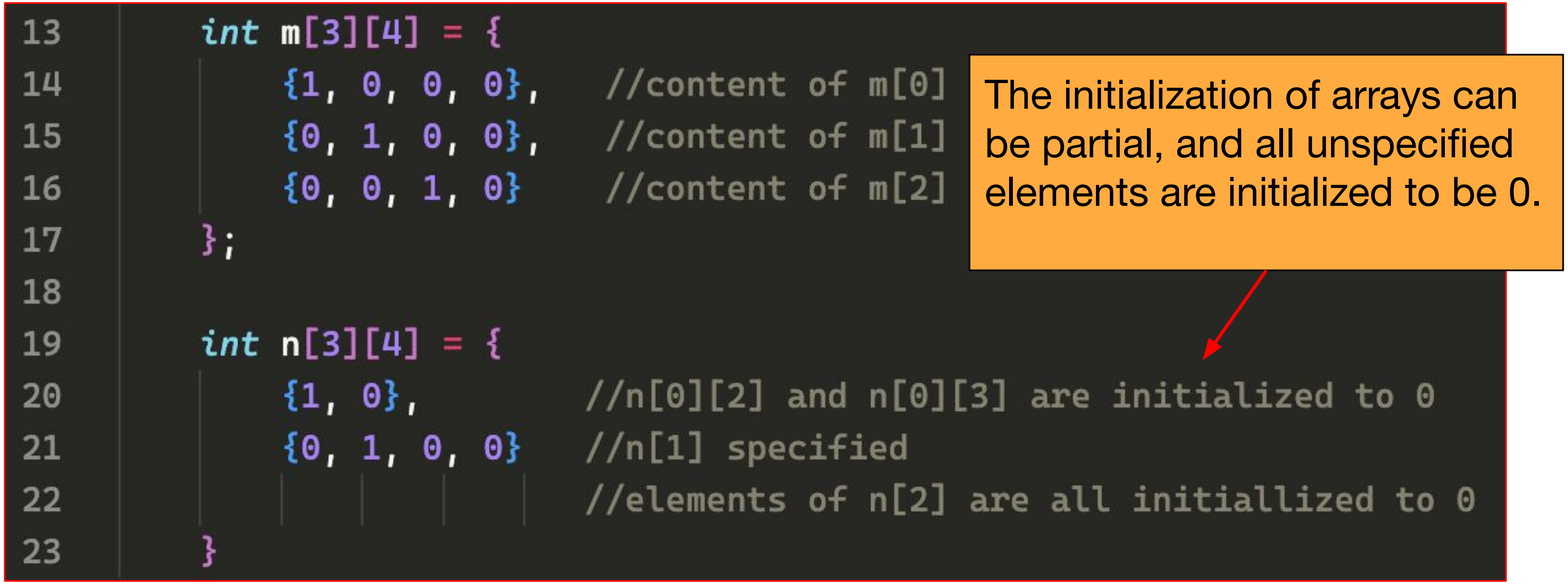
an array of integers of size 4 `int[4]`
is not the same thing as
an array of integers of size 5 `int[5]`

Higher-dimensional arrays

- `int m[3][4]` ⇒ it declares an **array of arrays** of integers of 4 elements.
  - More precisely, it declares an array of 3 elements where each element is itself an array of integers of 4 elements;
    - m[0] is a first array of four elements
    - m[1] is a second array of four elements
    - m[2] is a third array of four elements
  - The declaration allocates a piece of memory containing:

    3 x 4 x 4 bytes = 48 bytes = 12 x 32 bits

Immediate initialization

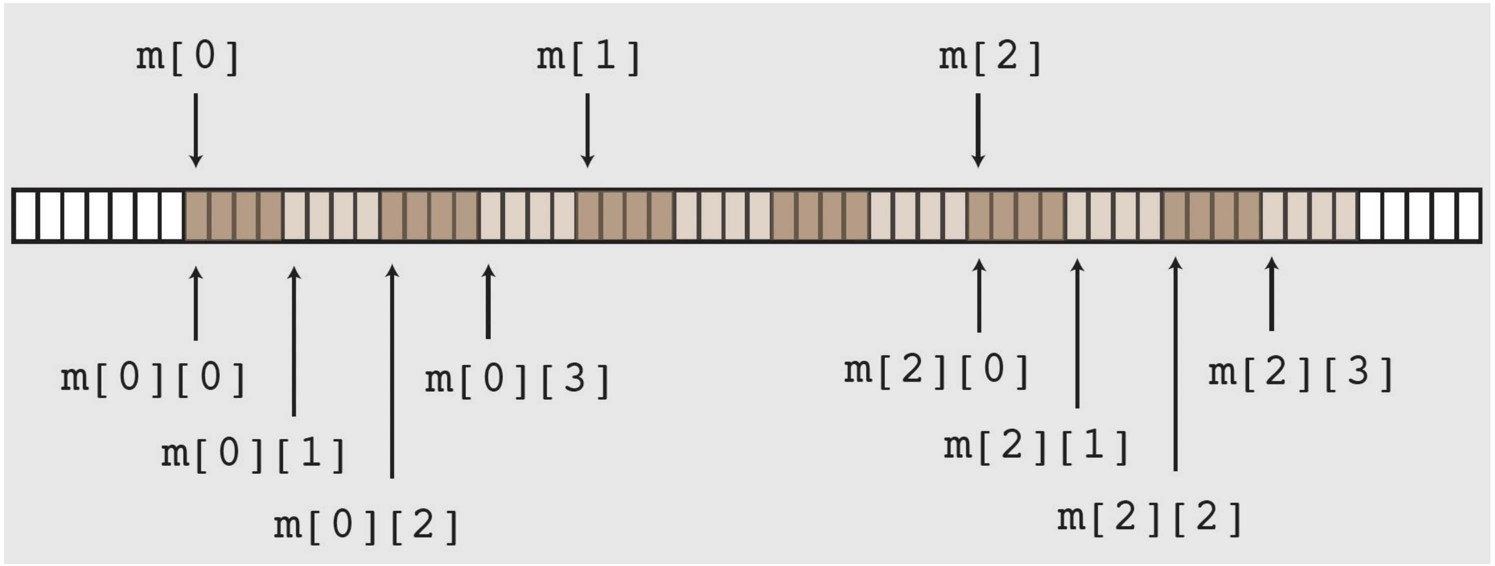


```
    int m[3][4] = {
        {1, 0, 0, 0},   //content of m[0]
        {0, 1, 0, 0},   //content of m[1]
        {0, 0, 1, 0}    //content of m[2]
    };

    int n[3][4] = {
        {1, 0},        //n[0][2] and n[0][3] are initialized to 0
        {0, 1, 0, 0}   //n[1] specified
                       //elements of n[2] are all initiallized to 0
    }
```

The initialization of arrays can be partial, and all unspecified elements are initialized to be 0.

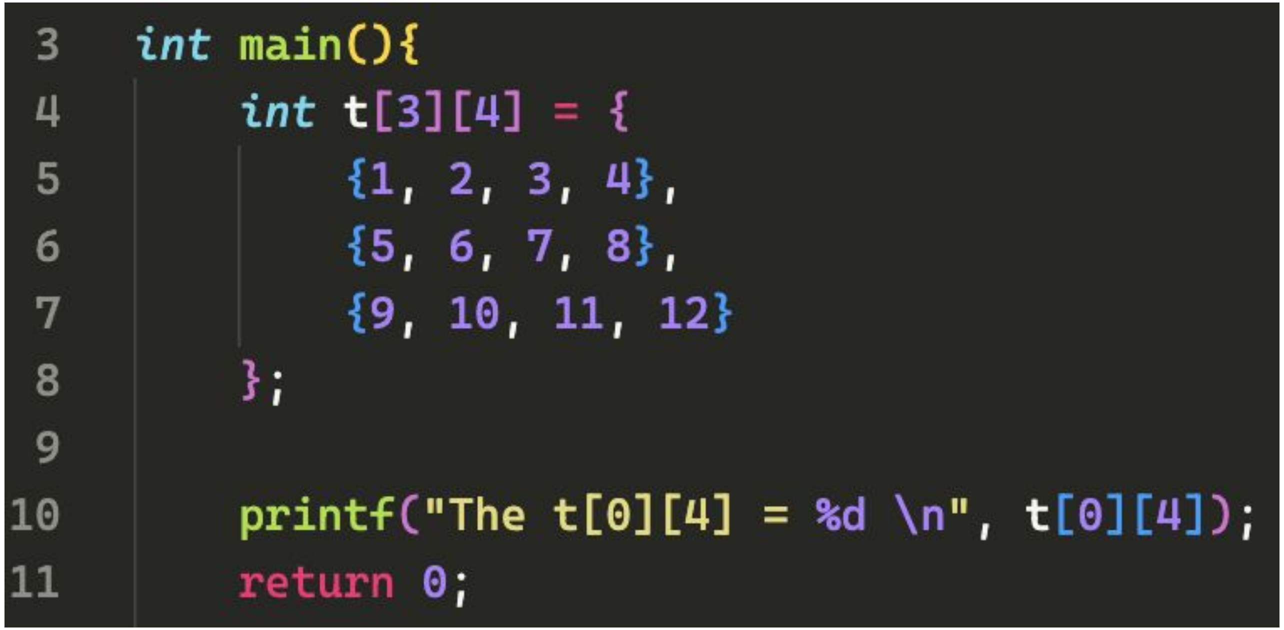
Higher-dimensional arrays: organisation of the memory



Warning: again, no verification of the indices …

Exercise

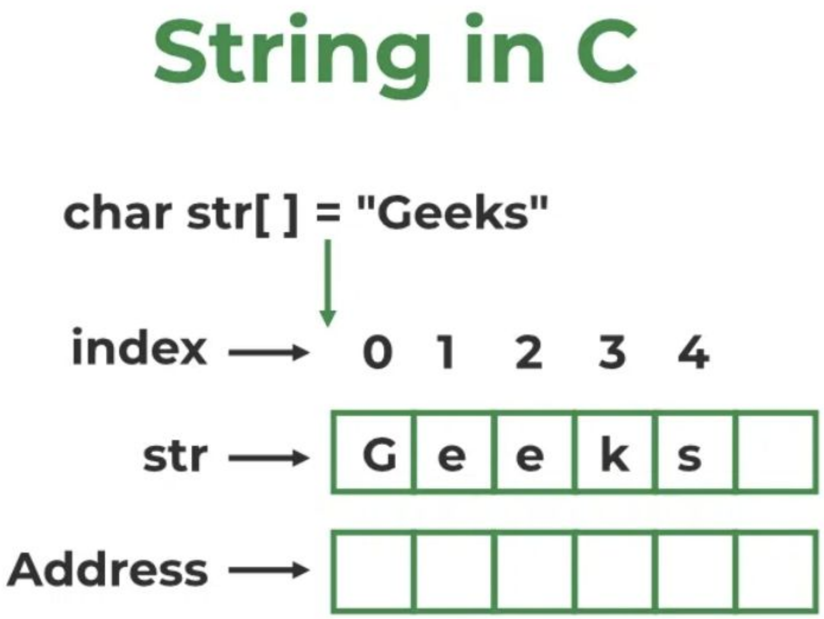
What will be printed on the screen with the following code?



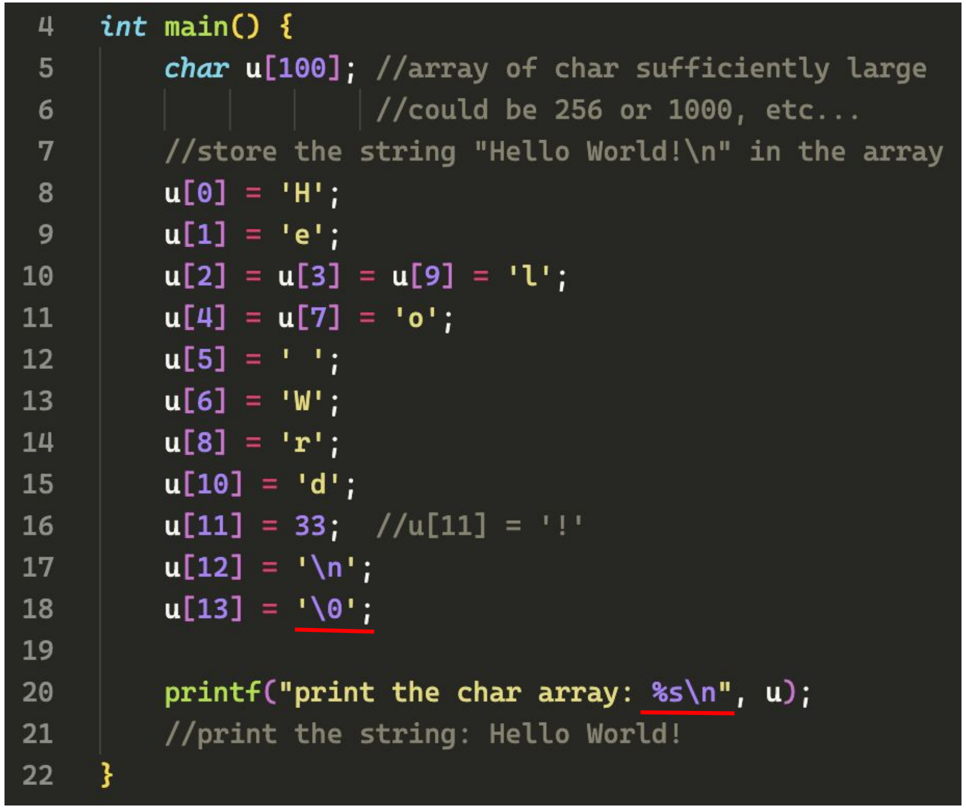
```c
int main(){
    int t[3][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };

    printf("The t[0][4] = %d \n", t[0][4]);
    return 0;
```

Character arrays



Character arrays and strings



```c
int main() {
    char u[100]; //array of char sufficiently large
                 //could be 256 or 1000, etc...
    //store the string "Hello World!\n" in the array
    u[0] = 'H';
    u[1] = 'e';
    u[2] = u[3] = u[9] = 'l';
    u[4] = u[7] = 'o';
    u[5] = ' ';
    u[6] = 'W';
    u[8] = 'r';
    u[10] = 'd';
    u[11] = 33;  //u[11] = '!'
    u[12] = '\n';
    u[13] = '\0';

    printf("print the char array: %s\n", u);
    //print the string: Hello World!
}
```

In C, a string is an array of char type elements.

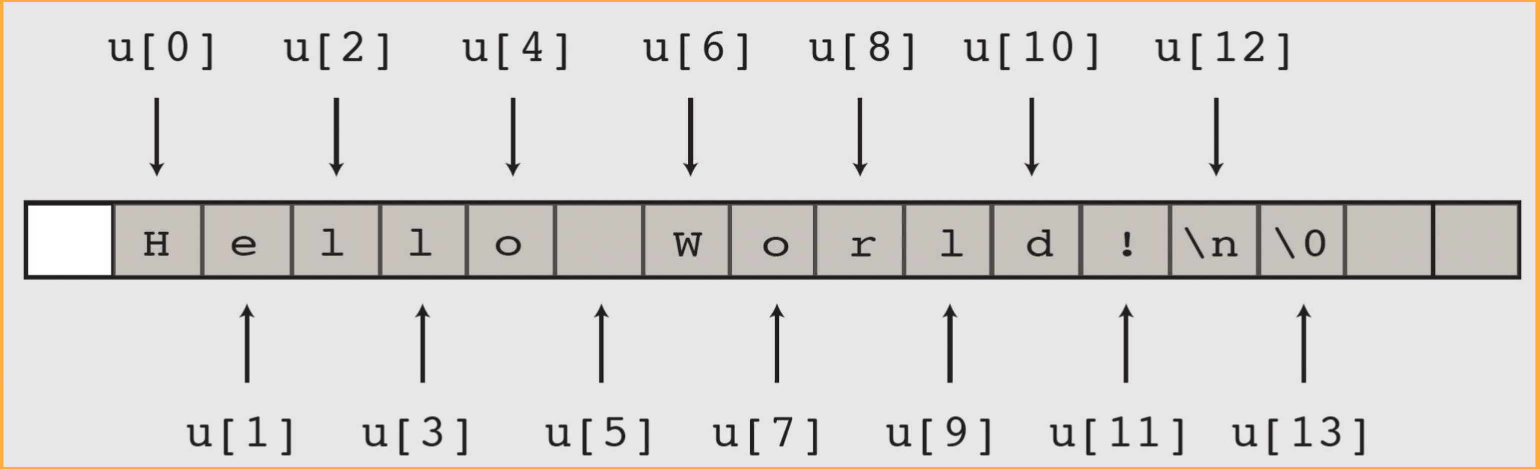
But it has an format identify `%s`, which will treat the array of char elements as a string.

'\0' is called null character. It indicates the end of a string.

Note: in C, a character is wrapped by single quotes ''; a string is wrapped by double quotes "".

Character arrays and strings

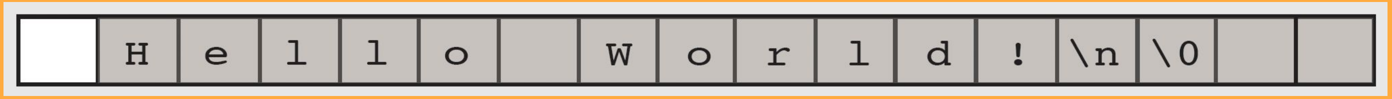
- Location of the character array `u` in memory



Remark: this should be compared with the representation of an array of integers.

Character arrays and strings

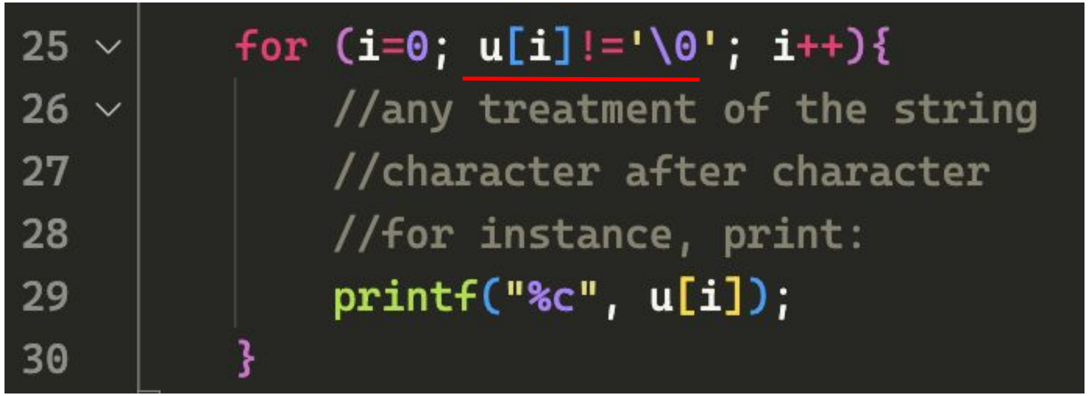
- A string of characters is represented as a character array.
  - The end of the string is indicated by a character `\0` in the array.
  - The character `\0` is called the <null character> and its ASCII value is 0.
  - The null character is not printed on the screen.



```
printf("%s", u)
```

- Inserts at the `%s` all the characters in the array between `u[0]` and the first encountered element with value `\0` in the array.
- The result may be quite surprising when the array `u` is not initialized!!!

Character arrays and strings



```
    for (i=0; u[i]!='\0'; i++){
        //any treatment of the string
        //character after character
        //for instance, print:
        printf("%c", u[i]);
    }
```

We often use a `for` loop to process each character in the string

Immediate initialization of a character array by a string of characters

- `char u[100] = “abc”;//the double quotes wraps the string`

The character array `u` is allocated with a size of 100 elements, but

- `u[0]` is initialized to `‘a’`
- `u[1]` is initialized to `‘b’`
- `u[2]` is initialized to `‘c’`
- **All the other** elements of the array are initialized to the value `‘\0’`
- In particular, the element `u[3]` is initialized `‘\0’` and thus indicates the end of the string characters.

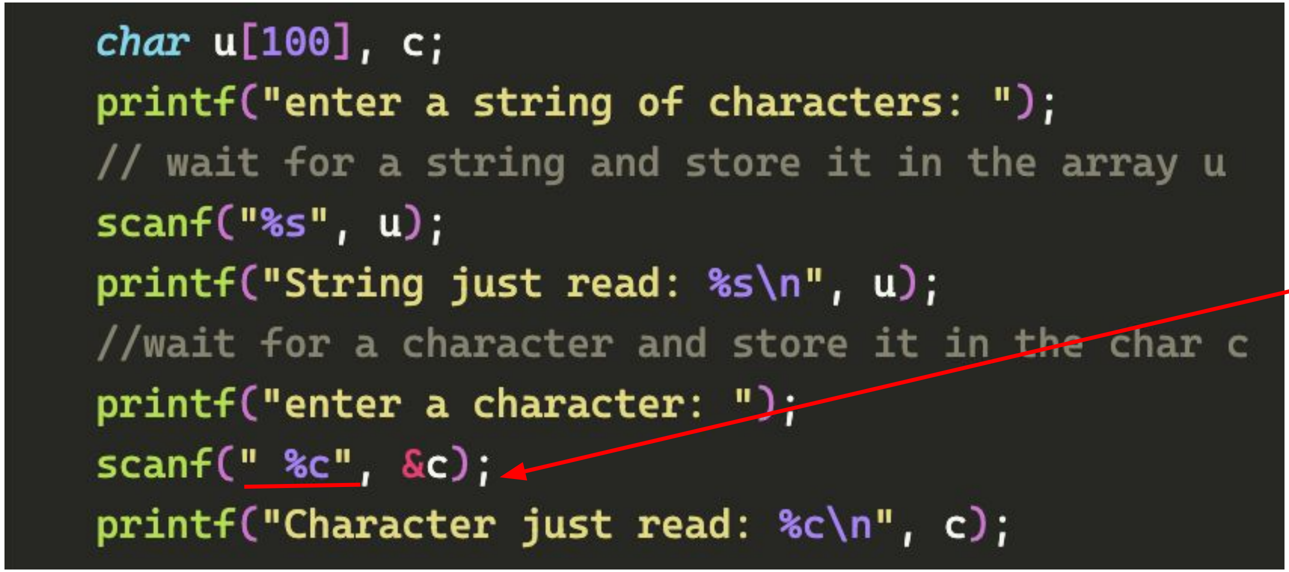
Immediate initialization of a character array by a string of characters

- `char u[] = “abc”;`

The character array `u` is allocated just with the appropriate size, which means in that case with a size of 4 = 3 + 1 elements.

- `u[0]` is initialized to `‘a’`
- `u[1]` is initialized to `‘b’`
- `u[2]` is initialized to `‘c’`
- `u[3]` is initialized `‘\0’` to indicate the end of the string.

Read a string from the keyboard



```
char u[100], c;
printf("enter a string of characters: ");
// wait for a string and store it in the array u
scanf("%s", u);
printf("String just read: %s\n", u);
//wait for a character and store it in the char c
printf("enter a character: ");
scanf(" %c", &c);
printf("Character just read: %c\n", c);
```

A space in front of the 1st argument is needed for the 2nd `scanf` call. The space is crucial, it instructs scanf to skip any white space (like new line) that might be left in the input buffer.

Note: there is no `&` in `scanf("%s", u)` because `u` already denotes the address of the array.

Read a string from the keyboard

```c
char u[100];

printf("Enter a string of characters : ");

// wait for a string and store it in the array u
scanf("%s",u);

printf("String just read : %s\n", u);
```

```
Enter a string of characters : Marvelous afternoon
String just read : Marvelous
```

The `scanf` instruction:

- starts from the first character different from the space character
- stops at the first encountered space (and at the end of the string otherwise)
- adds a null character at the end of the string
- and throws an abort exception when the array is not sufficiently large…

Exercise 1

Write a program which asks the user to enter a string and then computes and prints the length of the string.

Exercise 2

Write a program which asks the user a string as well as a character and then computes and prints the number of times this character occurs in the string.

Exercise 3

Write a program which asks the user to enter a string of characters and then writes the string in the reverse order!

Note: the purpose of the program is not to reverse the string of characters but simply write it in the reverse order.

Analysis

Since the task of the program is only to print in reverse order, we can simply,

- go to the end of the string and then
- read the string backwards and print each character encountered.

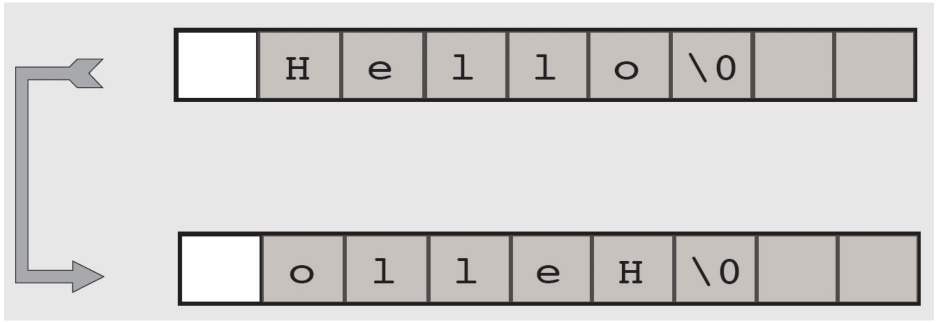
What we will need:

- the character array itself
- a loop with counter i to count the number of characters until one reaches the first null character in the array u
- another loop to come back to the beginning of the string.

Exercise 4

Write a program which asks the user to enter a string of characters then reverse the string and prints it!

Note: this time, the purpose of the program is to reverse the string itself:



Exercise 5

The left rotation of a non-empty string is the string obtained by shifting to its first character to the end of the string.

abcde	⇒	bcdea
ab	⇒	ba
a	⇒	a

Write a program which:

- reads a string character
- stores it in a character array u
- and if the string is not empty, transforms it (inside the same array) into its left rotation
- prints the resulting string in the end.

Exercise 6

More difficult:

- Write a program which asks user to enter a text and then computes (and prints) the number of words in the string.

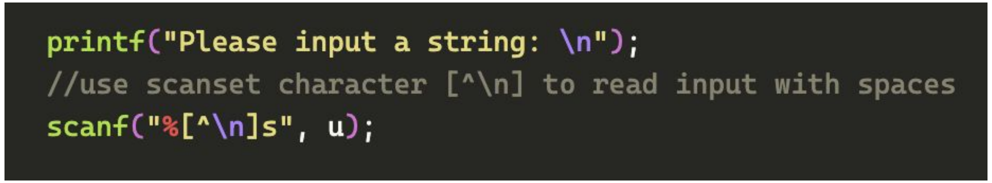
Conventions:

- the words are separated by space
- there can be several spaces between words
- there can be several spaces at the beginning of the string
- the string can be empty or contain only spaces

Additional rule:

- the string can be read only once

Hint: using [^\n] to read input with spaces



```
printf("Please input a string: \n");
//use scanset character [^\n] to read input with spaces
scanf("%[^\n]s", u);
```

Analysis

Every position i in the string u can be:

- either on a word
- or on a space

In the array u, an index i contains the last letter of a word precisely when

- `u[i]` does not contain a space character (hence it is on a word)
- `u[i+1]` contains: either a space character `' '` or the null character `'\0'`

It is thus sufficient to count the last letters of a word.

Exercise 7

The right rotation of a non-empty string is the string obtained by shifting to its first character to the end of the string.

abcde	⟹	eabcd
ab	⟹	ba
a	⟹	a

Write a program which:

- reads a string character
- stores it in a character array u
- and if the string is not empty, transforms it (inside the same array) into its left rotation
- prints the resulting string in the end.

Analysis

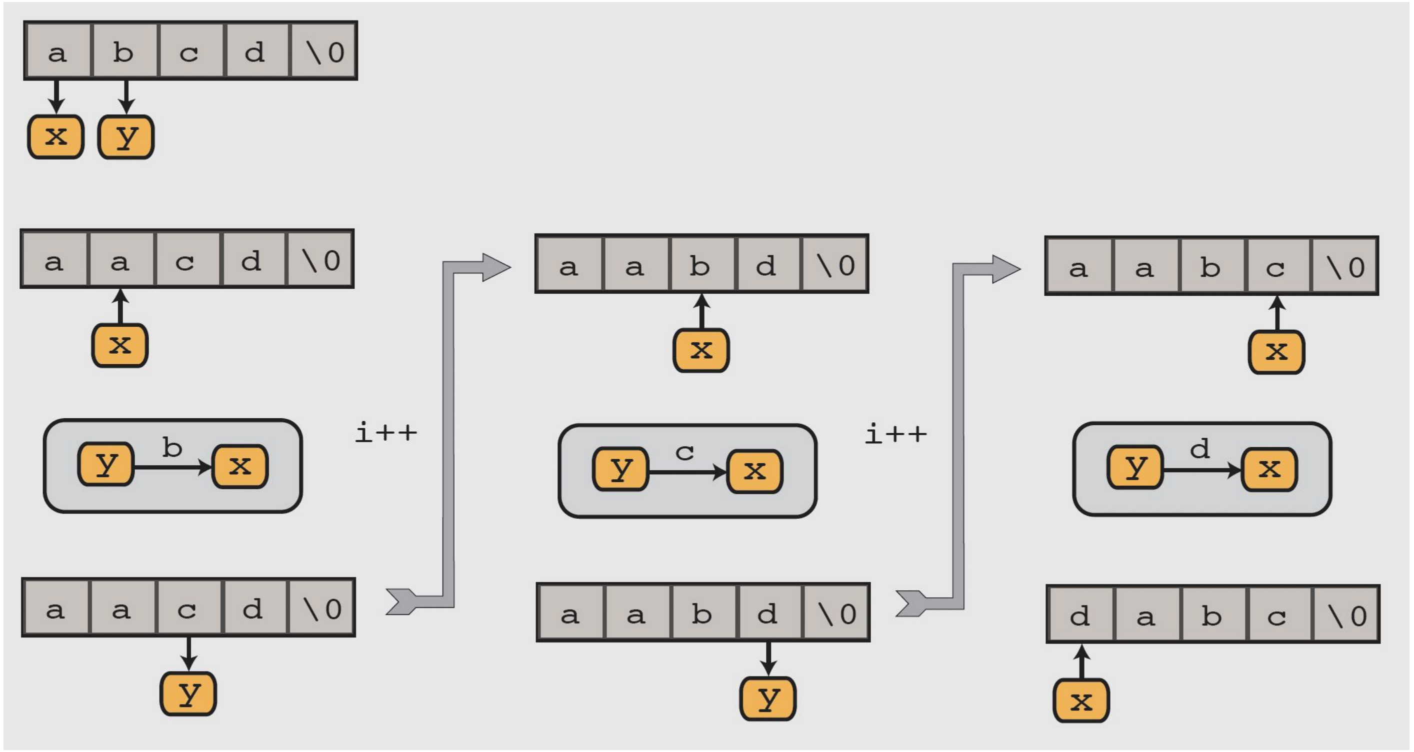
- Every character in the string should be shifted to the right

Beware: If you are not careful, each character will overwrite the next one and the second character b will be lost!

- How much does one need to remember during the loop?
- And on which order should one proceed?
- This is much more subtle than for the left rotation!

Hint: one needs two variables for keeping track of the relevant information.

Analysis



a b c d \0
x
y
a a c d \0
x
y b x
a a c d \0
y
i++
a a b d \0
x
y c x
a a b d \0
y
i++
a a b c \0
x
y d x
d a b c \0
x

Exercise 8

The purpose of this exercise is to select randomly a character in a string without repetition like “abcdef”.

Three main constraints:

- The choice should be fair in the sense that every character in the string should have the same probability to be chosen
- The string should be read only once
- The program should not start by computing the length of the string

Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
// select a random character in a string

int main() {
  int i, selected;
  char u[100]="abc";

  srand((unsigned)time(NULL));

  if (u[0] == '\0'){printf("Your string is empty");}
  else{
    selected = 0;                     // start by selection element 0
    for (i=1; u[i] != '\0' ; i++){
      if (rand() % (i + 1) == 0){     // pick a number between 0 and i
        selected=i;                   // if zero select this element i
      };
    }
    printf("The selected letter is %c at position %d\n", u[selected], selected+1);
  }
  return 0;
}
```

The scanf function

The input/output devices are treated just as files by UNIX.

In particular, among the UNIX files, one finds:

- The standard input (stdin) ⇒ file number 0
- The standard output (stdout) ⇒ file number 1
- The standard error (stderr) ⇒ file number 2

The function scanf reads the characters from the standard input. It consumes the characters one after the other in the order in which they were entered.

Each time the function scanf attempts to consume a character and the standard input is empty:

- The function scanf returns to the shell
- The program carries on its execution the first time the return key has been pressed

The scanf function

The scanf function is not very safe but it is possible to know when it fails:

The instruction `scanf(“%d %d %d”, &i, &j, &k)` is at the same time an expression of type (signed) integer!

The value of this expression is equal to:

- in case of success: the number of integer variables read from the standard input
- in case of failure: the signed integer EOF=-1