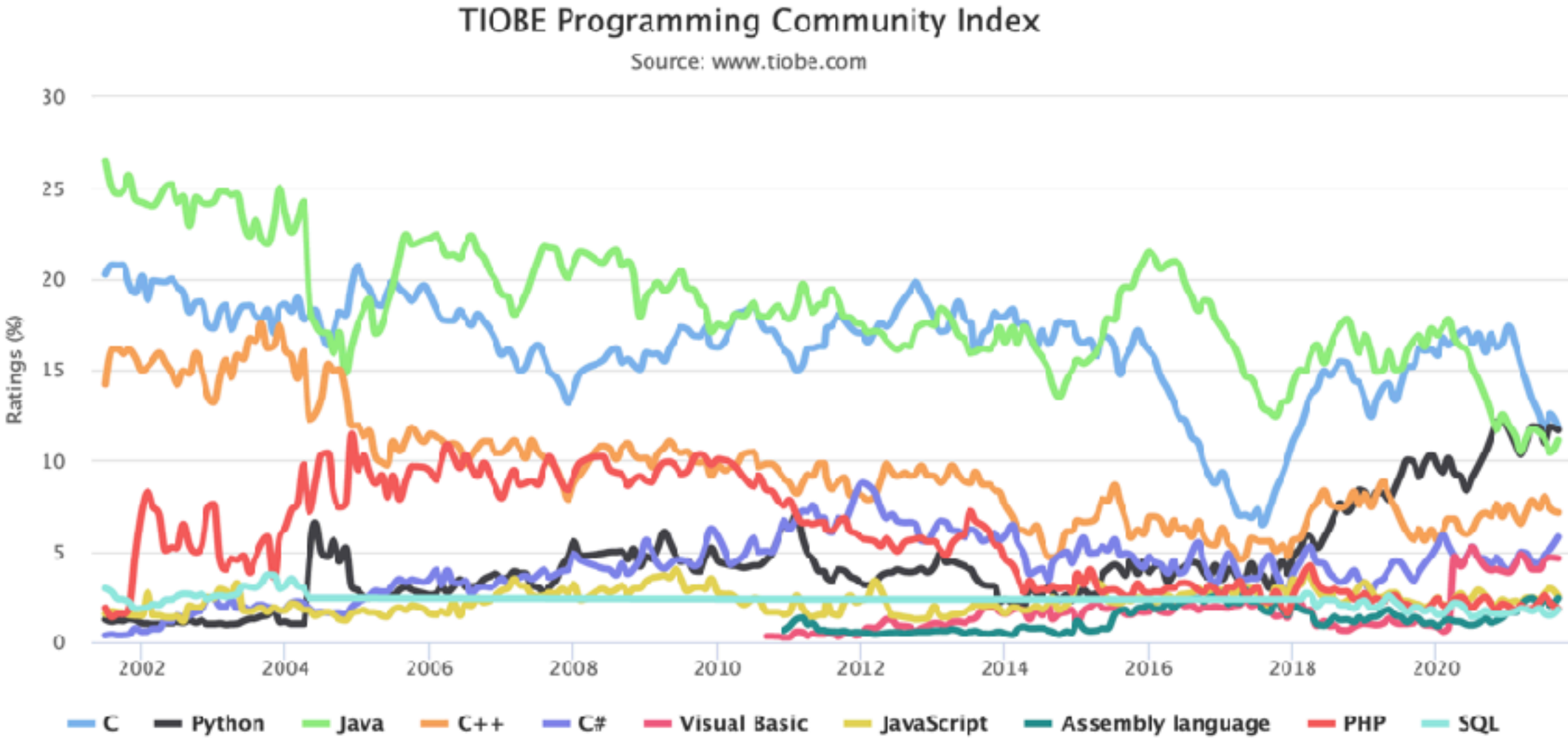
Introduction to C

Olivier Mattelaer
UCLouvain
CP3 & CISM

Why C

Tiobe Ranking



- C is #1 (even if it drops in absolute value)
  - ➡ 4 of the 5 top program are C related
  - ➡ Including Python and C++
- C is also useful for Cuda

Program of today

- Basic of C
  - ➡ Type of language
  - ➡ Hello World
  - ➡ Variable
    - ✦ Type of variable
  - ➡ Arrays
  - ➡ Pointers
  - ➡ Functions
  - ➡ Conditional
  - ➡ Data structure
  - ➡ Dynamical memory

This Afternoon

- Basic of C++
  - Introduction to Class/object in C++
  - (Multi) Inheritance

C language

C (programming language)

From Wikipedia, the free encyclopedia

"C Programming Language" redirects here. For the book, see The C Programming Language.

C (/siː/, as in the letter *c*) is a general-purpose, procedural computer programming language supporting structured programming, lexical variable scope, and recursion, with a static type system. By design, C provides constructs that map efficiently to typical machine instructions. It has found lasting use in applications previously coded in assembly language.

- C is quite low level language
  - ➡ Allows to generate very efficient machine code
    - ✦ Efficiency of the code depends of the language but also of the algorithm

Hello World

```
1  // my first program in C
2  #include <stdio.h>
3
4  int main()
5  {
6    printf("Hello World!\n");
7  }
```

http://cpp.sh/3okrv

- line1: Comment
  - ➡ also /* … */
- line 2: preprocessor directive:
  - ➡ Include a section of standard C code in the code
- line 3: empty line: do nothing (but clarity for human reader)
- line 4: declaration of a function
  - ➡ main is a special function which is run automatically
  - ➡ starts and stops with the braces (line 5 and 7)
- Statement. Send character to the output device
  - ➡ Note the **semi-column** at the end of the line

Compile the code

C++

Simplest command

Make FILENAME_NO_EXT

- Make is NOT a compiler but a program that knows how to compile
  - ➡ No extension to FILENAME

Calling the compiler:

cc -o EXECNAME input.c

- Convention to call c code with .c
- On cluster: "module load foss"

Problem

https://ideone.com/

Select C (bottom left)

http://www.cpp.sh/2dd

https://www.tutorialspoint.com/compile_c_online.php

Run the code

./EXECNAME

Simple code print multiplication table

http://cpp.sh/4odwq

```c
// my first program in C
#include <stdio.h>

int main()
{

  printf("Multiplication table of 5:\n");
  printf(" 5 * 1 = 5 \n");
  printf(" 5 * 2 = 10 \n");
  printf(" 5 * 3 = 15 \n");
  printf(" 5 * 4 = 20 \n");
  printf(" 5 * 5 = 25 \n");
  printf(" 5 * 6 = 30 \n");
  printf(" 5 * 7 = 35 \n");
  printf(" 5 * 8 = 40 \n");
  printf(" 5 * 9 = 45 \n");
  printf(" 5 * 10 = 50 \n");
}
```

- What's wrong with this code?
  - ➡ Maintainability

Tarball: multiplication_table.c

Variable

http://cpp.sh/522d2

```c
// my first program in C
#include <stdio.h>

int main()
{

  int i = 5;
  printf("Multiplication table of %d:\n", i);
  printf(" %d * 1 = %d \n", i, i);
  printf(" %d * 2 = %d \n", i, 2*i);
  printf(" %d * 3 = %d \n", i, 3*i);
  printf(" %d * 4 = %d \n", i, 4*i);
  printf(" %d * 5 = %d \n", i, 5*i);
  printf(" %d * 6 = %d \n", i, 6*i);
  printf(" %d * 7 = %d \n", i, 7*i);
  printf(" %d * 8 = %d \n", i, 8*i);
  printf(" %d * 9 = %d \n", i, 9*i);
  printf(" %d * 10 = %d \n", i, 10*i);
}
```

- Make "5" a parameter
  - ➡ Abstract the code for any value

```c
int i = 5;
```

- Note that
  - I say that this is an integer
  - That it's (initial) value is 5

Tarball: multiplication_table_var.c

While loop

```
1  // my first program in C
2  #include <stdio.h>
3
4  int main()
5  {
6
7    int i = 5 ;
8    printf("Multiplication table of %d:\n", i);
9    int j=1;
10   while(j<11){
11         printf(" %d * %d = %d \n", i,j, i*j);
12         j = j +1;
13
14   }
15
16 }
```

cpp.sh/9dn5g

- Spaces are not important (line9)
  - ➡ "=" is the assignment operation not a mathematical operation
  - ➡ "j" will change value while looping (line 10-14)

Tarball: multiplication_table_while.c

For loop

cpp.sh/75vpk

```c
// my first program in C
#include <stdio.h>

int main()
{

  int i = 5.;
  printf("Multiplication table of %d:\n", i);
  for (int j=0; j<10; j++){
            printf(" %d * %d = %d \n", i,j+1, i*(j+1));
  }

}
```

- j++: means “add one to the value of j”
- Quite common to count from 0 in C
- “j” is not defined outside the loop !!
  - ➡ Variable have “scope” (limited range)
    - ✦ Nice for code scalability

Tarball: multiplication_table_for.c

Loop

- For (int i=0; i< …; i++) {}
- while(condition) {code}
- Do{ code }while(condition);

Loop special keyword

- continue
  - Go to the next step in loop (bypass any following lines in the loop for this step)
- break
  - Stop the loop (resume main code)

Variable

```c
// my first program in C
#include <stdio.h>

int main()
{

  int i = 5;
  float x=1.0;
  double c =1.0;
  char a = 'h';
```

- No type for string
  - ➡ But wait for it
- Boolean supported since 99
  - ➡ Requires “#include stdbool.h”

```c
printf("How to print: %d %c %f %f:\n", i,a,x,c);
```

- Note you can not define twice the same variable name
- Variable name have a “scope”, only available locally

Tarball: variable_demo.c

Tarball: multiplication_table_fct.c

Functions

```c
// my first program in C
#include <stdio.h>

void print_table(int tableof, int maxmul){

  for(int j=1; j<(maxmul+1); j++){
      printf("the product of %d and %d is %d\n", tableof, j, tableof*j);
      }

}


int main()
{
 print_table(4,10);
 print_table(5,11);

}
```

cpp.sh/24uno

- Function allows to reuse a piece of code with argument
- Other variable are not passed to the function
  - ➡ You can define a variable with the same name in both block. They will not conflict and not share the value
- Argument are not modified by the function

http://cpp.sh/3ssg5

What if I want to change a variable via a function?

- That's where the address/pointer are useful

cpp.sh/2pqel

Address

```c
int i = 5;
```

- A variable contains a value
  - ➡ That value can change with time
  - ➡ That value is store on RAM at a given place
  - ➡ This place is called the “address” of the variable

http://cpp.sh/932uo

```c
// my first program in C
#include <stdio.h>

int main()
{

  int i = 5;
  printf("Multiplication table of %d:\n", i);
  printf("i is store in ram at adress %p", &i);
}
```

```
Multiplication table of 5:
i is store in ram at adress 0x7078f5bfb7bc
```

Tarball: address.c

Address

```
int i = 5;
```

- A variable contains a value
  - ➡ That value can change with time
  - ➡ That value is store on RAM at a given place
  - ➡ This place is called the “address” of the variable
- Seems a useless concept
  - ➡ The place in RAM is not predictable
- Useful because you can ask to change a value at a given address
  - ➡ i = 5 : change the content of the variable i
    - ✦ Replace a book by a new one
  - ➡ *address = 5 : change the content at a given RAM position
    - ✦ Replace the book which is on a given shelve

Can I store the address in a variable?

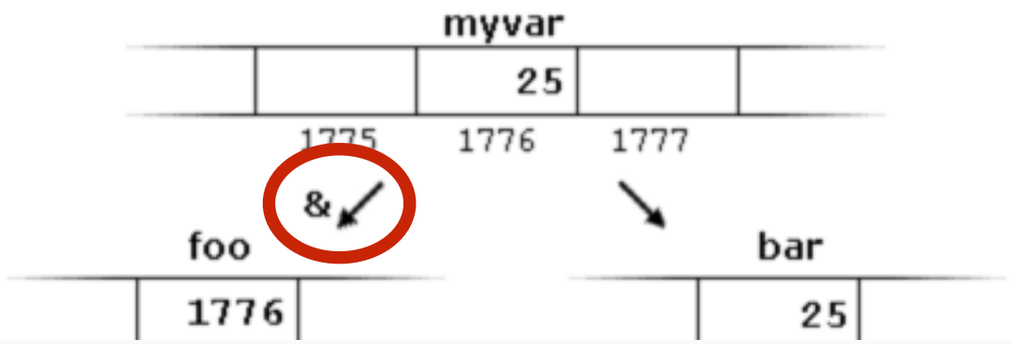
- Yes you can store the address.
  - ➡ As C is strongly typed, you have a type for that

```
int* pi = &i;
printf("i=%d is store in ram at adress %p\n", i, pi);
```

  - ➡ Each type of numbers have various size (number of bit) in memory, so we have a type of address for any type of value.
  - ➡ This is call pointer.
  - ➡ “easy syntax”: add a “*” to the name of the original type
    - ✦ float*, bool*, char*
- Possible to get the value associate to a pointer:
  - ✦ *pi
- Change the value store at a given adress
  - ✦ *pi = 2

Basic of pointer

Pointer = variable that give the position in memory of the variable




```c
// my first program in C
#include <stdio.h>

int main()
{

  int i = 5;   // i is a normal variable
  int* p_i = &i; // p_i is a pointer of an integer (int*) which is assigned as initial value the address of i

  int j = *p_i; // j is an integer which takes as initial value the value stored in the adress of p_i
  // so here this is a complex way to write int j = i;

  printf("value of j is  %d:\n", j);

  *p_i = 6; // change the value written at address p_i
   printf("value of i  is  %d:\n", i); // i modified since it has address p_i
   printf("value of j  is  %d:\n", j); // j is still on 5 since it has his own adress

   //let's proof that they have the same address
   printf("address of i  is  %p, address of j is %p, p_i is %p:\n", &i, &j,p_i);

}
```

Tarball: pointer.c [cpp.sh/5rbv4](cpp.sh/5rbv4)

What if I want to change a variable via a function?

- That's where the address/pointer are useful

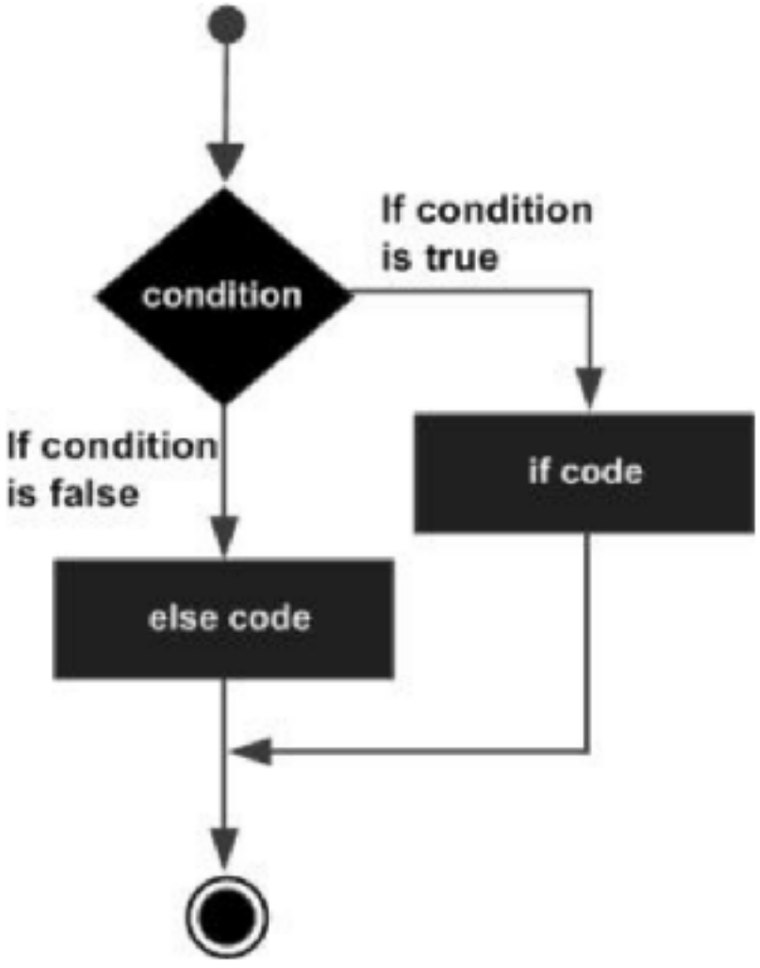
```c
/* function declaration */
void swap(int* x, int* py);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   swap(&a, &b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}

/* function definition to swap the values */
void swap(int* px, int* py) {

   int temp;
   temp = *px;    /* save the value at address px */
   *px = *py;      /* put the value from adress py into
   *py = temp;    /* put temp into adress py */

   return;
}
```

- You can modify what is store at a given memory location

- So you pass the address and modify the value store at that address

cpp.sh/2pqel Tarball: swap_function.c

If statement



- Checking condition and react accordingly is the core of programming

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 100;

   /* check the boolean condition */
   if( a < 20 ) {
      /* if condition is true then print the following */
      printf("a is less than 20\n" );
   } else {
      /* if condition is false then print the following */
      printf("a is not less than 20\n" );
   }

   printf("value of a is : %d\n", a);

   return 0;
}
```

- One liner:

```c
int x = (a>0 ? 2 : 4);
printf("x= %d\n", x);
```

https://www.tutorialspoint.com/compile_c_online.php

and/or operation

- Combining condition is of curse crucial

Operator	Meaning
&&	AND
||	OR
!	NOT

```c
if ( a && b ) {
   printf("Line 1 - Condition is true\n" );
}

if ( a || b ) {
   printf("Line 2 - Condition is true\n" );
}

/* lets change the value of  a and b */
a = 0;
b = 10;

if ( a && b ) {
   printf("Line 3 - Condition is true\n" );
} else {
   printf("Line 3 - Condition is not true\n" );
}

if ( !(a && b) ) {
   printf("Line 4 - Condition is true\n" );
}
```

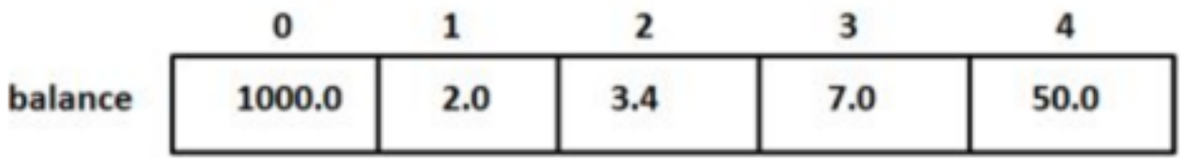
https://www.tutorialspoint.com/compile_c_online.php

Array

- Let's represent a list of number
- The size of an array is fixed!

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

```
balance[4] = 50.0;
```




```c
#include <stdio.h>

int main () {

   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;

   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ ) {
      n[ i ] = i + 100; /* set element at location i to i + 100 */
   }

   /* output each array element's value */
   for (j = 0; j < 10; j++ ) {
      printf("Element[%d] = %d\n", j, n[j] );
   }

   return 0;
}
```

Tarball: array.c

https://www.tutorialspoint.com/compile_c_online.php

Array and function

- Array are actually pointers…
  - ➡ Those two codes are identical

http://tpcg.io/h9ymMaep

```c
float average(int* myarray, int size){

    float average;
    for (int i =0; i<size; i++){
        printf("Element[%d] = %d\n", i, myarray[i] );
        average += myarray[i];
    }
    average /= size;
    return average;
}
```

```c
float average(int myarray[], int size){

    float average;
    for (int i =0; i<size; i++){
        printf("Element[%d] = %d\n", i, myarray[i] );
        average += myarray[i];
    }
    average /= size;
    return average;
}
```

Tarball: array_are_pointer.c

- You can pass a sub-array to a function

```c
printf( "average  from index 5 is %f\n", average(&n[5], 5));
```

Strings

- No native “strings” type
- You can use an array of char

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

```
char greeting[] = "Hello";
```

- A series of functions simplify handling of strings
  - ➡ Via “include <string.h>”

1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.

Data structure

- Can we have a special data-type with metadata
  - ➡ Like a "formation"
    - ✦ With the number of student
    - ✦ The name of the formation
    - ✦ The name of the teacher

```c
struct Formation {
   char  title[50];
   char  speaker[50];
   int   nb_student;
};
```

```c
int main( ) {

   struct Formation Lect_C;
   struct Formation Lect_Cpp;

   /* Formation C  initialization*/
   strcpy( Lect_C.title, "C Programming");
   strcpy( Lect_C.speaker,  "O. Mattelaer");
   Lect_C.nb_student = 10;



   /* print Book1 info */
   printf( " Formation \"%s\" given by \"%s\" has %d student",
           Lect_C.title, Lect_C.speaker, Lect_C.nb_student);
```

http://tpcg.io/umjalDnr

Tarball: data_structure.c

More on Data structure

Tarball: data_structure_pointer.c

- Can be passed to functions
- Can have pointer
  - ➡ Can be modified within function
  - ➡ Note special syntax to access attribute from pointer
    - ✦ “address->attribute_name”
    - ✦ (*address.attribute_name)

```c
struct Formation {
   char  title[50];
   char  speaker[50];
   int   nb_student;
};

void print_stat(struct Formation formation){

    /* print Book1 info */
   printf( " Formation \"%s\" given by \"%s\" has %d student\n",
           formation.title, formation.speaker, formation.nb_student);

}

int main( ) {

   struct Formation Lect_C;
   struct Formation Lect_Cpp;

   /* Formation C  initialization*/
   strcpy( Lect_C.title, "C Programming");
   strcpy( Lect_C.speaker,  "O. Mattelaer");
   Lect_C.nb_student = 10;


   print_stat(Lect_C);
   return 0;
}
```

```c
void print_stat(struct Formation* formation){

    /* print Book1 info */
   printf( " Formation \"%s\" given by \"%s\" has %d student\n",
           formation->title, formation->speaker, formation->nb_student);
    formation->nb_student +=1;

}
```

Dynamical memory

- You do not always know at compile time the size of all your array

```
int* vector;
int size = 3;
vector = malloc(size * sizeof(int));
```

Array of arbitrary size!!

http://tpcg.io/RRBXOAkj

Tarball:dynamical_memory.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {


    int* vector;
    int size = 3;
    vector = malloc(size * sizeof(int));

    if( vector == NULL ) {
      fprintf(stderr, "Error - unable to allocate required memory\n");
      return 1;
   }
    vector[0] = 1;
    vector[1] = 2;
    vector[2] = 3;

    int i =4;
    if(i<3){
        size +=1;
        vector = realloc( vector, size * sizeof(char) );
        if( vector == NULL ) {
            fprintf(stderr, "Error - unable to allocate required memory\n");
            return 1;
        }
        vector[3] = 4;
    }
    printf("size is %d\n", size);
    for(int j=0; j<size; j++){
        printf("%d ", vector[j]);

    }
    free(vector);

}
```

Conclusion

- You need to play with it
  - ➡ Coding is learning by exercise/exploration
  - ➡ Read book on coding style
    - ✦ How to present you code (space/comment/indentation)
    - ✦ Type of good structure/…
- Good understanding of C is key since it defines the basic notion for many language (including Python)
- A lot of this is to learn syntaxes but not only
  - ➡ You need to understand the abstraction