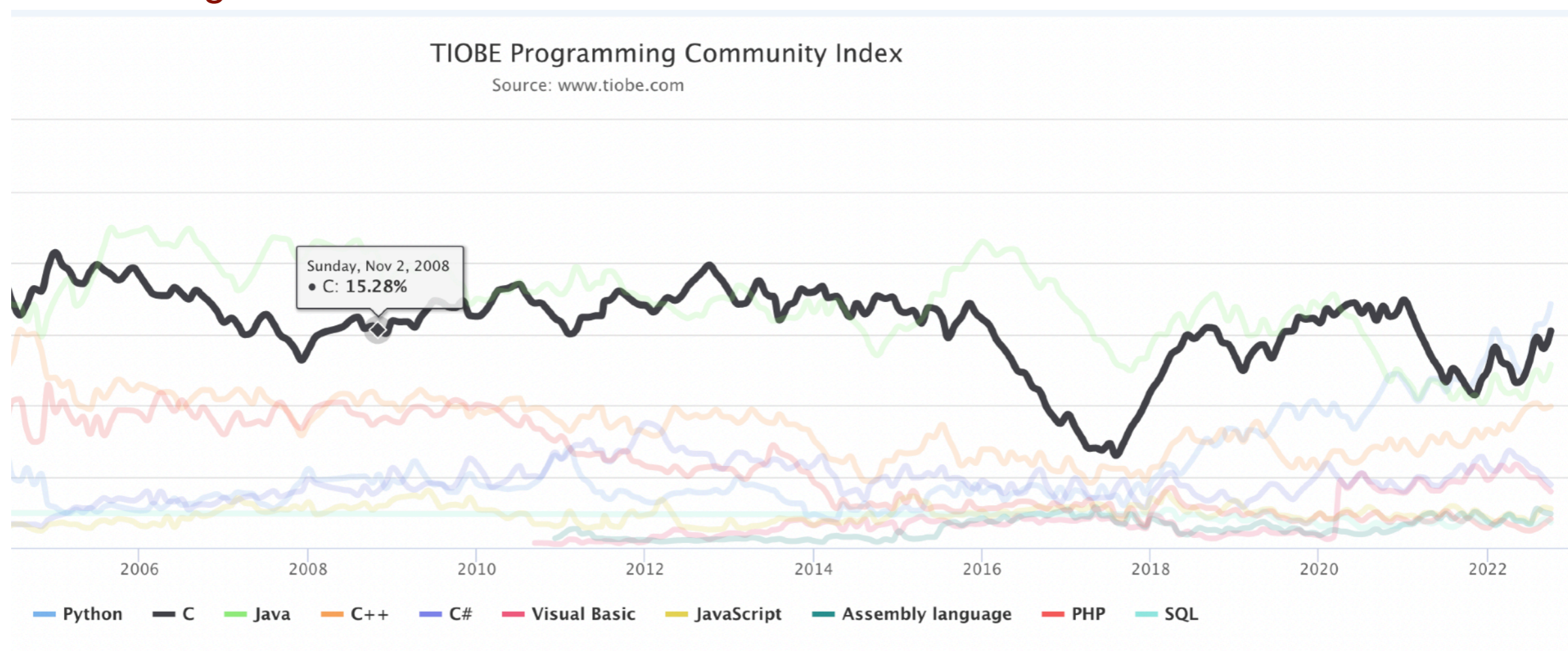


Introduction to C

Olivier Mattelaer
UCLouvain
CP3 & CISM

Why C

Tiobe Ranking



- C is currently #2 (#1 or #2 since 2001)
 - ➔ 4 of the 5 top program are C related
 - ➔ Including Python and C++
- C is concept are key for other language (Cuda)

Program of today

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

Warning:

- I'm not a C/C++ expert
- This explains the basic. Not the latest up-to-date standard/feature/...

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>

- line 1: 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>

```
1 // my first program in C
2 #include <stdio.h>
3
4 int main()
5 {
6
7     printf("Multiplication table of 5:\n");
8     printf(" 5 * 1 = 5 \n");
9     printf(" 5 * 2 = 10 \n");
10    printf(" 5 * 3 = 15 \n");
11    printf(" 5 * 4 = 20 \n");
12    printf(" 5 * 5 = 25 \n");
13    printf(" 5 * 6 = 30 \n");
14    printf(" 5 * 7 = 35 \n");
15    printf(" 5 * 8 = 40 \n");
16    printf(" 5 * 9 = 45 \n");
17    printf(" 5 * 10 = 50 \n");
18 }
```

- What's wrong with this code?

Variable

<http://cpp.sh/522d2>

```
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     printf(" %d * 1 = %d \n", i, i);
10    printf(" %d * 2 = %d \n", i, 2*i);
11    printf(" %d * 3 = %d \n", i, 3*i);
12    printf(" %d * 4 = %d \n", i, 4*i);
13    printf(" %d * 5 = %d \n", i, 5*i);
14    printf(" %d * 6 = %d \n", i, 6*i);
15    printf(" %d * 7 = %d \n", i, 7*i);
16    printf(" %d * 8 = %d \n", i, 8*i);
17    printf(" %d * 9 = %d \n", i, 9*i);
18    printf(" %d * 10 = %d \n", i, 10*i);
19 }
```

- Make “5” a parameter
 - ➔ Abstract the code for any value

```
int i = 5;
```

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

While loop

cpp.sh/9dn5g

```
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 }
```

- 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

```
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     for (int j=0; j<10; j++){
10         printf(" %d * %d = %d \n", i, j+1, i*(j+1));
11     }
12
13
14 }
```

- `j++`: means “add one to the value of `j`”
- Quite common to count from 0 in 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

```
1 // my first program in C
2 #include <stdio.h>
3
4 int main()
5 {
6
7     int i = 5;
8     float x=1.0;
9     double c =1.0;
10    char a = 'h';
11
```

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

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

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

Functions

```
1 // my first program in C
2 #include <stdio.h>
3
4 void print_table(int tableof, int maxmul){
5
6     for(int j=1; j<(maxmul+1); j++){
7         printf("the product of %d and %d is %d\n", tableof, j, tableof*j);
8     }
9
10 }
11
12
13 int main()
14 {
15     print_table(4,10);
16     print_table(5,11);
17
18
19 }
```

cpp.sh/24uno

<http://cpp.sh/3ssg5>

- Function allows to reuse a piece of code with argument

Functions

```
1 // my first program in C
2 #include <stdio.h>
3
4 void print_table(int tableof, int maxmul){
5
6     for(int j=1; j<(maxmul+1); j++){
7         printf("the product of %d and %d is %d\n", tableof, j, tableof*j);
8     }
9
10 }
11
12
13 int main()
14 {
15     print_table(4,10);
16     print_table(5,11);
17
18
19 }
```

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

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

<http://cpp.sh/932uo>

```
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     printf("i is store in ram at adress %p", &i);
10 }
```

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


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 address

◆ *pi = 2

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

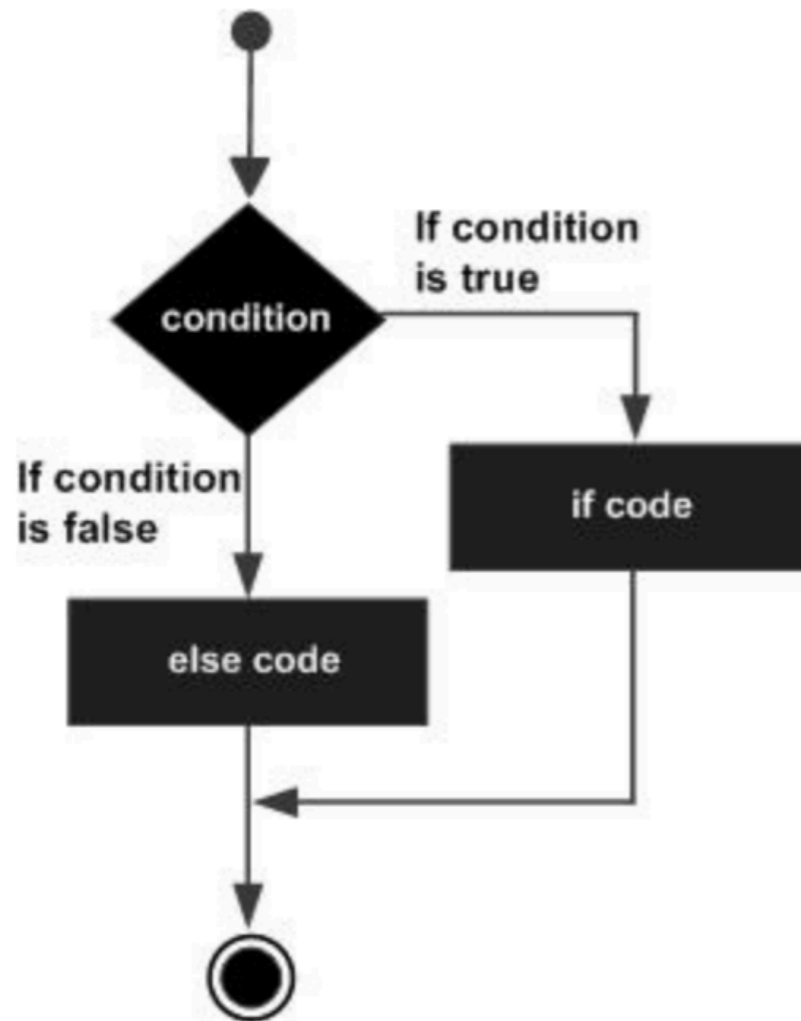
- That's where the address/pointer are useful

```
3  /* FUNCTION DECLARATION */
4  void swap(int* x, int* py);
5
6  int main () {
7
8  /* local variable definition */
9  int a = 100;
10 int b = 200;
11
12 printf("Before swap, value of a : %d\n", a );
13 printf("Before swap, value of b : %d\n", b );
14
15 swap(&a, &b);
16
17 printf("After swap, value of a : %d\n", a );
18 printf("After swap, value of b : %d\n", b );
19
20 return 0;
21 }
22
23 /* function definition to swap the values */
24 void swap(int* px, int* py) {
25
26 int temp;
27 temp = *px; /* save the value at address px */
28 *px = *py; /* put the value from address py into
29 *py = temp; /* put temp into adress py */
30
31 return;
32 }
```

- You can modify what is store at a given memory location
- So you pass the address and modify the value store at that address

If statement

- Checking condition and react accordingly is the core of programming



```
1 #include <stdio.h>
2
3 int main () {
4     /* local variable definition */
5     int a = 100;
6
7     /* check the boolean condition */
8     if( a < 20 ) {
9         /* if condition is true then print the following */
10        printf("a is less than 20\n" );
11    } else {
12        /* if condition is false then print the following */
13        printf("a is not less than 20\n" );
14    }
15
16    printf("value of a is : %d\n", a);
17
18    return 0;
19
20 }
```

- One liner:

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

and/or operation

- Combining condition is of course crucial

Operator	Meaning
&&	AND
 	OR
!	NOT

```
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;
```

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

```
#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

Tarball: array_are_pointer.c

<http://tpcg.io/h9ymMaep>

```
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;  
}
```

```
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_fct.c

- You can pass a sub-array to a function

```
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

```
struct Formation {  
    char title[50];  
    char speaker[50];  
    int 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 Book1 info */  
    printf( " Formation \"%s\" given by \"%s\" has %d student",  
           Lect_C.title, Lect_C.speaker, Lect_C.nb_student);  
}
```

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)

```
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;
}
```

```
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

- Computer have two type of memory
 - Static memory (stack) assigned when the program starts
 - Compiler might not assigned enough (rarely)
 - Dynamic memory (heap) assigned on the flight
 - Only limitation is the amount of RAM
- What happens if you do not know the size of an array?
- Or if you need to change it's size?

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

Array of arbitrary size!!

Dynamical allocation

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main() {
6
7
8     int* vector;
9     int size = 3;
10    vector = malloc(size * sizeof(int));
11
12    if( vector == NULL ) {
13        fprintf(stderr, "Error - unable to allocate required memory\n");
14        return 1;
15    }
16    vector[0] = 1;
17    vector[1] = 2;
18    vector[2] = 3;
19
20    int i =4;
21    if(size<i){
22        size +=1;
23        vector = realloc( vector, size * sizeof(char) );
24        if( vector == NULL ) {
25            fprintf(stderr, "Error - unable to allocate required memory\n");
26            return 1;
27        }
28        vector[3] = 4;
29    }
30    printf("size is %d\n", size);
31    for(int j=0; j<size; j++){
32        printf("%d ", vector[j]);
33    }
34    }
35    free(vector);
36
37 }
```

Exercise

- Compute the Fibonacci sequence
 - ➔ 1 1 2 3 5 8 13 ...
 - ➔ Use at least one function for doing so
 - ➔ Up to 50 term

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