## header files .h

- function declarations
- pre-processor directives (#...)
- user-defined types
- inclusion of other headers

## source files .c

- function definitions (main())
->
- global variables
- inclusion of headers

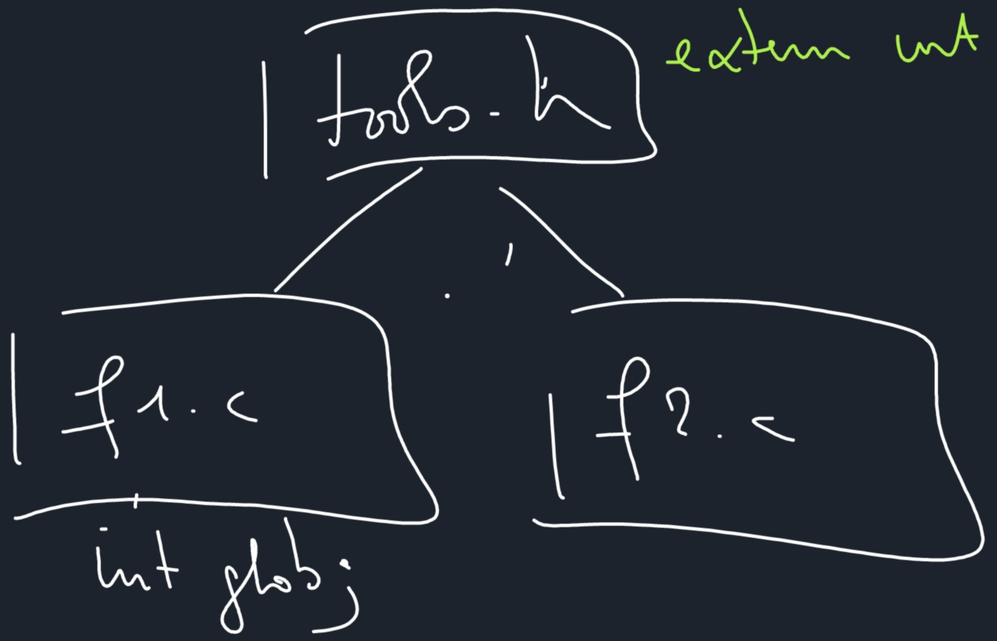
types:

- single
  - discrete
    - user defined
      - enumerations
        - enum { RED, BLUE, WHITE } myColor;
    - predefined
      - pointers
      - char, short, int, long
        - char c = 'A';
        - char c = 65;
  - floating point
    - float
    - double
- structured
  - ~~arrays~~
  - structures
    - typedef struct { double re; double im; } Complex;
    - Complex z1;
  - unions
    - struct Complex { double re; double im; };
    - struct Complex z1;

Variable statuses: automatic, static, extern, register

| | auto | static |
|---|---|---|
| Scope | - in {} : the {}<br>- otherwise : global | - in {} : the {}<br>- otherwise : the file |
| life duration | - in {} : exec {}<br>- otherwise : whole prog exec | whole prog exec |

```
int nTimes () {
static int n = 0;
        ⋮
return n++;
}
```




```
register int i;
for (i=0; i<N; i++)
    doSomething(i);
```

returnType name (params);

double avg (double x, double y) { return (x + y)/2; }

void

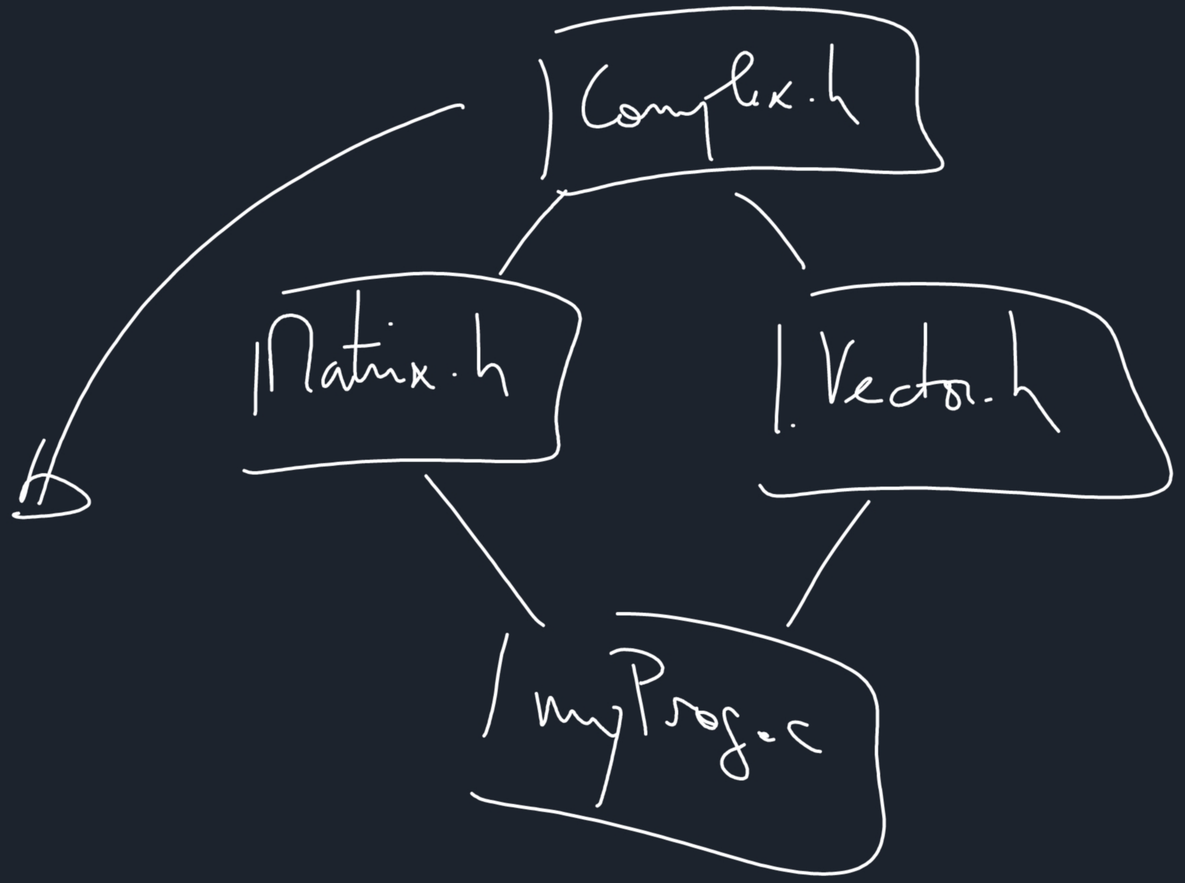
# Main pre-processor tasks

```
#include <stdio.h>
#include "myType.h"
```

```
#ifndef __COMPLEX_H
#define __COMPLEX_H

typedef struct {
  ...
} Complex;

#endif
```



```
#define BSIZE 1024
#define max(a,b) a<b ? b : a
int i=2, j=3, *j
k = max(i++, j++);
```

```
k = i++ < j++ ? j++ : i++
```

expected: i=3, j=4, k=3

got : i=3, j=5, k=4


i : 3
j : 4

j : 5
k = 4

# I/O functions.

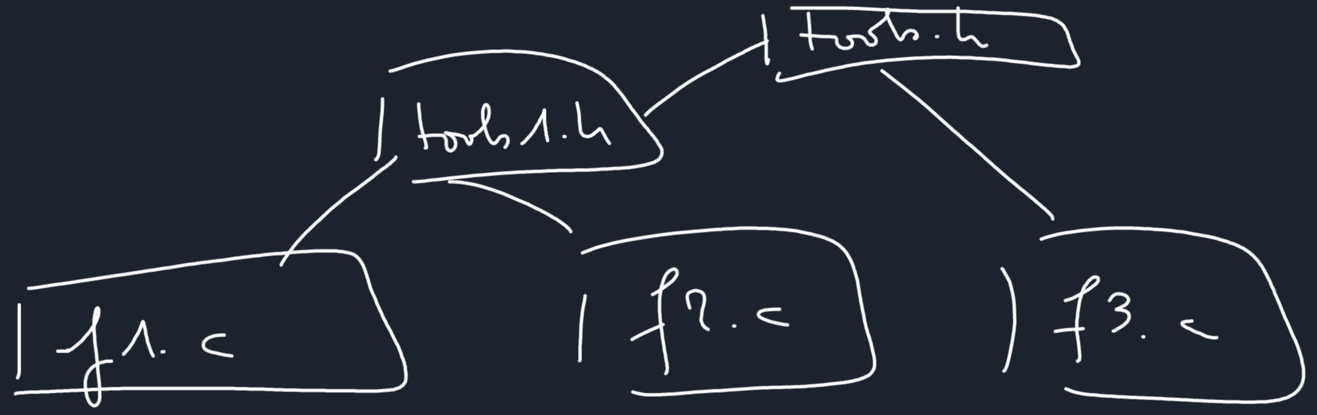
```
#include <stdio.h>

int printf (format, ....);
```

- strings "bonjour"
- special chars \t, \n ....
- format markers %d integers
  %f float
  %s strings
  %c char

```
printf("avg of %lf and %lf is %lf\n",
        x, y, avg(x,y));
```

```
scanf("%lf%lf", &x, &y);
```




# makefile

OBJ = f1.o f2.o f3.o

myProg : $(OBJ)
<tab> cc -o myProg $(OBJ)

f1.o : f1.c toolsh1.h toolsh.h
<tab> cc -c f1.c

) → f.o : toolsh1.h toolsh.h

```
depend:
	sed '/^#DEP/q' makefile > mk.tmp
	cc -M *.c >> mk.tmp
	mv mk.tmp makefile
```

#DEPEND : don't delete this line !

# Pointers

Why?

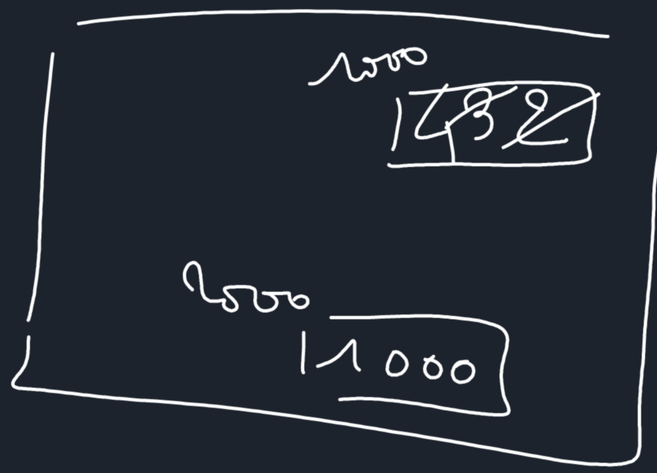
1. create dynamic structures
2. enable side effects in functions
2.5 optimize parameter passing in function

def: a pointer is a variable whose value is a memory address

2 operators: unary * : "object pointed to by"
unary & : "address-of"

definition and usage

```
int i = 2;
```


```
int *p;
p = &i;
i = 3;
*p = 5;
```

→ `int *p = &i;`

stack

ST

| name | @ |
|---|---|
| i | 1000 |
| p | 2000 |

# Dynamic memory allocation

```
Bag * createBag (int capa) {
    Article b[capa];
    ...
    return theBag;
}
```

```
void free(void * p);
```

& functions:

```
void* malloc(int nBytes);
```

```
{ int i;
  ...
}
```

```
{ int * p
= (int *)
malloc(10);
...
}
```

# Side effects in functions

pb: pass_by_value is the only mechanism in C

```
void triple(int *a){
    (*a) = 3 * (*a);
}
int main(){
    int i = 2;
    triple(&i);
    printf("%d\n", i);
    return 0;
}
```

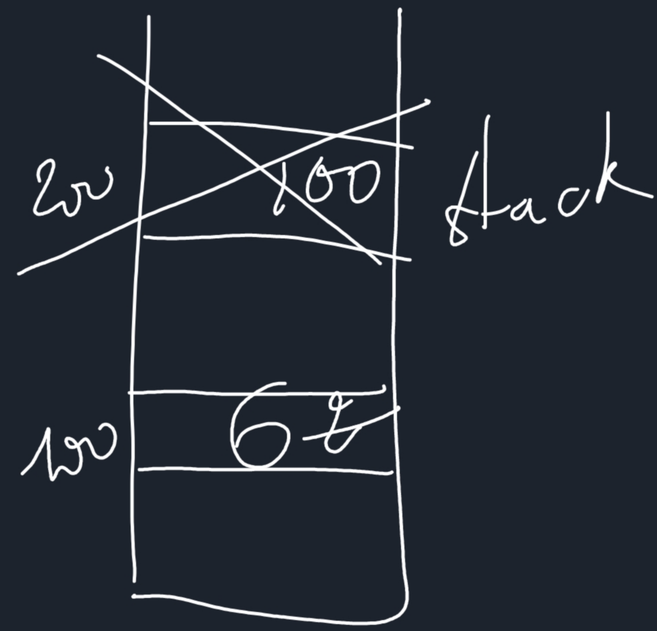
```
x = 2; y = 3;
z = avg(x, y);
```

```
double avg(double a, double b){...}
```

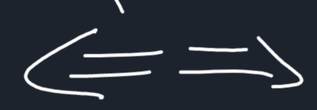
Here, a is 2 and b is 3

ST

| name | @ |
|---|---|
| i | 100 |
| a | 200 |



In C, one cannot set to a parameter (from inside a function) in a permanent way

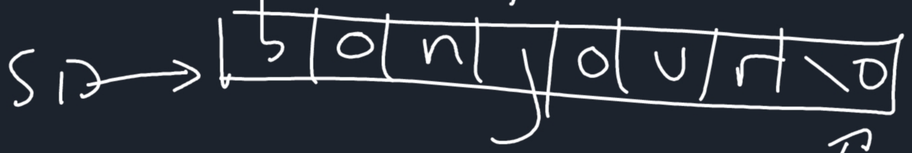


## Strings

```
#include <string.h>
int strlen(char *s);
int strcpy(char *dst, char *src);
int strcmp(char *s1, char *s2);
```

→ | 0 if =
| <0 if s1 < s2
| >0 if s1 > s2

```
char *s = "bonjour";
```

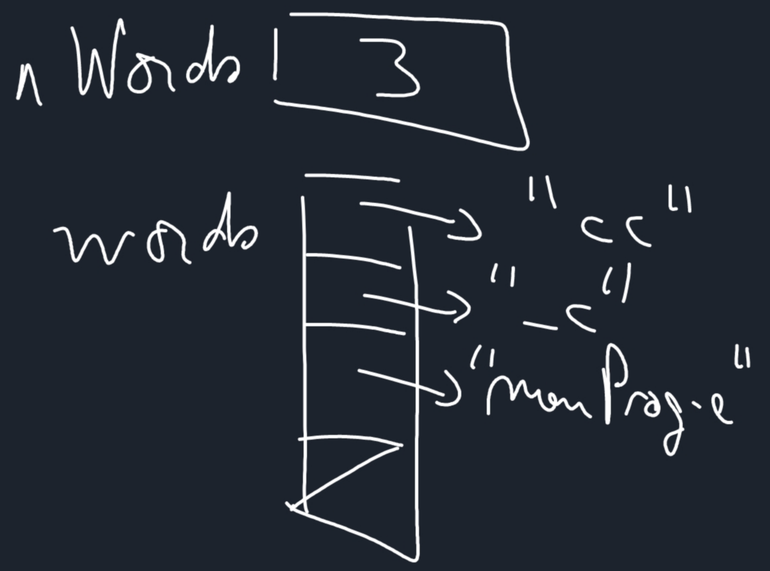



```
*s = 'B';
s[0] = 'B';
if (s1 == s2) ....
if (!strcmp(s1, s2)) ....
```

# Command line parameters

```
$ cc -c monProg.c
```

```
int main(int nWords, char* words[]){ ... }
```




HOME /home/me

atoi("3") → 3

```
$ ./sum 3 4 -1 2
8
```

```
/* sum.c */
#include <stdio.h>
#include <stdlib.h>
int main(int nWords, char* words[])
{
    int sum=0, i;
    for(i=1; i<nWords; i++)
        sum += atoi(words[i]);
    printf("%d\n", sum);
    return 0;
}
```

```
if ...
then
    ...
else
    ...
fi
```

```
grep root /etc/passwd
grep route
1
grep root /etc/password
2
```

```
res = sscanf(words[i], "%d", &cur)
if (res == 1) sum += cur;
```