A-


```c
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
{
    int i;
    for(i = 1; i < argc ; i++) {
        printf("%s ", argv[i]);
    }
    putchar('\n');
    exit(EXIT_SUCCESS);
    /*
    Question 1
    % mecho "Hello world !"
    -> argv = ["mecho", "Hello world !"]

    % mecho Hello world !
    -> argv = ["mecho", "Hello", "world", "!"]

    Question 2
    Ce programme va nous montrer les arguments insÃ©rÃ©
    (car la boucle commence Ã  1).
    int argc = la quantitÃ© de l'argument+1
    char *argv[] = l'argument

    Pour exÃ©cute le program :
    ./module/division
    */
}
```

✓ ✓ ✓

éviter les accents / caractères spéciaux dans les fichiers de code.

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

/*Question 1*/
float * search_dicho(float v, float *tab, int size){
    float *res = NULL;
    int mid = (size/2);
    if(size < 1){
        res = NULL;
    }
    else if(tab[mid] == v){
        res = tab + mid;
    }
    else if(tab[mid] > v){
        res = search_dicho(v, tab, mid);
    }
    else if(tab[mid] < v){
        res = search_dicho(v, &tab[mid + 1], mid);
    }
    return res;
}

/*Question 2*/
int main(void){
    float tab[]={1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9 ,2.0};
    float * a = search_dicho(1.1, tab, SIZE);
    float * b = search_dicho(1.9, tab, SIZE);
    float * c = search_dicho(2.1, tab, SIZE);

    if ((a == tab) && (b == tab+8) && (c == NULL)){
        printf("OK ;)\n");
        exit(EXIT_SUCCESS);
    } else{
        printf("KO ;(\n");
        exit(EXIT_FAILURE);
    }
        return 0;
}
```

size-mid-1 (pour les cas où size est pair)

```c
#include <stdio.h>
int put_digit(int digit)
{
    if (0<= digit && digit <= 9){
    int code_ascii = digit + 48;
    if (putchar(code_ascii) == EOF){
        return -1;
    }
    return 0;
    }
    return -1;
}

int putdec(long int nombre)
{
    int index = 1000000000;
    if (nombre == 0) putchar('0');
    if (nombre < 0) {
        nombre = -nombre;
        putchar('-');
    }
    while(index > 0 && ((nombre / index) % 10 == 0)){
        index = index / 10;
    }
    while(index > 0){
        int c = (nombre / index) % 10;
        c = (c<0) ? -c : c;
        put_digit(c);
        index = index / 10;
    }
    return 0;
}

/*Question 1*/
void division (int *x, int *y, int *q, int *r){
    *q = *x / *y;
    *r = *x % *y;
}

/*Question 2*/
int main(void){
    int x, y, q=0, r=0;

    printf("Entrez la valeur de diviseur : ");
    scanf("%d", &x);

    printf("Entrez la valeur de dividende : ");
    scanf("%d", &y);

    division (&x, &y, &q, &r);

    printf("\nLe quotient : ");
    putdec(q);
    printf("\nLe reste : ");
    putdec(r);
    printf("\n");

    return 0;
}
```

On ne cherche pas à modifier x et y donc on utilisera plutôt des int ici.

Attention à la division par 0.

Il est prudent de vérifier la valeur de retour de scanf (ici, ça devrait être 1).

```c
#include <stdio.h>

/*Question 1*/
void swap_int(int *a, int *b){
    int x;
    x = *a;
    *a = *b;
    *b = x;
}
/*Question 2*/
int main(void){
    int a, b;
    printf("Entrez la premiÃ¨re valeur : ");
    scanf("%d", &a);

    printf("Entrez la deuxiÃ¨me valeur : ");
    scanf("%d", &b);

    printf("int 1 : %d, int 2 : %d \n", a, b);

    swap_int(&a, &b);

    printf("AprÃ¨s avoir changÃ© la valeur\n");
    printf("int 1 : %d, int 2 : %d \n", a, b);

    return 0;
}
```

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

/*
Question 1
un type de int ** (pointeur qui pointe Ã  un pointeur)
*/

/*Question 2*/
void swap_ptr(int **p, int **q){
    int a,*b=&a,**c=&b;
    *c = *p;
    *p = *q;
    *q = *c;
}

/*Question 3*/
int main() {
    int a, b;
    int *p = &a;
    int *q = &b;

    swap_ptr(&p, &q);

    if ((p == &b) && (q == &a)) {
        printf("OK ;)\n");
        exit(EXIT_SUCCESS);
    } else {
        printf("KO ;(\n");
        exit(EXIT_FAILURE);
    }
}
```

Il un peu compliqué : on doit stocker la valeur pointée par p, de type int*, donc il suffit d'avoir un pointeur de type int* pour le tampon :

```
int *b;
b = *p;
:
```

Note : on peut déclarer un pointeur sans forcément l'initialiser avec une adresse immédiatement (ou, mieux, en l'initialisant à NULL).

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <ctype.h>

/*Question 1*/
char* skip_spaces(char s[]){
    int i;
    for (i = 0; s[i] == ' '; i++){
        if(s[i] != ' ')
            return NULL;
    }
    return &s[i];
}

int main(int argc, char *argv[])
{
    char * strip;
    int i;
    assert(argc == 2);

    printf("argv : %s\n", argv[1]);
    strip = skip_spaces(argv[1]);
    printf("strip : %s\n", strip);
    for (i=0 ; strip[i]; i++)
        strip[i] = toupper(strip[i]);

    printf("strip : %s\n", strip);
    printf("argv : %s\n", argv[1]);

    exit(EXIT_SUCCESS);
    /*
    Question 2
    % ./module/strip_spaces_tst "FOO BAR"
    argv  : FOO BAR
    strip : FOO BAR
    strip : FOO BAR
    argv  : FOO BAR
    - Ã§a ne change rien car il n'y a pas d'espace devant et les lettres sont dÃ
©jÃ  en majuscules

    % ./module/strip_spaces_tst "       FOO BAR"
    argv  :        FOO BAR
    strip : FOO BAR
    strip : FOO BAR
    argv  :        FOO BAR
    - il supprime l'espace devant le texte

    % ./module/strip_spaces_tst "Foo Bar"
    argv  : Foo Bar
    strip : Foo Bar
    strip : FOO BAR
    argv  : FOO BAR
    - il rend le mot en majuscule car la fonction toupper qui a importÃ© par <ct
ype.h>

    % ./module/strip_spaces_tst "       Foo Bar"
    argv  :        Foo Bar
    strip : Foo Bar
    strip : FOO BAR
    argv  :        FOO BAR
    - il supprime l'espace devant le texte & il rend le mot en majuscule car la
fonction toupper qui a importÃ© par <ctype.h>
     */

}
```

ça n'est jamais vrai car on sort de la boucle si s[i] != ' ', et on ne retourne jamais NULL, car au pire, on retournera l'adresse de '\0'.

Mais pourquoi argv[1] passe-t-il aussi en majuscules ?