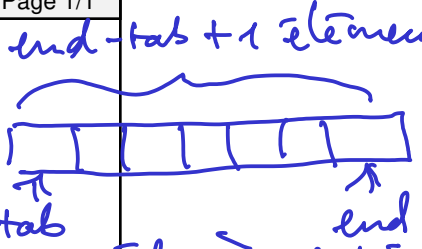
```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

/*Question 1*/
float * search_interval(float v, float *tab, float *end){
    int size = end - tab;
    int mid = size/2;

    if (tab[mid] == v)
        return &tab[mid];

    else if (size < 1)
        return NULL;

    else if ( v < tab[mid] ){
        return search_interval(v, tab , end - mid );
    } else {
        return search_interval(v, tab + mid + 1, end);
    }
}

/*Question 2*/
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;
}

int main(void){
    float tab[SIZE]={1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9 ,2.0};
    float * a = search_interval(1.1, tab, &tab[SIZE-1]);
    float * b = search_interval(1.9, tab, &tab[SIZE-1]);
    float * c = search_interval(2.1, tab, &tab[SIZE-1]);

    /*verification*/
    float * d = search_dicho(1.1, tab, SIZE);
    float * e = search_dicho(1.9, tab, SIZE);
    float * f = search_dicho(2.1, tab, SIZE);

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



```c
#include <stdio.h>
#define MAX 10

typedef int (fct)(int);

/*Question 1,2,3*/
int filter_int(const int *src, int *dst, unsigned int size, fct *fa){
    int i,n;
    int* src_int = (int*)src;
    int* dst_int = (int*)dst;
    for(n=i=0; i<size; i++){
        if( fa(src_int[i]) ){
            dst_int[n++] = src_int[i];
        }
    }
    return n;
}

int f(int a){
    return (a % 2 == 0)? 1:0;
}

int main(void){
    int i,end;
    int src[MAX] = {1,2,3,4,5,6,7,8,9,10};
    int dest[MAX];
    end = filter_int(src,dest,MAX,&f);

    printf("les entiers pairs : \n");
    for(i=0; i < end ;i++){
        printf("%d\n",dest[i]);
    }
    return 0;
}
```

inutile : ce sont déjà des int*

✓

inutile : return (a%2==0);

✓

```c
#include <stdlib.h>             /* pour random() */
#include <string.h>             /* pour memcmp() */
#include <assert.h>             /* pour assert() */
#include <stdio.h>
#define SIZE 1021

/*Question 1 copie generique*/
void mmemcpy(void *to, const void *from, unsigned int size){
  char *cto = (char*)to;
  const char *cfrom = (char*)from;
  while (size--)
    *cto++ = *cfrom++;
}
/*Question 1 echange generique*/
void memswap(void* a, void* b, int size)
{
    void *tmp = malloc(size);
    memcpy(tmp, a, size);
    memcpy(a, b, size);
    memcpy(b, tmp, size);
    free(tmp);
}

/*Question 2 copie generique*/
void test_mmemcpy()
{
    char     tc_orig[SIZE], tc_dest[SIZE];
    long int ti_orig[SIZE], ti_dest[SIZE];
    int i;

    /* initialisation */
    for(i=0 ; i<SIZE ; i++) {
        tc_orig[i] = random() % 256;
        tc_dest[i] = random() % 256;
        ti_orig[i] = random();
        ti_dest[i] = random();
    }

    /* copie */
    mmemcpy(tc_dest, tc_orig, SIZE);
    mmemcpy(ti_dest, ti_orig, SIZE * sizeof(long int));

    /* vÃ©rification */
    assert(memcmp(tc_orig, tc_dest, SIZE) == 0);
    assert(memcmp(ti_orig, ti_dest, SIZE * sizeof(long int)) == 0);
}

/*Question 2 echange generique*/
void test_mmemswap()
{
    char     a[SIZE], b[SIZE];
    long int c[SIZE], d[SIZE];
    char     testa[SIZE], testb[SIZE];
    long int testc[SIZE], testd[SIZE];
    int i;

    /* initialisation */
    for(i=0 ; i<SIZE ; i++) {
        a[i] = random() % 256;
        b[i] = random() % 256;
        c[i] = random();
        d[i] = random();
    }

    memcpy(a, testa, SIZE);
    memcpy(b, testb, SIZE);
    memcpy(c, testc, SIZE);
    memcpy(d, testd, SIZE);
```

Vous pouvez utiliser votre version !

1) Vérifier que l'allocation a réussi (tmp != NULL)
2) En général, on évite d'utiliser l'allocation dynamique pour les données locales (risque d'erreur plus important). Ici, ça peut cependant être nécessaire si size est supérieur à la taille de la pile.

inverser : memcpy(dest, src, size)

* sizeof(long int)


```c
    /* echange */
    memswap(b,a, SIZE);
    memswap(c, d, SIZE * sizeof(long int));

    /* vÃ©rification */
    assert(memcmp(testb, a, SIZE) == 0);
    assert(memcmp(testa, b, SIZE) == 0);
    assert(memcmp(testd, c, SIZE) == 0);
    assert(memcmp(testc, d, SIZE) == 0);
}

int main(void){
    test_mmemcpy();
    test_mmemswap();
    printf("Tous les tests sont passes !\n");
    return 0;
}
```

* sizeof(long int)

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

extern char **environ;

/*Question 1*/
int nvar(){
    /*                  Autre facon de le faire
    char **penv = environ;
    int i = 0;
    for(penv = environ; *penv != (char *) 0;penv++){
        i++;
    }
    return i;
    */

    int i = 0;
    while(environ[i]) {
        i++;
    }
    return i;

}

/*Question 2*/
void mprintenvAll(){
    int i = 0;
    while(environ[i]) {
        printf("%s\n", environ[i++]);
    }
}

/*Question 3*/
char * mprintenv(char *s){
    int i = 0, n;
    while(environ[i]) {
        for(n = 0; toupper(*s) == environ[i][n]; n++){
            s = s+sizeof(char);
        }
        if(*s == '\0' && environ[i][n] == '='){
            return environ[i]+n+1;
        }
        i++;
    }
    return NULL;
}

int main(int argc, char *argv[]){

    int i;
    char * toUp;

    printf("\nnvar() : %d\n",nvar());

    /*Question 2*/
    /* printf("mprintenv() : \n\n");
    mprintenvAll();  */

    for(i = 1; i < argc ; i++) {
        toUp = mprintenv(argv[i]);
        printf("%s\n",toUp);
    }
    exit(EXIT_SUCCESS);
    return 0;
}
```

✓

✓

✓

les noms de variables d'environnement peuvent être sensibles à la casse (selon le système d'exploitation employé).

1 → Arithmétique des pointeurs : +1 avance d'un "élément" de la taille du type pointé (ici, c'est sans conséquences car sizeof(char) == 1).

→ Risque de dépassement de chaîne si les deux chaînes sont identiques.

appeler cette version si argc == 1

Attention : la fonction peut retourner NULL !