C

Wieslaw Zielonka
zielonka@irif.fr

rappel

```
#include <stdlib.h>

void *malloc(size_t size)

void *calloc(size_t count, size_t size)

void *realloc(void *ptr, size_t size)

void free(void *ptr)
```

rappel

```
#include <string.h>

void *memcpy(void *dst, const void *src, size_t n)

void *memmove(void *dst, const void *src, size_t n)

void *memset(void *s, int c, size_t n)

void *memchr(const void *ptr, int val, size_t len)
```

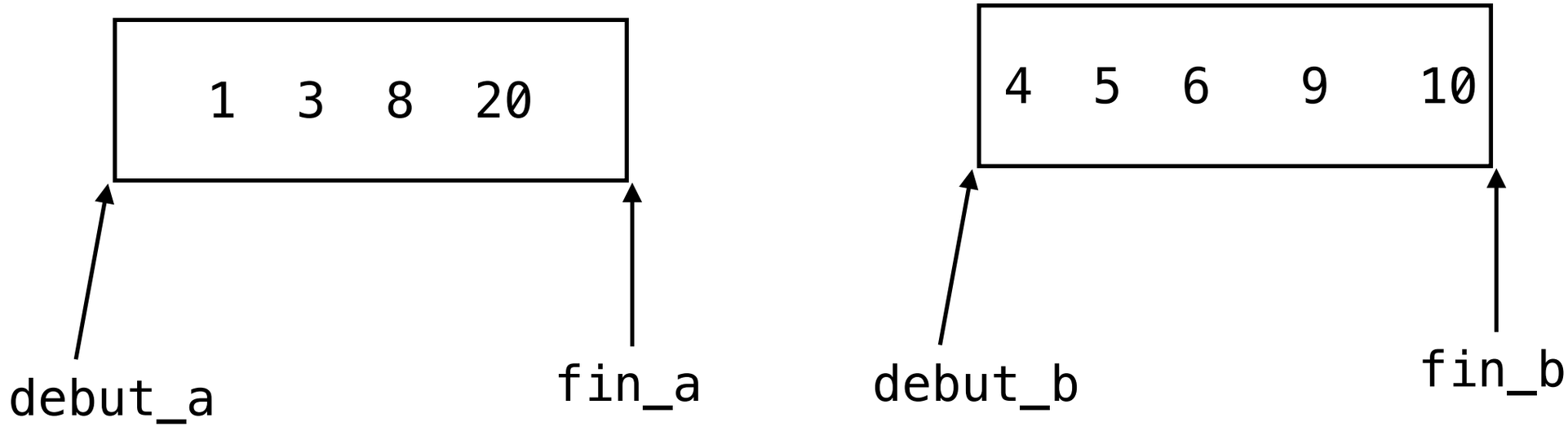
`memchr()` cherche la première occurence de val parmi `len` octets à partir de l'adresse `ptr.` La fonction retourne le pointeur vers le caractère retrouvé ou `NULL` s'il n'y a pas `val` dans la zone recherchée. Chaque octet c est comparé avec val avec

```
(unsigned char) c == (unsigned char) val
```

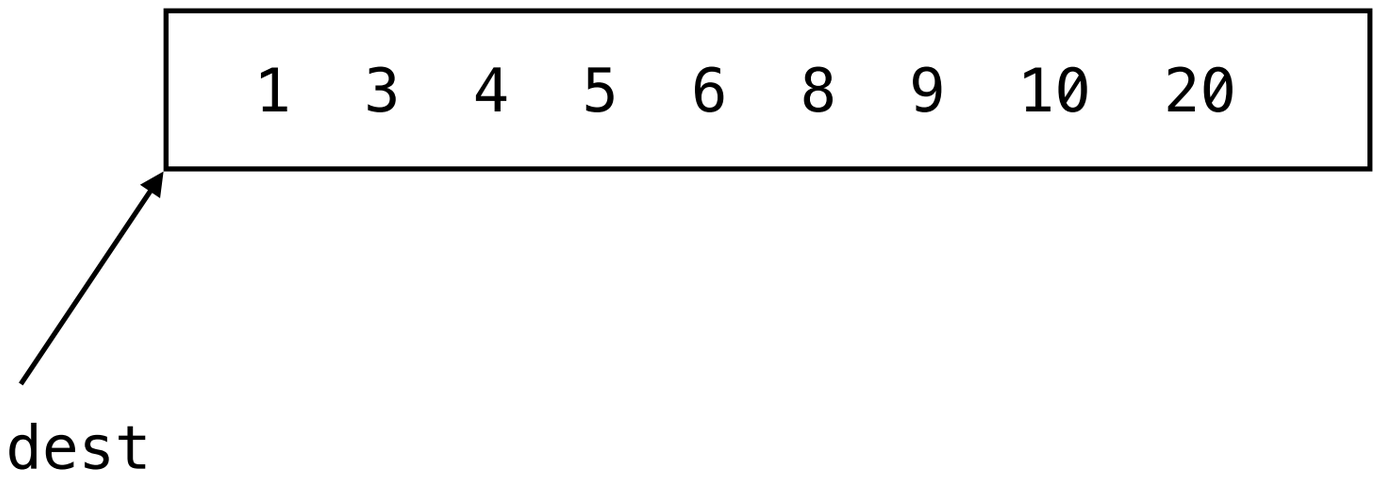
```
int memcmp(const void *ptr1, const void *ptr2, size_t len)
```

memcmp() compare deux suites de `len` octets dans deux zones de mémoire à l'adresse ptr1 et ptr2 respectivement. La fonction retourne l'entier négatif si la première suite est inférieure dans l'ordre lexicographique, `0` si les deux suites sont identiques, et `1` si la premier suite est supérieure dans l'ordre lexicographique.

exemple : fusion de tableaux triés de int




```
fusion(int *debut_a, int *fin_a, int *debut_b, int * fin_b, int *dest)
```



exemple : fusion de deux tableaux triés

```
void fusion(int *debut_a, int *fin_a,
            int *debut_b, int *fin_b, int *dest){


  while(debut_a < fin_a && debut_b < fin_b ){
    if(*debut_a <= *debut_b)
       *dest++ = *debut_a++;
    else
       *dest++ = *debut_b++;
  }


  if( debut_a < fin_a)
       memmove(dest, debut_a,
               sizeof(int) * (fin_a - debut_a));
    else
       memmove(dest, debut_b,
               sizeof(int) * (fin_b - debut_b));
}


int main(){
    int ta[]={-3,-8,99,120,500};
    int tb[]={-100,-2,40,155};
    int *tab = malloc( sizeof(ta) + sizeof(tb) );
    if(tab == NULL){ exit(1); }
    fusion(ta, ta+5, tb, tb+4, tab);
    ...........
}
```

assert(condition)

```
#include <assert.h>

assert(condition);
```

Si la condition est FALSE (s'évalue à 0) alors

- assert() envoie sur la sortie standard un message :

assertion failed, nom de la fonction, le nom de fichier source, le numéro de la ligne dans le fichier source et

- assert() exécute abort() ce qui termine le programme.

En définissant la constante NDEBUG avant #include <assert.h> on désactive les assertions:

```
#define NDEBUG

#include <assert.h>
```

Une autre possibilité pour désactiver les assertions ajouter l'option `-DNDEBUG` à la compilation.

```
int i;

.......

assert( i < val  && i>= 0);
```