Pointeurs génériques
void *

Pointeurs génériques

void *malloc(size_t size)

malloc retourne un pointeur générique parce que malloc ne "sait" pas quelles données seront stockées dans la mémoire allouée

vois *memmove(void *dst, const void *src, size_t len)

les adresses est et srrc sont de pointeurs génériques, memmove() ne "sait" rien (et n'a pas besoin de savoir) quel est le type de données recopiées à l'adresse dst.

L'arithmétique de pointeur pour le pointeurs génériques

L'arithmétique de pointeurs ne s'applique pas aux pointeurs génériques.

```
int tab[] = {1, 2, 3, 4, 5};

void *p = &tab[2];

void *q = &tab[4];

void *r = p+2; /* p+2 n'est pas défini sur un pointeur générique p */

ptrdiff_t d = q - p; /* q - p n'est pas défini quand p et q sont génériques */
```

Implementer un vecteur générique en C

```
struct vect_dyn{
  size_t size_elem; /* longueur d'un element en octets */
  size_t capacite;  /* le nombre d'éléments dans le vecteur */
  void   *debut;       /* adresse du premier élément du vecteur */
};
```

debut est un pointeur générique, on n'assume rien sur le type de données stockées dans le vecteur

Créer un vecteur générique

```
/* creer un vecteur dynamique de capacite  d'éléments
 * len_elem – la taille d'un element en d'octets */
vect_dyn * vect_dyn_creer(size_t len_elem, size_t capacité){

  vect_dyn * p = malloc(sizeof( vect_dyn ));
  if( p == NULL )  return NULL;

  p->capacite = capacite;
  p->size_elem = len_elem;

  p->debut = calloc( capacite, len_elem );
  if( p-> debut == NULL ){  free(p); return NULL; }

  return p;
}
```

L'adresse de i-ème élément ou NULL

```
/* retourner l'adresse de i-ème élément
 * ou NULL si i >= t->libre
 */
static void  *vect_dyn_adr(vect_dyn *t, size_t i){
  if( i >= t->capacite )
    return NULL;
  return (char *)t->debut + i * t->size_elem;
}
```

Notez le retypage :

`(char *)t->debut`

pour pouvoir faire l'arithmétique de pointeur en octets.

Rappel : `sizeof( char ) == 1` en C

Mettre une nouvelle valeur dans i-ème élément

```
void  vect_dyn_put_ieme(vect_dyn *t, size_t i, void  *x){
  /* est-ce que i-ème élément existe ? */
  assert( i < t->capacite );
  /* recopier la nouvelle valeur depuis l'adresse x */
  memmove( vect_dyn_adr(t, i), x, t->size_elem);
  return ;
}
```

récupérer la valeur de i-ème élément

```
/*  copier la valeur de i-ème élément a l'adresse x  */
void vect_dyn_get_ieme(vect_dyn *t, size_t i, void  *x){
  assert( i < t->capacite );
  memmove(x, vect_dyn_adr(t,i), t->size_elem);
  return ;
}
```

modifier le nombre d'éléments

```
int vect_dyn_agrandir( vect_dyn *t, size_t n ){
  if( n == t->capacite )
    return;
  void *ptr = realloc( t->debut, t->size_elem * n );
  if( ptr == NULL) return -1
  t->debut = ptr;
  size_t c = t->capacite;
  t->capacite = n;

  /* mettre à 0 les nouveaux éléments */
  if( n > c )
    memset( vect_dyn_adr(t, c) , 0, (n-c) * t->size_elem);
  return 0;
}
```

rechercher un élément

```
/* vect_dyn_find retourne l'indice du premier élément
 * égal à la valeur à l'adresse e.
 * Si aucun élément trouvé alors retourne -1
 */
int vect_dyn_find( vect_dyn * t, void *e){
  for(size_t i=0; i < t->capacite ; i++){
    if( memcmp( vect_dyn_adr(t, i) , e, t->size_elem) == 0 )
      return  (int)i;
  }

  return -1;
}
```

fichier vect_dyn.h

```
#ifndef VECT_DYN_H
#define  VECT_DYN_H

typedef struct vect_dyn vect_dyn;

extern vect_dyn *vect_dyn_creer(size_t t_elem);
extern void   vect_dyn_put_ieme( vect_dyn *t, size_t i, void  *x );
extern void    vect_dyn_get_ieme( vect_dyn *t, size_t i, void  *x );
extern int    vect_dyn_nbelems( vect_dyn *t );
extern void   vect_dyn_detruire( vect_dyn *t );
extern int    vect_dyn_supprimer_ieme(vect_dyn *t, int i);
extern int    vect_dyn_find( vect_dyn *t, void *e);
#endif
```