```c
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "flist.h"

struct pair
{
  void * car;
  void * cdr;
};

const flist nil = NULL;


flist
cons(void * obj, flist l)
{
  flist pair = (flist) malloc(sizeof(struct pair));

  assert(pair);
  pair -> car = obj;
  pair -> cdr = l;
  return pair;
}


void *
car(flist l)
{
  return l -> car;
}


flist
cdr(flist l)
{
  return l -> cdr;
}


int
is_null(flist l)
{
  return l == nil;
}

#ifdef MAIN

#include <stdio.h>

int main(int argc, char * argv[])
{
  flist l = nil;
  int i;

  for (i = 1; i < argc; i++)
    l = cons(argv[i], l);
  printf("(");
  while (! is_null(l))
    {
      printf("%s ", (char *)car(l));
      l = cdr(l);
```



```c
    }
  printf(")\n");
  exit(EXIT_SUCCESS);
}

#endif
```

```
#ifndef FLIST_H
#define FLIST_H

typedef struct pair * flist;

extern const flist nil;

extern flist
cons(void * obj, flist l);

extern void *
car(flist l);

extern flist
cdr(flist l);

extern int
is_null(flist l);

#endif
```

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

#include "mon_malloc.h"
#include "flist.h"

struct liste
{
    void  *car;
    fliste cdr;
};

const fliste Vide = NULL;


fliste
fl_cons(void * obj, fliste l)
{
    fliste res = (fliste) mon_malloc(sizeof(struct liste));

    res -> car = obj;
    res -> cdr = l;
    return res;
}


void *
fl_car(fliste l)
{
    assert(l != Vide);
    return l -> car;
}


fliste
fl_cdr(fliste l)
{
    assert(l != Vide);
    return l -> cdr;
}


int
fl_vide(fliste l)
{
    return l == Vide;
}


int
fl_longueur(fliste l)
{
    return (fl_vide(l)? 0 : 1 + fl_longueur(fl_cdr(l)));
}


fliste
fl_concatener(fliste l1, fliste l2)
{
    return (fl_vide(l1)? l2 : fl_cons(fl_car(l1), fl_concatener(fl_cdr(l1), l2))
);
```



```
}


fliste
fl_prefixe(fliste l, int i)
{
    if ( i == 0 )
        return Vide;
    else {
        assert(l != Vide);
        return fl_cons(fl_car(l), fl_prefixe(fl_cdr(l), i-1));
    }
}


fliste
fl_suffixe(fliste l, int i)
{
   if ( i == 0 )
        return l;
    else {
        assert(l != Vide);
        return fl_suffixe(fl_cdr(l), i-1);
    }
}


fliste
fl_supprimer(void * obj, fliste l, int (* comp)(void *, void *))
{
    if (fl_vide(l))
        return l;
    else if (comp(fl_car(l), obj))
        return fl_supprimer(obj, fl_cdr(l), comp);
    else return fl_cons(fl_car(l), fl_supprimer(obj, fl_cdr(l), comp));
}


fliste
fl_appliquer(void *(* f) (void *), fliste l)
{
    return (fl_vide(l)? Vide : fl_cons(f(fl_car(l)), fl_appliquer(f, fl_cdr(l)))
);
}
```

```
#ifndef _FLIST_H_
#define _FLIST_H_

typedef struct liste *fliste;

extern const fliste Vide;

extern fliste
fl_cons(void * obj, fliste l);

extern void *
fl_car(fliste l);

extern fliste
fl_cdr(fliste l);

extern int
fl_vide(fliste l);

extern int
fl_longueur(fliste l);

extern fliste
fl_concatener(fliste l1, fliste l2);

extern fliste
fl_egalite(fliste l1, fliste l2, int (* comp)(void *, void*));

extern fliste
fl_prefixe(fliste l, int i);

extern fliste
fl_suffixe(fliste l, int i);

extern fliste
fl_appliquer(void *(* f) (void *), fliste l);

extern fliste
fl_supprimer(void * obj, fliste l, int (* comp)(void *, void *));

#endif
```

```c
#include "mon_malloc.h"
#include <malloc.h>
#include <stdio.h>

#define ALLOC_EXIT(_TEST_, _MSG_, _F_, _L_) if ((_TEST_)) \
{\
  fprintf(stderr, "%s : fichier %s ligne %d.\n", (_MSG_), (_F_), (_L_));\
  exit(EXIT_FAILURE);\
}

void *mon_malloc_securise(size_t taille, char *fichier, int ligne)
{
    void *resultat = malloc(taille);
    ALLOC_EXIT((resultat == NULL), "Erreur d'allocation", fichier, ligne);
    return resultat;
}

void *mon_realloc_securise(void *ptr, size_t taille, char *fichier, int ligne)
{
    void *resultat = realloc(ptr, taille);
    ALLOC_EXIT((resultat == NULL), "Erreur d'allocation", fichier, ligne);
    return resultat;
}

void *mon_calloc_securise(size_t nbelts, size_t taille_elt, char *fichier, int l
igne)
{
    void *resultat = calloc(nbelts, taille_elt);
    ALLOC_EXIT((resultat == NULL), "Erreur d'allocation", fichier, ligne);
    return resultat;
}

void mon_free_securise(void *ptr, char *fichier, int ligne)
{
    ALLOC_EXIT((ptr == NULL), "Erreur de liberation", fichier, ligne);
    free(ptr);
}
```

```
#ifndef _MON_MALLOC_H_
#define _MON_MALLOC_H_

#include <stdlib.h>

extern
void *mon_malloc_securise(size_t taille, char *fichier, int ligne);
#define mon_malloc(taille) (mon_malloc_securise(taille, __FILE__, __LINE__))

extern
void *mon_realloc_securise(void *ptr, size_t taille, char *fichier, int ligne);
#define mon_realloc(taille) (mon_realloc_securise(void *ptr, taille, __FILE__, _
_LINE__))

extern
void *mon_calloc_securise(size_t nbelts, size_t taille_elt, char *fichier, int l
igne);
#define mon_calloc(taille) (mon_calloc_securise(size_t nbelts, taille, __FILE__,
 __LINE__))

extern
void mon_free_securise(void *ptr, char *fichier, int ligne);
#define mon_free(ptr) (mon_free_securise(ptr, __FILE__, __LINE__))

#endif /* _MON_MALLOC_H_ */
```

```
#include <stdio.h>
#include <stdlib.h>
#include "flist.h"

static void
afficher(fliste l);

static int
carre(int n);

int
main(int argc, char * argv[])
{
    fliste l = Vide;
    fliste l2;
    int i;

    for (i = 1; i < argc; i++)
        l = fl_cons(atoi(argv[i]), l);
    afficher(l);
    printf("Longueur %d\n", fl_longueur(l));
    l2 = fl_appliquer((void * (*)(void *))carre, l);
    afficher(l2);

    l2 = fl_concatener(l, l2);
    afficher(l2);
    exit(EXIT_SUCCESS);
}

static void
afficher(fliste l)
{
    printf("(");
    while (! fl_vide(l))
    {
        printf("%d", (int) fl_car(l));
        l = fl_cdr(l);
        printf("%s", fl_vide(l)? ")" : " ");
    }
    printf("\n");
}


static int
carre(int n)
{
    return n*n;
}
```