L2 Programmation Impérative

TD5 Modularité et compilation séparée

1 Construction de l'exécutable

1. Écrire les commandes dans un terminal afin de construire l'exécutable d'un programme à partir de 2 sources, `toto.c` qui contient :

```
#include <stdio.h>

void toto (int i)
{
  printf ("i = %d\n",i);
}
```

et `titi.c` qui contient :

```
extern void toto (int);

int main (int argc, char * argv[])
{
  toto(argc);
  toto(3);
}
```

2. Écrire le fichier `Makefile` pour cette construction de l'exécutable.

2 Fichier Makefile

1. Écrivez un `Makefile` permettant de produire l'exécutable `prog` à partir des fichier `prog.c` et `lire.c` suivants en utilisant la compilation séparée.

`prog.c` :

```
#include <stdio.h>

extern int lire_entier (void);

int main (void) {
  int i, x;
  for (i = 1; i <= 10; i++) {
    x = lire_entier ();
    printf ("Vous avez saisi l'entier %d\n", x);
  }
  return 0;
}
```

`lire.c` :

```
#include <stdio.h>

int lire_entier () {
  int x;
  printf ("Entrez un entier svp !\n");
```

```
  scanf ("%d", &x);
  return x;
}
```

2. Même question si l'on choisit de remplacer dans `prog.c` l'instruction `printf(...)` par l'appel d'une fonction `ecrire_entier (x)` fournie dans un fichier `ecrire.c`.
3. On suppose que l'on a exécuté avec succès la commande `make`, puis que l'on modifie `ecrire.c`. Quelles commandes de compilation seront effectuées si l'on exécute de nouveau `make` ?

3 Liste doublement enchaînée

Construction d'un noeud dans une liste doublement enchaînée d'entiers :

```
struct noeud {
  int valeur;
  struct noeud *suivant;
  struct noeud *precedent;
} ;
```

Pour présenter une liste doublement enchaînée, on utilise une deuxième structure :

```
struct dliste {
  struct noeud *tete;
  struct noeud *queue;
} ;
```

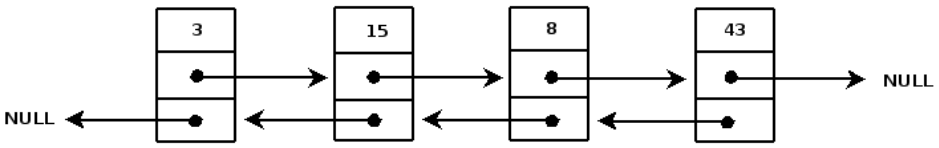

FIGURE 1 – Exemple d'une liste de 4 entiers

On se propose de créer une librairie de gestion de listes doublement enchaînées.

1. Écrire un fichier `liste.h` décrivant l'interface de manipulation de listes doublement enchaînées. Les méthodes utilisables devront être :
   - `initialiser()`, initialisant les champs d'une structure représentant une liste vide,
   - `liste_vide()`, retournant `vrai` si une liste est vide et `faux` sinon,
   - `est_present()`, retournant un pointeur sur un noeud contenant un entier recherché s'il est présent dans la liste,
   - `ajouter_en_tete()`, ajoutant un entier en tête de la liste s'il n'y est pas présent,
   - `cardinal()`, retournant le nombre d'éléments dans la liste,
   - `afficher_liste()`, affichant le contenu de la liste.
2. Écrire un fichier `liste.c` implémentant les fonctionnalités ci-dessus.
3. Compiler `liste.c` en un fichier objet et écrire un fichier `essai_liste.c` pour tester les fonctions sur les listes. Créer un fichier `Makefile` pour effectuer la création de l'exécutable `essai_liste`.