Université de Bordeaux
Département Licence, **4TIN302U**

6 décembre 2016
9h30-10h00

Devoir surveillé N2
Corrigé

Nom : Prénom : Groupe :

1 Utilisation d'une pile

On considère la fonction `mystere` suivante :

```
fonction mystere(val n : entier): entier;
var i,a,b: entier;
var P: pile de entier
debut
  P= creerPile();
  P= empiler(P, 0);
  P= empiler(P, 1);
  i=0;
  tant que i < n faire
    b= valeur(P);
    P= depiler(P);
    a= valeur(P);
    P= depiler(P);
    P= empiler(P, b);
    P= empiler(P, a+b);
    i++;
  fin tant que
  tant que ! pileVide(P) faire
    a= valeur(P);
    P= depiler(P);
  fin tant que
  detruirePile(P);
  retourner a;
fin
```

1. Simulez l'exécution de l'appel `mystere(7)` et complétez les tableaux suivant montrant l'évolution des variables et de la pile. Montrez l'évolution de la pile en la dessinant après chaque modification.

i	0	0	1	2	3	4	5	6	7 stop	7 stop
b	/	1	1	2	3	5	8	13	13	13
a	/	0	1	1	2	3	5	8	21	13
P	1 0	1 1	2 1	3 2	5 3	8 5	13 8	21 13	13	vide

2. Quelle valeur retourne l'appel `mystere(7)` ?

   La valeur retournée est 13

3. Dans le cas général quelle valeur retourne la fonction `mystere` ?

   La fonction calcule et retourne le nombre de Fibonacci au rang n.

2 File

Le sujet de cet exercice concerne le type abstrait `File` tel qu'il a été décrit en cours.

2.1 Question de cours :

Rappelez le principe de fonctionnement d'une file.

Une file est un type absrait basé sur le principe FIFO (First In First Out). Voir cours pour plus de détails.

2.2 Implémentation File :

Une **liste simplement chainée circulaire** est une liste simplement chainée telle que le suivant du dernier élément de la liste est le premier élément de la liste. Par exemple, le schéma ci-dessous peut représenter la liste d'entiers 5, 0, 8, 7, 12, 3, 17 stockés dans une liste circulaire L.
On remarquera que la tête de liste peut-être placée sur n'importe quel élément de la liste.

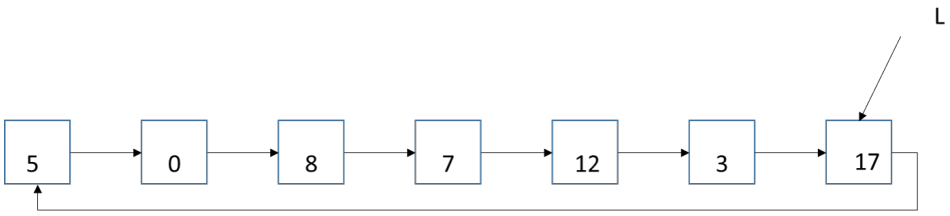

FIGURE 1 – Liste circulaire

Pour implémenter une file on propose d'utiliser une liste simplement chainée circulaire.
On considère l'implémentation suivante :

Implémentation File

```
struct File{
  int val;
  struct File* next;
}

typedef struct File* File;
```

On considère la fonction suivante :

queFaisJe

```
File
queFaisJe(File F, int x){
    File p = file_alloc();
    p->val = x;
    if(F== NULL){
        p->next=p;
    }
    else{
       p->next = F->next;
       F->next= p;
   }
    F=p;
    return F;
                }
```

On considère la file F représentée sur le schéma suivant :

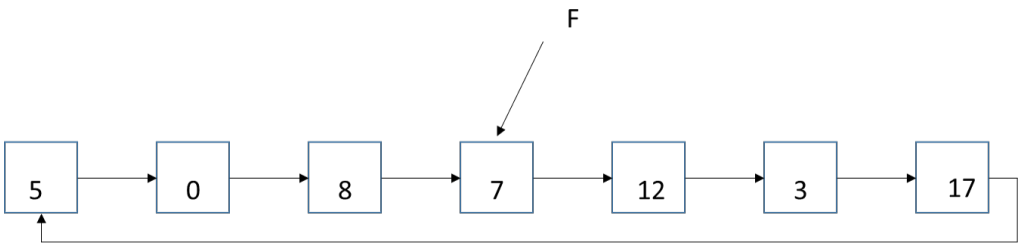

FIGURE 2 – File

1. Dessinez `F` après l'instruction `F = queFaisJe(F,9)`

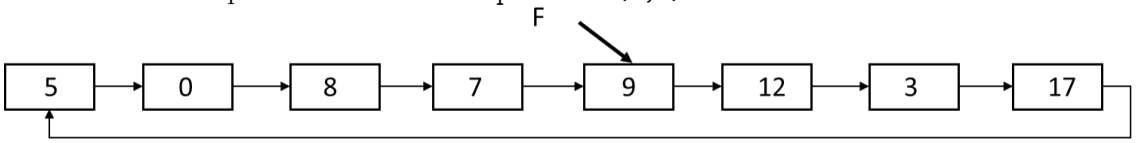

FIGURE 3 – File

2. Que fait la fonction `queFaisJe` dans le cas général ?

   La fonction`queFaisJe` *enfile un nouvel élément dans la File.*

   Quelle est sa complexité ?

   Sa complexité est en $O(1)$ car on ne parcourt pas la file. On peut donc dire que c'est une primitive du type abstrait File

3. Compte tenu des réponses aux questions précédentes, indiquez où se trouve la queue et la tête de file dans cette implémentation.

   Dans cette implémentation il faut comprendre que F ne pointe pas sur la tête de la File mais sur la queue.
   De fait $F->next$ pointe sur la tête. On a ainsi les 2 accès nécessaires à la manipulation d'une file.

4. Implémentez la primitive `defiler(F)` en O(1)

defiler

```
File
defiler(File F){
    assert(!fileVide(F)); /* File Vide */
    if(F==F->next){ /* Dernier element a defiler*/
      detruire_File (F);
      return NULL;
    }
    File p= F->next;
    F->next= p->next;
    file_free(p);
    return F;
                }
```

3 Annexe : Primitives des différents Types Abstraits vus en cours

Liste simplement chainée

```
fonction creerListe(): SCList;
fonction detruireListe(L : SCList) : vide;
fonction listeVide(L : SCList) : booleen;
fonction valeur(L : SCList) : objet;
fonction suivant(L : SCList): SCList;
fonction insererEnTete(L : SCList, v: objet): SCList;
fonction supprimerEnTete(L : SCList): SCList;
fonction insererApres(L : SCList, p :SCList, v: objet):SCList;
fonction supprimerApres(L : SCList, p :SCList): SCList;
```

Liste doublement chainée

```
fonction creerListe(): DCList;
fonction detruireListe(D: DCList) : vide;
fonction listeVide(D: DCList) : booleen;
fonction valeur(D: DCList) : objet;
fonction suivant(D: DCList): DCList;
fonction precedent(D: DCList): DCList;
fonction insererEnTete(D: DCList, v: objet): DCList;
fonction supprimerEnTete(D: DCList): DCList;
fonction insererApres(D: DCList, p: DCList, v: objet): DCList;
fonction supprimerApres(D: DCList, p: DCList): DCList;
fonction insererAvant(D: DCList, p: DCList, v: objet): DCList;
fonction supprimerAvant(D: DCList, p: DCList): DCList;
```

Pile

```
fonction creerPile(): pile de objet;
fonction detruirePile(P: pile de objet): vide;
fonction pileVide(P: pile de objet): booleen;
fonction valeur(P: pile de objet): objet;
fonction depiler(P: pile de objet): pile de objet;
fonction empiler(P: pile de objet, val: objet): pile de objet;
```

File

```
fonction creerFile(): file de objet;
fonction detruireFile(F: file de objet): vide;
fonction fileVide(F: file de objet): booleen;
fonction valeur(F: file de objet): objet;
fonction defiler(F: file de objet): file de objet;
fonction enfiler(F: file de objet, val: objet): file de objet;
```