Travaux Dirigés d'Algorithmique n°8

Cours d'Informatique de Deuxième Année

—Deug SSM M3-MI—

Nous allons regarder la structure de pile et en particulier nous allons l'utiliser pour dérécursiver des fonctions récursives. Encore une fois, le vocabulaire change : nous parlerons du sommet de la pile pour signifier le premier élément, d'empilement d'un élément pour faire référence à l'insertion en haut de la pile d'un nouvel élément et de dépilement pour l'opération consistant à supprimer le sommet de la pile.

▸ **Exercice 1.** *Comment implémenter le type de données abstrait "pile". Donner les déclarations C correspondantes.*

▸ **Exercice 2.** *Écrire les fonctions élémentaires suivantes :*

1. `EstVide` *prenant en paramètre une pile pointée par* `p`*, qui testera si* `p` *est vide ;*
2. `Empiler` *prenant en paramètre un entier* `x` *et une pile pointée par* `p`*, empilant l'élément* `x` *sur la pile* `p`*. La fonction retournera un pointeur sur la pile obtenue ;*
3. `Depiler` *prenant en paramètre une pile pointée par* `p`*, dépilant le sommet de la pile* `p`*. La fonction retournera la valeur contenue dans la structure correspondant au sommet de la pile.*

▸ **Exercice 3.** *On considère des expressions arithmétiques en notation postfixée sur des chiffres positifs et les opérateurs binaires : +, -, ×, /. On souhaite écrire un algorithme utilisant une pile pour évaluer de telles expressions.*

▸ **Exercice 4.** (**Parcours préfixe d'un arbre binaire**) *On considère les déclarations suivantes implantant la structure d'arbre binaire :*

```
typedef struct noeud
{ int Val;                    /* valeur dans le noeud                   */
  struct noeud * Gauche;      /* pointeur sur le sous-arbre gauche      */
  struct noeud * Droit;       /* pointeur sur le sous-arbre droit       */
} StrNoeud, * PtrNoeud;       /* nouveaux types : StrNoeud et PtrNoeud */
```

Nous souhaitons simuler un parcours préfixe d'un arbre binaire à l'aide d'une pile. La fonction récursive que nous voulons dérécursiver est donc la suivante :

```
void Prefixe(PtrNoeud abr)
{ if (abr!=NULL)
  { printf("%d ", abr->Val);
    Prefixe(abr->Gauche);
    Prefixe(abr->Droit);
  }
}
```

1. *quelle(s) modificication(s) doit-on apporter à la structure de pile définie précédemment afin de pouvoir l'utiliser pour simuler le parcours d'un arbre binaire ?*
2. *expliciter l'algorithme prenant en entrée un arbre binaire et réalisant le parcours préfixe (non récursif) de l'arbre binaire en utilisant une pile ;*
3. *en déduire la fonction correspondante implantant cet algorithme ;*
4. *évaluer la hauteur maximale de (au sens du nombre maximal d'éléments simultanément présents dans) la pile.*

▸ **Exercice 5.** (**exercice supplémentaire**) *On considère les mots sur les deux lettres* `(` *et* `)`*, c'est-à-dire faits de parenthèses ouvrantes et fermantes. De tels mots seront dits bien parenthésés si et seulement si, lorsqu'on les parcourt de gauche à droite et que l'on compte le nombre de parenthèses ouvrantes et fermantes, on a toujours au moins autant de parenthèses ouvrantes que de parenthèses fermantes. A la fin de la lecture du mot, ces deux nombres doivent être égaux. Par exemple* `()()` *et* `(()(()()))` *sont bien parenthésés alors que les mots* `)(` *et* `(()))` *ne le sont pas.*

1. *donner un algorithme utilisant une pile permettant de vérifier qu'un mot est ou n'est pas bien parenthésé ;*
2. *en déduire la fonction correspondante.*

▸ **Exercice 6.** (**exercice supplémentaire**) *Utiliser la structure de pile définie au début pour supprimer la récursion dans la fonction suivante de calcul du* `i`*-ième nombre de Fibonacci :*

```
int Fibonacci(int i)
{ if ((i==0) or (i==1)) return 1;
  return Fibonacci(i-1) + Fibonacci(i-2);
}
```