Introduction à l'algorithmique et à la programmation
Les piles en C

Les piles statiques

Présentation

- La pile «dernier entré premier sortie» (LIFO: last in, first out)

  - Imaginez une pile de pièces :

    a) vous pouvez ajouter des pièces une à une en haut de la pile
    b) vous pouvez aussi en enlever depuis le haut de la pile.
    c) Il est impossible d'enlever une pièce depuis le bas de la pile.

- La file «premier entré, premier sortie» (FIFO: first in, first out)

  - … par exemple la file d'attente au restaurant universitaire entre personnes civilisées

Dans ce chapitre, nous étudierons les piles STATIQUES. Il existe également des piles et files dynamiques (au programme de deuxième année)

Les piles statiques

Les structures de base pour les piles:

- La structure t_element

```
typedef struct{
        <type> <c_composant1>;
        <type> <c_composant2>;
        <type> <c_composant3>;
}t_element;
```

- La structure t_pile

```
# define MAX 1024
typedef struct{
        t_element tabElts[MAX];
        int nbElts;
}t_pile;
```

- ATTENTION: t_pile n'est pas un tableau mais une structure dont un composant est un tableau

- Une variable de type t_pile n'est pas une adresse.

Les piles statiques

Les opération de base de la pile (FIFO)

- Empiler un élément
- Dépiler un élément
- Initialiser la pile
- Vider la pile
- Donner le sommet de la pile (le dernier élément)
- La pile est-elle vide?
- La pile est-elle pleine?

Les piles statiques

Empiler un élément (on suppose que la pile est partiellement remplie)

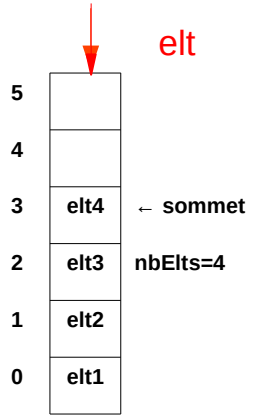
- Le prototype de la procédure empiler()

```
void empiler(t_pile* adrPile, t_element elt);
// la pile sera modifiée par la procédure
// l'élément n'est pas modifié par la procédure
```

- la définition de la procédure empiler()

  - Remarque: la pile n'est pas pleine

```
void empiler(t_pile *adrPile, t_element elt){
        if (!(estPleine(*adrPile)) {
                adrPile->tabElts[adrPile->nbElts]=elt ;
                adrPile->nbElts=adrPile->nbElts+1;
                // Attention aux indices du tableau
        }
        else{
                printf("La pile est pleine");
        }
}
```



Les piles statiques

Empiler un élément … suite

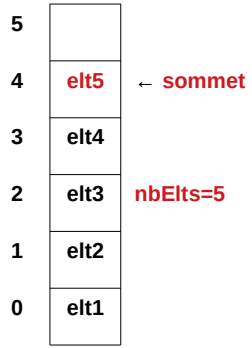
- Le prototype de la procédure empiler()

```
void empiler(t_pile* adrPile, t_element elt);
// la pile sera modifiée par la procédure
// l'élément n'est pas modifié par la procédure
```

- la définition de la procédure empiler()

  - Remarque: la pile n'est pas pleine

```
void empiler(t_pile *adrPile, t_element elt){
        if (!(estPleine(*adrPile)) {
                adrPile->tabElts[adrPile->nbElts]=elt ;
                adrPile->nbElts=adrPile->nbElts+1;
                // Attention aux indices du tableau
        }
        else{
                printf("La pile est pleine");
        }
}
```



Les piles statiques

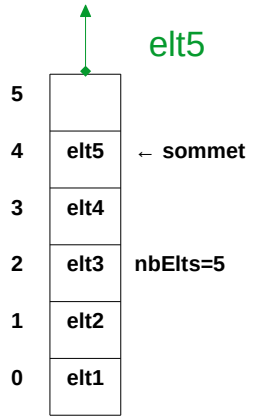
Dépiler un élément (on suppose que la pile est partiellement remplie)

- Le prototype de la fonction depiler()

```
t_element depiler(t_pile* adrPile);
// la pile sera modifiée par la fonction
// la fonction retourne l'élément dépilé
```

- la définition de la fonction depiler()
  - Remarque: la pile n'est pas vide
  - On définira ELTVIDE;

```
t_element depiler(t_pile *adrPile){
        t_element elt;
        if (!(estVide(*adrPile))) {
                elt =adrPile-> tabElts[adrPile-> nbElts-1] ;
                (adrPile-> nbElts)--;
                // nettoyage de la pile
                adrPile->tabElts[adrPile-> nbElts] = ELTVIDE;
        }
        else{
                elt = ELTVIDE;
        }
        return elt;
}
```



Dépiler un élément … suite

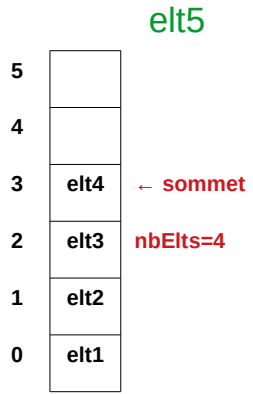
- Le prototype de la fonction depiler()

```
t_element depiler(t_pile* adrPile);
// la pile sera modifiée par la fonction
// la fonction retourne l'élément dépilé
```

- la définition de la fonction depiler()

  - Remarque: la pile n'est pas vide

```
t_element depiler(t_pile *adrPile){
        t_element elt;
        if (!(estVide(*adrPile))) {
                elt =adrPile-> tabElts[adrPile-> nbElts-1] ;
                (adrPile-> nbElts)--;
                // nettoyage de la pile
                adrPile->tabElts[adrPile-> nbElts] = ELTVIDE;
        }
        else{
                elt = ELTVIDE;
        }
        return elt;
}
```



Les piles statiques

Initialiser la pile:

- La fonction retournera une pile vide:
  - Nombre d'éléments à 0
  - Initialisation de tous les éléments de la pile
- Le prototype de la fonction initialiser():

```
t_pile initialiser();
```

- Définition de la fonction initialiser()
  - La pile est vie

```
t_pile initialiser() {
        t_pile p;
        for (int i=0; i<MAX; i++)
                p.tabElts[i] = ELTVIDE; // tous les éléments de la pile sont initialisés
        p.nbElts=0 ;
        return p;
}
```

Les piles statiques

Vider la pile:

- La procédure initialisera tous les éléments «non vide» de la liste:
- Le prototype de la procédure vider():

```
void vider(t_pile *adrPile);
```

- Définition de la fonction vider()

- Il faudra modifier uniquement les éléments non vide en partant par exemple du sommet de la pile.

```
void vider(t_pile *adrPile){
	for(int i = adrPile->nbElts -1; i >= 0; i--){
		adrPile-> tabElts[i] = ELTVIDE;
	}
	adrPile-> nbElts =0 ;
}
```

- Autre solution :

```
void vider2(t_pile *adrPile){
	while (adrPile->nbElts >=0){
		adrPile->tabElts[adrPile->nbElts-1] = ELTVIDE;
		(adrPile->nbElts)-- ;
	}
}
```

Récupérer le sommet de la pile

- La fonction sommet() retourne le dernier élément de la pile … sans dépiler.
- Prototype de la fonction sommet()

```
t_element sommet();
```

- Définition de la fonction sommet()

```
t_element sommet(t_pile p) { // la pile est un paramètre d'entrée
        t_element elt;
        if (!(estVide(p))) {
                // le sommet de la pile est à l'indice p.nbElts - 1
                elt = p.tabElts[p.nbElts-1];
        }
        else{
                elt = ELTVIDE;
        }
        return elt;
}
```

Les piles statiques

La pile est-elle vide? Pleine?

- La fonction estvide()

```
int estVide(t_pile p){
    return (p.nbElts == 0);
}
```

- La fonction estPleine()

```
int estPleine(t_pile p){
    return (p.nbElts==MAX);
}
```

Les piles statiques

… simplification du code : utilisation de variables intermédiaires

- Bien faire la différence entre tableaux et variables

```
void empiler1(t_pile *adrPile, t_element elt){
        // définition et initialisation des variables intermédiaires
        t_element *tab = adrPile->tabElts;// tab est une adresse
        int *adrNb = &(adrPile->nbElts);

        if (!(estPleine(*adrPile))){
                tab[*adrNb]=elt ;
                (*adrNb)++;
        }
        else{
                printf("La pile est pleine");
        }
}
```

Les piles statiques

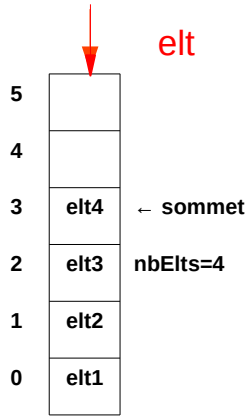
… simplification du code : utilisation de variables intermédiaires

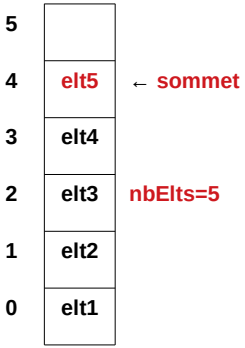
```
t_element depiler1(t_pile *adrPile){
        t_element elt;
        // définition et initialisation des variables intermédiaires
        t_element *tab = adrPile->tabElts;
        int *adrNb = &(adrPile->nbElts);

        if (!(estVide(*adrPile))){
                elt =tab[*adrNb-1] ;
                (*adrNb)--;
                tab[*adrNb] = ELTVIDE;
        }
        else{
                elt = ELTVIDE;      //à définir
        }
        return elt;
}
```

Les piles statiques

Les files (LIFO) :

- Enfiler = Empiler





Les piles statiques

Les files (LIFO) :

- Défiler