Université d'Aix-Marseille III - Faculté des Sciences et Techniques de Saint-Jérôme

Licence Sciences et Technologies - Mention Maths-Info

I3 - Algorithmique et Programmation

TD n°5

1. Liste chaînée et pile

Nous voulons créer un programme où un utilisateur peut saisir un nombre illimité d'entiers au clavier puis afficher tous les entiers dans l'ordre inverse de leur saisie.

Dans ce but, nous allons créer une pile d'entiers. Nous aurons besoin d'effectuer 4 opérations sur la pile : tester si la pile est vide, lire l'entier se trouvant au sommet de la pile, empiler un entier sur la pile ou dépiler un entier de la pile.

Nous allons représenter la pile sous forme d'une liste chaînée. Un chaînon de la liste sera donc défini ainsi :

```
struct ch
{
  int entier;
  struct ch * suiv;
};
typedef struct ch chainon;
```

Nous utiliserons alors le type suivant pour représenter les piles d'entiers :

```
typedef chainon * pile;
```

Pour une pile, il n'y a besoin que de pouvoir accéder au premier chaînon de la liste chaînée. Pour mémoriser la pile, on utilisera donc un pointeur sur le premier chaînon. Etant donnée une variable p de type `pile`, l'allocation de mémoire pour la variable dynamique pointée par p s'effectuera grâce à la ligne d'instruction :

```
p = (pile) malloc (sizeof (chainon));
```

1.1. Ecrire un programme qui initialise une pile d'entiers, empile 2 entiers saisis au clavier et les dépile pour les afficher dans l'ordre inverse de leur saisie. N'utilisez ni fonction ni procédure dans ce programme.

1.2. Ecrire la fonction *init* qui initialise à vide une pile.

1.3. Ecrire la fonction *pile_vide* qui prend une pile en paramètre et renvoie 1 si elle est vide et 0 sinon.

1.4. Ecrire la fonction *sommet* qui prend une pile en paramètre et retourne l'entier se trouvant au sommet de cette pile.

1.5. Ecrire la fonction *empile* qui prend une pile et un entier en paramètre, empile l'entier sur la pile et retourne la nouvelle pile. Faites-en une version procédurale.

1.6. Ecrire la fonction *depile* qui prend une pile en paramètre, retire son sommet et retourne la nouvelle pile. On supposera la pile non vide. Faites-en une version procédurale.

1.7. En utilisant une pile et les fonctions et procédures précédentes, écrire un programme C qui saisit des entiers au clavier jusqu'à ce que l'utilisateur entre l'entier zéro, puis affiche les entiers dans l'ordre inverse de leur saisie.

2. Liste chaînée et ensemble

Nous voulons effectuer des opérations simples sur des ensembles d'entiers : initialiser un ensemble, ajouter ou retirer un élément, calculer la cardinalité d'un ensemble, tester si deux ensembles sont égaux. Dans ce but, nous allons créer le type structuré défini en C de la façon suivante :

```
struct maillon
{
   int entier;
   struct maillon * suiv;
};
typedef struct maillon * ensemble;
```

Etant donnée une variable ens de type ensemble, l'allocation de mémoire pour la variable dynamique pointée par ens s'effectuera grâce à la ligne d'instruction :

```
ens = (ensemble) malloc  (sizeof (struct maillon));
```

2.1. Ecrire la procédure *initialise* qui permet d'initialiser un ensemble à $\emptyset$.

2.2. Ecrire la fonction *cardinal* qui calcule la cardinalité d'un ensemble.

2.3. Ecrire la fonction *appartient* qui retourne 1 si un entier (passé en paramètre) appartient à un ensemble (passé en paramètre) et retourne 0 sinon.

2.4. Ecrire la procédure *ajoute* qui permet d'ajouter un entier à un ensemble. Si l'entier fait déjà partie de l'ensemble, l'ensemble restera inchangé.

2.5. Ecrire la procédure *enleve* qui permet d'enlever un entier d'un ensemble. Si l'entier ne fait pas partie de l'ensemble, l'ensemble restera évidemment inchangé.

2.6. Ecrire un programme qui crée un ensemble vide, lui ajoute tous les multiples de 4 inférieurs à N, puis lui retire tous les entiers qui sont des multiples de 3, puis affiche son cardinal.

2.7. A partir de cette question, on considère que la liste chaînée est ordonnée dans l'ordre croissant. Ecrire la fonction *appartient* qui retourne 1 si un entier (passé en paramètre) appartient à un ensemble (passé en paramètre) et retourne 0 sinon.

2.8. Ecrire la procédure *ajoute* qui permet d'ajouter un entier à un ensemble. Si l'entier fait déjà partie de l'ensemble, l'ensemble restera simplement inchangé.

2.9. Ecrire la fonction *egal* qui, étant donnés deux ensembles, renvoie 1 si les deux ensembles sont égaux, 0 sinon.

3. Liste chaînée et file

Nous voulons effectuer des opérations simples sur des files d'entiers : initialiser une file, tester si une file est vide, ajouter ou retirer un élément de la file. Dans ce but, nous allons utiliser les types suivants :

```
struct maillon
{
  int entier;
  struct maillon * suiv;
};

typedef struct maillon * liste;

typedef struct
{
  liste tete;     /* tete de la file */
  liste queue;    /* queue de la file */
} file;
```

Pour une file, l'ajout s'effectue en queue de file tandis que la suppression d'un élément est réalisée en tête de file.

1.1. Ecrire la procédure *initialise* qui permet d'initialiser une file à vide.

1.2. Ecrire la fonction *vacuite* qui teste si la file est vide. Elle retourne 1 si la file est vide, 0 sinon.

1.3. Ecrire la procédure *ajout* qui ajoute un entier en queue de file.

1.4. Ecrire la procédure *suppression* qui supprime l'entier situé en tête de file.

1.5. Ecrire un programme qui crée une file, lui ajoute 3 entiers saisis au clavier, puis affiche tous les entiers de cette file en les supprimant au fur et à mesure.