UNIVERSITÉ TOULOUSE III PAUL SABATIER

L2 CUPGE 2020/2021

Contrôle Terminal – 1ère session

1h30 - Documents non autorisés

Nota : L'ordre de traitement des exercices n'importe pas.

Partie 1 (4 pts)

1. Donner trois exemples de structures de données linéaires ? (1,5 pt)
2. Donner la définition d'une pile et d'une file (1,5 pt)
3. Soit le code suivant :

```
#include <stdio.h>
#include <stdlib.h>


int main() {
            char *a;
            a = "EXAMEN";
            printf("%s\n", a);
            printf("%c\n", *a);
            return(1);
}
```

Qu'affiche ce code ? (1 pt)

Partie 2 – NPI / Notation Polonaise Inverse (5 pts)

La notation Polonaise Inverse (NPI) permet d'écrire une expression arithmétique sans avoir besoin d'utiliser des parenthèses. Par exemple, l'expression « (3+2) * (4+5) » pourra s'écrire « 3 2 **+** 4 5 **+** * »

Pour écrire cette expression, il est nécessaire d'utiliser une pile. Nous prendrons comme hypothèse de départ afin de simplifier le problème les éléments suivants que :

- seuls les nombres compris entre **0** et **9** sont possibles
- seuls les opérateurs **+** et * sont définis
- l'expression est terminée par le caractère « **.** »
- l'expression est stockée dans une chaîne de caractères. Le 1er élément du tableau est le 1er élément de l'expression. Ceci donne pour l'exemple précédent

3	2	+	4	5	+	*	.

- Nous supposerons l'existence préalable des fonctions de gestion de la pile suivante :
  - `pile creer_pile();`
  - `void empiler(pile p, int v);`
  - `int depiler(pile p);`
  - `void liberer_pile(pile p);`

1. Définir la structure de données de l'expression **e** contenant une expression en notation polonaise inverse (NPI) (1 pt)
2. Ecrire l'algorithme de la fonction `int eval(expression e)` qui permet d'évaluer et de donner le résultat d'une expression en NPI (2 pts)
3. On désire maintenant gérer les nombres complexes écrits sous la forme `a+ib`. Les opérations restent les mêmes que précédemment. Le nombre complexe `1+2i` s'écrira en NPI : `12i*+`
   Modifier l'algorithme précédent pour tenir compte de cette possibilité. Indiquer explicitement les parties modifiées (2 pts).

   Nota : il faut utiliser deux piles

../..

Partie 3 – pile sans face … (4 pts)

1. Supposons qu'avec une pile **S** vide au départ, on a effectué au total : 25 opérations « *empiler* », 12 opérations « *sommet* », et 10 opérations « *dépiler* », dont 3 ont généré une erreur « *Pile Vide* ». Quelle est la taille actuelle de la pile S ? (2 pts)
2. Écrire une fonction qui prend deux piles ordonnées d'entiers A et B (minimum placé au sommet) et qui crée une seule pile ordonnée (minimum placé au sommet) en utilisant uniquement les opérations « classiques » sur une pile (**pop, push, isEmpty** et **top**). Aucune autre structure de données n'est permise pour le traitement. (2 pts)

Partie 4 – liste (en)chaînée … (7 pts)

Soit les déclarations C suivantes :

```
typedef struct Node {
      int data;
      struct Node* next;
} Node;

typedef Node* List;
List head = NULL;
```

1. Écrire un programme principal qui permet d'insérer les éléments 0, 1, 4, 9, 16 et 25 (dans cet ordre), dans la liste **head**. (2 pts)
2. Écrire une fonction qui affiche dans l'ordre, les éléments impairs de la liste **head**. (1 pt)
3. Écrire une fonction <u>récursive</u> qui supprime tous les éléments pairs de la liste **head**. (1 pt)
4. Écrire une fonction qui teste si la liste **head** contient un élément donné « **x** ». (1 pt)
5. Écrire une fonction qui permet de transformer la liste **head** en une liste circulaire. (1 pt)
6. Sachant maintenant que la liste **head** est circulaire, écrire une fonction qui affiche sa longueur (c'est-à-dire le nombre de ses éléments). (1 pt)