Premier cycle Vendredi 29 Juin 2007

I3 - Algorithmique et Langage C

Durée : *3h00*
Documents autorisés : **AUCUN** (calculatrice comprise)

Remarques :
- Veuillez lire attentivement les questions avant de répondre.
- Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
- Rendez une copie propre.

1 Position d'une sous-chaîne (4 points)

Donnez le corps de la fonction suivante qui retourne l'indice de la première occurrence de $sousChaine$ dans $chaine$. Si $sousChaine$ n'est pas présente dans $chaine$, alors la fonction retourne -1.
- **fonction** position (chaine,sousChaine : **Chaîne de caractères) : Entier**

Vous pouvez utiliser les fonctions suivantes :
- **fonction** longueur (uneChaine : **Chaîne de caractères) : Naturel**
  ...avec $longueur("") = 0$
- **fonction** iemeCaractere (uneChaine : **Chaîne de caractères**, iemePlace : **Naturel) : Caractère**
  ...avec $0 < iemePlace \leq longueur(uneChaine)$

2 Palindrome (3 points)

Une chaîne de caractères est un palindrome si la lecture de gauche à droite et de droite à gauche est identique. Par exemple "radar", "été", "rotor", etc.

En plus des deux fonctions pécédentes, vous pouvez utiliser la fonction suivante :
- **fonction** sousChaine (uneChaine : **Chaîne de caractères**, debut, longueur : **Naturel) : Chaîne de caractères**
  ...avec :
  - $0 < debut \leq longueur(uneChaine)$,
  - $debut + longueur \leq longueur(uneChaine)$,
  - `sousChaine(s,d,0)=""`

Écrire une fonction **récursive** qui permet de savoir si une chaîne est un palindrome.

3 Pile et expression arithmétique (8 points)

On suppose dans cet exercice que l'on possède :
- Le type *Operateur* :
  - **Type** Operateur **=** {addition,soustraction,multiplication,division}
- Le type *ElementEA* (pour Élément d'une Expression Arithmétique) avec les opérations suivantes :

– **fonction** creerElementEAAPartirDUnReel (a : **Réel) :** ElementEA
– **fonction** creerElementEAAPartirDUnOperateur (op : Operateur) **:** ElementEA
– **fonction** estUnOperateur (e : ElementEA) **: Booléen**
– **fonction** obtenirOperateur (e : ElementEA) **:** Operateur
– **fonction** obtenirNombre (e : ElementEA) **: Réel**

3.1 Pile d'éléments d'expression arithmétique (3,5 points)

Définition

Une pile est un type de données qui permet de stocker des éléments avec un accès LIFO (Last In First Out, Dernier Entré Premier Sortie). Il peut être schématisé à l'aide de la figure 1 (ici les nombres 1 puis 2 ont été empilés).

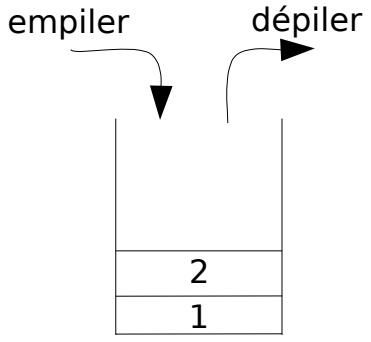

FIG. 1 – une pile

3.1.1 Utilisation d'une pile d'éléments d'expression arithmétique (0,5 points)

Soit le type *PileDElementsEA* avec les fonctions et procédures suivantes :
– **fonction** creerPileDElementsEA () **:** PileDElementsEA
…qui permet de créer une pile vide
– **fonction** estVide (laPile : PileDElementsEA) **: Booléen**
…qui permet de savoir si une pile est vide
– **procédure** empiler **(** **E/S** laPile : PileDElementsEA , **E** e : ElementEA **)**
…qui permet d'empiler *e* dans *laPile*
– **procédure** depiler **(** **E/S** laPile : PileDElementsEA , **S** e : ElementEA **)**
…qui permet de dépiler le sommet de la pile (on obtient cet élément à l'aide du paramètre formel *e*).

Soit `p` de type *PileDElementsEA*. Dessinez `p` (à l'image de la figure 1) à l'issue des instructions suivantes :

p ← creerPileDElementsEA()

empiler(p,creerElementEAAPartirDUnReel(5))

empiler(p,creerElementEAAPartirDUnReel(3))

empiler(p,creerElementEAAPartirDUnOperateur(addition))

empiler(p,creerElementEAAPartirDUnReel(2))

empiler(p,creerElementEAAPartirDUnOperateur(multiplication))

3.1.2 Conception (3 points)

On se propose de concevoir le type *PileDElementsEA* à l'aide d'une structure :
Type PileDElementsEA **= Structure**
lesElementsEA : **Tableau[**1...MAX**] d'**ElementEA
nbElementsEA : **Naturel**
finstructure
Donnez le corps des fonctions et procédures précédentes : *creerPileDElementsEA*, *estVide*, *empiler*, *depiler* (on considère qu'on utilise correctement la pile, on ne traite pas les cas d'erreurs).

3.2 Évaluation d'une expression arithmétique (4,5 points)

Comme l'indique la section 3.1.1, on peut stocker une expression arithmétique en notation polonaise inversée (nommée aussi postfixe) à l'aide d'une pile.

1. Donnez la représentation en notation polonaise inversée de l'expression arithmetique $(2+3) \times (4+5)$.
2. Dessinez une pile stockant cette expression.
3. À l'image de la suite d'instructions de la section 3.1.1, donnez la liste d'instructions qui permet de stocker l'expression arithmétique $(2+3) \times (4+5)$.
4. Donnez le corps de la procédure suivante qui permet d'évaluer une expression arithmétique stockée dans une pile en dépilant ses éléments. Le paramètre erreur permet de savoir si l'expression arithmétique stockée dans la pile est correcte.
   - **procédure** evaluer **(E/S** lExpression : PileDElementsEA , **S** valeur : **Réel**, erreur : **Booléen** **)**

4 Un peu de C (2 points)

Soit le programme C suivant :

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #define TAILLE_BUFFER 100
4
5  char* saisirChaine() {
6    char buffer[TAILLE_BUFFER];
7    scanf("%s",buffer);
8    return buffer;
9  }
10
11 void afficherChaine(char* chaine) {
12   printf("Vous avez tapé %s\n",chaine);
```

```
13 }
14
15 int main(int argc, char** argv) {
16   afficherChaine(saisirChaine());
17   return EXIT_SUCCESS;
18 }
```

On compile ce programme :

```
> gcc -c -Wall -pedantic main.c
main.c: In function 'saisirChaine':
main.c:8: warning: function returns address of local variable
```

Répondez aux questions suivantes (à chaque fois en 10 lignes maximum) :

1. Que signifient les options `-c`, `-Wall` et `-pedantic` de *gcc* ?
2. Que signifie le *warning* affiché par *gcc* ? Expliquez comment résoudre ce problème (donnez le code C) ?

5 Et pour finir, encore du C (3 points)

Qu'affiche le programme C suivant :

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

void f1 (int a, int *b) {
  a = *b;
}

void f2 (int *b, int c) {
  *b = c;
}

void f3 (int *a, int c) {
  *a=c+1;
}

int main (int argc, char** argv) {
  int v1, v2,v3;
  v1 = 5; v2 = 8; v3 = 10;
  printf ("Au depart : v1 = %d  v2 = %d  v3 = %d\n", v1, v2, v3);
  f1 (v1, &v2);
  printf ("Apres f1 : v1 = %d  v2 = %d  v3 = %d\n", v1, v2, v3);
  f2 (&v3, *(&v1));
  printf ("Apres f2 : v1 = %d  v2 = %d  v3 = %d\n", v1, v2, v3);
  f3 (&v3, v2);
  printf ("Apres f3 : v1 = %d  v2 = %d  v3 = %d\n", v1, v2, v3);
  return EXIT_SUCCESS;
}
```