Université Bordeaux 1 Licence Informatique 2010/2011

Algorithmique 1

Interrogation 1 groupe CSB3A2

NOM :	Prénom

Exercice 1

On considère ici les listes simplement chaînées d'entiers : `listeSC-ent`.
On écrira les fonctions demandées en utilisant exclusivement les primitives données en annexe.

1.1 Ecrire une fonction `ecartMax` qui retourne la différence entre le maximum et le minimum des éléments d'une liste d'entier `L` (la liste est supposée non vide).

1.2 Ecrire une fonction `tabToListe` qui, à partir d'un tableau d'entiers `T` indicé de `1` à `N`, construit une liste simplement chaînée d'entiers contenant les éléments de `T` dans le même ordre.

1.3 Ecrire une fonction `supprimeDernier` qui supprime le dernier élément de la liste L (supposée non vide).

1.4 Ecrire une fonction `insereRang` qui insère l'entier `x` dans la liste `L` de sorte qu'il occupe le rang `k` dans la liste. On supposera la valeur de `k` cohérente avec la liste ($1 \leq k \leq$ longueur(`L`)+1).

Exercice 2

Ecrire 2 versions d'une fonction qui calcule x^n par la méthode alexandrine pour x réel et n entier positif : une version récursive non terminale et une version récursive terminale.
On rappelle la version itérative :

Itérative

```
fonction puissIter(val x :réel, val n :entier) :réel ;
var p :réel
début
  p=1 ;
  tant que n>0 faire
    si estImpair(n) alors
      p=p*x ;
    finsi
    x=x*x ;
    n=n/2 ;
  fintantque
  retourner(p) ;
fin
```

Récursive non terminale

Récursive terminale

Exercice 3

On considère dans cet exercice les listes doublement chaînées d'entiers implémentées par allocation dynamique : `listeDC-ent`

Ecrire deux versions de la fonction `insererEnFin` qui insère l'entier `x` à la fin de la liste `L` :

- ✓ une version utilisant les primitives données en annexe,
- ✓ une version utilisant directement l'implémentation (sans utiliser les primitives).

Annexe

Liste simplement chaînée d'entiers

```
cellule = structure
          valElem : entier ;
          pointSuiv : ^cellule ;
          finstructure

curseur = ^cellule

listeSC_ent : structure
              premier : curseur ;
              cle : curseur ;
              finstructure

fonction creerListe(ref L : liste_SC_ent) : vide
fonction debutListe(ref L : liste_SC_ent) : vide
fonction valeur(val L : liste_SC_ent) : entier
fonction suivant(ref L : liste_SC_ent) : vide
fonction listeVide(val L : liste_SC_ent) : booleen
fonction insererEnTete(ref L : liste_SC_ent ; val x :entier) : vide
fonction insererApres(ref L : liste_SC_ent ; val x :entier) : vide
fonction supprimerEnTete(ref L : liste_SC_ent) : vide
fonction supprimerApres(ref L : liste_SC_ent) : vide
fonction estFinListe(val L : liste_SC_ent) : booleen
fonction estDernier(val L :listeSC_ent) :booleen
```

Liste doublement chaînée d'entiers

```
cellule = structure
          valElem : entier ;
          pointPrec : ^cellule ;
          pointSuiv : ^cellule ;
          finstructure

curseur = ^cellule

listeDC_ent : structure
              premier : curseur ;
              dernier : curseur ;
              cle : curseur ;
              finstructure

fonction creerListe(ref L : liste_DC_ent) : vide
fonction debutListe(ref L : liste_DC_ent) : vide
fonction finListe(ref L : liste_DC_ent) : vide
fonction valeur(val L : liste_DC_ent) : entier
fonction suivant(ref L : liste_DC_ent) : vide
fonction precedent(ref L : liste_DC_ent) : vide
fonction listeVide(val L : liste_DC_ent) : booleen
fonction insererEnTete(ref L : liste_DC_ent ; val x :entier) : vide
fonction insererApres(ref L : liste_DC_ent ; val x :entier) : vide
fonction supprimerEnTete(ref L : liste_DC_ent) : vide
fonction supprimerApres(ref L : liste_DC_ent) : vide
fonction estFinListe(val L : liste_DC_ent) : booleen
```