Algorithmique 1 : Devoir Surveillé 1

Listes

Durée : 40mn
Sans documents

Exercice 1.1 *Listes simplement chaînées*

Écrire une fonction
`tab_to_listeSC_car(val T: tableau de car, val n: int): listeSC_car`
qui à partir d'un tableau de `n` objets du type `car` produit une liste simplement chaînée du type `listeSC_car`.
On respectera l'ordre des éléments et on utilisera les primitives de table 1.
Par exemple pour le tableau `T={'a','b','c','d','e','f','g','h'}`, le premier élément de la liste contiendra la valeur 'a', le deuxième, 'b' et etc., le dernier sera 'h'.

Exercice 1.2 *Listes doublement chaînées*

Dans cet exercice on considère le type `listeDC_car`, liste doublement chaînée implémentée par l'allocation dynamique vue en cours.
Le rappel sur la définition et l'ensemble de primitives est donné sur table 2.

1. Écrire une fonction `afficher_etat_listeDC(val l: listeDC_car):vide` qui pour une liste `l` passée en paramètre affiche les valeurs pointées par le premier, le dernier et l'élément clé de la liste `l`.
   On supposera donnée une fonction `put_car(val c: car): vide` qui affiche la valeur de l'objet du type `car` passé en paramètre.
2. Écrire une fonction `afficher_liste_de_caracteres(val l: listeDC_car):vide` qui affiche l'ensemble d'objets contenus dans `l`.
   Par exemple le résultat de la suite de fonctions suivantes

```
creerListe(l);
insererEnTete(l,'a');
insererEnTete(l,'a');
afficher_etat_listeDC(l);
afficher_liste_de_caracteres(l);
```

   sera

```
premier= a cle=  a dernier=  a
 a  a
```

3. Soit la fonction `main` suivante :

```
fonction main(): vide
var l: listeDC_car;
debut
  creerListe(l);
  insererEnTete(l,'a');
  insererEnTete(l,'a');

  afficher_etat_listeDC(l);
  afficher_liste_de_caracteres(l);

  insererEnTete(l,'b');
  insererEnTete(l,'c');
  insererApres(l,'e');
  insererApres(l,'a');

  afficher_etat_listeDC(l);
  afficher_liste_de_caracteres(l);

  ajout_en_queue_sans_doublons(l, 'x');

  afficher_etat_listeDC(l);
  afficher_liste_de_caracteres(l);

  ajout_en_queue_sans_doublons(l, 'e');

  afficher_etat_listeDC(l);
  afficher_liste_de_caracteres(l);

  supprimer_derniere_occurrence(l, 'o');
  afficher_etat_listeDC(l);
  afficher_liste_de_caracteres(l);

  supprimer_derniere_occurrence(l, 'a');
  afficher_etat_listeDC(l);
  afficher_liste_de_caracteres(l);
fin
```

Donner l'affichage résultat de l'exécution de cette fonction.
On supposera connues les fonctions
`ajout_en_queue_sans_doublons(ref L: listeDC_car):vide` et
`supprimer_derniere_occurrence(ref L: listeDC_car):vide`
vues en TD.

Type	opération	primitives
Liste simplement chaînée ListeSC		
	accès	fonction valeur(val L : liste d'objet) : objet ; fonction debutListe(ref L : liste d'objet) : vide ; fonction suivant(ref L : liste d'objet) : vide ; fonction listeVide(val L : liste d'objet) : booléen ; fonction getCléListe(val L : liste d'objet) : curseur ;
	modification	fonction créerListe(ref L : liste d'objet) : vide ; fonction insérerAprès(ref L : liste d'objet, val x : objet) : vide ; fonction insérerEnTête(ref L : liste d'objet, val x : objet) : vide ; fonction supprimerAprès(ref L : liste d'objet) : vide ; fonction supprimerEnTête(ref L : liste d'objet) : vide ; fonction setCléListe(ref L : liste d'objet, val c : curseur) : vide ; fonction détruireListe(ref L : liste d'objet) : vide ;
Implémentation par tableau		
élémentListe= structure valeur : car ; indexSuivant : curseur ; finstructure ; stockListe= tableau[1..tailleStock] d'élémentListe ;		
listeSC_Car= structure tailleStock : entier ; vListe : stockListe ; premier : curseur ; premierLibre : curseur ; clé : curseur ; finstructure	 redef modif gestion	 fonction créer_liste(ref L : listeSC_Car) : vide ; fonction insérerAprès(val x : car, val L : listeSC_Car) : booléen ; fonction prendreCellule(ref L : listeSC_Car) : curseur ; fonction mettreCelluleEnTête(ref L : listeSC_Car, val P : curseur) : vide ;
Implémentation dynamique		
cellule= structure valeurElement : car ; pointeurSuivant : ^cellule ; finstructure		
listeSC_car= structure premier : curseur ; clé : curseur ; finstructure	redef accès redef modif gestion	fonction premier(ref L : listeSC_car) : vide ; fonction contenu(val L : listeSC_car) : car ; créer_liste(ref L : listeSC_car) : vide ; fonction new(ref c : ^ cellule) : vide ; fonction delete(ref c : ^ cellule) : vide ;

TAB. 1 – Listes simplement chaînées.

Type	opération	primitives
Liste doublement chaînée ListeDC		
	accès	fonction valeur(val L : liste d'objet) : objet ; fonction debutListe(ref L : liste d'objet) : vide ; fonction finListe(ref L : liste d'objet) : vide ; fonction suivant(ref L : liste d'objet) : vide ; fonction précédent(ref L : liste d'objet) : vide ; fonction listeVide(val L : liste d'objet) : booléen ; fonction getCléListe(val L : liste d'objet) : curseur ;
	modification	fonction créerListe(ref L : liste d'objet) : vide ; fonction insérerAprès(ref L : liste d'objet, val x : objet) : vide ; fonction insérerEnTête(ref L : liste d'objet, val x : objet) : vide ; fonction supprimerAprès(ref L : liste d'objet) : vide ; fonction supprimerEnTête(ref L : liste d'objet) : vide ; fonction setCléListe(ref L : liste d'objet, val c : curseur) : vide ; fonction détruireListe(ref L : liste d'objet) : vide ;
Implémentation dynamique		
cellule= structure valeurElement : car ; pointeurSuivant : ^cellule ; pointeurPrécédent : ^cellule ; finstructure		
listeSC_car= structure premier : curseur ; dernier : curseur ; clé : curseur ; finstructure	gestion	fonction new(ref c : ^ cellule) : vide ; fonction delete(ref c : ^ cellule) : vide ;

TAB. 2 – Listes doublement chaînées.