Cartouche du document

Année : ING 1
Matière : Programmation C
Activité : Examen

Objectifs

Cet examen teste vos aptitudes en programmation en C
- à traduire en C des algorithmes
- à corriger les erreurs d'un source C
- à manipuler les pointeurs
- à utiliser la gestion bufferisée de fichiers en C

Tous les documents (electronique et autres) sont autorisés.

La durée de l'examen est de 1h20 heures.

L'examen est assez long. Vous êtes donc notés sur 24.

Sommaire des exercices

Corps des exercices

1 - Tri fusion (7pts)

Enoncé :

Soit le type abstrait Liste vu en TD, sur lequel nous ajoutons l'opération d'extension réalisant l'algorithme de Tri Fusion ci dessous:

```
//L'opération FusionnerListesTriees a été vue et corrigée en TD d'algo
Constructeur Liste: triFusion (Liste l) : Liste
Références locales
Liste l1,l2

Debut
        l1 <- listeVide()
        l2 <- listeVide()
        Si split(L1,L2) Alors
        Debut
                retrouner fusionnerListesTriees(triFusion(l1),trifusion(l2))
        Fin
        Sinon
        Debut
```

```
                retourner l
        Fin
Fin

//On découpe la liste observée en deux sous listes
//contenant chacune la moitié des éléments (1 sur 2).
//L1 et L2 sont deux paramètres modifiés par cette opération
//L'opération retourne faux si la liste de départ est vide
Observateur Liste: split(Liste l1,Liste l2) : Booleen
Observé l

Références locales
Liste ltemp
Element e

Debut
        ltemp <- l
        Si estVide(ltemp) Alors
        Debut
                retourner Faux()
        Fin
        Sinon
        Debut
                Faire
                                Debut
                                        e <- premier(Ltemp)
                                        ajouter(l1,e)
                                        ltemp <- reste(ltemp)
                                        Si Non estVide(Ltemp) Alors
                                        Debut
                                                e <- premier(ltemp)
                                                ajouter(l2,e)
                                        Fin
                                Fin
                Tant que Non estVide(ltemp)
        Fin
Retourner Vrai()
Fin
```

Question 1)

Enoncé de la question

Implémenter en C, les opérations triFusion, split, fusionListesTriees. On pourra réutiliser des sources .h et .c faits en TDs et édités sur le site Arel.

On précisera bien ce qui doit être mis dans les fichiers d'entête (.h) et les fichiers sources (.c).

Solution de la question

Le fichier *trifusion.ta.c*

2 - Les pointeurs (7pts)

Enoncé :

On désire implémenter en C, une liste linéaire doublement chaînée (suivant et précédent). Pour cela on utilisera les types suivants :

```
struct SCell
{
        struct SCell * pprec;
        void * pinfo;
```

```
        struct SCell * psuiv;
} SCell;

typedef struct SCell * Cellule;

struct
{
        Cellule premiereCellule;
} SListe;

typedef struct SListe * Liste;
```

Question 1)

Enoncé de la question

Ecrire en C, la fonction qui permet de créer une liste linéaire doublement chaînée vide.

Solution de la question

Question 2)

Enoncé de la question

Ecrire en C, la fonction qui permet d'ajouter un élément en début de liste.

Solution de la question

Question 3)

Enoncé de la question

Ecrire en C, la fonction qui permet d'insérer un élément à une place donnée en paramètre sous forme d'un entier (On indicera la liste à partir de 0 pour la première cellule).

Solution de la question

Question 4)

Enoncé de la question

Ecrire en C, les fonctions resteAvant et resteApres. Ces deux fonctions sont les équivalentes de listeReste mais en tenant compte du double chaînage.

Solution de la question

Réponse globale à tout l'exercice

Le fichier *listedoublementchainee.c*

3 - Quelques erreurs de compilation (4pts)

Enoncé :

Corriger les erreurs de ce source (warning et syntax error). On utilisera la commande gcc avec

l'option -c pour s'arrêter à la traduction du code : **gcc -c ???.c** .

```
int main()
{
        float res1, res2;

        res = 0;
        for(i = 0; i < 10, i++ )
        {
                res1 = res1 + i
                res2 = res2 + i * i;
        }
        res1 /= 10;
        res2 = res2 /10 - res1 * res1;
        afficherIntervalle(res1,res2)
}

void afficherIntervalle(float res1, float res2)
{
        printf("[%f , %f]\n",res1 - sqrt(res2),res1 + sqrt(res2))
}
```

4 - Les fichiers (6pts)

Enoncé :

Dans cet exercice, on manipule un fichier qui contient dans l'ordre en non formaté :
- deux entiers n1 et n2
- n1 nombres réels
- n2 caractères

Question 1)

Enoncé de la question

Ecrire en C, une fonction qui reçoit le nom d'un tel fichier et qui affiche tous les réels puis tous les caractères contenus dans ce fichier.

Solution de la question

Réponse globale à tout l'exercice

Le fichier *fichierIntChar.c*

Annexes (Arel)

```
//L'implémentation du type Liste a été vu en cours
#include <Liste.ta.h>

// Observateur Liste: split(Liste l1,Liste l2) : Booleen
```

```
int split(Liste pl, Liste pl1, Liste pl2);

Liste fusionnerListesTriees(Liste pl1,Liste pl2);

//Constructeur Liste: triFusion (Liste l) : Liste
Liste triFusion(Liste pl)
{
  //Références locales
  //Liste l1,l2
  Liste pl1;
  Liste pl2;

//Debut
  pl1 = listeVide(); // l1 <-- listeVide()
  pl2 = listeVide(); // l2 <-- listeVide()

  if( split(pl,pl1, pl2) ) //Si split(l1,l2) Alors
  { //Debut
    //retourner fusionnerListesTriees(triFusion(l1),trifusion(l2))
    return fusionnerListesTriees(triFusion(pl1),triFusion(pl2));
  } //Fin
  else //Sinon
  { //Debut
    return pl; //retourner l
  } //Fin
//Fin
}

//On découpe la liste observée en deux sous listes
//contenant chacune la moitié des éléments (1 sur 2).
//L1 et L2 sont deux paramètres modifiés par cette opération
//L'opération retourne faux si la liste de départ est vide

// Observateur Liste: split(Liste l2) : Booleen
// Observé l
int split(Liste pl, Liste pl1, Liste pl2)
{
//Références locales
Liste pltemp; // Liste ltemp
Element pe; // Element e

//Debut
  pltemp = pl; //ltemp <Inf/>- l
```

```
  if(listeEstVide(pltemp)) // Si estVide(ltemp) Alors
  { //Debut
    return faux(); // retourner Faux()
  } // Fin
  else // Sinon
  { //Debut
    do // Faire
    { // Debut
      pe = listePremier(pltemp); // e <Inf/>- premier(ltemp)
      listeAjouter(pl1,pe); // ajouter(l1,e)
      pltemp = listeReste(pltemp); // ltemp <Inf/>- reste(ltemp)
      if( ! listeEstVide(pltemp)) // Si Non estVide(ltemp) Alors
      { // Debut
        pe = listePremier(pltemp); // e <Inf/>- premier(ltemp)
        listeAjouter(pl2,pe); // ajouter(l2,e)
      } // Fin
    } // Fin
    while( ! listeEstVide(pltemp)); // Tant que Non estVide(ltemp)
  } // Fin
return vrai(); // Retourner Vrai()
} // Fin
```

```
#include <stdio.h>
#include <malloc.h>
#include <algoC.h>

typedef struct SCell
{
struct SCell * pprec;
void * pinfo;
struct SCell * psuiv;
} SCell;

typedef struct SCell * Cellule;

typedef struct
{
  Cellule premiereCellule;
} SListe;
typedef SListe * Liste;
```

```
Liste listeVide();
Liste listeAjouterDebutListe(Liste pl, Element pe);
Liste listeInserer(Liste pl, int place, Element pe);

Liste listeVide()
{
  Liste pl = malloc(sizeof(SListe));

  pl->premiereCellule = NULL;
  return pl;
}

Liste listeAjouterDebutListe(Liste pl, Element pe)
{
  Cellule pcell = malloc(sizeof(SCell));

  pcell->pprec = NULL;
  pcell->psuiv = pl->premiereCellule;
  pcell->pinfo = pe;

  pl->premiereCellule = pcell;

  return pl;
}

Liste listeInserer(Liste pl, int place, Element pe)
{
  if(place == 0)
  {
    return listeAjouterDebutListe(pl,pe);
  }
  else
  {
    int i;
    Cellule pcell = pl->premiereCellule;
    Cellule pnewCell;

    for(i = 0; i < place ET pcell <> NULL; i++)
      pcell = pcell->pprec;

    // mauvaise place
    if(pcell == NULL)
      return pl;
```

```
    pnewCell = malloc(sizeof(SCell));

    pnewCell->pprec = pcell->pprec;
    pnewCell->pinfo = pe;
    pnewCell->psuiv = pcell;

    if(pcell->pprec <> NULL)
      pcell->pprec->psuiv = pnewCell;

    return pl;
  }
}


Liste listeResteAvant(Liste pl)
{
  if(pl->premiereCellule == NULL)
    return NULL;
  else
  {
    Liste plReste = malloc(sizeof(SListe));

    plReste->premiereCellule = pl->premiereCellule->pprec;
    return plReste;
  }
}

Liste listeResteApres(Liste pl)
{
  if(pl->premiereCellule == NULL)
    return NULL;
  else
  {
    Liste plReste = malloc(sizeof(SListe));

    plReste->premiereCellule = pl->premiereCellule->psuiv;
    return plReste;
  }
}
```

```c
#include <stdio.h>

void afficheFicIntChar(char * nomFic)
{
  FILE * pf;
  int nbCar, nbFloat;
  int i;
  char valCar;
  float valFloat;


  pf = fopen(nomFic,"r");

  if(pf == NULL)
    return;

  fread(&nbFloat,sizeof(int), 1, pf);
  fread(&nbCar,sizeof(int), 1, pf);

  printf("Les nombres réels\n");
  for(i = 0; i < nbFloat; i++) {
    fread(&valFloat,sizeof(float), 1, pf);
    printf("%d) %f\n",i+1,valFloat);
  }

  printf("\n\nLes caractères\n");
  for(i = 0; i < nbCar; i++) {
    fread(&valCar,sizeof(char), 1, pf);
    printf("%d) %c\n",i+1,valCar);
  }

  fclose(pf);
}
```