Examen de Fin de Semestre

(Durée 1h30)

Exercice 1: (3points)

Soit un entier N donnée en base 10, écrire une fonction récursive qui convertit N en binaire.

Exercice 2: (4points)

Soient deux piles[1] P1 et P2 d'entiers, écrire une fonction ***NB_Diff*** qui retourne le nombre d'éléments de P1 qui n'existent pas dans P2 (Ne pas restaurer P1 et P2).

Exercice 3: (13 points)

Soit un texte ***T*** (taille max=1000) constitué de mots séparés par un seul blanc.

(Remarque : Il y a un blanc après le dernier mot)

a) Définir le type ***texte***.

b) Ecrire une fonction ***RechPosMot*** qui recherche dans ***T*** un mot donnée ***M*** et retourne sa position s'il existe sinon retourne -1, ainsi que l'adresse du mot suivant s'il y en a.

Prototype : char * RechPosMot(texte T, char* M, int* pos) ;

Exemple : Le texte ***T*** « Mes meilleurs vœux de santé bonheur et réussite »
Le mot ***M*** « santé » se trouve à la position 5
L'adresse du mot suivant est un pointeur sur le mot « bonheur »
Et si ***M*** est le dernier mot, l'adresse du mot suivant sera ***NULL***

c) Un mot peut se répéter plusieurs fois dans le texte. En utilisant la fonction *RechPosMot*, écrire une fonction ***PositionsMot*** qui étant donnée un mot ***M*** retourne une liste chainée[2] ***Tetepos*** contenant toutes les positions de ***M*** dans le texte ***T.***

Remarques : - Il faut donner la déclaration de la liste,
- Les positions dans *Tetepos* doivent être dans l'ordre croissant, (sans faire de Tri bien sur)

---

[1] Les fonctions sur les piles, dont les prototypes sont donnés ci-dessous, sont supposées prédéfinies :
pile initpile() ; int pilevide(pile p) ; typelem sommetpile(pile p) ;
void empiler(pile *p , typelem x) ; void desempiler(pile *p, typelem* x) ;

[2] Les fonction suivantes : ***liste créer_noeud() ; void ajout_tete(liste * tete, typelem E) ;***
void ajout_Apres(liste* prd, typelem E) ; sont supposées prédéfinies

d) Ecrire une fonction ***ExtraireMots*** qui retourne une liste chainée ***Lmot*** contenant chaque mot du texte ***T*** ainsi que la liste de ces positions.

Remarques : - Il faut donner la déclaration de la liste (avec *element* qui est une structure contenant le mot (taille max=30) et un pointeur (*Tetepos*) sur la liste des positions)
- La liste va contenir même les mots qui se répètent et qui seront supprimés par la suite.
- L'ordre des mots n'a aucune importance

e) Soit la liste ***Lmot*** obtenue précédemment, écrire une fonction ***Suppr_Repet*** qui supprime[3] les mots qui se répètent.

f) En utilisant la liste ***Lmot***, écrire une fonction ***Mot_Frequent*** qui retourne le mot qui se répète le plus dans le texte ***T***.

g) Ecrire le programme qui étant donnée un texte ***T***, extrait les mots et construit la structure de point d'entrée ***Lmot,*** supprime de ***Lmot*** les mots qui se répètent puis affiche le mot le plus fréquent.

Rappel :

strcmp(S1, S2) = 0 si S1=S2 ,
> 0 si S1 suit S2,
< 0 si S1 précède S2

strcpy(S1, S2) copie le mot S2 dans S1
strlen(S) retourne la taille de la chaine sans la marque de fin
strncpy(S1, S2, n) copie n caractères de S2 vers S1

[3] Les fonctions suivantes : ***void supprim_tete(liste *tete) ;***
void supprim_Après(liste prd, liste p) ; sont supposées prédéfinies

Corrigé de l'examen de fin de semestre

Exercice1:

```
int binaire1(int n)
{ if(n!=0) return(n%2+10*binaire(n/2));
  else return 0;
}
```

Exercice2:

```
typedef int  typelem ;
pile Diff_Poly(pile p1, pile p2)
{ typelem x1, x2; pile R=initpile(); int cpt=0; int bool;
  while (!pilevide(p1))
   {  desempiler(&p1,&x1); bool=1;
      while(!pilevide(p2) &&  bool==1)
       {   desemiler(&p2,x2); empiler(&R, x2);
           if (x1==x2) bool=0;
       }
     if (bool==1) cpt++;
    while (!pilevide(R) ) {desmpiler(&R, &x2); empiler(&p2,x2);
   }
}
```

Exercice3:

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

#define maxT 1000
#define maxM  30
typedef char texte[maxT];
typedef char chaine[maxM];

typedef struct ne1 *liste1;
typedef struct ne1 {int element; liste1 svt;} noeud1;

typedef struct { chaine mot; liste1 svt;} element;

typedef struct ne2 *liste2;
typedef struct ne2 {element E; liste2 svt; } noeud2;
```

```
 /**************** Ces fonctions sont non demandées **********************/
liste1 creer_noeud1()
{ liste1 p=(liste1)malloc(sizeof(noeud1));
 if (!p) { printf("erreur d'allocation\n"); exit(-1); }
 return p;
}

liste2 creer_noeud2()
{ liste2 p=(liste2)malloc(sizeof(noeud2));
 if (!p) { printf("erreur d'allocation\n"); exit(-1); }
 return p;
}

void ajout_tete1(liste1 *tete1, int E)
{ liste1 p=creer_noeud1();
 p->element=E;
 p->svt=*tete1;
 *tete1=p;
}

void ajout_tete2(liste2 *tete2, element E)
{ liste2 p=creer_noeud2();
 p->E=E;
 p->svt=*tete2;
 *tete2=p;
}

void ajout_Apres1(liste1 *prd, int E)
{ liste1 p=creer_noeud1();
 p->element=E;
 p->svt=(*prd)->svt;
 (*prd)->svt=p;
 *prd=p;
}
void ajout_Apres2(liste2 *prd, element E)
{ liste2 p=creer_noeud2();
 p->E=E;
 p->svt=(*prd)->svt;
 (*prd)->svt=p;
 *prd=p;
}
/*************************************************************************/
```

```
char *  RechPosMot(texte T, chaine M, int *pos)
{ int j, trouv=0;
  char *p=T;
     while((*p)!='\0' && trouv==0)
    { j=0;
      while (M[j]==(*p) && M[j]!='\0' && (*p)!=' ') {j++; p++; }
      if (M[j]=='\0' && (*p)==' ') trouv=1;
      while ((*p)!=' ') p++;
      (*pos)++;      p++;
    }
    if (trouv==1)
        { if ((*p)=='\0') return NULL;        else return p;      }
    else { *pos=-1; return NULL; }
}

liste1 PositionMot(texte T,chaine M)
{ liste1 tetepos=NULL, prd=NULL;
  int pos=0;
  char *p;
  p=RechPosMot(T,M,&pos);
  if (pos !=-1) ajout_tete1(&tetepos,pos);
  prd=tetepos;
  while (p!=NULL)
  { p=RechPosMot(p,M,&pos);
    if (pos!=-1) ajout_Apres1(&prd,pos);
  }
  return tetepos;
}

/*********************  Non demandées********* ************************/
void affichePos(liste1 tetepos)
{    while(tetepos!=NULL){ printf("%d\t",tetepos->element); tetepos=tetepos->svt; }
    printf("\n");
}
void affiche(liste2 lmot)
{ liste1 p;
   while (lmot!=NULL)
   { printf("%s\t",lmot->E.mot);
     affichePos(lmot->E.svt);
     lmot=lmot->svt;
     printf("\n");
   }
}
/*************************************************************************/
```

```
liste2 ExtraireMots(texte T)
{ liste2 lmot=NULL;    int i=0,j;    chaine M; element E;
  while(T[i]!='\0')
  { j=0;
    while(T[i]!=' ')    { M[j]=T[i]; i++; j++; }
    M[j]='\0';     strcpy(E.mot,M);    E.svt=PositionMot(T,M);
    ajout_tete2(&lmot,E);
    i++;
  }
  return lmot;
}

void liberer(liste1 *tetepos)
{ liste1 p;
    while (*tetepos!=NULL)
    { p=*tetepos;      *tetepos=(*tetepos)->svt;
      free(p);
    }
}

/********************** Non demandée ***************************/
void supprim_Apres(liste2 prd, liste2 p)
{ prd->svt=p->svt;
  liberer(&p->E.svt);
  free(p);
}
/*****************************************************************/
void supprim_repetition(liste2 lmot)
{ liste2 p, q,prd=lmot;
   while (lmot!=NULL)
     { q=lmot->svt;
       while (q!=NULL)
         { if (strcmp(lmot->E.mot,q->E.mot)==0) { supprim_Apres(prd,q); q=prd->svt;}
           else {prd=q;q=q->svt;}
         }
       lmot=lmot->svt;
     }
}
```

```
void FrequenceMax(liste2 lmot, chaine M)
{ int f, fmax=0;
  liste2 s=NULL;
  liste1 tetepos;

  while(lmot!=NULL)
  { tetepos=lmot->E.svt;
    f=0;
    while(tetepos!=NULL) { tetepos=tetepos->svt; f++; }
    if (f>fmax) { fmax=f; s=lmot; }
    lmot=lmot->svt;
  }
  strcpy(M,s->E.mot);
}



main()
{ texte T; chaine M; liste2 lmot;
  printf("Donnez votre texte \n");
  fflush(stdin);
  gets(T);
  lmot=ExtraireMots(T);
  printf("Affiche le contenu de la structure\n");
  affiche(lmot);
  supprim_repetition(lmot);
  printf("Apres suppression \n");
  affiche(lmot);
  FrequenceMax(lmot,M);
  printf("Le mot le plus fréquent est : %s", M);
  getch();
}
```