CRO - 3TC - examen 2021-2022

Durée : 2h heures, tous documents autorisés

Remarques :

— Le barème est donné à titre indicatif (il pourra être modifié éventuellement).

— Pour évaluer les programmes C, on prendra en compte (dans l'ordre décroissant d'importance) : la validité de l'algorithme, la validité syntaxique du programme, la clarté du programme et les commentaires, la forme générale de la présentation.

Attention : version corrigée

1 Compilation (3 pts)

1. Expliquer (en moins de 10 lignes) quel est le mode de passage de paramètres pour les fonctions en C et comment peut-on faire lorsque l'on veut qu'une fonction modifie une variable passée en paramètre.
2. Écrire le fichier Makefile pour le projet C comportant les fichiers suivants :
   — deux fichiers de fonctions C : `liste.c` et `arbre.c` (ainsi que leur fichiers `.h` associés)
   — Un fichier principal : `testStructureDonnees.c` contenant une fonction `main()`

```
all: testStructureDonnees

testStructureDonnees: testStructureDonnees.o liste.o arbre.o
      gcc -o testStructureDonnees testStructureDonnees.o liste.o arbre.o

testStructureDonnees.o: testStructureDonnees.c liste.h arbre.h
gcc -c testStructureDonnees.c -o testStructureDonnees.o

liste.o: liste.c liste.h
gcc -c liste.c -o liste.o

arbre.o: arbre.c arbre.h
gcc -c arbre.c -o arbre.o
```

2 Quelques exercices (6pts)

2.1 les N premiers carrés

Ecrire une fonction C :

`void nPremierCarres(int n)`

qui affiche à l'écran les `n` premiers entiers qui sont des carrés (i.e. les entiers p qui sont le carré d'un autre entier q : $p = q \times q$). Par exemple `nPremierCarres(3)` affichera :

```
1 4 9
```

(On compte 1=1×1 comme un carré).

```c
#include <stdio.h>

void nPremiersCarres(int n)
{
  for (int i=1; i<= n; i++)
        printf("%d ",i*i);
}

int main()
{
  int n=1;

  while (n!=0)
    {
      printf("Entrez un entier (0 pour sortir):");
      scanf("%d",&n);
      nPremiersCarres(n);
    }
  return 0;
}
```

2.2 Miroir

Ecrire une fonction C

`int miroir(int n)`

qui renvoie, à partir d'un entier `n` en entrée, son entier *miroir*, c'est à dire l'entier constitué du même nombre de chiffres et dont les chiffres sont dans l'ordre inverse de ceux de `n`. Par Exemple : si l'entrée `n` est 123 le résultat est 321, si l'entrée est 10278, le résultat est 87201, etc.

Dans cette exercice, on ne se préocupera pas des éventuels débordements de capacité. On rappelle que l'opérateur modulo en C est : '%'.

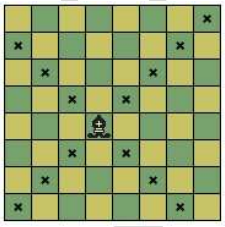
```
#include <stdio.h>

int miroir(int n)
{
  int temp=n, res=0, chiffre;

  while (temp!=0)
    {
      chiffre=temp % 10;
      res=10*res+chiffre;
      temp=temp/10;
    }
  return res;

}

int main()
{
  int n=20987;

  printf("miroir de %d: %d\n",n,miroir(n));
  return 0;
}
```

2.3 La diagonale du fou

Ecrire une fonction C `fou` :

`void fou(int echiquier[8][8], int i, int j)`

qui prend en entrée un tableau 8x8 rempli par des zéros et la position (i, j) d'une case de ce tableau. Cette fonction marque toutes les cases des deux diagonales qui passent par la position (i, j) en y mettant des '1'. Les cases que l'on cherche à marquer sont les case menacées si un fou était placé dans la case (i, j) sur un échiquier (comme illusté sur la figure ci-dessous). On supposera que les cases sont numérotées de 0 à 7 dans chaque dimension.



```
#include <stdio.h>

void fou(int tab[8][8], int i, int j)
 {
   tab[i][j] = 15;
   for (int i1=0;i1<8;i1++)
     for (int j1=0;j1<8;j1++)
       {
 if ((i1+j1 == i+j) && (i != i1))
   tab[i1][j1]=1;
       }
   for (int i1=0;i1<8;i1++)
     for (int j1=0;j1<8;j1++)
       {
 if ((i1-j1 == i-j) && (i != i1))
   tab[i1][j1]=1;
       }

 }

 void afficherEchiquier(int tab[8][8])
 {
  for (int i=0;i<8;i++)
    {
    for (int j=0;j<8;j++)
      {
printf("%2X ",tab[i][j]);
      }
    printf("\n");
    }
}

int main()
{
  int echiquier[8][8];

  for (int i=0;i<8;i++)
    for (int j=0;j<8;j++)
      echiquier[i][j]=0;

  fou(echiquier, 3,4);
  afficherEchiquier(echiquier);
  return 0;
}
```

3 Noeuds à distance 2 dans un graphe (4pts)

Dans cet exercice nous utiliserons une représentation de graphe orienté à base de matrice d'adjacence. Le type proposé pour un graphe est indiqué ci-dessous. Le champ `adj` représente la matrice d'adjacence (matrice carrée), `adj[i][j]` représentant l'existence, ou pas d'un arc entre le noeud `i` et le noeud `j`. `adj[i][j]` prend donc 2 valeurs uniquement : 0 (absence d'arc entre `i` et `j`) ou 1 (arc de `i` à `j`).

```
struct graph {
  int nb_sommets;
  int **adj;
};
typedef struct graph GRAPHE;
```

On cherche à savoir, pour un noeud `x` donné, quels sont les noeuds `y` qui sont à une distance d'au plus 2 de `x`. Un noeud `y` est *à distance 2* du noeud `x` si il existe un chemin

de longueur 2 qui va de `x` à `y`. Par exemple sur le graphe de la figure 1, les noeuds à une distance d'au plus 2 du noeud 1 sont : 2, 3, 4, 5, 6. Les noeuds à une distance d'au plus 2 du noeud 5 sont 7, 8, 3, il n'y a aucun noeud à distance 1 ou 2 du noeud 8.

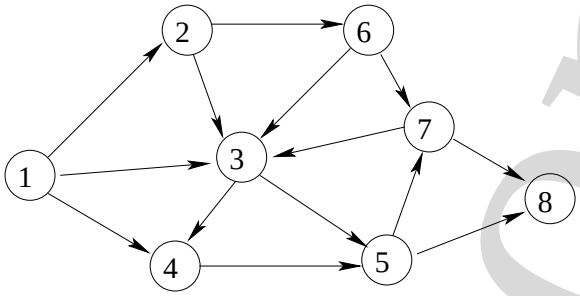


FIGURE 1 – Un graphe orienté `g` quelconque. Par exemple l'appel `distance2(g,1)` (i.e. quels sont les noeuds à distance au plus 2 du noeud 1) affichera : "2 3 4 5 6". Ici les numéros des noeuds vont de 1 à 8 mais on pourra considérer que les noeuds sont numérotés de 0 à N-1 pour un graphe de N Noeuds.

Ecrire une fonction C `distance2` :

```
void distance2(GRAPHE g, int x)
```

qui prend en entrée un graphe g et *affiche à l'écran*[1] les noeuds qui sont à distance inférieure ou égale à 2 du noeud `x`. Les noms des noeuds sont juste leur indice allant de 0 à `g.nb_sommets-1`. L'ordre d'affichage n'a pas d'importance, mais on s'attachera à ne pas afficher plusieurs fois le même noeud.

1. Pour plus de simplicité on ne retournera pas de résultat dans cette fonction, il faudrait mettre en place une gestion de liste.

```c
#include <stdio.h>
#include <stdlib.h>
#include "graphe.h"



void distance2(GRAPHE g, int x)
{
  int *mark;

  mark=(int *)malloc(g.nb_sommets*sizeof(int));
  for (int i=0; i<g.nb_sommets; i++)
    mark[i]=0;

  for (int i=0; i<g.nb_sommets; i++)
    {
      if ((g.adj[x][i]==1))
        {
          if (mark[i]==0)
            {
              mark[i]=1;
              printf(" %d ",i);
            }
        }
      for (int j=0; j<g.nb_sommets; j++)
        {
          if ((g.adj[x][i]==1) && (g.adj[i][j]==1))
            {
              if (mark[j]==0)
                {
                  mark[j]=1;
                  printf(" %d ",j);
                }
            }
        }
    }
  printf("\n");
}

GRAPHE init_graph(GRAPHE g)
{
  g.nb_sommets = 8;
  g.adj=(int **)malloc(g.nb_sommets*sizeof(int *));

  for (int i=0; i<g.nb_sommets;i++)
    g.adj[i]=(int *)malloc(g.nb_sommets*sizeof(int));

  //graphe du sujet
  g.adj[0][1]=1;
  g.adj[0][2]=1;
  g.adj[1][2]=1;
  g.adj[2][3]=1;
  g.adj[1][5]=1;
  g.adj[5][2]=1;
  g.adj[2][4]=1;
  g.adj[5][6]=1;
  g.adj[6][2]=1;
  g.adj[4][6]=1;
  g.adj[6][7]=1;
  g.adj[4][7]=1;

  return g;
}

void print_graph(GRAPHE g)
{
  for (int i=0;i<g.nb_sommets;i++)
    {
      for (int j=0;j<g.nb_sommets;j++)
        {
          printf("%2X ",g.adj[i][j]);
        }
      printf("\n");
    }
}

int main()
{
  GRAPHE g;

  g=init_graph(g);
  print_graph(g);
  printf("distance 2 de 0:\n");
  distance2(g,0);
  printf("distance 2 de 4:\n");
  distance2(g,4);
  printf("distance 2 de 7:\n");
  distance2(g,7);
}
```

4 Liste d'entiers (7pts)

Nous allons définir une structure de donnée permettant de manipuler des listes d'entiers. Cette structure pourra être utilisée par exemple pour le résultat par la fonction `distance2` de la question précédente, puisque nous ne savons pas exactement combien d'entier vont être renvoyés par cette fonction.

Dans cette partie on utilisera la structure de donnée suivante pour définir des listes d'entiers :

```
struct cell {
      int numero;
      struct cell *suivant;
     };
typedef struct cell NOEUD;

typedef NOEUD *LISTE;
```

1. Ecrire une fonction C `printList` :
   `void printList(LISTE l1)`
   qui affiche la liste `l1` passée en argument.
2. Ecrire une fonction C `newList` :
   `LISTE newList(int numeroNoeud)`
   qui crée une nouvelle liste à 1 élément constitué du noeud ayant la valeur `numeroNoeud`.
3. Ecrire une fonction C `appendList` :
   `LISTE appendList(LISTE l1, int numeroNoeud)`
   qui prend en argument une liste `l1` et renvoie la liste dans laquelle on a rajouté à `l1` le noeud contenant la valeur `numeroNoeud` *en fin de liste*. On s'attachera à réutiliser la fonction `newList`.
4. Ecrire une fonction C `freeList` :
   `void freeList(LISTE *pliste)`
   qui prend en entrée un pointeur sur une liste (`pliste`) et qui libère tous les éléments de la liste pointée par `pliste`. On affichera à l'écran la valeur des noeuds désalloués au fur et à mesure de leur libération.

bonus Ecrire une fonction C :
`void sortList(LISTE *pliste)`
Qui trie dans l'ordre croissant les éléments de la liste pointée par `pliste`.

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

void printList(LISTE liste1)
{
  LISTE visitor;
  visitor=liste1;
  printf("|-");
  while (visitor!=NULL)
    {
      printf("-> %d ",visitor->numero);
      visitor=visitor -> suivant;
    }
  printf("|\n");
}

LISTE newList(int numeroNoeud)
{
  LISTE new;
  new=(LISTE)malloc(sizeof(struct cell));
  if (new==NULL)
    exit(-1);
  new->suivant=NULL;
  new->numero = numeroNoeud;
  return(new);
}

LISTE appendList(LISTE l1, int numeroNoeud)
{
  if (l1==NULL)
    return newList(numeroNoeud);
  else
    {
      LISTE temp=appendList(l1->suivant,numeroNoeud);
      l1->suivant = temp;
      return l1;
    }
}

void freeList(LISTE *pliste)
{
  if (pliste==NULL)
    exit(-1);
  else
    {
      if (*pliste==NULL)
        return;
      else
        {
          freeList(&((*pliste)->suivant));
          printf (" <- %d ",(*pliste)->numero);
          free(*pliste);
          *pliste=0;
        }
      return;
    }
}

/* bubble sort */
void sortList(LISTE *pliste)
{
  int changed=1;

  while (changed)
    {
      changed = 0;
      LISTE visitor=*pliste;
      while (visitor->suivant!=NULL)
        {
          if (visitor->numero > visitor->suivant->numero)
            {
              int temp=visitor->numero;
              visitor->numero = visitor->suivant->numero;
              visitor->suivant->numero = temp;
              changed = 1;
            }
          visitor = visitor -> suivant;
        }
    }

}
int main()
{
  LISTE l=NULL;
  l=appendList(l,1);
  l=appendList(l,4);
  l=appendList(l,3);
  printList(l);
  printf("libération:\n");
  freeList(&l);
  printf("then:\n");
  printList(l);
  l=appendList(l,4);
  l=appendList(l,3);
  l=appendList(l,1);
  l=appendList(l,2);
  printList(l);
  sortList(&l);
  printf("after Sort:\n");
```