Licence Informatique
Université Bordeaux I

Initiation à la Programmation
Session de septembre 2000

L'épreuve dure 3 heures. Une préparation manuscrite de deux feuilles format A4 est le seul document autorisé.
Les programmes sont à rédiger en langage C - ansi. On respectera (un minimum) le mode d'indentation utilisé en cours. On utilisera systématiquement la macro-fonction `assert()` *pour signaler toute erreur de programmation ou anomalie d'utilisation. La notation tiendra compte de la correction des programmes mais aussi de la qualité du codage (lisibilité, maintenabilité et extensibilité).*

Exercice 1. Qu'affiche le programme suivant ?

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

int
f(int *a, int b)
{
  int c;
  c = *a + 1;
  *a = b + 1;
  b = c;
  return b;
}

int
main(void)
{
  int a[2], *c = a;

  for (a[0] = a[1] = 0; a[0] < 5; a[0]++)
    {
      printf("%d %d ", a[0], a[1]);
      printf("%d ", f(c, a[0]));
      printf("%d\n", f(c + 1, a[1]));
    }

  return EXIT_SUCCESS;
}
```

Exercice 2. La commande `extraire` permet de sélectionner certaines lignes de caractères présentes sur l'entrée standard ; voici les options *mutuellement exclusives* de cette commande :

`-l liste_croissante_de_numeros_de_lignes` :
affiche les lignes dont le numéro figure dans la liste de numéros donnée en paramètre ;

`-p num` :
affiche les `num` premières lignes ;

`-d num` :
affiche les `num` dernières lignes.

Sans se soucier d'éventuelles incohérences lors de l'appel de la commande :

1. écrire la fonction principale (`main`) qui, après avoir analysé la ligne de commande, appelle une fonction spécifique à chacune des options ;

2. implémenter les options `-l` et `-p` ;
3. spécifier en français un algorithme traitant l'option `-d` (soyez précis quant à la gestion de la mémoire).

Exercice 3. On s'intéresse à la traduction de listes chainées d'entiers en tableaux et vice-versa. On cherchera à optimiser un minimun la complexité des fonctions.

1. Définir les types `liste_entier` et `tableau_entier` ;
2. écrire la fonction `tableau_vers_liste(liste_entier *liste,  tableau_entier tableau)` qui crée une liste contenant (dans le même ordre) les éléments d'un tableau passé en paramètre ;
3. écrire la fonction `liste_vers_tableau(tableau_entier *tableau, liste_entier liste)` qui crée un tableau contenant (dans le même ordre) les éléments d'une liste passée en paramètre.

Exercice 4. Il s'agit de proposer un module traitant de graphes orientés dont les sommets sont étiquetés par des entiers. Ce module proposera les fonctionnalités suivantes :

- créer un graphe vide ;
- détruire un graphe ;
- ajouter un sommet étiqueté à un graphe ;
- supprimer un sommet étiqueté à un graphe ;
- créer un arc entre deux sommets étiquetés d'un graphe ;
- créer un sous graphe à partir d'un graphe et d'un tableau d'étiquettes de sommets.

Pour représenter un graphe on utilisera sa représentation matricielle (matrice d'adjacence), l'ordre des sommets étant celui défini par un tableau contenant les étiquettes ; à cette fin on utilise la structure suivante :

```
struct graphe
{
  int nombre_de_sommets ;
  int *etiquettes ;
  int *matrice ;
};
```

Exemple : voici deux représentations en mémoire du graphe à trois sommets notés 1,4 et 6 et à deux arcs (1,4) et (6,6) (un arc de 1 vers 4 et un autre de 6 vers 6).

Pour l'ordre des sommets 1 4 6 on a :

```
{ 3 , ADRESSE1, ADRESSE2 } avec {1, 4, 6} en  ADRESSE1 et

                                {0, 1, 0,
                                 0, 0, 0,
                                 0, 0, 1 }   en ADRESSE2
```

et pour l'ordre des sommets 4 6 1 on a :

```
{ 3 , ADRESSE3, ADRESSE4 } avec {4, 6, 1} en  ADRESSE3 et

                                {0, 0, 0,
                                 0, 1, 0,
                                 1, 0, 0 }   en ADRESSE4
```

Proposer un fichier en-tête `graphe.h`, un fichier source `graphe.c` et un fichier makefile permettant la compilation d'un programme utilisant ce module et contenu dans le fichier `mon_graphe.c` .

Annexes

Les définitions des fonctions suivantes sont extraites du «Linux Programmer's Manual».

```
int atoi(const char *nptr);                               <stdlib.h>
The  atoi()  function  converts the initial portion of the string
pointed to by nptr to int.

int getchar(void);                                          <stdio.h>
reads the next character from stdin and returns it as an unsigned char
cast to an int, or EOF on end of file or error.

int strcmp(const char *s1, const char *s2);                <string.h>
The strcmp() function compares the two strings s1 and s2.  It returns
an integer less than, equal to, or greater than zero if s1 is found,
respectively, to be less than, to match, or be greater than s2.

void *malloc(size_t size);                                 <stdlib.h>
malloc() allocates size bytes and returns a pointer to the allocated
memory.  The memory is not cleared.

void free(void *ptr);                                       <stdlib.h>
free() frees the memory space pointed to by ptr, which must have been
returned by a previous call to malloc(), calloc() or realloc().
Otherwise, or if free(ptr) has already been called before, undefined
behaviour occurs.  If ptr is NULL, no operation is performed.

void *realloc(void *ptr, size_t size);                      <stdlib.h>
realloc()  changes the size  of the memory  block pointed to by ptr to
size bytes.  The contents will be unchanged to the minimum of the old
and new sizes; newly allocated mem- ory will be uninitialized.  If ptr
is NULL, the call is equivalent to malloc(size); if size is equal to
zero, the call is equivalent to free(ptr).  Unless ptr is NULL, it
must have been returned by an earlier call to malloc(), calloc() or
realloc().
```

Exemple de fichier Makefile

```
CC = gcc
CFLAGS = -g -Wall
OBJ= test_proc.o proc.o gqueue.o

all:	test_proc

test_proc:	$(OBJ)
	$(CC) $(CFLAGS) -o test_proc $(OBJ)

test_proc.o: test_proc.c proc.h

proc.o: proc.c proc.h gqueue.h

gqueue.o: gqueue.c gqueue.h
```

Corrigé

Barème (sur 25)
(I) 3
(II) 2 - 4 - 2
(III) 1 - 2 - 2
(IV) 1 (makefile) - 2 (graphe.h) - 6 (graphe.c)

Exercice 1

```
0 0 1 1
2 1 3 2
4 2 5 3
```

Commentaires : cet exercice a été réussi par la moitié des étudiants.

Exercice 2

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <assert.h>



#define OPTION_LISTE_LIGNES "-l"
#define OPTION_NUM_PREMIERES_LIGNES "-p"
#define OPTION_NUM_DERNIERES_LIGNES "-d"


void afficher_liste_lignes(char *liste[], int longueur_liste);
void afficher_num_premieres_lignes(int num);
void afficher_num_dernieres_lignes(int num);


/* Question 1 ************************************************** */
int
main(int argc, char *argv[])
{
  if (argc < 3)
    return EXIT_FAILURE;

  if (!strcmp(argv[1], OPTION_LISTE_LIGNES))
    afficher_liste_lignes(argv + 2, argc - 2);
  else if (!strcmp(argv[1], OPTION_NUM_PREMIERES_LIGNES))
    afficher_num_premieres_lignes(atoi(argv[2]));
  else if (!strcmp(argv[1], OPTION_NUM_DERNIERES_LIGNES))
    afficher_num_dernieres_lignes(atoi(argv[2]));
  else
    return EXIT_FAILURE;

  return EXIT_SUCCESS;
}


/* Question 2 ************************************************** */
void
afficher_num_premieres_lignes(int num)
{
  int caractere_lu;
  while (num > 0 && (caractere_lu = getchar()) != EOF)
    {
      putchar(caractere_lu);
      if (caractere_lu == '\n')
        num--;
    }
}
```

```
void
afficher_liste_lignes (char *liste [], int longueur_liste )
{
  int ligne_courante = 1;
  int position_liste = 0;
  int prochaine_ligne = atoi (liste [0]);
  int caractere_lu ;

  if (longueur_liste == 0)
    return ;

  while ((caractere_lu = getchar ()) != EOF)
    {
      if (ligne_courante == prochaine_ligne )
        putchar (caractere_lu );
      if (caractere_lu == '\n')
        {
          ligne_courante ++;
          if (ligne_courante > prochaine_ligne )
            {
              position_liste ++;
              if (position_liste != longueur_liste )
                prochaine_ligne = atoi (liste [position_liste ]);
              else
                return ;

            }
        }
    }
}


/* Question 3 ************************************************** */


/* voici une fonction equivalente a getline de stdio ,
   a la difference qu'on retourne 0 ssi on a rencontre EOF */

static int
lire_ligne (char **ligne , size_t * taille )
{
  int caractere_lu , nb_caracteres_lus = 0;

  while ((caractere_lu = getchar ()) != EOF)
    {
      if (*taille <= nb_caracteres_lus )
        {
          *taille = 2 * *taille + 1;
          *ligne = realloc (*ligne , *taille * sizeof (char ));
          assert (*ligne != NULL );
        }
      (*ligne )[nb_caracteres_lus ] = caractere_lu ;
      nb_caracteres_lus ++;
      if (caractere_lu == '\n')
        break ;
    }

  *taille = nb_caracteres_lus + 1;
  *ligne = realloc (*ligne , *taille * sizeof (char ));
  assert (*ligne != NULL );
  (*ligne )[nb_caracteres_lus ] = '\0';

  return caractere_lu != EOF;
}


void
afficher_num_dernieres_lignes (int num)
{
  char **les_lignes ;
  int *longueur_ligne ;
```

```
  int ligne_courante = 0;
  int i;

  les_lignes = (char **) calloc(num, sizeof(char *));
  longueur_ligne = (int *) calloc(num, sizeof(int));
  assert(les_lignes != NULL && longueur_ligne != 0);

  while (lire_ligne(&les_lignes[ligne_courante % num],
                    &longueur_ligne[ligne_courante % num]))
    ligne_courante++;

  for (i = 1; i <= num; i++)
    if (longueur_ligne[(ligne_courante + i) % num] != 0)
      printf("%s", les_lignes[(ligne_courante + i) % num]);
}
```

Commentaires :

Question 1 Comparer directement deux chaines de caractères est une erreur assez classique : l'expression `argv[1] == "-l"` vérifie simplement que le pointeur `argv[1]` est égale à l'adresse de la chaine `"-l"`, il fallait utiliser la fonction `strcmp()` (donnée en annexe !). Trop d'étudiants ont perdu du temps en essayant de traiter (incomplétement) d'éventuelles erreurs d'appel - aucun point n'a été attribué pour cela. Enfin quelques étudiants ont repris la structure du programme `main()` de l'exercice 2 (commande wc) de la session de janvier 2000, cette solution était plus complexe que nécessaire.

Question 2 Beaucoup d'étudiants ont utilisé (par manque de simplicité) des chaines alors qu'un buffer d'un seul caractère est suffisant ! Pire : certains étudiants (*JAVAnais ?*) ont additionné pointeurs et caractères (`chaine += caractere`) croyant ainsi éviter le problème qu'ils s'étaient eux même imposé.

Question 3 Une seule personne a répondu de façon satisfaisante à cette question difficile puisqu'ici il fallait gérer habilement la mémoire – il ne fallait ni mémoriser le texte en entier, ni prétendre lire deux fois le texte présent sur l'entrée standard.

Exercice 3

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

struct chainon
  {
    int valeur;
    struct chainon *suivant;
  };


typedef struct liste
  {
    int cardinal;
    struct chainon *sommet;
  }
 *liste_entier;

typedef struct tableau
  {
    int cardinal;
    int *elements;
  }
 *tableau_entier;


void
tableau_vers_liste(liste_entier * liste, tableau_entier tableau)
{
  int i;
  struct chainon **pp_chainon;
```

```
  assert(tableau != NULL);
  *liste = (liste_entier) malloc(sizeof(struct liste));
  assert(*liste != NULL);

  (*liste)->cardinal = tableau->cardinal;

  for (i = 0, pp_chainon = &((*liste)->sommet);
       i < tableau->cardinal;
       i++, pp_chainon = &(*pp_chainon)->suivant)
    {
      *pp_chainon = (struct chainon *) malloc(sizeof(struct chainon));
      assert(*pp_chainon != NULL);
      (*pp_chainon)->valeur = tableau->elements[i];
    }

  *pp_chainon = NULL;
}

void
liste_vers_tableau(tableau_entier * tableau, liste_entier liste)
{
  int i;
  struct chainon *p_chainon;

  assert(liste != NULL);
  *tableau = (tableau_entier) malloc(sizeof(struct tableau));
  assert(*tableau != NULL);
  (*tableau)->cardinal = liste->cardinal;
  if (liste->cardinal == 0)
    {
      (*tableau)->elements = NULL;
      return;
    }

  (*tableau)->elements = (int *) malloc(liste->cardinal * sizeof(int));
  assert((*tableau)->elements != NULL);

  for (i = 0, p_chainon = liste->sommet;
       i < liste->cardinal;
       i++, p_chainon = p_chainon->suivant)
    (*tableau)->elements[i] = p_chainon->valeur;
}



int
main()
{

  int i, tab[] =
  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  struct tableau t =
  {10, tab};
  liste_entier l;
  tableau_entier t1;
  tableau_vers_liste(&l, &t);
  liste_vers_tableau(&t1, l);
  for (i = 0; i < t1->cardinal; i++)
    printf("%d␣", t1->elements[i]);
}
```

Commentaires :

Question 1 Si la présence du cardinal était facultative pour le type liste chainée (on dispose d'un marqueur de fin), elle était incontournable pour les tableaux dynamiques (plus de la moitié des candidats a commis cette erreur, certains allant jusqu'à utiliser strlen() pour calculer la taille du tableau). Quelques JAVAnais ont encore frappé en introduisant des structures récursives qui, occupant théoriquement une infinité d'octets en mémoire, sont refusées par le compilateur.

Question 2 La résolution de ce type de question fait partie du b.a.ba de la programmation C.

Exercice 4

Makefile

```
CC = gcc
CFLAGS = -g -Wall
OBJ = mon_graphe.o graphe.o

mon_graphe : $(OBJ)
	$(CC)  $(CFLAGS) -o mon_graphe $(OBJ)

mon_graphe.o : mon_graphe.c graphe.h

graphe.o : graphe.c  graphe.h
```

graphe.h

```
#ifndef _GRAPHE_H_
#define  _GRAPHE_H_

typedef struct graphe *graphe;

extern void graphe_creer(graphe * g);
extern void graphe_detruire(graphe * g);
extern void graphe_ajouter_sommet(int etiquette, graphe g);
extern void graphe_supprimer_sommet(int etiquette, graphe g);
extern void graphe_creer_arc(int source, int destination, graphe g);
extern void graphe_creer_sous_graphe(graphe * sous_g, int *sommets,
                                     int nb_sommets, graphe g);
extern void graphe_afficher(graphe g);

#endif
```

graphe.c

```
#include "graphe.h"
#include <malloc.h>
#include <assert.h>
#include <stdio.h>

#define CASE(un_graphe,x,y,dimension) (un_graphe)->matrice[(x) * (dimension) + (y)]

struct graphe
  {
    int nombre_de_sommets;
    int *etiquettes;
    int *matrice;
  };


static int
indice_etiquette(int etiquette, graphe g)
{
  int i;
  for (i = 0; i < g->nombre_de_sommets; i++)
    if (etiquette == g->etiquettes[i])
      return i;
  return -1;
}

static void
retailler_graphe(graphe g, int nouvelle_dimension)
{
  g->nombre_de_sommets = nouvelle_dimension;
  g->etiquettes = realloc(g->etiquettes, nouvelle_dimension * sizeof(int));
  g->matrice = realloc(g->matrice, nouvelle_dimension * nouvelle_dimension * sizeof(int));
}

void
graphe_creer(graphe * g)
{
  *g = (graphe) malloc(sizeof(struct graphe));
```

```
  assert(*g != NULL);
  (*g)->nombre_de_sommets = 0;
  (*g)->matrice = (*g)->etiquettes = NULL;
}


void
graphe_detruire(graphe * g)
{
  assert(*g != NULL);
  free((*g)->matrice);
  free((*g)->etiquettes);
  free(*g);
  *g = NULL;
}

void
graphe_ajouter_sommet(int etiquette, graphe g)
{
  int ancienne_dimension, i, j;
  assert(g != NULL);

  ancienne_dimension = g->nombre_de_sommets;

  retailler_graphe(g, g->nombre_de_sommets + 1);

  for (i = ancienne_dimension - 1; i >= 0; i--)
    for (j = ancienne_dimension - 1; j >= 0; j--)
      CASE (g, i, j, g->nombre_de_sommets) = CASE (g, i, j, ancienne_dimension);

  for (i = 0; i < g->nombre_de_sommets; i++)
    CASE (g, i, ancienne_dimension, g->nombre_de_sommets) =
      CASE (g, ancienne_dimension, i, g->nombre_de_sommets) = 0;

  g->etiquettes[g->nombre_de_sommets - 1] = etiquette;
}

void
graphe_supprimer_sommet(int etiquette, graphe g)
{
  int i, j, position_etiquette;
  int future_dimension;

  assert(g != NULL && g->etiquettes != NULL);
  position_etiquette = indice_etiquette(etiquette, g);
  assert(position_etiquette != -1);
  future_dimension = g->nombre_de_sommets - 1;

  for (i = 0; i < g->nombre_de_sommets; i++)
    if (i != position_etiquette)
      for (j = 0; j < g->nombre_de_sommets; j++)
        if (j != position_etiquette)
          CASE (g, i - (i > position_etiquette), j - (j > position_etiquette), future_dimensi
            CASE (g, i, j, g->nombre_de_sommets);

  for (i = position_etiquette + 1; i < g->nombre_de_sommets; i++)
    g->etiquettes[i - 1] = g->etiquettes[i];

  retailler_graphe(g, future_dimension);
}

void
graphe_creer_arc(int source, int destination, graphe g)
{
  int s, d;
  assert(g != NULL && g->matrice != NULL);
  s = indice_etiquette(source, g);
  d = indice_etiquette(destination, g);
  assert(s != -1 && d != -1);
  CASE (g, s, d, g->nombre_de_sommets) = 1;
}
```

```
void
graphe_creer_sous_graphe (graphe * sous_g, int *sommets,
                          int nb_sommets, graphe g)

/* L'ordre des etiquettes de sous_g est determine par le tableau
   "sommets". Le calcul est facilite par l'utilisation d'un tableau
   "position" : position[i] contient l'indice du sommet d'etiquette
   sommets[i] dans le graphe g. */
{
  int i, j, *position;

  graphe_creer (sous_g);

  if (nb_sommets == 0)
    return;

  assert (g != NULL && sommets != NULL);

  position = (int *) malloc (nb_sommets * sizeof (int));
  (*sous_g)->nombre_de_sommets = nb_sommets;
  (*sous_g)->etiquettes = (int *) malloc (nb_sommets * sizeof (int));
  (*sous_g)->matrice = (int *) calloc (nb_sommets * nb_sommets, sizeof (int));
  assert (position != NULL && (*sous_g)->etiquettes != NULL
          && (*sous_g)->matrice != NULL);

  for (i = 0; i < nb_sommets; i++)
    {
      (*sous_g)->etiquettes [i] = sommets [i];
      position [i] = indice_etiquette (sommets [i], g);
      assert (position [i] != -1);
    }

  for (i = 0; i < nb_sommets; i++)
    for (j = 0; j < nb_sommets; j++)
      CASE (*sous_g, i, j, nb_sommets) =
        CASE (g, position [i], position [j], g->nombre_de_sommets);


  free (position);
}

void
graphe_afficher (graphe g)
{
  int i, j;
  assert (g != NULL);

  if (g->nombre_de_sommets == 0)
    printf ("({},{})");

  for (i = 0; i < g->nombre_de_sommets; i++)
    printf ("\t%d", g->etiquettes [i]);

  for (i = 0; i < g->nombre_de_sommets; i++)
    {
      printf ("\n%d", g->etiquettes [i]);
      for (j = 0; j < g->nombre_de_sommets; j++)
        printf ("\t%d", CASE (g, i, j, g->nombre_de_sommets));
    }
}
```

Commentaires : peu d'étudiants respectent le principe du masquage de l'implémentation (le contenu d'un type structuré ne doit jamais apparaitre dans un ".h") qui est pourtant à la base de la programmation modulaire. Si les fonctions de création et de destruction d'un graphe ont parfois été correctement traitées, il en va tout autrement pour les autres fonctions : clairement la technique de manipulation des tableaux dynamiques à plusieurs dimensions n'a pas été assimilée (la fonction d'ajout d'un arc n'est pas difficile si l'on maitrise cette technique). Pour terminer, les fonctions de suppression et d'ajout d'un sommet présentées ici ont été réécrites plusieurs fois par le correcteur.