Implémentation basique

Exercice 1

Trivial, cf. les TP précédents.

Exercice 2

```
void initialise_matrice_uniforme(float *c, int n)
{  int i,j;
   float valeur = 1.0/(n*n);
   for(i=0;i<n;i++)
   {  for(j=0;j<n;j++)
         *(c+n*i+j) = valeur;
   }
}
```

Exercice 3

```
void calcule_voisinage(SDL_Surface* image, int x, int y, Uint32* voisinage, int
n)
{  int i,j,a,b;
   for(i=0;i<n;i++)
   {  a = x+i-n/2;
      for(j=0;j<n;j++)
      {  b = y+j-n/2;
         *(voisinage+n*i+j) = couleur_pixel_surface(image,a,b);
      }
   }
}
```

Exercice 4

```
Uint32 calcule_couleur(SDL_Surface* image, int x, int y, float *c, int n){
   Uint32 voisinage[n][n], resultat;
   int i,j;
   float sr=0, sv=0, sb=0, poids;
   Uint8 r,v,b;

   // on calcule le voisinage de (x,y)
   calcule_voisinage(image, x, y, voisinage, n);

   // on calcule la somme ponderee pour chaque composante
   for(i=0;i<n;i++)
   {  for(j=0;j<n;j++)
      {  convertis_couleur(voisinage[i][j], &r, &v, &b);
         poids = *(c+n*i+j);
         sr = sr + poids*r;
         sv = sv + poids*v;
         sb = sb + poids*b;
      }
   }

   // on calcule la nouvelle couleur
   resultat = convertis_rvb((int)sr, (int)sv, (int)sb);
   return resultat;
}
```

Exercice 5

```
   SDL_Surface* floute_rectangle(SDL_Surface* source, int x, int y, int l, int h,
float *c, int n)
   {  SDL_Surface* cible;
      int i,j;
      Uint32 coul;

      // on clone l'image originale
      cible = clone_surface(source);

      // on traite chaque pixel du rectangle
      for(i=x;i<x+l;i++)
      {  for(j=y;j<y+h;j++)
         {  coul = calcule_couleur(source, i, j, c, n);
            allume_pixel_surface(cible,i,j,coul);
         }
      }

      return cible;
   }
```

3 Gestion des bords

Exercice 6

```
   void calcule_voisinage2(SDL_Surface* image, int x, int y, Uint32* voisinage, int
n)
   {  int i,j,a,b;
      for(i=0;i<n;i++)
      {  a = x+i-n/2;
         if(a<0 || a>=image->w)
            a = x+(n-1-i)-n/2;
         for(j=0;j<n;j++)
         {  b = y+j-n/2;
            if(b<0 || b>=image->h)
               b = y+(n-1-j)-n/2;
            *(voisinage+n*i+j) = couleur_pixel_surface(image,a,b);
         }
      }
   }
```

Exercice 7

```
   SDL_Surface* floute_image(SDL_Surface* source, float *c, int n)
   {  SDL_Surface* resultat = floute_rectangle(source,0,0,source->w,source->h,c,n);
      return resultat;
   }
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 29 | flous avancés

1 Flou gaussien

Exercice 1

```
void normalise_matrice(float *c, int n)
{  int i,j;

   // calcul de la somme des valeurs
   float somme = 0;
   for(i=0;i<n;i++)
   {  for(j=0;j<n;j++)
         somme = somme + *(c+n*i+j);
   }

   // normalisation des valeurs
   for(i=0;i<n;i++)
   {  for(j=0;j<n;j++)
         *(c+n*i+j) = *(c+n*i+j) / somme;
   }
}
```

Exercice 2

```
void initialise_matrice_gaussienne(float *c, int n, float sigma)
{  int i,j;
   float valeur,coef,expo;
   coef = 1 / (pow(sigma,2)*2*PI);

   // on calcule les valeurs d'apres la loi de Gauss
   for(i=0;i<n;i++)
   {  for(j=0;j<n;j++)
      {  expo = - (pow(i-n/2,2) + pow(j-n/2,2)) / (2*pow(sigma,2));
         valeur = coef * pow(EXP,expo);
         *(c+n*i+j) = valeur;
      }
   }

   // on normalise tout ca
   normalise_matrice(c,n);
}
```

2 Flou cinétique

Exercice 4

```
void initialise_matrice_cinetique(float *c, int n, float coef_dir, int
multiplicite)
{  int i,j,k,d;
   float valeur;

   // on met des zeros partout
   for(i=0;i<n;i++)
   {  for(j=0;j<n;j++)
         *(c+n*i+j) = 0;
   }
```

```
    // on trace la droite principale et ses paralleles
    if(coef_dir>-1 && coef_dir<1)
    {  for(k=0;k<multiplicite;k++)
       {  valeur = multiplicite - k;
          for(d=-1;d<=1;d=d+2)
          {  for(i=0;i<n;i++)
             {  j = (int)(coef_dir*(i-n/2)) + n/2 + d*k;
                if(j>=0 && j<n)
                   *(c+n*i+j) = valeur;
             }
          }
       }
    }
    else
    {  for(k=0;k<multiplicite;k++)
       {  valeur = multiplicite - k;
          for(d=-1;d<=1;d=d+2)
          {  for(j=0;j<n;j++)
             {  i = (int)((j-n/2)/coef_dir) + n/2 + d*k;
                if(i>=0 && i<n)
                   *(c+n*i+j) = valeur;
             }
          }
       }
    }

    // on normalise
    normalise_matrice(c,n);
}
```

3 Flou radial

Exercice 6

```
float calcule_coefficient_directeur(int x1, int y1, int x2, int y2)
{  float resultat;
   float numerateur = y2 - y1;
   float denominateur = x2 - x1;

   if(denominateur==0)
   {  if(numerateur==0)
         resultat = 0;
      else
         resultat = FLT_MAX;
   }
   else
      resultat = numerateur / denominateur;

   return resultat;
}
```

Exercice 7

```
SDL_Surface* floute_image_radial(SDL_Surface* source, int x, int y)
{  SDL_Surface* cible;
   int i,j,n=11,multiplicite=1;
   float c[n][n],coef_dir;
   Uint32 coul;

   // on clone l'image originale
   cible = clone_surface(source);

   // on traite chaque pixel du rectangle
   for(i=0;i<source->w;i++)
   {  for(j=0;j<source->h;j++)
      {  coef_dir = calcule_coefficient_directeur(x,y,i,j);
         initialise_matrice_cinetique(c,n,coef_dir,multiplicite);
         coul = calcule_couleur(source, i, j, c, n);
```

```
                allume_pixel_surface(cible,i,j,coul);
            }
        }

        return cible;
    }
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

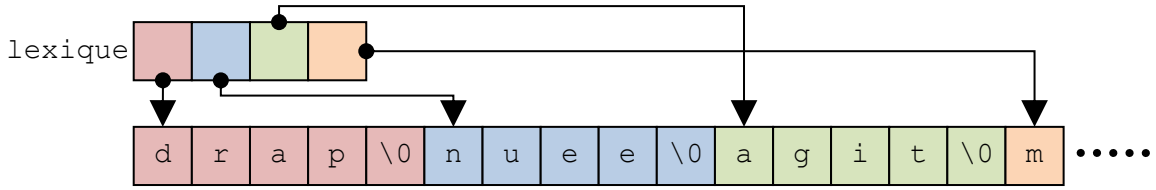
Damien Berthet & Vincent Labatut

Correction TP 30 | gestion d'un lexique

1 Définition du lexique

Exercice 1

La variable `lexique` désigne un tableau de pointeurs sur des chaînes de caractères constantes.



- `*lexique` est le contenu de la première case du tableau `lexique`, i.e. un pointeur sur une chaîne de caractères, de type `char*`. Une chaine de caractères est un tableau, donc il s'agit en fait de l'adresse de la première case de ce tableau. Autrement dit, cette expression est l'adresse de la première lettre du lexique (ici, la lettre `'d'`).
- `lexique+1` est l'adresse de la deuxième case du tableau `lexique`. C'est donc l'adresse d'un pointeur sur la chaine `"nuee"`, et son type est par conséquent `char**`.
- `*(lexique+2)` représente le contenu de la 3ème case du tableau `lexique`, et son type est donc `char*`. Comme pour `*lexique`, on obtient l'adresse de la première lettre de la chaîne pointée (i.e. `'a'`, dans notre cas).
- `*(lexique+1)+1` est l'adresse de la 2ème lettre de la chaîne pointée par le pointeur `*(lexique+1)`. Son type est donc `char*`, et il s'agit de l'adresse de la lettre `'u'` dans le mot `"nuee"`.
- `*(*(lexique+2)+2)` est une valeur littérale de type `char` et sa valeur est `'i'` : 3ème lettre du 3ème mot de `lexique`.
- De la même façon, `*(*(lexique+3)+4)` est la 5ème lettre du 3ème mot, donc une valeur de type `char` correspondant au caractère de fin de chaîne `'\0'`.

Exercice 2

```
void affiche_lexique()
{  int i;
   for(i=0;i<TAILLE_LEXIQUE;i++)
      printf("lexique[%d]=\"%s\"\n",i,lexique[i]);
}
```

2 Recherche dans le lexique

Exercice 3

```
int cherche_prefixe(char *prefixe)
{  int i,resultat=0;
   printf("La chaine \"%s\" est un prefixe des mots suivants :\n",prefixe);
```

```
    for(i=0;i<TAILLE_LEXIQUE;i++)
    {  if(est_prefixe(prefixe,lexique[i]))
       {  resultat++;
          printf("%3d.lexique[%d]=\"%s\"\n",resultat,i,lexique[i]);
       }
    }

    return resultat;
}
```

Exercice 4

```
int cherche_premier()
{  int i,comp,resultat=1;

    for(i=1;i<TAILLE_LEXIQUE;i++)
    {  comp = compare_chaines(lexique[resultat],lexique[i]);
       if(comp>0)
          resultat = i;
    }

    return resultat;
}
```

Exercice 5

```
int cherche_premier_partiel(int debut)
{  int i,comp,resultat=debut;

    for(i=debut;i<TAILLE_LEXIQUE;i++)
    {  comp = compare_chaines(lexique[resultat],lexique[i]);
       if(comp>0)
          resultat = i;
    }

    return resultat;
}
```

3 Tri du lexique

Exercice 6

```
void permute_mots(int i, int j)
{  char *p = lexique[i];
   lexique[i] = lexique[j];
   lexique[j] = p;
}
```

Exercice 7

```
void trie_lexique()
{  int i,premier;

   for(i=0;i<TAILLE_LEXIQUE;i++)
   {  premier = cherche_premier_partiel(i);
      permute_mots(i,premier);
   }
}
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 31 | allocation dynamique

1 Organisation de la mémoire

Exercice 1

Le programme modifié est (les ajouts sont indiqué en turquoise) :

```
void fonction()
{  short g,h,i;
   short *p,*q,*r;
   if((p=(short*)malloc(sizeof(short)))==NULL
      || (q=(short*)malloc(sizeof(short)))==NULL
      || (r=(short*)malloc(sizeof(short)))==NULL)
      printf("fonction:malloc: ERREUR lors de l'allocation.\n");
   printf("&g:%p &h:%p &i:%p \n",&g,&h,&i);
   printf(" p:%p  q:%p  r:%p \n",p,q,r);
}

int main()
{  short d,e,f;
   short *m,*n,*o,*s,*t,*u;
   if((m=(short*)malloc(sizeof(short)))==NULL
      || (n=(short*)malloc(sizeof(short)))==NULL
      || (o=(short*)malloc(sizeof(short)))==NULL)
      printf("main:malloc: ERREUR lors de l'allocation.\n");
   printf("&a:%p &b:%p &c:%p \n",&a,&b,&c);
   printf("&d:%p &e:%p &f:%p \n",&d,&e,&f);
   printf(" m:%p  n:%p  o:%p \n",m,n,o);

   fonction();

   short j,k,l;
   if((s=(short*)malloc(sizeof(short)))==NULL
      || (t=(short*)malloc(sizeof(short)))==NULL
      || (u=(short*)malloc(sizeof(short)))==NULL)
      printf("main:malloc: ERREUR lors de l'allocation.\n");
   printf("&j:%p &k:%p &l:%p \n",&j,&k,&l);
   printf(" s:%p  t:%p  u:%p \n",s,t,u);

    return EXIT_SUCCESS;
}
```

Vous remarquerez que pour les pointeurs, on affiche directement leur contenu (qui est une adresse) et non pas leur adresse.

Avant la modification, on a un résultat du type :

```
&a:0x601030        &b:0x601032        &c:0x601034
&d:0x7fff23cbc8ce  &e:0x7fff23cbc8cc  &f:0x7fff23cbc8ca
&g:0x7fff23cbc89e  &h:0x7fff23cbc89c  &i:0x7fff23cbc89a
&j:0x7fff23cbc8c8  &k:0x7fff23cbc8c6  &l:0x7fff23cbc8c4
```

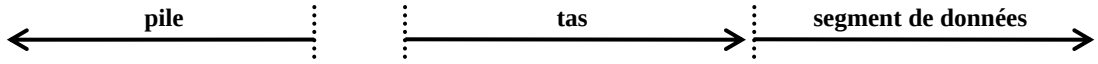
Après la modification, on a :

```
&a:0x601040        &b:0x601042        &c:0x601044
&d:0x7fffd7b2a68e  &e:0x7fffd7b2a68c  &f:0x7fffd7b2a68a
 m:0x1b72010        n:0x1b72030        o:0x1b72050
&g:0x7fffd7b2a62e  &h:0x7fffd7b2a62c  &i:0x7fffd7b2a62a
 p:0x1b72070        q:0x1b72090        r:0x1b720b0
&j:0x7fffd7b2a688  &k:0x7fffd7b2a686  &l:0x7fffd7b2a684
 s:0x1b720d0        t:0x1b720f0        u:0x1b72110
```

On peut faire les remarques suivantes :

- On peut observer la conséquence sur les adresses de `j`, `k` et `l` de la déclaration des pointeurs dans le `main` : les adresses sont décalées par rapport à avant la modification.
- Le début du tas (`0x1b72010`) est localisé entre le début de la pile (`0x7fffd7b2a68e`) et celui du segment de données (`0x601040`).
- Comme pour le segment de données, les adresses des nouvelles zones allouées sont obtenues par incrémentation (`0x1b72010`, `0x1b72030`, `0x1b72050`, `0x1b72070`…), alors que c'est par décrémentation pour la pile (`0x7fffd7b2a68e`, `0x7fffd7b2a68c`, `0x7fffd7b2a68a`…).

La représentation graphique de la mémoire ressemble donc à :



Attention : l'organisation exacte dépend du contexte (compilateur, système d'exploitation, etc.). En particulier, le tas n'est pas forcément localisé entre la pile et le segment de données.

2 Tableaux d'entiers

Exercice 2

- Fonction `affiche_tableau` :

```
void affiche_tableau(short *tab, int taille)
{  int i;
   printf("{");
   for(i=0;i<taille;i++)
      printf(" %d",tab[i]);
   printf(" }");
}
```

- Fonction `alloue_tableau1` :

```
short* alloue_tableau1(int taille)
{  short *p,i;
   // réservation de l'espace mémoire
   if((p = (short*)malloc(taille*sizeof(short))) == NULL)
      printf("alloue_tableau1: erreur lors du malloc\n");
   else
   {  // initialisation à 1
      for(i=0;i<=taille;i++)
         p[i]=1;
   }
   return p;
}
```

- Fonction `main` :

```
int main(int argc, char **argv)
{  short *t;
   t = alloue_tableau1(N);
   affiche_tableau(t,N);printf("\n");

   exit(EXIT_SUCCESS);
}
```

Exercice 3

- Fonction `alloue_tableau2` :

```
void alloue_tableau2(int taille, short **tab)
{  short *p,i;
   // réservation de l'espace mémoire
   if((p = (short*)malloc(taille*sizeof(short))) == NULL)
      printf("alloue_tableau2 : erreur lors du malloc\n");
```

```
      else
      {  // initialisation à 0
         for(i=0;i<=taille;i++)
         p[i]=1;
      }
      *tab = p;
   }
```

- Fonction `main` :

```
   int main(int argc, char **argv)
   {  short *t;
      alloue_tableau2(N,&t);
      affiche_tableau(t,N);printf("\n");

      exit(EXIT_SUCCESS);
   }
```

3 Chaînes de caractères

Exercice 4

- Fonction `saisis_chaine_tampon` :

```
   void saisis_chaine_tampon(char **chaine)
   {  char c[100],*p;
      int i=0,taille;
      // saisie de la chaine
      gets(c);
      // calcul de la taille de la chaine
      while(c[i]!='\0')
         i++;
      taille=i+1;
      // réservation de l'espace mémoire
      if((p = (char*)malloc(taille*sizeof(char))) == NULL)
         printf("saisis_chaine_tampon: erreur lors du malloc\n");
      else
      {  // copie de la chaine
         for(i=0;i<=taille;i++)
            p[i]=c[i];
         // mise à jour du paramètre
         *chaine = p;
      }
   }
```

- Fonction `main` :

```
   int main(int argc, char **argv)
   {  char* chaine;
      printf("Entrez une chaine de caracteres :\n");
      saisis_chaine_tampon(&chaine);
      printf("La chaine saisie est : \"%s\"\n",chaine);

      exit(EXIT_SUCCESS);
   }
```

Exercice 5

- Fonction `saisis_chaine_direct` :

```
   void saisis_chaine_direct(char **chaine)
   {  char c,*p=NULL;
      int i=0;
      do
      {  c=getche();
         if((p = (char*)realloc(p, (i+1)*sizeof(char))) == NULL)
            printf("saisis_chaine_direct: erreur lors du realloc\n");
         else
         {  p[i]=c;
            i++;
         }
```

```
    }
    while(c!='\r' /*pour windows*/ && c!='\n' /*pour linux*/);
    p[i-1]='\0';
    *chaine = p;
}
```

- Fonction `main` :

```
int main(int argc, char **argv)
{  char* chaine;
   printf("Entrez une chaine de caracteres :\n");
   saisis_chaine_direct(&chaine);
   printf("La chaine saisie est : \"%s\"\n",chaine);

   exit(EXIT_SUCCESS);
}
```

4 Tableaux de pointeurs

Exercice 6

```
void remplis_mots()
{  int i;

   // saisie
   for(i=0;i<N;i++)
   {  printf("Entrez le mot n.%d : ",i);
      saisis_chaine_tampon(&mots[i]);
   }

   // affichage
   for(i=0;i<N;i++)
      printf("Mot n.%d : %s\n",i,mots[i]);
}
```

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 32 | générateur pseudo-aléatoire

2 Implémentation

Exercice 1

- Fichier `alea.h` :

```
#ifndef ALEA_H_
#define ALEA_H_

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>

#define A 137
#define C 187
#define M 256

unsigned char genere_nombre_lehmer();

#endif /* ALEA_H_ */
```

- Fichier `alea.c` :

```
#include "alea.h"

unsigned char u = 0;

unsigned char genere_nombre_lehmer()
{  u = (u*A + C) % M;
   return u;
}
```

Remarquez que `A`, `C` et `M` sont des constantes définies dans `alea.h`, alors que `u` est une variable globale définie dans `alea.c`.

Exercice 2

```
unsigned char init_graine_lehmer()
{  time_t t;
   if(time(&t)==-1)
      printf("init_graine_lehmer: erreur lors de l'execution de time()\n");
   else
      u = t%M;
   return u;
}
```

Exercice 3

```
unsigned char* genere_tableau_lehmer(int n)
{  int i;
   unsigned char *resultat;
   if((resultat=(unsigned char*)malloc(n*sizeof(unsigned char))) == NULL)
      printf("genere_tableau_lehmer: erreur lors du malloc\n");
   else
   {  for(i=0;i<=n;i++)
         resultat[i] = genere_nombre_lehmer();
   }
   return resultat;
```

```
}
```

3 Vérification

Exercice 4

```
void calcule_stats(unsigned char *tab, int n, int *min, int *max, float *moy,
float *et)
{  int i;
   float somme = tab[0];
   *min = tab[0];
   *max = tab[0];

   for(i=1;i<n;i++)
   {  somme = somme + tab[i];
      if(tab[i]<*min)
         *min = tab[i];
      if(tab[i]>*max)
         *max = tab[i];
   }
   *moy = somme/n;

   somme = 0;
   for(i=0;i<n;i++)
      somme = somme + pow(tab[i]-*moy,2);
   *et = sqrt(somme/(n-1));
}
```

Exercice 5

```
void calcule_repartition(unsigned char *tab, int n, int **dist)
{  int i;
   if((*dist=(int*)malloc(M*sizeof(int))) == NULL)
      printf("calcule_repartition: erreur lors du malloc\n");
   else
   {  for(i=0;i<M;i++)
         (*dist)[i] = 0;
      for(i=0;i<=n;i++)
         (*dist)[tab[i]]++;
   }
}
```

Exercice 6

Fichier `main.c` demandé :

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

#include "alea.h"
#include "histogramme.h"

int main(int argc, char *argv[])
{  int g,n,*dist;
   unsigned char *tab;
   g = init_graine_lehmer();
   printf("graine: %d\n",g);
   n = 10000;
   tab = genere_tableau_lehmer(n);
   calcule_repartition(tab,n,&dist);
   histogramme_horizontal(dist,M);

   return EXIT_SUCCESS;
}
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 33 | nombres hexadécimaux

2 Opérations simples

Exercice 1

```
int longueur_nombre(char *n)
{   int i=0;
    while(n[i]!='\0')
       i++;
    return i;
}
```

Exercice 2

```
int verifie_nombre(char *n)
{  int resultat=1;
   int i=0;
   while(n[i]!='\0' && resultat)
   {  if((n[i]>='0' && n[i]<='9') || (n[i]>='A' && n[i]<='F'))
         i++;
      else
         resultat=0;
   }
   return resultat;
}
```

3 Comparaisons

Exercice 3

```
int compare_chiffres(char c1, char c2)
{  int resultat;
   if(c1>='0' && c1<='9')
      if(c2>='0' && c2<='9')
         // c1 et c2 sont des chiffres
         resultat=c1-c2;
      else
         // c1 est un chiffre, c2 est une lettre
         resultat=-1;
   else
      if(c2>='0' && c2<='9')
         // c1 est une lettre, c2 est un chiffre
         resultat=1;
      else
         // c1 et c2 sont des lettres
         resultat=c1-c2;
   return resultat;
}
```

Exercice 4

```
int compare_nombres(char *n1, char *n2)
{  int l1=longueur_nombre(n1),l2=longueur_nombre(n2);
   int resultat=l1-l2,i=0;
   if(resultat==0)
   {  while(i<l1 && n1[i]==n2[i])
         i++;
```

```
        if(i==l1)
           resultat=0;
        else
           resultat=compare_chiffres(n1[i],n2[i]);
     }
     return resultat;
  }
```

4 Conversion

Exercice 5

```
  char convertit_chiffre(int x)
  {  char resultat;
     if(x<10)
        resultat='0'+x;
     else
        resultat='A'+(x-10);
     return resultat;
  }
```

Exercice 6

```
  char* convertit_nombre(int x)
  {  int taille=0,temp=x,reste,i=0;
     char *resultat,chiffre;
     // on calcule le nombre de chiffres du résultat
     while(temp!=0)
     {  temp=temp/16;
        taille++;
     }
     // on rajoute l'espace pour le '\0'
     taille++;
     // allocation dynamique
     if((resultat = (char *) malloc(sizeof(char)*taille))==NULL)
        printf("convertit_nombre : erreur lors du malloc\n");
     // construction du nombre en base 16
     temp=x;
     resultat[taille-1]='\0';
     while(temp!=0)
     {  // extraction du dernier chiffre
        reste=temp%16;
        temp=temp/16;
        // conversion du chiffre en base 16
        chiffre = convertit_chiffre(reste);
        // on place le chiffre dans le nombre
        resultat[taille-2-i]=chiffre;
        i++;
     }
     return resultat;
  }
```

5 Modifications

Exercice 7

```
  void incremente_nombre(char **n)
  {  int i,longueur;
     char *p=*n,retenue='0',*q;
     // on calcule la longueur du nombre
     longueur=longueur_nombre(p);
     i=longueur-1;
     // on effectue l'incrémentation
     do
     {  switch(p[i])
        {  case 'F':
              retenue='1';
```

```
                p[i]='0';
                break;
            case '9':
                retenue='0';
                p[i]='A';
                break;
            default:
                retenue='0';
                p[i]++;
        }
        i--;
    }
    while(i>=0 && retenue=='1');
    // s'il y a une retenue, on agrandit la mémoire allouée à la chaîne
    if(retenue!='0')
    {  if((q = (char *)realloc(p,sizeof(char)*longueur+1))==NULL)
          printf("incremente : erreur lors du realloc\n");
       else
       {  for(i=longueur-1;i>0;i--)
             q[i]=p[i-1];
          q[i]=retenue;
          *n=q;
       }
    }
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 34 | répertoire téléphonique

1 Numéros de téléphone

Exercice 2

```
void affiche_numero(int* numero)
{  int *p = numero;
   while(*p!=FIN)
   {  printf("%d",*p);
      p++;
   }
}
```

Exercice 3

```
void saisis_numero(int** numero)
{  char c;
   int *p=NULL;
   int i = 0;
   do
   {  // on saisit un seul caractere
      c = getche();
      // on verifie si c'est un chiffre
      if(c>='0' && c<='9')
      {  // on agrandit le tableau courant
         if((p = (int*)realloc(p, (i+1)*sizeof(int))) == NULL)
            printf("saisis_numero: erreur lors du realloc\n");
         // on y place le nouveau chiffre
         else
         {  p[i] = c - '0';
            i++;
         }
      }
   }
   while(c!='\r' /*pour windows*/ && c!='\n' /*pour linux*/);
   // on rajoute le -1 final
   p[i] = FIN;
   // on passe le resultat pas adresse
   *numero = p;
}
```

2 Contacts

Exercice 5

- Fonction d'affichage :

```
void affiche_contact(t_contact contact)
{  printf("%s : ",contact.nom);
   affiche_numero(contact.numero);
}
```

- Fonction de saisie :

```
void saisis_contact(t_contact* contact)
{  // nom
   char c,*p=NULL;
   int i = 0;
   printf("Entrez le nom du contact : ");
```

```
    do
    {  c = getche();
       if((p = (char*)realloc(p, (i+1)*sizeof(char))) == NULL)
          printf("saisis_chaine_direct: erreur lors du realloc\n");
       else
       {  p[i]=c;
          i++;
       }
    }
    while(c!='\r' /*pour windows*/ && c!='\n' /*pour linux*/);
    p[i-1]='\0';
    contact->nom = p;

    // numero
    printf("Entrez le numero de %s : ",contact->nom);
    saisis_numero(&(contact->numero));
}
```

3 Répertoire

Exercice 7

```
void initialise_repertoire (t_repertoire* repertoire)
{  repertoire->contacts = NULL;
   repertoire->taille = 0;
}
```

Exercice 8

```
void ajoute_contact(t_repertoire* repertoire, t_contact contact)
{  int taille = repertoire->taille;
   t_contact *p = repertoire->contacts;

   if((p = (t_contact*)realloc(p, (taille+1)*sizeof(t_contact))) == NULL)
      printf("ajoute_contact: erreur lors du realloc\n");
   else
   {  // ici on peut faire une affectation superficielle entre deux valeurs
      // de type structure (ce qui n'est pas toujours vrai, en general)
      p[taille] = contact;
      repertoire->taille = taille + 1;
      repertoire->contacts = p;
   }
}
```

Exercice 9

```
void affiche_repertoire(t_repertoire repertoire)
{  int i;

   printf("Taille du repertoire : %d contact",repertoire.taille);
   if(repertoire.taille>1)
      printf("s");
   printf(".\n");

   for(i=0;i<repertoire.taille;i++)
   {  printf("%3d. ",i+1);
      affiche_contact(repertoire.contacts[i]);
      printf("\n");
   }
}
```

Exercice 10

```
t_contact* cherche_nom(t_repertoire repertoire, char* nom)
{  int i=0, comp;
   t_contact *resultat = NULL;
   while(resultat==NULL && i<repertoire.taille)
   {  comp = compare_chaines(nom,repertoire.contacts[i].nom);
      if(comp==0)
```

```
            resultat = &repertoire.contacts[i];
         else
            i++;
      }
      return resultat;
   }
```

Exercice 11

```
   void supprime_contact(t_repertoire* repertoire, t_contact* contact)
   {  //  on  calcule  la  position  du  contact  en  utilisant  l'arithmetique  des
pointeurs
      // on suppose que le parametre contact n'est pas NULL et que ce contact
      // apartient bien au repertoire.
      int index = contact-repertoire->contacts;

      int taille = repertoire->taille;
      int i;
      for(i=index;i<taille-1;i++)
         repertoire->contacts[i] = repertoire->contacts[i+1];

      repertoire->taille = taille - 1;
   }
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 35 | fichiers et arguments d'un programme

1 Arguments d'un programme

Exercice 1

```
void affiche_arguments(int argc, char *argv[])
{  int i;
   printf("Nombre d'arguments recus : %d\n",argc);
   printf("Nom du programme : %s\n",argv[0]);
   for(i=1;i<argc;i++)
      printf("Argument %d : %s\n",i,argv[i]);
}
```

2 Accès non-formaté à un fichier

Exercice 3

```
int ecris_chaine()
{  int resultat=0;

   // on saisit la chaine
   char chaine[10];
   printf("entrez la chaine : ");
   scanf("%s",chaine);

   // on ouvre le fichier
   FILE *fp;
   if((fp = fopen("donnees.txt", "w")) == NULL)
   {  //traitement en cas d'erreur
      printf("ecris_chaine: erreur dans fopen\n");
      return -1;
   }

   // on écrit la chaîne
   while(chaine[resultat] !='\0')
   {  if((fputc(chaine[resultat],fp)) == EOF)
      {  // traitement de l'erreur
         printf("ecris_chaine: erreur dans fputc\n");
         return -1;
      }
      resultat++;
   }

   // on ferme le fichier
   if((fclose(fp)) == EOF)
   {  // traitement de l'erreur
      printf("ecris_chaine: erreur dans fclose\n");
      return -1;
   }

   return resultat;
}
```

Exercice 4

```
int lis_chaine()
{  char chaine[10];
```

```
    FILE *fp;
    int resultat = 0;

    // on ouvre le fichier
    if((fp = fopen("donnees.txt", "r")) == NULL)
    {  //traitement en cas d'erreur
       printf("lis_chaine: erreur dans fopen\n");
       return -1;
    }

    // on lit la chaîne
    while((chaine[resultat]=fgetc(fp)) != EOF)
       resultat++;
    if(!feof(fp))
    {  // traitement de l'erreur
       printf("lis_chaine: erreur dans fgetc\n");
       return -1;
    }
    chaine[resultat]='\0';

    // on ferme le fichier
    if((fclose(fp)) == EOF)
    {  // traitement de l'erreur
       printf("lis_chaine: erreur dans fclose\n");
       return -1;
    }

    // on affiche la chaine
    printf("La chaine lue est : \"%s\"\n",chaine);

    return resultat;
}
```

Exercice 5

```
int copie_fichier(char *source, char *cible)
{  int resultat = 0;

   // on ouvre le fichier source en lecture
   FILE *src;
   if((src = fopen(source, "r")) == NULL)
   {  //traitement en cas d'erreur
      printf("copie_fichier: erreur dans fopen\n");
      return -1;
   }
   // on ouvre le fichier cible en écriture
   FILE *cbl;
   if((cbl = fopen(cible, "w")) == NULL)
   {  //traitement en cas d'erreur
      printf("copie_fichier: erreur dans fopen\n");
      return -1;
   }

   // on lit la source ligne par ligne
   // en écrivant chaque ligne dans la cible
   char tampon[50];
   while(fgets(tampon,50,src) != NULL)
   {  if((fputs(tampon,cbl)) == EOF)
      {  // traitement de l'erreur
         printf("copie_fichier: erreur dans fputs\n");
         return -1;
      }
      printf("%s",tampon);
      resultat++;
   }
   if(!feof(src))
   {  // traitement de l'erreur
      printf("copie_fichier: erreur dans fgets\n");
```

```
        return -1;
    }

    // on ferme les fichiers
    if((fclose(src)) == EOF)
    {  // traitement de l'erreur
       printf("copie_fichier: erreur dans fclose\n");
       return -1;
    }
    if((fclose(cbl)) == EOF)
    {  // traitement de l'erreur
       printf("copie_fichier: erreur dans fclose\n");
       return -1;
    }

    printf("\n");
    return resultat;
}
```

Exercice 6

```
int main(int argc, char *argv[])
{  if(argc<3)
   {  printf("Erreur: Il n'y a pas assez de paramètres (%d au lieu de 2)
                                                                \n",argc-1);
      exit(EXIT_FAILURE);
   }
   else if(argc>3)
   {  printf("Erreur: Il y a trop de paramètres (%d au lieu de 2)\n",argc-1);
      exit(EXIT_FAILURE);
   }
   else
      copie_fichier(argv[1],argv[2]);

   return EXIT_SUCCESS;
}
```

3 Accès formaté à un fichier

Exercice 7

```
int sauve_date(t_date d, FILE *fp)
{  int resultat = 0;
   if(fprintf(fp,"%d/%d/%d",d.jour,d.mois,d.annee) == EOF)
   {  // traitement de l'erreur
      printf("sauve_date: erreur dans fprintf\n");
      resultat = -1;
   }
   return resultat;
}
```

Exercice 8

```
int charge_date(t_date *d, FILE *fp)
{  int resultat = 0;
   if(fscanf(fp,"%d/%d/%d",&(d->jour),&(d->mois),&(d->annee))== EOF)
   {  // traitement de l'erreur
      printf("charge_date: erreur dans fscanf\n");
      resultat = -1;
   }
   return resultat;
}
```

Exercice 9

La modification réalisée dans le fichier introduit une valeur trop grande pour le type entier, ce qui explique la valeur incorrecte (par rapport à celle du fichier) affichée.

Exercice 10

Lors de la lecture, `fscanf` tente absolument de respecter le format, et cherche donc des `'/'` qui n'existent pas, ce qui provoque l'affichage de valeurs erronées.

Exercice 11

- Enregistrement d'un étudiant dans un fichier :

```
int sauve_etudiant(t_etudiant e, FILE *fp)
{  int resultat = 0;
   if(fprintf(fp,"%s\t%s\t%d\t",e.prenom,e.nom,e.genre) == EOF)
   {  printf("sauve_etudiant: erreur dans fprintf\n");
      resultat = -1;
   }
   else
      resultat = sauve_date(e.naissance,fp);
   return resultat;
}
```

- Chargement d'un étudiant depuis un fichier :

```
int charge_etudiant(t_etudiant *e, FILE *fp)
{  int resultat = 0;
   if(fscanf(fp,"%s\t%s\t%d\t",e->prenom,e->nom,&(e->genre))== EOF)
   {  printf("charge_etudiant: erreur dans fscanf\n");
      resultat = -1;
   }
   else
      resultat = charge_date(&(e->naissance),fp);
   return resultat;
}
```

Exercice 12

- Enregistrement d'une promotion dans un fichier :

```
int sauve_promotion(t_promo p, FILE *fp)
{  int resultat=0,temp;

   // on sauve la taille de la promotion
   if(fprintf(fp,"%d\n",p.taille) == EOF)
   {  printf("sauve_promotion: erreur dans fprintf\n");
      resultat = -1;
   }

   // puis on sauve chaque etudiant
   while(resultat>-1 && resultat<p.taille)
   {  temp = sauve_etudiant(p.etudiants[resultat],fp);
      if(temp==-1)
         resultat = temp;
      else
      {  if(fprintf(fp,"\n") == EOF)
         {  printf("sauve_promotion: erreur dans fprintf\n");
            resultat = -1;
         }
         else
            resultat++;
      }
   }

   return resultat;
}
```

- Chargement d'une promotion depuis un fichier :

```
int charge_promotion(t_promo *p, FILE *fp)
{  int resultat=0,temp;

   // on charge la taille de la promotion
   if(fscanf(fp,"%d\n",&(p->taille))== EOF)
   {  printf("charge_promotion: erreur dans fscanf\n");
```

```
            resultat = -1;
        }

        // puis on charge chaque etudiant
        while(resultat>-1 && resultat<p->taille)
        {   temp = charge_etudiant(&(p->etudiants[resultat]),fp);
            if(temp==-1)
                resultat = temp;
            else
            {   if(fscanf(fp,"\n") == EOF)
                {   printf("sauve_promotion: erreur dans fscanf\n");
                    resultat = -1;
                }
                else
                    resultat++;
            }
        }

        return resultat;
    }
```