Démonstration inf 3135
Construction et maintenance de logiciel.

Bruno Malenfant

janvier 2010

Semaine 1. (14 janvier)

Sujet : pas de démo,

Semaine 2. (21 janvier)

Sujet : Unix, Édition de texte, utilisation de gcc, connection a distance.

Problème 2.1 (10 minutes)

Leurs montrer comment se connecter en lab sur leurs compte 'rayon1'.

Problème 2.2 (30 minutes)

Leurs faire des exemples d'utilisation des commandes suivantes : ls, cp, mv, touch, mkdir, cd. (Et toute autre commandes que vous jugez utile.) Voir la commande 'man' pour montrer les commandes Unix/Linux.

Problème 2.3 (20 minutes)

Leurs montrer un éditeur de texte de votre choix (par exemple : emacs, xemacs, eclypse,...). Comme exemple leurs faire écrire un programme (a la 'hello world'). revoir la commande 'man' avec 'printf' cette fois pour montrer que les routines C sont aussi dans le manuel.

Problème 2.4 (20 minutes)

Leurs montrer comment utiliser gcc.

- comment exécuter le résultat a.out (avec le ./)
- -o : change le nom de l'exécutable
- -E : pré-compilation seulement, faire des exemples avec #define, #ifdef, #include, sur des petits fichiers.
- -c : compile sans l'edition de lien

Pour cela vous pouvez utiliser les deux fichiers suivants. Montrez les différence en ne plaçant pas de `#define PRESENT`, en le plaçant avant les `#include` et après les `#include`.

exemple, fichier exa.c

```
#ifdef PRESENT

void f( int x, int y ) {
  return x + y;
}

#else

void f( int x, int y ) {
  return x - y;
}

#endif
```

exemple fichier main.c

```
#include "exa.c"
#include <stdio.h>

int main(){
  printf( "%i\n", f( 4, 2 ) );
  return 0;
}
```

Problème 2.5 (10 minutes)

Leurs donner une/des technique(s) pour se connecter à distance.

Semaine 3. (28 janvier)

Sujet : Expression, if et boucle.

Problème 3.1 (15 minutes)

Écrivez un code qui calcule la somme des nombres de 0 à n, où n est une constante dans le programme. Cette somme sera ensuite affichée.

Solution :

```
#include <stdio.h>

int main()
{
  const int n = 10;
  int i = 0;
  int somme = 0;

  for( i = 1; i <= n; i++ )
  {
    somme += i;
  }

  printf( "resultat : %i\n", somme );

  return 0;
}
```

Problème 3.2 (10 minutes)

Modifiez le code du problème 3.1 de tel sorte que seulement les nombres pair soient additionnés.

Solution :

```
#include <stdio.h>

int main()
{
  const int n = 15;
```

```
  int i = 0;
  int somme = 0;

  for( i = 0; i <= n; i += 2 )
  {
    somme += i;
  }

  printf( "resultat : %i\n", somme );

  return 0;
}
```

Problème 3.3 (15 minutes)

Écrivez un programme qui affiche les nombres de 48 à 126. Un nombre par ligne doit être affiché.

Solution :

```
#include <stdio.h>

int main()
{
  int i = 0;

  for( i = 48; i <= 126; i++ )
  {
    printf( "%i\n", i );
  }

  return 0;
}
```

Problème 3.4 (10 minutes)

Modifiez le programme 3.3 pour qu'il affiche la valeur caractère et la valeur numérique des nombre de 48 à 126.

Solution :

```
#include <stdio.h>

int main()
```

```
{
  int i = 0;

  for( i = 48; i <= 126; i++ )
  {
    printf( "%i => %c\n", i, i );
  }

  return 0;
}
```

Problème 3.5 (10 minutes)

Modifiez le programme 3.4 pour qu'il n'affiche les caractères que pour les valeurs 48 à 57, 65 à 90 et 97 à 122.

Solution :

```
#include <stdio.h>

int main()
{
  int i = 0;

  for( i = 48; i <= 126; i++ )
  {
    if( ( i >= 48 && i <= 57 )
         ||
        ( i >= 65 && i <= 90 )
         ||
        ( i >= 97 &&  i <= 122 ) )
    {
      printf( "%i => %c\n", i, i );
    }
  }

  return 0;
}
```

Semaine 4. (4 février)

Sujet : Lisibilité

Problème 4.1 (15 minutes)

Leur montrer le fonctionnement de printf : %s, %i, %c, ...

Problème 4.2 (10 minutes)

Voici plusieurs versions différentes pour le même problème. Discutez des avantages et désavantage de chacun.

Version 1 :

```
#include <stdio.h>

int main()
{
  int i = 0;

  for( i = 48; i <= 126; i++ )
  {
    if( ( i >= 48 && i <= 57 )
         ||
        ( i >= 65 && i <= 90 )
         ||
        ( i >= 97 && i <= 122 ) )
    {
      printf( "%i => %c\n", i, i );
    }
  }

  return 0;
}
```

- Bien écrit, le 'if' est très lisible.
- Plusieurs itération de boucles inutiles puisqu'il n'affiche pas tout.

Version 2 :

```c
#include <stdio.h>

int main()
{
  int i = 0;

  for( i = 48; i <= 126; i++ )
  {
    if( i >= 48 && i <= 57 )

    {
      printf( "%i => %c\n", i, i );
    }

    if( i >= 65 && i <= 90 )
    {
      printf( "%i => %c\n", i, i );
    }

    if( i >= 97 && i <= 122 )
    {
      printf( "%i => %c\n", i, i );
    }
  }

  return 0;
}
```

- La pire version des trois.
- Code répété inutilement.
- Plusieurs itération de boucles inutiles puisqu'il n'affiche pas tout.

Version 3 :

```c
#include <stdio.h>

int main()
{
  int i = 0;

  for( i = 48; i <= 57; i++ )
  {
    printf( "%i => %c\n", i, i );
```

```
  }

  for( i = 65; i <= 90; i++ )
  {
    printf( "%i => %c\n", i, i );
  }

  for( i = 97; i <= 122; i++ )
  {
    printf( "%i => %c\n", i, i );
  }

  return 0;
}
```

- Aucun 'if', aucun tours de boucle inutile.
- Le code reste très lisible.

Problème 4.3 (15 minutes)

Réécrivez ce code pour qu'il soit plus lisible :

```
#include <stdio.h>

int main()
{
  double i = -40.0;

  printf( "+------------+------------+------------+------------+\n" );
  printf( "|  Celsius   |   Kelvin   | Fahrenheit |  Rankine   |\n" );
  printf( "+------------+------------+------------+------------+\n" );
  while( i <= 40.0 ) {
    double f = i * 9 / 5 + 32;
    double r = i + 273.15 ;

    printf( "| %10.2f | %10.2f | %10.2f | %10.2f |\n", i, i + 273.15, f, r * 9 / 5 );

    i += 5.0;
  }
  printf( "+------------+------------+------------+------------+\n" );


  return 0;
}
```

Solution :

```c
#include <stdio.h>

int main() {
  const double ZERO_KELVIN = 273.15;
  const double FACTEUR_CONVERSION = 9.0 / 5.0;
  const double ZERO_FAHRENHEIT = 32.0;
  double celsius = 0.0;

  printf( "+------------+------------+------------+------------+\n" );
  printf( "|  Celsius   |   Kelvin   | Fahrenheit |  Rankine   |\n" );
  printf( "+------------+------------+------------+------------+\n" );
  for( celsius = -40.0; celsius <= 40.0; celsius += 5.0 ) {
    double kelvin = celsius + ZERO_KELVIN;
    double fahrenheit = celsius * FACTEUR_CONVERSION + ZERO_FAHRENHEIT;
    double rankine = kelvin * FACTEUR_CONVERSION;

    printf( "| %10.2f | %10.2f | %10.2f | %10.2f |\n",
            celsius, kelvin, fahrenheit, rankine );
  }
  printf( "+------------+------------+------------+------------+\n" );

  return 0;
}
```

Problème 4.4 (30 minutes)

Voici un code C, ce code n'est pas très bien écrit, dites pourquoi.

```c
#include <stdio.h>
#include <math.h>

#define TAILLE 20

int main(){
  double tab[TAILLE] =
    { 2, 6, 4, 8, 1, 6, 9, 4, 2, 8, 3, 6, 2, 6, 2, 6, 87, 2, 4, 67 };
  int i = 0;

  double moyenneArithmetique = 0.0;
  double moyenneGeometrique = 1.0;
  double moyenneHarmonique = 0.0;
  double moyenneQuadratique = 0.0;
```

```
  for( i = 0; i < TAILLE; i++ ) {
    moyenneArithmetique += tab[i];
  }
  for( i = 0; i < TAILLE; i++ ) {
    moyenneGeometrique *= tab[i];
  }
  for( i = 0; i < TAILLE; i++ ) {
    moyenneHarmonique += ( 1.0 / tab[i] );
  }
  for( i = 0; i < TAILLE; i++ ) {
    moyenneQuadratique += ( tab[i] * tab[i] );
  }

  moyenneArithmetique /= TAILLE;
  moyenneGeometrique = pow( moyenneGeometrique, 1.0 / TAILLE );
  moyenneHarmonique = TAILLE / moyenneHarmonique;
  moyenneQuadratique /= TAILLE;
  moyenneQuadratique = sqrt( moyenneQuadratique );

  printf( "Moyenne arithmetique : %f\n", moyenneArithmetique );
  printf( "Moyenne geometrique : %f\n", moyenneGeometrique );
  printf( "Moyenne harmonique : %f\n", moyenneHarmonique );
  printf( "Moyenne quadratique : %f\n", moyenneQuadratique );

  return 0;
}
```

réponse : Ce code contient quatres calcules imbriqué entre eux, cela enlève de la lisibilité. Aussi il y a quatres boucles similaires, cela est moins performant qu'une seule boucles.

Essayez de corrigé le code pour le rendre plus performant :

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

#define TAILLE 20

int main(){
  double tab[TAILLE] =
    { 2, 6, 4, 8, 1, 6, 9, 4, 2, 8, 3, 6, 2, 6, 2, 6, 87, 2, 4, 67 };
  int i = 0;

  double moyenneArithmetique = 0.0;
  double moyenneGeometrique = 1.0;
```

```
  double moyenneHarmonique = 0.0;
  double moyenneQuadratique = 0.0;

  for( i = 0; i < TAILLE; i++ ) {
    moyenneArithmetique += tab[i];
    moyenneGeometrique *= tab[i];
    moyenneHarmonique += ( 1.0 / tab[i] );
    moyenneQuadratique += ( tab[i] * tab[i] );
  }

  moyenneArithmetique /= TAILLE;
  moyenneGeometrique = pow( moyenneGeometrique, 1.0 / TAILLE );
  moyenneHarmonique = TAILLE / moyenneHarmonique;
  moyenneQuadratique /= TAILLE;
  moyenneQuadratique = sqrt( moyenneQuadratique );

  printf( "Moyenne arithmetique : %f\n", moyenneArithmetique );
  printf( "Moyenne geometrique : %f\n", moyenneGeometrique );
  printf( "Moyenne harmonique : %f\n", moyenneHarmonique );
  printf( "Moyenne quadratique : %f\n", moyenneQuadratique );

  return 0;
}
```

Dans ce cas bien que le code soit plus performant, il reste illisible.
Faites une version plus lisible.

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

#define TAILLE 20

int main(){
  double tab[TAILLE] =
    { 2, 6, 4, 8, 1, 6, 9, 4, 2, 8, 3, 6, 2, 6, 2, 6, 87, 2, 4, 67 };
  int i = 0;


  double moyenneArithmetique = 0.0;
  for( i = 0; i < TAILLE; i++ ) {
    moyenneArithmetique += tab[i];
  }
  moyenneArithmetique /= TAILLE;
  printf( "Moyenne arithmetique : %f\n", moyenneArithmetique );
```

```
  double moyenneGeometrique = 1.0;
  for( i = 0; i < TAILLE; i++ ) {
    moyenneGeometrique *= tab[i];
  }
  moyenneGeometrique = pow( moyenneGeometrique, 1.0 / TAILLE );
  printf( "Moyenne geometrique : %f\n", moyenneGeometrique );


  double moyenneHarmonique = 0.0;
  for( i = 0; i < TAILLE; i++ ) {
    moyenneHarmonique += ( 1.0 / tab[i] );
  }
  moyenneHarmonique = TAILLE / moyenneHarmonique;
  printf( "Moyenne harmonique : %f\n", moyenneHarmonique );


  double moyenneQuadratique = 0.0;
  for( i = 0; i < TAILLE; i++ ) {
    moyenneQuadratique += ( tab[i] * tab[i] );
  }
  moyenneQuadratique /= TAILLE;
  moyenneQuadratique = sqrt( moyenneQuadratique );
  printf( "Moyenne quadratique : %f\n", moyenneQuadratique );


  return 0;
}
```

Ici le code est plus lisible, il est facile de voir que nous pouvons rendre ce code procédurale en isolant les différentes moyenne, c'est un bon indice que le code est mieux divisé.

Problème 4.5 (15 minutes)

Voici un code C :

```
#define TAILLE 10

int main() {
  int tab[TAILLE] = {4, 6, 8, 3, 9, 4, 6, 2, 6, 7};
  int i = 0;

  int x = 8;
  int rep = 0;
```

```
  for( i = 0; i < TAILLE; i++ ){
    if( tab[i] > x ) {
      rep = i;
      i = TAILLE;
    }
  }
  if( rep != TAILLE ) {
    // trouve
  } else {
    // non-trouve
  }
  return 0;
}
```

Ce code utilise une boucle et pour sortir il place la valeur de l'itérateur plus haut que TAILLE. Ceci est une très mauvaise technique, réécrivez ce code pour qu'il soit acceptable.

```
#define TAILLE 10

int main() {
  int tab[TAILLE] = {4, 6, 8, 3, 9, 4, 6, 2, 6, 7};
  int i = 0;

  int x = 8;
  int rep = 0;

  while( i < TAILLE && tab[i] < x ) {
    i++;
  }
  if( rep != TAILLE ) {
    // trouve
  } else {
    // non-trouve
  }
  return 0;
}
```

Semaine 5. (11 février)

Sujet : Cast, sizeof, tableau, allocation et pointeur.

Problème 5.1 (10 minutes)

Faire fonctionner le code suivant, pour qu'ils voient l'utilisation de sizeof.

```
#include <stdio.h>

int main()
{
  printf("char : %i octet\n", sizeof( char ) );
  printf("short : %i octets\n", sizeof( short ) );
  printf("int : %i octets\n", sizeof( int ) );
  printf("long : %i octets\n", sizeof( long ) );
  printf("float : %i octets\n", sizeof( float ) );
  printf("double : %i octets\n", sizeof( double ) );
  printf("long double : %i octets\n", sizeof( long double ) );

  return 0;
}
```

Problème 5.2 (10 minutes)

Un autre programme d'exemple pour voir le fonctionnement de sizeof et du cast.

```
#include <stdio.h>

int main() {
  short c = 100;
  printf( "sizeof( c ) = %i\n", sizeof( c ) );
  printf( "sizeof( (double)c ) = %i\n", sizeof( (double)c ) );

  char t1[10];
  int t2[10];
  int t3[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
  printf( "sizeof( t1 ) = %i\n", sizeof( t1 ) );
  printf( "sizeof( t2 ) = %i\n", sizeof( t2 ) );
  printf( "sizeof( t3 ) = %i\n", sizeof( t3 ) );
```

```
  return 0;
}
```

Problème 5.3 (15 minutes)

Leurs montrer le code suivant qui contient une allocation dynamique d'un tableau. (leurs expliquer le fonctionnement de malloc.) (leurs montrer l'acces normal au tableau.)

```
#include <stdio.h>

int main()
  const TAILLE_TAB = 20;
  int * tab = (int *)malloc( sizeof( int ) * TAILLE_TAB );
  int i = 0;

  for( i = 0; i < TAILLE_TAB; i ++ )
    tab[i] = 5 * i + 2;


  return 0;
```

Problème 5.4 (20 minutes)

Construire une fonction qui recoit un tableau de double et sa taille en entrées. Cette routine retourne la moyenne arithmétique des éléments du tableau.

Problème 5.5 (15 minutes)

Écrire une procédure qui reçoit deux pointeurs sur des entiers et échange les valeurs pointées.

Problème 5.6 (15 minutes)

Écrire une fonction qui reçoit deux pointeurs d'entier représentants des tableaux et une valeur entière (n). La procédure copie 'n' valeur du premier tableau vers le deuxieme.

Problème 5.7 (15 minutes)

Construire une procédure qui alloue un tableau, l'initialise à 0 et retourne un pointeur sur le début du tableau. La procédure reçoit une valeur indiquant le nombre de case à allouer.

Semaine 6. (18 février)

Sujet : Revision intra

Voici des questions d'examens, certaines ont déjà des réponses.

Problème 6.1 (15 minutes)

Que va afficher le code suivant :

```
#include <stdio.h>

int main() {
  int u = 1;
  int v = 5;
  char i[6][6];
  char e[] = {'w','w','a','w','w','w','s','s','s','z','s','s','s','s','s'};
  int j = 0;
  int k = 0;

  for( j = 0; j < 6; j ++ ){
    for( k = 0; k < 6; k ++ ){
      i[j][k] = '.';
    }
  }

  for( j = 0; j < 15; j ++ ){
    i[u][v] = '0';
    u = u + ((e[j] == 's' && u != 5)?1:0) - ((e[j] == 'a' && u != 0)?1:0);
    v = v + ((e[j] == 'z' && v != 5)?1:0) - ((e[j] == 'w' && v != 0)?1:0);
  }

  for( j = 0; j < 6; j ++ ){
    for( k = 0; k < 6; k ++ ){
      printf( "%c", i[k][j] );
    }
    printf( "\n" );
  }

  return 0;
}
```

réponse :

```
OOOO..
O..OOO
O.....
OO....
.O....
.O....
```

Problème 6.2 (15 minutes)

Écrivez une routine qui replace les éléments d'un tableau en ordre **décroissant**. Vous pouvez utiliser l'algorithme de tri que vous voulez.

```
void trier( int * vecteur, int nbrElement )
```

Problème 6.3 (15 minutes)

Que va affichez le programme suivant :

```
#include <stdio.h>
char p( char * a ) {
  (*a)++;
  a++;
  return *a;
}

void r( char * a, char * b ) {
  *a = *b;
}

void s( char * a, char c, char i ) {
  for( i-- ; i >= 0; i-- ) {
    *(a + i) = c;
  }
}

int main( void ) {
  int i = 0;
  char a = 30;
  char b = 40;
  char c[a];
  char * d = c;
  s( c, b, a );
  p(d);
  p(d);
```

```
  d += 2;
  r( d, c );
  p( c );
  r( d - 1, c );
  p( c );
  p( c );
  p( c );
  r( d + 1, c );
  for( i = 0; i < 10; i++ ) {
    printf( "%i ", c[i] );
  }
  printf( "\n" );
  return 0;
}
```

réponse :
46 43 42 46 40 40 40 40 40 40

Problème 6.4 (10 minutes)

Expliquez la différence entre cohésion communicationnelle et cohésion temporelle.

Réponse : la cohésion communicationnelle utilise une données de départ qui sert à toutes les tâches. La cohésion temporelle n'a pas de données communes aux tâches.

Problème 6.5 (10 minutes)

Nous avons un module de traitement de **Date**. Ce module contient une fonction qui calcule si une date est bissextile et ensuite utilise ce résultat pour calculer le nombre de jours dans l'année. Cette fonction retourne le nombre de jour dans l'année. Proposez un nom pour cette fonction. Quel est la cohésion de cette routine à l'intérieur du contexte de **Date**.

Réponse : nombreJoursAnnée, cohésion fonctionnelle. (séquentielle est acceptable comme réponse.)

Problème 6.6 (20 minutes)

Soit les déclarations suivantes :

```
struct _date {
  unsigned short jour;
  unsigned short mois;
  unsigned int annee;
};
```

```
struct _entree {
  struct _date date;
  char * description;
  int montant_cent;
};
```

Chaque entrée représente la description d'un item. Cette description contient un maximum de 50 caractères. Le champs `montant_cent` indique la somme payée pour cette item en cent. Le champ `date` indique la date de l'achat. Une facture est représentée par un tableau d'entrées (struct _entree *). Vous devez écrire une procédure qui affiche la facture d'un client. Cette procédure affiche la facture à raison de un item par ligne. Une ligne commence par la date affichée selon le format suivant : aaaa/mm/jj. Ensuite vous devez afficher 3 espaces. La description est affichée après les trois (3) espaces. Si la description contient moins de 50 caractères alors des espaces sont affichées jusqu'à ce qu'il y ait 50 caractères affichés. Ensuite affichez 5 espaces supplémentaires. Enfin affichez le montant avec un point séparant les dollars des cents. Le montant est suivit d'une espace et du caractère '$'. Le montant est affiché sur 10 cases. Le montant doit être aligné à droite. Après avoir affiché tout les items vous affichez le total. Le total n'a pas de date. La description pour le total est 'Total'. Le total est affiché dans le même format que les montants. Remarque : la facture n'a pas d'entrée pour le total, votre procédure doit le calculer. Exemple d'affichage :

```
2007/05/04   Televiseur                                          1299.99 $
2007/05/11   Livre "The Java Language Specification"              49.99 $
2007/05/13   Imprimante                                           69.99 $
             Total                                              1419.97 $
```

Voici l'entête de la procédure :

```
void afficher_facture( struct _entree * a_facture, int a_nombreEntree );
```

Semaine 7. (25 février)

Sujet :

Semaine 8. (11 mars)

Sujet : Chaine de caracteres, arguments du main

Problème 8.1 (20 minutes)

Construire une routine qui parcours une chaîne de caractères et remplace les majuscules par des minuscules.

Problème 8.2 (10 minutes)

Nous avons une chaîne de caractères existante (constante) et vous devez utiliser strlen pour allouer de l'espace afin de faire un double de la chaîne avec strcpy.

Problème 8.3 (30 minutes)

Construire une routine de tri (sélection ou insertion) sur un tableau de chaîne de caractères en utilisant strcmp.

Problème 8.4 (10 minutes)

Exécutez le programme suivant avec des arguments sur la ligne de commande :

```
#include <stdio.h>

int main( int argc, char * argv[] ) {
  int i = 0;

  for( i = 0; i < argc; i++ )
    printf( "%s\", argv[i] );


  return 0;
}
```

Problème 8.5 (10 minutes)

Construire un programme qui tri les arguments reçus en ordre alphabétique et ensuite les affiche. (utilisez les programme des derniers numéro.)

Semaine 9. (18 mars)

Sujet : Makefile et compilation séparée

Problème 9.1 (30 minutes)

Sur mon site il y a un projet complet : 'Petit projet et sa doc'. telecharger le 'code' et defaire l'archive avec 'gunzip' et 'tar xvf'.

- Regardez avec comment les .h sont fais (define, ifndef, ...)
- Regardez aussi la divisions .h/.c, faires remarquer quel fichier contient les typedef, struct, ...
- Regardez le makefile.
- Faire compiler le projet avec la commande 'make'

Problème 9.2 (20 minutes)

Construire un projet simple :

- Un fichier avec le main.
- 1 module avec sa version .h et .c. Ajoutez une procedure dans ce module. Faire un appel de cette procedure dans le main.
- Le .c doit inclure le .h, le main doit inclure le .h.
- Faire une compilation separe du projet sans makefile.
- Construire le makefile

Semaine 10. (25 mars)

Sujet : Script bash

Problème 10.1 (15 minutes)

Écrivez et essayez le script suivant, n'oubliez pas de rendre le script exécutable. Ce script demande le nom d'un fichier pour placer le résultat, n'utilisez pas un fichier existant car il sera effacé par le script. Testez ce script dans un répertoire contenant des fichier de code c. Puisque vous allez travailler un script il est important d'utiliser un répertoire contenant du code auquel vous ne tenez pas. (faites une copie d'un répertoire.)

```
#!/bin/bash

if [ -e$1 ]
then
  \rm -f $1
fi

touch $1

for fichier in `ls *.c`
do
  echo >> $1
  echo "/*** fichier ${fichier} ***/" >> $1
  cat -b ${fichier} >> $1
done
```

Problème 10.2 (10 minutes)

Modifiez le script précédant pour qu'il ajoute les fichiers .h dans l'archive.

Problème 10.3 (15 minutes)

Réécricez le script suivant et étudiez le.

```
#!/bin/bash

if [ -e$1 ]
```

```
then
  \rm -f $1
fi

touch $1

for fichier in `ls *.[ch]`
do
  echo >> $1
  echo "/*** fichier ${fichier} ***/" >> $1
  grep "include" ${fichier} >> $1
done
```

Problème 10.4 (20 minutes)

Réécrivez le script du numéro précédant pour qu'il s'appele récursivement dans les sous répertoires du répertoire courrant.

Semaine 11. (1 avril)

Sujet : Programmation défensive : assert

Problème 11.1 (10 minutes)
Regarder les assertions de la routines additionner (GrandNombres.c) et son contrat (GrandNombres.h), voir les relations entre les deux. (dans l'onglet exemple : assertion, erreur, exception.

Problème 11.2 (15 minutes)
Écrire une fonction qui à la signature suivante :

```
int iemElement( int * tableau, int taille, int index );
```

La routine retourne la valeur à la position index du tableau. Elle s'assure que index est valide (entre 0 et taille) et que le pointeur de tableau n'est pas NULL, le tout avec des assertions.

Problème 11.3 (20 minutes)

Voici la description de deux structures de données :

```
typedef struct {
  int exceptionLevee;
  double discriminant;
}
ExceptionDiscriminantNegatif;

typedef struct {
  double premiereValeur;
  double deuxiemeValeur;
}
PaireDouble;
```

Écrivez le code de la routine suivante avec ses erreurs et assertions.

```
/**
 * Cette fonction calcule les zéros de la fonction quadratique
 * ax^2 + bx + c
 * Le pointeur d'exception ne peu pas etre NULL (assertion)
 * Le discriminant est calculé : b^2 - 4ac
 * Si le résultat est négatif alors exceptionLevee de la structure
 *   d'exception est placé à vrai et le champs discriminant prend
```

```
 *   la valeur du discriminant.  La routine se termine sans calculer
 *   de résultat.
 * Calculer les deux résultats :
 *  1 : (-b + discriminant) / (2a)
 *  2 : (-b - discriminant) / (2a)
 * Si 2a donne 0 alors affichez et message d'erreur et quittez avec exit.
 * Placez les résultats dans une structure PaireDouble et la retourner.
 * Remarque : si le discriminant == 0 alors les deux résultats seront égaux.
 */
PaireDouble zeroQuadratique( double a_a, double a_b, double a_c,
                             ExceptionDiscriminantNegatif a_exception );
```

Problème 11.4 (20 minutes)

Modifiez le code du numéro précédant pour enlever l'exception. Pour cela la valeur de retour peu avoir trois possibilité :

- 2 valeurs
- une seule valeur
- exception discriminant négatif

Utilisez une Union pour le choix du résultat et un type enuméré pour identifier le résultat.

Problème 11.5 (10 minutes)

Pour la fonction zeroQuadratique remplacez l'erreur par une exception division par zéro. Vous pouvez travailler sur la version de votre choix pour cette fonction.

Problème 11.6 (10 minutes)

Toujours pour la fonction quadratique, ajoutez des assertions à la fin du programme qui vérifie que les valeurs de retour sont correcte. Pour cela utilisez les dans la formule et vérifiez que cela donne une valeur près de zéro. (avec des valeurs point flottant il n'est jamais assuré qu'après un calcule la valeur garde une précision parfaite.)

Semaine 12. (8 avril)

Sujet : CVS

Voir CVS.
page : http://www.labunix.uqam.ca/ tremblay/INF3135/
document : Gestion de la configuration des logiciels (SCM)

- intro : p5
- init : p6
- CVSROOT : p9
- import : p10
- checkout : p11
- update : p15
- commit : p17
- add : p20

Semaine 13. (15 avril)

Sujet : Tests Unitaires

Problème 13.1 (10 minutes)

Construire les tests unitaires pour la routine suivante :
`int iemElement( int * tableau, int taille, int index );`

La routine retourne la valeur à la position index du tableau. Elle s'assure que index est valide (entre 0 et taille), que le pointeur de tableau n'est pas NULL et que la taille est plus grande que 0, le tout avec des assertions.

réponse

```
int t1[1] = 5;
int t2[2] = 5, 10;
int t3[5] = 5, 10, 15, 20, 25;

//  taille = 1, index = 0.
assert( 5 == iemElement( t1, 1, 0 ) );

// taille = 2, index = 0.
assert( 5 == iemElement( t2, 2, 0 ) );

// taille = 2, index = 1.
assert( 10 == iemElement( t2, 2, 1 ) );

// taille = 5, index = 0.
assert( 5 == iemElement( t3, 5, 0 ) );

// taille = 5, index = 2.
assert( 15 == iemElement( t3, 5, 2 ) );

// taille = 5, index = 4.
assert( 25 == iemElement( t3, 5, 4 ) );
```

Ne pas tester le cas de pointeur NULL ou d'un tableau de taille 0. Ces cas ne sont pas permis donc ils ne sont pas testés.

Problème 13.2 (60 minutes)

Soit le .h suivant :

```
#ifndef BANQUE_H
#define BANQUE_H

/**
 * Type de données abstrait contenant la description d'un
 * client.
 */
typedef struct _client Client;

/**
 * Type de données abstrait contenant la description d'un
 * compte.
 */
typedef struct _compte Compte;

/**
 * Fonction de construction d'une structure Client.
 * @param a_nom (char *) une chaine contenant le nom du
 *  client.
 * @precondition a_nom != NULL
 * @param a_prenom (char *) une chaine contenant le prenom
 *  du client
 * @precondition a_prenom != NULL
 * @param a_noClient (int) un entier identifiant de facon
 *  unique le client.
 * @precondition a_noClient > 0
 * @return Un pointeur sur un Client.  Les chaines 'a_nom'
 *  et 'a_prenom' ont ete copiees dans la structure.
 */
Client * creerClient( char * a_nom, char * a_prenom, int a_noClient );

/**
 * Fonction de construction d'une structure Compte.
 * @param a_noIdentification (int) un numero d'identification
 *  unique pour le compte.
 * @param a_noClient (int) un numero identifiant le proprietaire
 *  du compte.
 * @precondition a_noClient > 0
 * @return Un pointeur sur un Compte.  L'actif du compte
 *  est place a zero (0) lors de la creation.
 */
Compte * creerCompte( int a_noIdentification, int a_noClient );

/**
 * Procedure modifiant l'actif d'un compte pour ajouter
 * de l'argent.
```

```
 * @param a_compte (Compte *) le compte a modifier.
 * @precondition a_compte != NULL
 * @param a_montant (long) montant (en cent) du depot.
 * @precondition a_montant > 0
 * @postcondition actif = actif + a_montant
 */
void depot( Compte * a_compte, long a_montant );

/**
 * Procedure modifiant l'actif d'un compte pour soustraire
 * de l'argent.
 * @param a_compte (Compte *) le compte a modifier.
 * @precondition a_compte != NULL
 * @param a_montant (long) montant (en cent) du retrait.
 * @precondition a_montant > 0
 * @exception e_transactionApprouve cette valeur est placee
 *  a un (1) s'il y avait assez d'argent pour le retrait.
 *  sinon elle est placee a zero (0).
 * @postcondition actif = actif - a_montant
 */
void retrait( Compte * a_compte, long a_montant, int * e_transactionApprouve );

/**
 * Fonction retournant l'actif d'un compte.
 * @param a_compte (Compte *) le compte a consulter.
 * @precondition a_compte != NULL
 * @return l'actif du compte.
 */
long actif( Compte * a_compte );

/**
 * Fonction afin de calculer l'actif total de tout les
 * comptes d'un client.
 * @param a_client (Client *) le client a traiter.
 * @precondition a_client != NULL
 * @param a_comptes (Compte *[]) un tableau de tout les
 *  les comptes du systeme.
 * @precondition a_comptes != NULL
 * @param nombreCompte (int) la taille du tableau a_comptes
 * @return l'avoir total des comptes appartenant au client.
 */
long calculerAvoir( Client * a_client, Compte * a_comptes[],
    int a_nombreCompte );

#endif
```

a) Construisez les tests unitaires pour les routines suivantes : depot, actif.

```
Compte * c1 = creerCompte( 1, 1 );
Compte * c2 = creerCompte( 2, 2 );

assert( 0 == actif( c1 ) );
depot( c1, 1 );
assert( 1 == actif( c1 ) );
depot( c2, 14649 );
assert( 14649 == actif( c2 ) );
depot( c1, 1 );
assert( 2 == actif( c1 ) );
depot( c2, 111 );
assert( 14760 == actif( c2 ) );
```

Remarque : puisque ces routines ne tiennent pas compte des débordements il est impossible de faire des tests sur la borne supérieure.

b) Les constructeurs nous donne des valeurs construites. Pour réussir à les tester il faut connaitre le contenue des structures :

```
/**
 * structure decrivant un client.
 * @field nom (char *) une chaine de caractere contenant le
 *  nom du client.
 * @invariant nom != NULL
 * @field prenom (char *) une chaine de caractere contenant
 *  le prenom du client.
 * @invariant prenom != NULL
 * @field noClient (int) un numero identifiant le client de
 *  facon unique.
 * @invariant noClient > 0
 */
struct _client
  char * nom;
  char * prenom;
  int noClient;
;

/**
 * structure decrivant un compte.
 * @field noIdentification (int) un numero identifiant de
 *  facon unique un compte.
 * @field noClient (int) un numero identifiant le proprietaire
 *  du compte.
 * @invariant noClient > 0
 * @field actif (long) le montant d'argent que contient le
 *  compte.  Ce montant est en cent.
```

```
 * @invariant actif >= 0
 */
struct _compte
  int noIdentification;
  int noClient;
  long actif;
;
```

Ensuite vous utilisez la routine et vérifiez le contenue des différents champs. Écrivez les tests unitaires pour creerCompte et creerClient.

```
Compte * co = creerCompte( 1, 5 );
assert( 1 == c->noIdentification );
assert( 5 == c->noClient );
assert( 0 == c->actif );

char * prenom = "Bruno";
char * nom = "Malenfant";
Client * cl = creerClient( nom, prenom, 1 );
assert( nom != cl->nom );
assert( 0 == strcmp( nom, c1->nom ) );
assert( prenom != cl->prenom );
assert( 0 == strcmp( prenom, cl->prenom ) );
assert( 1 == cl->noClient );
```

c) La routine retrait utilise une exception. Il faut lui passer une variable entrées/sorties pour le retour d'exception. Faites les testes pour retrait, n'oubliez pas de tester l'exception.

```
Compte * c1 = creerCompte( 1, 1 );
int exc = 0;

retrait( c1, 1, &exc );
assert( 0 == actif( c1 ) );
assert( exc );

exc = 0;
depot( c1, 1 );
retrait( c1, 1, &exc );
assert( 0 == actif( c1 ) );
assert( ! exc );

depot( c1, 1 );
retrait( c1, 2, &exc );
assert( 1 == actif( c1 ) );
assert( exc );
```

```
exc = 0;
depot( c1, 1000 );
retrait( c1, 1, &exc );
assert( 1000 == actif( c1 ) );
assert( ! exc );

retrait( c1, 500, &exc );
assert( 500 == actif( c1 ) );
assert( ! exc );

retrait( c1, 1000, &exc );
assert( 500 == actif( c1 ) );
assert( exc );

exc = 0;
retrait( c1, 500, &exc );
assert( 0 == actif( c1 ) );
assert( ! exc );
```

d) Discutez des méthodes pour tester calculerAvoir. (Remplir le tableau de compte)

Semaine 14. (22 avril)

Sujet : Revision examen

Problème 14.1 (15 minutes)

Nous voulons construire un logiciel en utilisant une technique de développement dirigée par les tests.

a) Quels sont les avantages à avoir une cohésion fonctionnelle pour nos routines dans un tel type de développement ?

réponse :

cohésion fonctionnelle -> une seule tache simple -> peu de test a faire -> tres rapide pour revenir au vert.

b) En quoi l'utilisation d'outils tel que Junit peuvent-ils nous assiter dans un développement dirigé par les tests ?

réponse :

automatisation et classification des tests. Permet de tester rapidement et souvent.

Problème 14.2 (30 minutes)

Soit les déclarations suivantes :

```
typedef enum { LIQUIDE, GRANULE, SOLIDE } TypeIgredient;
typedef union {
  float litre;
  unsigned short gramme;
  unsigned char unite;
} Mesure;
typedef struct {
  char * nom;
  Mesure quantite;
  TypeIgredient type;
} Ingredient;
typedef struct {
  Ingredient * disponible;
  int nbrIngredient;
} Cuisine;
typedef struct {
  char * nom;
  Ingredient * demande;
  int nbrIngredient;
} Recette;
```

La structure `Cuisine` représente les aliments disponibles dans une cuisine. Chaque élément du tableau '`disponible`' contient le nom de l'ingrédient (char *) et la quantité disponible (le nom sert à identifier l'ingrédient). Pour un ingrédient de type `LIQUIDE` nous mesurons la quantité en litre. Pour un ingrédient de type `GRANULE` nous mesurons la quantité en gramme. Pour un ingrédient de type `SOLIDE` nous mesurons la quantité en nombre d'unité disponible.

La structure `Recette` représente une recette avec son `nom`, la liste des ingrédients `demandé` pour la recette et le nombre d'ingrédient dans le tableau '`demande`'.

Nous voulons écrire une fonction ayant la signature suivante :

```
/**
 * Fontion permettant de verifier si nous pouvons faire une
 * recette.
 * @return Cette fonction retourne true si tout les ingredients
 * de la recette sont disponible en quantite suffisante dans
 * la cuisine.  Elle retourne faux sinon.
 * @param recette : la liste des ingredients d'une recette a faire.
 * @pre recette != NULL.
 * @param cuisine : la liste des ingredients disponible.
 * @pre cuisine != NULL.
 */
int ingredientDisponible( Recette * recette, Cuisine * cuisine );
```

a) Écrivez le code pour cette fonction. Votre code doit utiliser les techniques de programmation défensive : verifier les pré-conditions, vérifier les accès pouvant causer des problèmes (pointeur, tableau).

réponse :

```
int ingredientDisponible( Recette * recette, Cuisine * cuisine ) {
  assert( recette != NULL );
  assert( demande != NULL );
  int resultat = 1;
  int i = 0;
  int j = 0;
  for( i = 0; i < recette->nbrIngredient; i++ ) {
    int trouve = 0;
    for( j = 0; j < cuisine->nbrIngredient; j ++ ) {
      assert( 0 <= i && i < recette->nbrIngredient );
      assert( 0 <= j && j < cuisine->nbrIngredient );
      assert( recette->demande[i].nom != NULL );
      assert( cuisine->disponible[j].nom != NULL );
      if( 0 == strcmp( recette->demande[i].nom, cuisine->disponible[j].nom ) ) {
        trouve = 1;
        switch( recette->demande[i].type ) {
          case LIQUIDE :
```

```
            if( recette->demande[i].quantite.litre >
                cuisine->disponible[j].quantite.litre ) {
              resultat = 0;
            }
            break;
          case GRANULE :
            if( recette->demande[i].quantite.gramme >
                cuisine->disponible[j].quantite.gramme ) {
              resultat = 0;
            }
            break;
          case SOLIDE :
            if( recette->demande[i].quantite.unite >
                cuisine->disponible[j].quantite.unite ) {
              resultat = 0;
            }
            break;
        }
      }
    }
    if( !trouve ) {
      resultat = 0;
    }
  }
  return resultat;
}
```

b) Décrivez les cas de test unitaire (boîte noire) complet pour tester cette fonction. Cette description doit indiquer chaque test avec les arguments utilisés et les résultats attendus.

Problème 14.3 (30 minutes)

Soit le code suivant :

```
 1 Palette * lirePalette( char * a_nomFichier ) {
 2   Palette * resultat = creerPalette();
 3   FILE * fichierCouleur = fopen( a_nomFichier, "r" );
 4   if( NULL == fichierCouleur ) {
 5     fprintf( stderr, "erreur lors de louverture du fichier.\n" );
 6   }
 7   do {
 8     double rouge, vert, bleu;
 9     int n = fscanf( fichierCouleur, "%lf %lf %lf", &rouge, &vert, &bleu );
10     if( 3 == n ) {
11       Couleur * couleur = ( Couleur * ) malloc( sizeof( Couleur ) );
```

```
12        if( NULL == couleur ) {
13          fprintf( stderr, "erreur d'allocation.\n" );
14        }
15        couleur->rouge = rouge;
16        couleur->vert  = vert;
17        couleur->bleu  = bleu;
18        couleur->alpha = 1;
19        ajouterCouleur( resultat, couleur );
20      } else if ( n != -1 ) {
21        fprintf( stderr, "erreur de lecture dans le fichier de couleur.\n" );
22      }
23    } while( ! feof( fichierCouleur ) );
24    return resultat;
25  }
```

a) Tracez le graphe représentant ce code.

b) Calculez la complexité cyclomatique de ce graphe.

Réponse : 6

c) Donnez les chemins indépendants permettant de faire la couverture du graphe.

Réponse :

2-4-7-10-12-15-23-24
2-4-5-7-10-12-15-23-24
2-4-7-10-12-13-15-23-24
2-4-7-10-20-22-23-24
2-4-7-10-20-21-22-23-24
2-4-7-10-12-15-23-7-10-12-15-23-24

Problème 14.4 (10 minutes)

Quel sont les avantages d'un système d'exception par rapport à une technique où un message d'erreur est affiché suivit d'un arrêt du programme.

Réponse :

Les exceptions permettent la reprise de l'exécution. Elles permettent aussi à l'utilisateur de corriger l'information qu'il entre.