# TD 5 BIS

## Traitement d'erreurs

Le mécanisme habituel pour la gestion d'erreurs est basé sur les exceptions. Dans cette solution, on sépare la détection des erreurs et les traitements des erreurs différentes. L'exemple en méta-langage, proche du Java illustre un cas. Dans cet exemple, une fonction
`int calculs(int, int, int)` reçoit trois entiers et elle doit les manipuler.

Les valeurs doivent être strictement supérieures à 0. S'il y a une valeurs qui est négative ou 0, il faut générer une exception. L'inégalité triangulaire doit être satisfaite (la somme de deux valeurs arbitraires doit être supérieure à la troisième valeur), sinon, une exception doit être lancée. Puis, la somme des trois valeurs doit être supérieure au produit de deux premières. Une troisième exception indique quand cette troisième condition n'est pas justifiée. S'il n'y a pas de problème à signaler, la fonction doit retourner une somme pondérée des trois valeurs (voir dans le code).

```
int calculs(int i, int j, int k) {
afficher(Calculs pour i,j,k);
if (i==0 || j==0 || k ==0)
      throw exception1;
if (i+j<k || j+k<i || k+i <j)
      throw exception2;
if (i+j+k > i*j)
      throw exception3;
return (i + 2*j + 3*k)/5;
}

static int main() {
int val1, val2, val3;
int res;
while(1) {
  afficher(Donnez trois valeurs a examiner );
  saisir(val1, val2, val3);
  afficher(Les valeurs sont : val1, val2, val3);
  try {
      res = calculs(val1, val2, val3);
      }
  catch (exception1){
      afficher(Valeur zero parmi : val1, val2, val3);
      }
  catch (exception2){
      afficher(Probleme triangulaire :  val1, val2, val3);
      }
  catch (exception3){
      printf("Probleme ecart :   val1, val2, val3);
      }
 } //while
}
```

**Q1** Proposez une solution en C avec `setjmp _ longjmp.` Les cas d'erreurs peuvent être codés par des entiers 1,2,3...

## Deuxième version, introduire les mots-clés

La structure `try… catch …` peut être introduite par des macros `#define`.

Regardons l'exemple suivant.

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

#define TRY do{ jmp_buf ex_buf__; if( !setjmp(ex_buf__) ){
#define CATCH } else {
#define ETRY } }while(0)
#define THROW longjmp(ex_buf__, 1)

int
main(int argc, char** argv) {
   TRY
   {
      printf("In Try Statement\n");
      THROW;
      printf("I do not appear\n");
   }
   CATCH
   {
      printf("Got Exception!\n");
   }
   ETRY;

   return 0;
}
```

Ce qui donne les substitutions suivantes (vérifiées avec `gcc -E` ...)

```
main(int argc, char** argv){
    do{ jmp_buf ex_buf__; if( !setjmp(ex_buf__) ){
    {
       printf("In Try Statement\n");
       longjmp(ex_buf__, 1);
       printf("I do not appear\n");
    }
    } else {
    {
       printf("Got Exception!\n");
    }
    } }while(0);

    return 0;
}
```

Pour ajouter des exceptions, on peut faire des définitions :

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

#define TRY do{ jmp_buf ex_buf__; switch( setjmp(ex_buf__) ){ case 0:
```

```
#define CATCH(x) break; case x:
#define ETRY } }while(0)
#define THROW(x) longjmp(ex_buf__, x)

#define FOO_EXCEPTION (1)
#define BAR_EXCEPTION (2)
#define BAZ_EXCEPTION (3)

int main(int argc, char** argv){
   TRY
   {
      printf("In Try Statement\n");
      THROW( BAR_EXCEPTION );
      printf("I do not appear\n");
   }
   CATCH( FOO_EXCEPTION )
   {
      printf("Got Foo!\n");
   }
   CATCH( BAR_EXCEPTION )
   {
      printf("Got Bar!\n");
   }
   CATCH( BAZ_EXCEPTION )
   {
      printf("Got Baz!\n");
   }
   ETRY;

   return 0;
}
```

Vous pouvez étudier la solution proposée sur https://web.archive.org/web/20091104065428/http://www.di.unipi.it/~nids/docs/longjump_try_trow_catch.html

**Q2)** Transformez votre programme en utilisant les mots-clés (TRY, CATCH, ETRY, THROW) proposées.