# PROJET MALG

## Groupe 8


**Préparé par**

**CRAPANZANO Claire**
**FERRÉ Nicolas**
**MOLLARD Romaric**
**RUCHOT Guillaume**


**Mai 2017**

# Sommaire

# 1 Introduction

# 2 Programme premier

## 2.1 Description de l'algorithme

Ce programme demande à l'utilisateur un nombre n quelconque (entier), et le programme affiche alors les n premiers nombres premiers en partant de zero. Si n est négatif ou nul, le programme ne doit rien afficher.

## 2.2 Traduction PlusCal

```
//TODO Copy code
```

## 2.3 Preuve TLA+

...

## 2.4 Preuve FramaC

Nous avons prouvé deux éléments grace à l'outil FramaC :
1. Le programme ne retourne que des nombres premiers
2. Le programme retourne le nombre de résultats demandé
Pour celà nous avons rajouté quelques données dans l'algorithme afin par exemple de comptabiliser le nombre de printf, qui est matérialisé par la variable 'nbprintf' initialisée à 0.

```
int main(){
  int n, i = 3, count, c;

  /*@ assert i==3; */

  printf("Enter the number of prime numbers required\n");
  scanf("%d",&n);

  //On protÃ¨ge n des valeurs nÃ©gatives pour framac
  if(n<0){
     n=0;
  }

  //On ajoute une variable pour compter le nombre d'affichages total
  int nbprintf = 0;

  /*@ assert n>=0; */
  /*@ assert nbprintf==0; */

  if (n>=1){
    printf("First %d prime numbers are :\n", n);
    printf("2\n"); //Preuve 1. Les rÃ©sultats sont tous des nombes premiers
   //On incrÃ©mente nbprintf
   nbprintf++;
```

```
}

//On s'assure qu'on a affichÃ© 2 pour commencer
/*@ assert n>=1 ==> nbprintf==1; */ //Preuve 2. On a n==nombre de printf
/*@ assert n<1 ==> nbprintf==0; */ //Preuve 2. On a n==nombre de printf


//Il faut prouver que la boucle termine

/*@ loop invariant 2 <= count;
    loop invariant i>count;
    loop invariant nbprintf<=n;
    loop invariant n>=1 ==> count<=n+1;
    loop invariant n>=1 ==> nbprintf==count-1;
    loop invariant n<1 ==> nbprintf==0;
*/
for (count = 2; count <= n; ){

  /*@ assert count<=n;*/

  /*@ loop invariant 2 <= c <= i;
      loop invariant 2 <= count;
      loop invariant i>count;
      loop invariant nbprintf==count-1;
      loop invariant count<=n;
      loop invariant nbprintf<=n;
      loop invariant c>=3 ==> i%(c-1)!=0;
      loop invariant \forall int w; 2<=w<c ==> i%w!=0;
  */
  for (c = 2; c<= i-1;c++){
    if (i%c == 0) {
      /*@ assert i%c==0; */
      break;
    }
    /*@ assert i%c!=0;*/
  }
  /*@ assert i%c==0 || i==c; */
  /*@ assert \forall int w; 2<=w<c ==> i%w!=0;*/
  /*@ assert count<=n;*/
  /*@ assert nbprintf==count-1;*/

  if (c == i){
    /*@ assert i==c; */

    //On prouve que i est premier
    /*@ assert \forall int w; 2<=w<i ==> i%w!=0;*/ //Preuve 1. Les rÃ©sultats sont tous
        des nombes premiers
    printf("%d\n",i);

   //On incrÃ©mente nbprintf
   nbprintf++;
    /*@ assert nbprintf==count;*/

    count++;
    /*@ assert count<=n+1;*/
    /*@ assert nbprintf==count-1;*/
    /*@ assert nbprintf<=n;*/
```

```
    }

    i++;
    /*@assert count<=n+1;*/
  }
  /*@ assert count>=n+1; */
  /*@ assert n>=1 ==> count==n+1; */

  //On prouve qu'il y a le bon nombre de rÃ©sultats
  /*@ assert n>=1 ==> nbprintf==n; */
  /*@ assert n<1 ==> nbprintf==0; */
  /*@ assert n>=0; */


  /*@ assert nbprintf==n; */ //Preuve 2. On a n==nombre de printf

  return 0;
}
```

Nous avons donc bien validé /*@ assert nbprintf==n; */ et /*@ assert $\forall intw; 2 <= w < i ==> i$

# 3 Tri 1 : Sélection

## 3.1 Description de l'algorithme

Ce programme propose à l'utilisateur de créer une liste de n éléments, puis lui demande de rentrer les valeurs de cette liste une par une. Le programme affiche ensuite la liste triée par la méthode du tri sélection.

## 3.2 Traduction PlusCal

## 3.3 Preuve TLA+

## 3.4 Preuve FramaC

# 4 Autre algorithme

## 4.1 Description de l'algorithme

Ce programme mistère calcule les éléments de la formule de récurrence

$$r(i) = 2 * r(i-1) + 2 * r(i-2), r(0) = 1, r(1) = 1$$

Après résolution, ceci correspond à la formule suivante :

$$r(i) = 1/2 * ((1 - \sqrt{(3)})^i + (1 + \sqrt{(3)})^i)$$

## 4.2 Traduction PlusCal

```
EXTENDS TLC,Integers, Reals

(*
--algorithm Algo3 {
variables a,b,c,i,r;
{
     a:=1;
     b:=1;
     i:=2;
     if(x==0){
        r:=1;
     }else{
        if(x==1){
            r:=1;
        }else{
            while(i-1 < x)
            {
                i:=i+1;
                c:=a;
                a:=b;
                b:=2*c+2*b;
            };
            r:=b;
        }
    }
}

}

*)

\* BEGIN TRANSLATION
CONSTANT x
VARIABLES a, b, c, i, r, pc
```

```
vars == << a, b, c, i, r, pc >>

Init == (* Global variables *)
       /\ a = 1
       /\ b = 1
       /\ c = 0
       /\ r = 0
       /\ i = 2
       /\ pc = "Lbl_1"

Lbl_1 == /\ pc = "Lbl_1"
        /\ a' = 1
        /\ b' = 1
        /\ i' = 2
        /\ IF x=0
             THEN /\ r' = 1
                  /\ pc' = "Done"
             ELSE /\ IF x=1
                       THEN /\ r' = 1
                            /\ pc' = "Done"
                       ELSE /\ pc' = "Lbl_2"
                            /\ r' = r
        /\ c' = c

Lbl_2 == /\ pc = "Lbl_2"
        /\ IF i-1 < x
             THEN /\ i' = i+1
                  /\ c' = a
                  /\ a' = b
                  /\ b' = 2*c'+2*b
                  /\ pc' = "Lbl_2"
                  /\ r' = r
             ELSE /\ r' = b
                  /\ pc' = "Done"
                  /\ UNCHANGED << a, b, c, i >>

Next == Lbl_1 \/ Lbl_2
          \/ (* Disjunct to prevent deadlock on termination *)
             (pc = "Done" /\ UNCHANGED vars)

Spec == Init /\ [][Next]_vars

Termination == <>(pc = "Done")

\* END TRANSLATION
```

## 4.3 Preuve TLA+

La preuve TLA+ consiste en une preuve partielle du programme.
La traduction automatique produit trois états :
Init : qui représente le programme avant le premier if
Lbl_1 : qui représente les tests et conditions if / else
Lbl_2 : qui représente le while et donc le calcul de la formule décrite ci-dessus.
La variable pc nous permet de connaitre l'état actuel du programme.

```
TestPreLoop == i>2 \/ b = 1 \* On a pas commencÃⓒ la boucle donc b=1
TestPostLoop == i\leq2 \/ b = 2*c+2*a \* On a commencÃⓒ la boucle donc b= 2*c + 2*a
TestEnd == pc#"Done" \/ ( (x\leq1 /\ r=1) \/ (x>1 /\ r=2*c+2*a) ) \*TerminÃⓒ donc soit
    c'est pas passÃⓒ dans la boucle et c'est 1 soit c'est la formule rÃⓒcursive
```

Ceci nous permet de vérifier la formule de récurrence (en admettant c==b[i-1] et a==b[i-2]) et que les résultats pour i=0 et i=1 sont bien 1, soit les valeurs de base de la formule récurrente.

## 4.4 Preuve FramaC

# 5 Conclusion

## 5.1 Difficultés rencontrées

## 5.2 Temps et méthodes de travail