**Compte-rendu projet LO21**

I. Objectif du projet :

Le projet a pour objectif la définition et la manipulation de types abstraits de données « Individu » et « Population ». Un individu est représenté par une suite de bits et une population est une suite d’Individus. Il s’agit d’une version simplifiée d’algorithmes génétiques.

II. Choix de conception et d’implémentation des structures de données et démarche :

Tout d’abord, notre projet contient 4 nouveaux types abstraits qui sont les suivants :  
 - Bit  
 - individu  
 - population  
 - param\_qualite

Les types « Bit », « individu » et « population » sont des listes chaînées, dont la structure est imposée par le projet. « Bit » désigne un charactère qui peut prendre la valeur ‘0’ ou ‘1’, « individu » est une liste de bit, qui pointe vers une structure « bit », et « population » est une liste d’individu, qui pointe vers une structure « personne ». Ces types ont donc la structure suivante :

unsigned char Bit

bit :  
 Bit valeur  
 individu suivant  
bit\* individu

personne :  
 individu valeur  
 population suivant  
personne\* population

Le type param\_qualite est un type abstrait que nous avons introduit. Il permet de regrouper la valeur de tous les paramètres dont dépend le calcul de la qualité d’un individu. Ce sont des paramètres qui sont demandés à l’utilisateur dès le lancement du programme. Nous l’avons introduit afin de réduire le nombre d’arguments qui doivent être passés aux différents sous-programme ayant besoin de ces paramètres. La structure de ce type est le suivante :

param\_qualite :  
 char type\_fonction  
 double A  
 double B

type\_fonction est un caractère qui permet de décider la fonction mathématique utilisé pour calculer la qualité d’un individu. Il peut prendre les valeurs suivantes :  
 - ‘s’ : fonction carré  
 - ‘l’ : fonction logarithme népérien  
 - ‘c’ : fonction cosinus  
A et B sont deux réels permettant de paramétrer le résultat.

Afin d’organiser le projet au mieux, nous avons réunis les structures de données dans un dossier unique, appelé « structures », contenant les quatre nouveaux types dans trois fichiers distincts (le type « Bit » est dans le même fichier que le type « individu »). Ces fichiers sont des header, ce qui nous permet de les inclure facilement dans des fonctions de notre code. Ceci a aussi été fait un souci de réutilisation de ces types dans des projets ultérieurs.

III. Algorithmes des sous-programmes :

Nous avons écrit les algorithmes de sous-programmes permettant de réaliser toutes les opérations nécessaires afin de simuler la fluctuation de la population. Afin de distinguer les variables et les types, nous mettrons les types en majuscule et les variables en minuscule. Ces fonctions sont les suivantes :

Fonction ENTIER longueur\_indiv

Compte le nombre de bits que contient un individu et renvoie le résultat.

Lexique :

- suc() : élément suivant de la liste

- est\_vide() : test si la liste est vide

Donnée :

- INDIVIDU personne

Début :

ENTIER nombre = 0

Tant que non est\_vide(personne) faire

nombre = nombre + 1

personne = suc(personne)

Fin tant que

longueur\_indiv <- nombre

Fin

Fonction INDIVIDU ajouter\_indiv\_q

Ajoute un bit à un individu en fin de liste et renvoie le nouvel individu.

Lexique :

- suc() : élément suivant de la liste

- longueur\_indiv() : longueur de la liste en paramètre

- valeur() : valeur du bit pointé dans une liste de bits (individu)  
 - creer\_indiv() : alloue de l’espace dans la mémoire pour stocker un individu

Données :

- INDIVIDU personne

- BIT valeur

Début :

INDIVIDU nouv\_bit = creer\_indiv()

valeur(nouv\_bit) = valeur

suc(nouv\_bit) = indéfini

Si longueur\_indiv(personne) = 0 alors

personne = nouv\_bit

Sinon

INDIVIDU temp = personne  
 ENTIER i

Pour i de longueur\_indiv(personne) à 2 par -1 faire

temp = suc(temp)

Fin Pour

suc(temp) = nouv\_bit

Fin Si

ajouter\_indiv\_q <- personne

Fin