**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 :

1. Type abstrait « Individu »

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

Fonction INDIVIDU croiser\_indiv

Ajoute un nouvel individu en fonction de deux autres (naissance d'un enfant)

Lexique :

- suc() : élément suivant de la liste

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

- valeur() : valeur d'un individu

- ajouter\_indiv\_q() : ajoute un nouvel individu

Données :

- INDIVIDU personne1

- INDIVIDU personne2

- DOUBLE pCroise

Début :

INDIVIDU nouvel\_indiv = NULL

Pour i de longueur\_indiv(personne1) à 1 de -1 en -1 faire

Si (rand()/RAND\_MAX) <= pCroise alors

nouvel\_indiv = ajouter\_indiv\_q(nouvel\_indiv, valeur(personne1))

Sinon

nouvel\_indiv = ajouter\_indiv\_q(nouvel\_indiv, valeur(personne2))

Fin Si

personne1 = suc(personne1)

personne2 = suc(personne2)

Fin pour

croiser\_indiv <- nouvel\_indiv

Fin

Fonction ENTIER decoder\_indiv

Affiche la valeur décimale de la liste de bit d'un individu

Lexique :

- suc() : élément suivant de la liste

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

- valeur() : valeur d'un individu

Donnée :

- INDIVIDU personne

Début :

ENTIER somme = 0

Pour i de (longueur\_indiv(personne) - 1) à 0 de -1 en -1 faire

somme = somme + (valeur(personne) - '0') \* puissance(2, i)

personne = suc(personne)

Fin Pour

decoder\_indiv <- somme

Fin

Fonction INDIVIDU initialiser\_indiv\_iter

Initialise un individu composé de bits aléatoires de manière itérative

Lexique :

- ajouter\_indiv\_q() : ajoute un nouvel individu

Donnée :

- ENTIER longIndiv : correspond au nombre de bit auquel on va initialiser la personne

Début :

INDIVIDU personne = NULL

Pour i de 1 à longIndiv de 1 en 1 faire

personne = ajouter\_indiv\_q(personne, rand()%2 + '0')

Fin Pour

initialiser\_indiv\_iter <- personne

Fin

Fonction INDIVIDU initialiser\_indiv\_recur

Initialise un individu composé de bits aléatoires de manière récursive

Lexique :

- ajouter\_indiv\_q() : ajoute un nouvel individu

Données :

- Entier longIndiv : correspond au nombre de bit auquel on va initialiser la personne

Début :

Si longIndiv = 1 alors

initialiser\_indiv\_recur <- ajouter\_indiv\_q(NULL, rand()%2 + '0')

sinon

initialiser\_indiv\_recur <- ajouter\_indiv\_q(initialiser\_indiv\_recur(longIndiv - 1), rand()%2 + '0')

Fin SI

Fin

Fonction DOUBLE qualite\_indiv

Calcul la qualité d'un individu

Lexique :

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

- decoder\_indiv() : donne la valeur decimal de la liste de bit d'un individu

Données :

- INDIVIDU personne

- param\_qualite \*parametres

Début :

DOUBLE qualite

DOUBLE X = (decoder\_indiv(personne) / puissance(2, longueur\_indiv(personne))) \* (B(parametres) - A(parametres)) + A(parametres)

Selon (type\_fonction(parametres)) alors

cas type\_fonction(parametres) = 's' : qualite = -pow(X, 2)

cas type\_fonction(parametres) = 'l' : qualite = -log(X)

cas type\_fonction(parametres) = 'c' : qualite = -cos(X)

Fin Selon

qualite\_indiv <- qualite

Fin

1. Type abstrait « Population »

Fonction POPULATION ajouter\_pop\_q

Ajoute un individu à une population

Lexique :

- suc() : élément suivant de la liste

- longueur\_pop() : longueur de la population

- valeur() : individu de la popualtion

Données :

- INDIVIDU valeur

- POPULATION personnes

Début :

valeur(nouv\_personne) = valeur

suc(nouv\_personne) = NULL

Si (longueur\_pop(personnes) = 0) alors

personnes = nouv\_personne

Sinon

population temp = personnes

Pour i de longueur\_pop(personnes) à 2 de -1 en -1 faire

temp = suc(temp)

Fin Pour

suc(temp) = nouv\_personne

Fin Si

ajouter\_pop\_q <- personnes

Fin

Fonction POPULATION initialiser\_pop\_iter

Initialisation de manière aléatoire d’une liste d’individu de façon itérative

Lexique :

- initialiser\_indiv\_iter() : initialisation d'un individu

- ajouter\_pop\_q() : ajoute un individu à une population en queue de liste

Données :

- ENTIER TaillePop

- ENTIER longIndiv

Début :

POPULATION personnes = NULL

Pour i de 1 à TaillePop de 1 en 1 faire

personnes = ajouter\_pop\_q(personnes, initialiser\_indiv\_iter(longIndiv))

Fin Pour

initialiser\_pop\_iter <- personnes

Fin

Fonction POPULATION initialiser\_pop\_recur

Initialisation de manière aléatoire d’une liste d’individu de façon récursive

Lexique :

- initialiser\_indiv\_recur() : initialisation d'un individu

- ajouter\_pop\_q() : ajoute un individu à une population en queue de liste

Données :

- ENTIER TaillePop

- ENTIER longIndiv

Début :

Si (TaillePop = 1) alors

initialiser\_pop\_recur <- ajouter\_pop\_q(NULL, initialiser\_indiv\_recur(longIndiv))

Sinon

initialiser\_pop\_recur <- ajouter\_pop\_q(initialiser\_pop\_recur(TaillePop - 1, longIndiv), initialiser\_indiv\_recur(longIndiv))

Fin Si

Fin

Fonction ENTIER longueur\_pop

Compte le nombre d’individu que contient une population et renvoie le résultat.

Lexique :

- est\_non\_nul() : vérification pour savoir si la liste n'est pas nulle

- suc() : élément suivant de la liste

Données :

- POPULATION personnes

Début :

ENTIER nombre = 0

Tant que est\_non\_nul(personnes) faire

nombre = nombre +1

personnes = suc(personnes)

Fin Tant que

longueur\_pop <- nombre

Fin

Fonction POPULATION trier\_pop

La fonction trie les individus en fonction de leur qualité au moyen de Quicksort (tri rapide composé de deux étapes : Diviser et Régner)

Lexique :

- longueur\_pop() : longueur de la population

- suc() : élément suivant de la liste

- ajouter\_pop\_q() : ajoute un individu à une population en queue de liste

- valeur() : individu de la popualtion

- qualite\_indiv() : qualité d'un individu

- enchainer\_pop() :

Données :

- POPULATION personnes

- param\_qualite \*parametres

Début :

Si (longueur\_pop(personnes) = 0) alors

trier\_pop = NULL

Sinon

DOUBLE pivot <- qualite\_indiv(valeur(personnes), parametres)

POPULATION habitant = suc(personnes)

POPULATION inf\_pivot = NULL

POPULATION sup\_pivot = NULL

Pour i de longueur\_pop(personnes) à 2 de -1 en -1 faire

Si (qualite\_indiv(valeur(habitant), parametres) < pivot)

inf\_pivot = ajouter\_pop\_q(inf\_pivot, valeur(habitant))

Sinon

sup\_pivot = ajouter\_pop\_q(sup\_pivot, valeur(habitant))

Fin Si

habitant = suc(habitant)

Fin Pour

inf\_pivot = trier\_pop(inf\_pivot, parametres)

sup\_pivot = trier\_pop(sup\_pivot, parametres)

trier\_pop <- enchainer\_pop(enchainer\_pop(sup\_pivot, ajouter\_pop\_q(NULL, valeur(personnes))), inf\_pivot)

Fin Si

Fin

Fonction POPULATION selectionner\_pop

Selectionne les meilleurs individus d’une population en tronquant la liste et en la complétant par recopie des tselect premiers éléments

Lexique :

- longueur\_pop() : longueur de la population

- suc() : élément suivant de la liste

- ajouter\_pop\_q() : ajoute un individu à une population en queue de liste

- valeur() : individu de la popualtion

- habitant\_num() : numéro d'un habitant d'une population

Données :

- POPULATION personnes

- ENTIER tselect

Début :

POPULATION nouv\_pop = NULL

Si (tselect = 0) faire

selectionner\_pop <- nouv\_pop

Sinon

Entier nb\_habitant = longueur\_pop(personnes)

Pour i de 0 à nb\_habitant - 1 de 1 en 1 faire

Si (i/tselect > 0) alors

nouv\_pop = valeur(ajouter\_pop\_q(nouv\_pop, habitant\_num(nouv\_pop, i % tselect + 1)))

Sinon

nouv\_pop = ajouter\_pop\_q(nouv\_pop, valeur(personnes))

Fin Si

personnes = suc(personnes)

Fin Pour

selectionner\_pop <- nouv\_pop

Fin Si

Fin

Fonction POPULATION queue\_pop

Place un individu en fin d’une liste population

Lexique :

- longueur\_pop() : longueur de la population

- habitant\_num() : l'habitant d'une population situé à un certain rang

Données :

- POPULATION personnes

Début :

queue\_pop <- habitant\_num(personnes, longueur\_pop(personnes))

Fin

Fonction POPULATION habitant\_num

Lexique :

- suc() : élément suivant de la liste

Données :

- ENTIER n

- POPULATION personnes

Début :

Pour i de 1 à n - 1 de 1 en 1 faire

personnes = suc(personnes)

Fin Pour

habitant\_num <- personnes

Fin

Fonction POPULATION enchainer\_pop

Lexique :

- suc() : élément suivant de la liste

- longueur\_pop() : longueur de la population

- queue\_pop() : dernier élément de la liste

Données :

- POPULATION p1

- POPULATION p2

Début :

Si (longueur\_pop(p1) = 0) alors

enchainer\_pop <- p2

Sinon

suc(queue\_pop(p1)) = p2

enchainer\_pop <- p1

Fin Si

Fin

Fonction POPULATION enlever\_hab

Enlève un habitant de la population

Lexique :

- suc() : élément suivant de la liste

- habitant\_num() : l'habitant d'une population situé à un certain rang

Données :

- ENTIER num\_habitant

- POPULATION personnes

Début :

Si (num\_habitant = 1) alors

personnes = suc(personnes)

Sinon

suc(habitant\_num(personnes, num\_habitant - 1)) = habitant\_num(personnes, num\_habitant + 1)

Fin Si

enlever\_hab <- personnes

Fin

Fonction population croiser\_pop

Croisement de deux populations pour en donner une troisième

Lexique :

- enlever\_hab() : enlève un individu d'une population

- ajouter\_pop\_q() : ajoute un individu à une population en queue de liste

- longueur\_pop() : longueur de la population

- valeur() : individu de la popualtion

- croiser\_indiv() : naissance d'un enfant avec deux individu

- habitant\_num() : numéro d'un habitant d'une population

Données :

- DOUBLE pCroise

- ENTIER nbCroise

- POPULATION personnes

Début :

POPULATION nouvelle\_pop = NULL

Pour i de longueur\_pop(personnes)/2 à 1 de -1 en -1 faire

INDIVIDU personne1

INDIVIDU personne2

Entier num\_hab1 = 1 + rand() % (longueur\_pop(personnes))

personne1 = valeur(habitant\_num(personnes, num\_hab1))

personnes = enlever\_hab(personnes, num\_hab1)

Entier num\_hab2 = 1 + rand() % (longueur\_pop(personnes))

personne2 = valeur(habitant\_num(personnes, num\_hab2))

personnes = enlever\_hab(personnes, num\_hab2)

Pour j de 1 à nbCroise de 1 en 1 faire

nouvelle\_pop = ajouter\_pop\_q(nouvelle\_pop, croiser\_indiv(personne1, personne2, pCroise))

Fin Pour

Fin Pour

croiser\_pop <- nouvelle\_pop

Fin