Exercie n°1

Fichier SuiteAriGeo.ta.h : Langage

```
/**
* TYPE ABSTRAIT TypeSuite
* Concept : Ce type permet d'énumérer différents types de suites
**
Opérations de base
* Constructeur TypeSuite : suiteGeo() : TypeSuite
* Constructeur TypeSuite : suiteAri() : TypeSuite
* Fin des opérations de base
* FIN DE TYPE ABSTRAIT
*/*/
enum ETypeSuite{suiteGeo,suiteAri};
typedef enum ETypeSuite TypeSuite;
/**
* TYPE ABSTRAIT SuiteAriGeo
* Concept : Ce type permet de modéliser une suite arithmétique ou géométrique
*/*/
struct SuiteAriGeo
{
double u0;
double coeff;
TypeSuite ts;
};
typedef struct SSuiteAriGeo SuiteAriGeo;
/**
* Opérations de base
*/*/
Constructeur SuiteAriGeo creerSuiteAriGeo(double uO, dobule coeff, TypeSuite
ts);
Observateur double recUO(SuiteAriGeo sag);
Observateur double recCoeff(SuiteAriGeo sag);
Observateur TypeSuite recTypeSuite(SuiteAriGeo sag);
/**
* Axiomes
* recU0(creerSuiteAriGeo(u0,coeff,ts)) = u0
* recCoeff(creerSuiteAriGeo(u0,coeff,ts)) = coeff
•  recTypeSuite(creerSuiteAriGeo(u0,coeff,ts)) = ts
•
* Fin des axiomes
* FIN DE TYPE ABSTRAIT
**/*/
```

Fichier SuiteAriGeo.ta.c : Langage

```
/* TYPE ABSTRAIT SuiteAriGeo */*/
#include "SuiteAriGeo.ta.h"
SuiteAriGeo creerSuiteAriGeo(double u0, double coeff, TypeSuite ts)
{
SuiteAriGeo sag;
sag.u0 = u0;
sag.coeff = coeff;
sag.ts = ts;
return sag;
}
/* Opération de base */*/
Observateur double recUO( SuiteAriGeo sag)
{
return sag.u0;
}
/* Opération de base */*/
Observateur double recCoeff( SuiteAriGeo sag)
```

```
{
return sag.coeff;
}
/* Opération de base */*/
Observateur TypeSuite recTypeSuite( SuiteAriGeo sag)
{
return sag.ts;
}
/**
* Axiomes
* recU0(creerSuiteAriGeo(u0,coeff,ts)) == u0
* recCoeff(creerSuiteAriGeo(u0,coeff,ts)) == coeff
* recTypeSuite(creerSuiteAriGeo(u0,coeff,ts)) == ts
*/*/
```

Fichier suiteAriGeoAffNTermes.ta.c : Langage

```
#include <stdio.h>
#include <Algo.h>
/* Début de fonction : opération d'extension */*/
Transformateur FILE * afficherTermesSuiteAriGeo( FILE *
sortie,SuiteAriGeo sag,int n)
{
double u;
int i;
/* Début du code */*/
fprintf(sortie,"%s" ,"Caractéristiques de la SuiteAriGeo :" );
fprintf(sortie,"\n" );
if( estEgal(recType(sag),suiteAri) )
{
fprintf(sortie,"%s" ,"Suite arithmetique" );
fprintf(sortie,"\n" );
}
else
{
fprintf(sortie,"%s" ,"Suite géometrique" );
fprintf(sortie,"\n" );
}
fprintf(sortie,"%s" ,"Premier terme == " );
fprintf(sortie,"%f" ,recU0(sag));
fprintf(sortie,"\n" );
fprintf(sortie,"%s" ,"Raison == " );
fprintf(sortie,"%f" ,recCoeff(sag));
fprintf(sortie,"\n" );
if( estEgal(recTypeSuite(sag),suiteAri()) )
{
u = recU0(sag);
i = 1;
while( i <= n )
{
u = u + recCoeff(sag);
fprintf(sortie,"%s" ,"u(" );
fprintf(sortie,"%d" ,i);
fprintf(sortie,"%s" ,") == " );
fprintf(sortie,"%s" ,u);
fprintf(sortie,"\n" );
i = i+1;
}
}
else
{
u = recU0(sag);
i = 1;
while( i <= n )
{
```

```
u = u * recCoeff(sag);
fprintf(sortie,"%s" ,"u(" );
fprintf(sortie,"%d" ,i);
fprintf(sortie,"%s" ,") == " );
fprintf(sortie,"%f" ,u);
fprintf(sortie,"\n" );
i = i+1;
}
}
}
/* Fin de fonction : : opération d'extension*/*/
```

Exercice n°2

-Q1

```
Ensemble creerEVide()
{
   Ensemble E;
   E = (Ensemble) malloc (sizeof(struct SEnsemble));
   E->pelements = (int*) NULL;
   E->nbElements = 0;
   return E;
}
```

-Q2

```
Ensemble ajouterElement (Ensemble E, int e)
{
   int existant;
   int* tableau;

  existant = 0;

   // Préexistance de l'élément e dans E
   for (int i = 0; i < E->nbElements; i++)
   {
     if (E->pelements[i] == e)
     {
       existant = 1;
     }
   }

   if (!existant)
   {
     E->nbElements ++;
     tableau = (int*) malloc (sizeof(int*E->nbElements));
     for (int i=0; i < E->nbElements-1; i++)
     {
       tableau[i] = E->pelements[i];
     }
     i++;
     tab[i] = e;

     free(E->pelements);
     E->pelements = tableau;

   }

  return E;
}
```

-Q3 :

```
Enemsble suppElement(Enemsble E, int e)
{
{
   int existant;
   int* tableau;

   existant = -1;


   for (int i = 0; i < E->nbElements; i++)
   {
     if (E->pelements[i] == e)
     {
       existant = i;
     }
   }



   if (existant != -1)
   {
     E->nbElements --;
     tableau = (int*) malloc (sizeof(int*E->nbElements));



     for (int i=0; i < existant; i++)
     {
       tableau[i] = E->pelements[i];
     }


     for (int i=existant; i < nbElements-1; i++)
     {
       tab[i] = E->pelements[i+1];
     }

     free(E->pelements);
     E->pelements = tableau;

   }
  else
   {
        printf("L'element n'existe pas dans l'ensemble.\n")
  return E;
}

//  on a d'abord copié les élément avant celui trouvé puis ceux cités apres
```

Exercie n°3

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

int main()
{
   int i;
   int k;

      for(i = 0; i < 10; i++)
        {
            // to do
        }


      for(k = 0; k < 10; k++)
           {
               // to do
           }

  return 0;
}
```

Exercie n°4

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

void questionQuatre(char *nomFic)
{
	char *resNomFic;
	FILE *pointFich, *resPointFich;
	float chiffreLu;
	int compteur=0;
	float somme=0;

	resNomFic = malloc((strlen(nomFic) + 4) * sizeof(char));

	strcpy(resNomFic, nomFic);
	strcat(resNomFic, ".res");

	pointFich = fopen(nomFic, "r");

	if (pointFich)
	{

		resPointFich = fopen(resNomFic, "w");

		while(!feof(pointFich))
		{
			fscanf(pointFich, "%f", &chiffreLu);
			compteur++;
			somme = somme + chiffreLu;
		}

		fwrite(&somme, sizeof(float), 1, resPointFich);
		fwrite(&compteur, sizeof(int), 1, resPointFich);

		fclose(resPointFich);
		fclose(pointFich);
	}
}
```