Examen du 15/11/2004

corrigé

Exercice I

```
#include <stdio.h>

int main()
{
  int i, j, k;
  int *P;

  i=0;  // i recoit la valeur 0
  j=1;  // j recoit la valeur 1
  k=2;  // k recoit la valeur 2
  P=&i; // P pointe vers i (qui contient 0)
  j=*P; // j recoit ce qui est pointer par P donc 0
  i=k;  // i recoit 2 ce qui ne change pas j
  printf("i=%d, j=%d, k=%d\n",i,j,k);

  return 0;
}
```

Le programme affiche `i=2, j=0, k=2`.

Exercice II

On utilise la formule de Taylor

$$\sin(x) = \sum_{k=0}^{n} \frac{\sin^{[k]}(0)}{k!} x^k + \mathrm{o}(x^n)$$

```
#include <stdio.h>
#include <assert.h>
#include <math.h>

int fact(n)
  /* ENTREE : un entier naturel n
     SORTIE : n! */
{
  assert (n>=0);
  if (n==0) return 1;
  else return n*fact(n-1);
}

double* DLsin (int n)
     /* ENTREE : ordre du DL de sin en 0
```

```
        SORTIE : tableau des coefficients du DL */
{
  double* DL;
  int k, numerateur;

  assert (n>=0);
  DL=(double*)malloc((n+1)*sizeof(double));
  for (k=0;k<=n;k++) {
    switch (k%4) {
    case 0: // La derivee 4p-ieme de sin est sin et sin(0)=0
      numerateur=0; break;
    case 1: // La derivee (4p+1)-ieme de sin est cos et cos(0)=1
      numerateur=1; break;
    case 2: // La derivee (4p+2)-ieme de sin est -sin et -sin(0)=0
      numerateur=0; break;
    case 3: // La derivee (4p+3)-ieme de sin est -cos et -cos(0)=-1
      numerateur=-1; break;
    }
    // Le coefficient est 1/k! * derivee k-ieme de sin en 0
    // Pour eviter de faire une division entiere on fait un type casting
    DL[k]=(double)numerateur/fact(k);
  }

  return DL;
}

int main()
{
  double x=0.1; // point d'evaluation de sin
  int n=10; // nombre de termes du DL
  double *DL, s;
  int k;

  DL=DLsin(n);
  s=0; // variable d'accumulation initialisee a 0
  for (k=0;k<=n;k++)
    s+=DL[k]*pow(x,k);
  printf("sin(0.1)=%lf\n",s);
}
```

Exercice III

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

// La macro suivante simplie l'ecriture de irreductibleQ
#define min(a,b) ((a)<(b)?(a):(b))

typedef struct rationnel { int N, D; } Q;

Q creerQ (int num, int den)
     /* ENTREE : le numerateur num et le denominateur den!=0
```

```
        SORTIE : le nombre rationnel num/den */
{
  Q resultat;
  assert(den!=0);
  resultat.N=num;
  resultat.D=den;
  return(resultat);
}

void afficherQ (Q r)
     /* ENTREE : le nombre rationnel r
        Affichage de r sur le standard output */
{
  printf("%d/%d",r.N,r.D);
  // Le buffer n'est pas flushe pour permettre l'ecriture de plusieur
  // nombres rationnels sur une meme ligne, si necessaire.
}

double approximerQ (Q r)
     /* ENTREE : le nombre rationnel r
        SORTIE : une approximation reele de r */
{
  return((double)r.N/r.D);
  // Attention a ne pas oublie le type casting
  // pour ne pas faire une division entiere
}

int egalQ (Q r1, Q r2)
     /* ENTREE : deux nombres rationnels r1 et r2
        SORTIE : 1 si r1=r2 et 0 sinon */
{
  return(r1.N*r2.D==r1.D*r2.N);
  // Attention a bien faire les produits croises et ne pas simplement
  // comparer numerateurs et denomniateurs (2/3=4/6)
}

Q produitQ (Q r1, Q r2)
     /* ENTREE : deux nombres rationnels r1 et r2
        SORTIE : r1*r2 */
{
  Q resultat;
  resultat.N=r1.N*r2.N;
  resultat.D=r1.D*r2.D;
  return(resultat);
}

Q divisionQ (Q r1, Q r2)
     /* ENTREE : deux nombres rationnels r1 et r2!=0
        SORTIE : r1/r2 */
{
  Q resultat;
  assert(r2.N!=0);
  resultat.N=r1.N*r2.D;
  resultat.D=r1.D*r2.N;
```

```
  return(resultat);
}

Q sommeQ (Q r1, Q r2)
     /* ENTREE : deux nombres rationnels r1 et r2
        SORTIE : r1+r2 */
{
  Q resultat;
  resultat.D=r1.D*r2.D; // on met au meme denominateur
  resultat.N=r2.N*r1.D+r1.N*r2.D;
  return(resultat);
}

Q irreductibleQ (Q r)
     /* ENTREE : un nombre rationnel r
        SORTIE : r sous forme irreductible */
{
  int n;

  /* On commence par regarder si 2 est un facteur commun
     au numerateur et au denominateur */
  n=2;

  while (n<=min(r.N,r.D)) {
    if ((r.N%n==0)&&(r.D%n==0))
      {
// si c'est le cas on simplifie
r.N/=n;
r.D/=n;
      }
    // sinon on passe a l'entier suivant
    // (le nombre premier suivant serait suffisant)
    else n++;
  }

  return r;
}

int main()
{
  Q r, r1, r2, r3, r4;

  r1=creerQ(1,2);
  r2=creerQ(1,4);
  r3=creerQ(2,9);
  r4=creerQ(1,7);
  r=divisionQ(sommeQ(r1,r2),sommeQ(r3,r4));
  r=irreductibleQ(r);
  printf("r=");
  afficherQ(r);
  printf(" et sa valeur approchee est %lf\n",approximerQ(r));
  return 0;
}
```