```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <math.h>

double racine(double a, double eps) {
  double x0=a, x1=a;
  if (a==0) return 0;
  do { x0=x1;
    x1=1/2.*(x0+a/x0);
  } while (fabs(x0-x1)>(fabs(x0)+fabs(x1))*eps);
  return x1;
}

void ex1(){
  double reel;
  printf("Entrer un reel"); fflush(stdout);
  scanf("%lf", &reel);
  if (reel>=0) printf("La racine de %lf est %lf \n",reel, racine
      (reel,1E-4));
  else printf("La racine d'un negatif n'existe pas\n");
}

void produit(double a, double b, double* mp) { *mp = a*b; }

double moyenne(double a, double b) { return (a+b)/2; }

void moyProd(double a, double b, double* mp, double* ms) {
  produit(a,b,mp);
  *ms=moyenne(a,b);
}

void ex2() { double x1,x2,x3,x4;
  printf("Entrer 2  nombres positifs: "),  fflush(stdout);
  scanf("%lf %lf",&x1,&x2);
  printf("Les 2 nombres %lf %lf\n",x1,x2);
  moyProd(x1,x2,&x3,&x4);
  printf("Les somme et produit des 2 nombres sont %lf %lf\n",x3,x4);
}


typedef unsigned char PIXEL;
typedef struct {
  int nl,nc;
  PIXEL** val;
  } IMAGEUCHAR;

PIXEL** allouеMemoirePixel(int nl, int nc) { int i;
```



```c
    PIXEL** p=NULL;
    if (nl>0 && nc>0) {
        if ( (p=(PIXEL**)calloc(nl,sizeof(*p)))==NULL) return
            NULL;
        if ( (*p=(PIXEL*)calloc(nl*nc,sizeof(**p)))==NULL) {
            free(p); return NULL;}
        for (i=1; i<nl; i++) p[i]=p[i-1]+nc;
    }
    return p;
}

IMAGEUCHAR initImage() { IMAGEUCHAR im; im.nl=0; im.nc=0; im.
    val=NULL; return im;}

IMAGEUCHAR creationImageUChar(int nl, int nc) {
    IMAGEUCHAR im;
    im.val=allouеMemoirePixel(nl,nc);
    if (im.val!=NULL) {im.nl=nl; im.nc=nc;}
    else {im=initImage();}
    return im;
}

int estVideImageUChar(IMAGEUCHAR im) { return im.nl>0 && im.
    nc>0 ? 0 : 1; }

void setImageUChar(IMAGEUCHAR im, PIXEL valeur) {
  int i,j;
  if (!estVideImageUChar(im)) {
    for (i=0;i<im.nl; i++)
      for(j=0;j<im.nc; j++)
        im.val[i][j]=valeur;
  }
}

void libereImageUChar(IMAGEUCHAR* pim) {
    if (pim->val!=NULL) {
        free(*(pim->val));free(pim->val);
    }
    *pim=initImage(); // ou pim->nl=pim->nc=0; pim->val=NULL;
}

int egalTableauUChar(PIXEL* t1, PIXEL* t2, int nbOctet) {
    PIXEL* p1; PIXEL* p2;
    for (p1=t1,p2=t2; p1<t1+nbOctet; p1++,p2++)
      if (*p1!=*p2) return 0;
  return 1;
}

int egaleImageUChar(IMAGEUCHAR  im1, IMAGEUCHAR im2) {
```

```
    return (im1.nl==im2.nl && im1.nc==im2.nc &&
        egalTableauUChar(im1.val[0],im2.val[0],im1.nl*im1.nc)
        );
}

PIXEL distance( IMAGEUCHAR im, IMAGEUCHAR patch, int is,int
    js) {
    int di,dj; unsigned int d=0;

    for (di=0; di<patch.nl; di++) {
        for (dj=0; dj<patch.nc; dj++) {
                d += abs(im.val[is+di-patch.nl/2][js+dj-patch
                    .nc/2]-patch.val[di][dj]);
        }
  }
  return((d)/patch.nl/patch.nc);
}

IMAGEUCHAR templateMatching(IMAGEUCHAR im, IMAGEUCHAR modele)
    { int i,j;
  IMAGEUCHAR res;

  res=creationImageUChar(im.nl,im.nc);
  setImageUChar(res,255);
  for (i=modele.nl/2; i<im.nl-modele.nl/2; i++) {
    printf("\b\b\b\b\b\b\b%3.2lf %%",i/(double)im.nl*100); fflush
        (stdout);
    for (j=modele.nc/2; j<im.nc-modele.nc/2; j++) {
       (res.val)[i][j]=distance(im,modele,i,j);
    }
    }
    puts("");
  return res;
}


int lectureImagePgmTexte(char* fic, IMAGEUCHAR * im) { int i,
    valmax,a,b;
  FILE* fp=fopen(fic,"r");
  char s[1024];

  if (fp==NULL) return -1;
  if ( (fscanf(fp,"%s",s) !=1) || strncmp(s,"P2",2)) return 1;
  if (fscanf(fp,"%d %d ",&a,&b) !=2) return 2;
  *im=creationImageUChar(b,a);
  if (im->val==NULL) return 2;
  if (fscanf(fp,"%d",&valmax)!=1) return 3;
  for (i=0;i<im->nc*im->nl; i++)
    if (fscanf(fp,"%d",(*(im->val))+i)!=1) return 4;
```



```
  fclose(fp);
  return 0;
}


int ecritureImagePgmBinaire(char* fic, IMAGEUCHAR * im) {
  FILE* fp=fopen(fic,"w");

  if (fp==NULL) return -1;
  fprintf(fp,"%s","P5\n");
  fprintf(fp,"%d %d\n",im->nc,im->nl);
  fprintf(fp,"%d\n",255);
  fwrite(im->val[0],im->nc*im->nl,1,fp);
  fclose(fp);
  return 0;
}

int ex3(int ac, char** av) {
        /* Variables contenant les images */
      IMAGEUCHAR im1, modele, imdistance;
      int res;

        /*
          On verifie que le programme est lance  avec un nom
              de fichier
        */
      if ( (ac != 3)  || av[1]==NULL ){
          printf("Usage : %s image modele\n",av[0]); exit(1);
      }

        /* Lecture de l'image, des nombres de lignes et
            colonnes */
      printf("Lecture image %s\n",av[1]);
      res=lectureImagePgmTexte(av[1],&im1);
      if (res) { printf("Lecture image %s Impossible\n",av[1]);
          return EXIT_FAILURE; }

      /* Lecture du modele, des nombres de lignes et colonnes
          */
      printf("Lecture image %s\n",av[2]);
      res=lectureImagePgmTexte(av[2],&modele);
      if (res) { printf("Lecture image %s Impossible\n",av[1]);
          return EXIT_FAILURE; }

        /* Calcul du block matching et sauvegarde d'une
            version 8 bits dans un fichier */
      printf("Block matching\n");
      imdistance=templateMatching(im1,modele);
      if (estVideImageUChar(imdistance)) return EXIT_FAILURE;
```

```
        ecritureImagePgmBinaire("distance.pgm",&imdistance);

        libereImageUChar(&im1);
        libereImageUChar(&modele);
          libereImageUChar(&imdistance);
        return EXIT_SUCCESS;
    }

     int main(int argc, char const *argv[]) {
      ex1();
      ex2();
      ex3(argc,argv);
    }
```