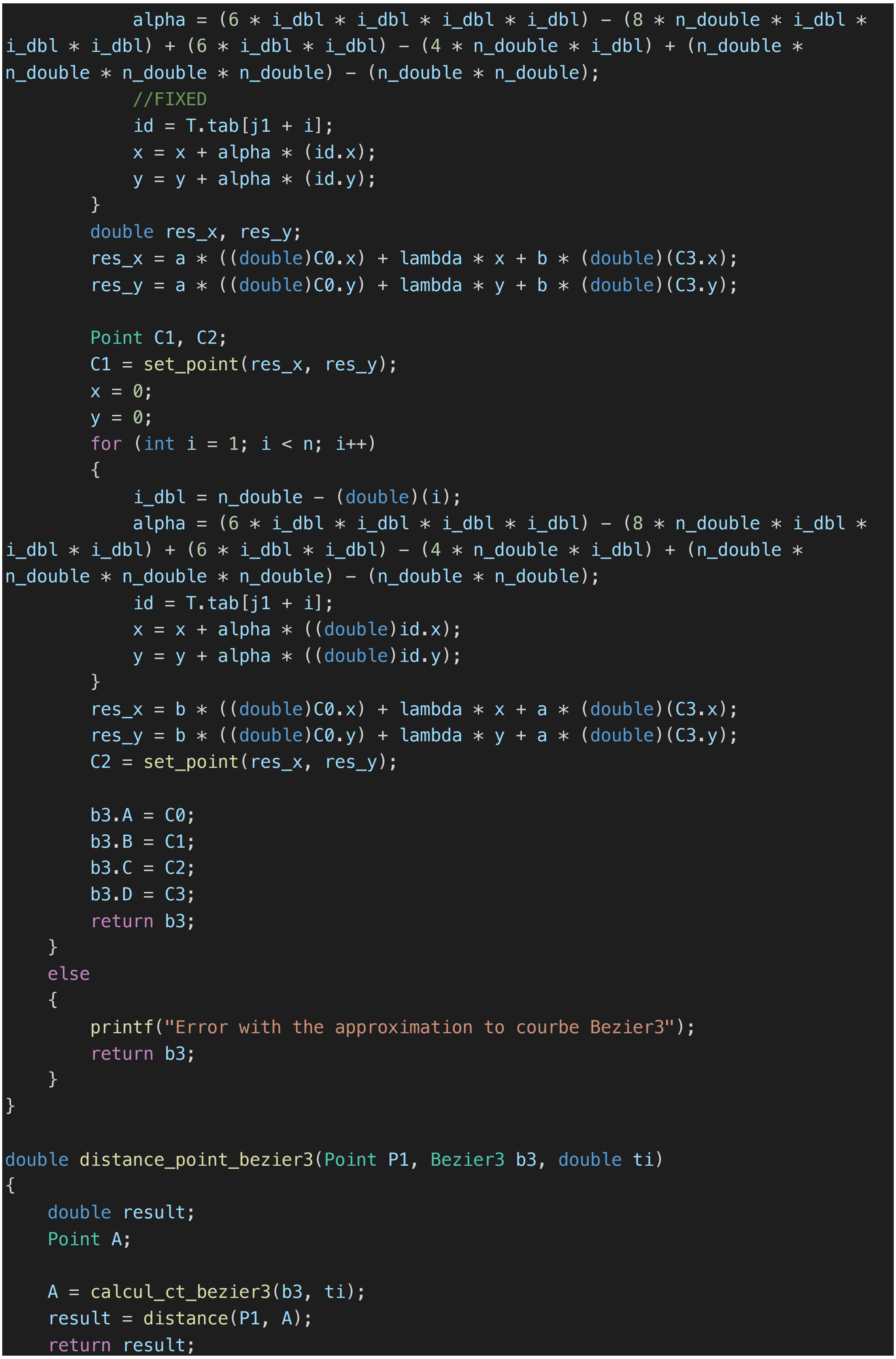
Tache 7 Partie 1

File contour.h:

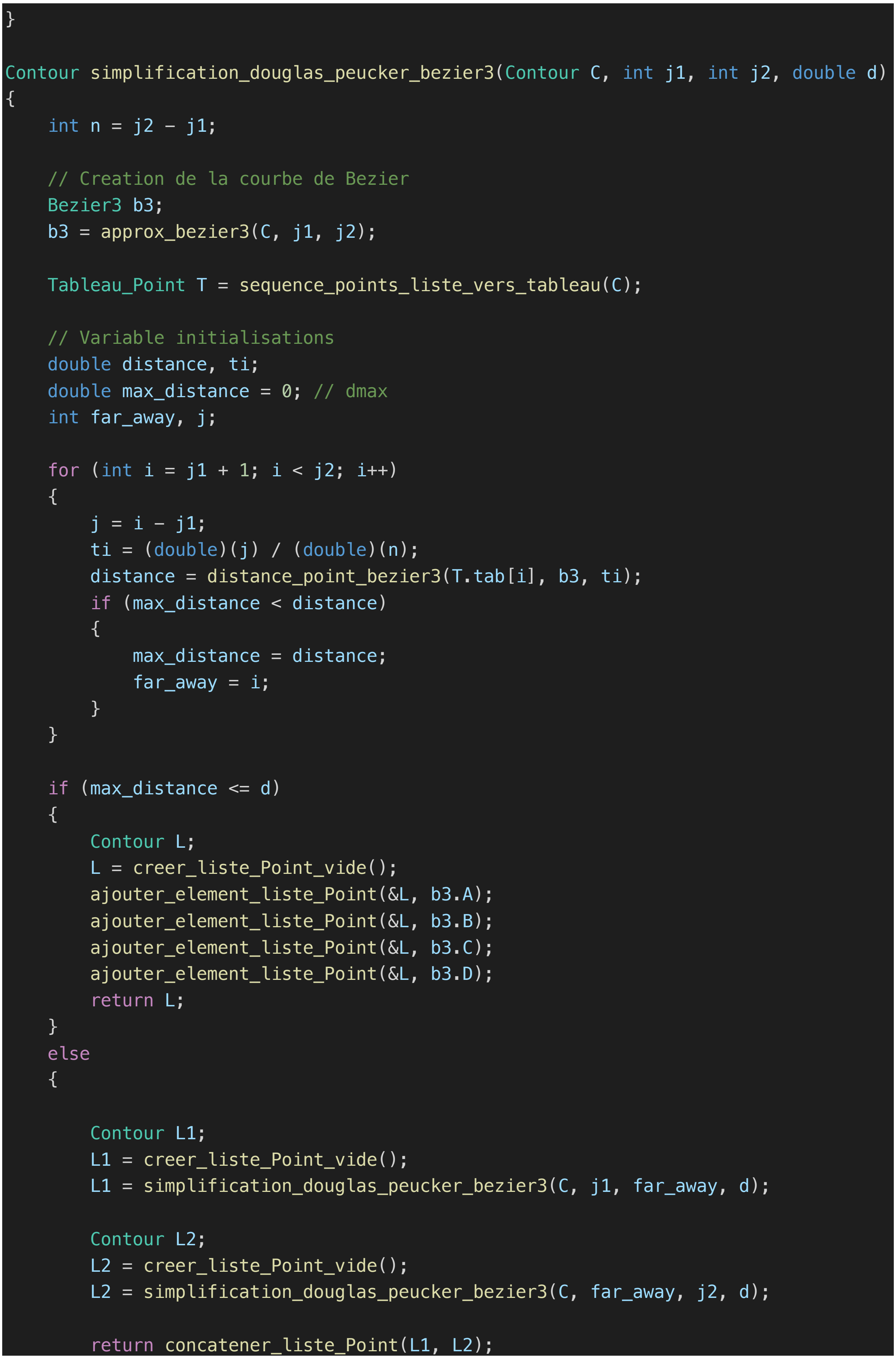
```
...
Bezier3 approx_bezier3(Contour c, int j1, int j2);

double distance_point_bezier3(Point P1, Bezier3 b3, double ti);

Contour simplification_douglas_peucker_bezier3(Contour C, int j1, int j2,double d);

void create_postscript_contours_bezier3(Liste_Contours c, char *file_name, int
hauteur, int largeur);
...
```

Source code de contour.c modifié :

```
...
void contours_data_bezier3(Liste_Contours c)
{
    Cellule_Liste_Contours *el;
    el = c.first;
    int nb = 0;
    int nb_beziers = 0;
    while (el != NULL)
    {
        nb++;
        Cellule_Liste_Point *e;
        e = (el->data).first;
        while (e != NULL)
        {
            e = e->suiv;
            e = e->suiv;
            e = e->suiv;
            e = e->suiv;
            nb_beziers++;
        }
        el = el->suiv;
    }
    printf("Nombre des contours: %d\n", nb);
    printf("Nombre des bezier totals: %d\n", nb_beziers);
    printf("\n");
}

...
Bezier3 approx_bezier3(Contour c, int j1, int j2)
{
    Bezier3 b3;
    int n = j2 - j1;
    Tableau_Point T = sequence_points_liste_vers_tableau(c);
    Point C0, C3;
    C0 = T.tab[j1];
    C3 = T.tab[j2];

    if (n == 1)
    {
        Point C1, C2;
        C1 = set_point((2 * C0.x + C3.x) / 3, (2 * C0.y + C3.y) / 3);
        C2 = set_point((C0.x + 2 * C3.x) / 3, (C0.y + 2 * C3.y) / 3);
        // Declaration de la courbe bezier
        b3.A = C0;
        b3.B = C1;
        b3.C = C2;
        b3.D = C3;
        return b3;
    }
    else if (n == 2)
    {
        Point C1, C2, P1;
        P1 = T.tab[j1 + 1];
        C1 = set_point((4 * P1.x - C3.x) / 3, (4 * P1.y - C3.y) / 3);
        C2 = set_point((4 * P1.x - C0.x) / 3, (4 * P1.y - C0.y) / 3);
        // Declaration de la courbe bezier
        b3.A = C0;
        b3.B = C1;
        b3.C = C2;
        b3.D = C3;
        return b3;
    }
    else if (n > 2)
    {
        double n_double;
        n_double = (double)(n);

        // Calcul a et b
        double a, b, lambda;
        a = (-15 * n_double * n_double * n_double + 5 * n_double * n_double + 2 *
n_double + 4) / (3 * (n_double + 2) * (3 * n_double * n_double + 1));
        b = ((10 * n_double * n_double * n_double - 15 * n_double * n_double +
n_double + 2) / (3 * (n_double + 2) * (3 * n_double * n_double + 1)));
        lambda = (70 * n_double) / (3 * (n_double * n_double - 1) * (n_double *
n_double - 4) * (3 * n_double * n_double + 1));
        // définir la fonction alpha(i) a faire
        double x = 0.0;
        double y = 0.0;
        Point id;
        double i_dbl, alpha;
        for (int i = 1; i < n; i++)
        {
            i_dbl = (double)(i);
```



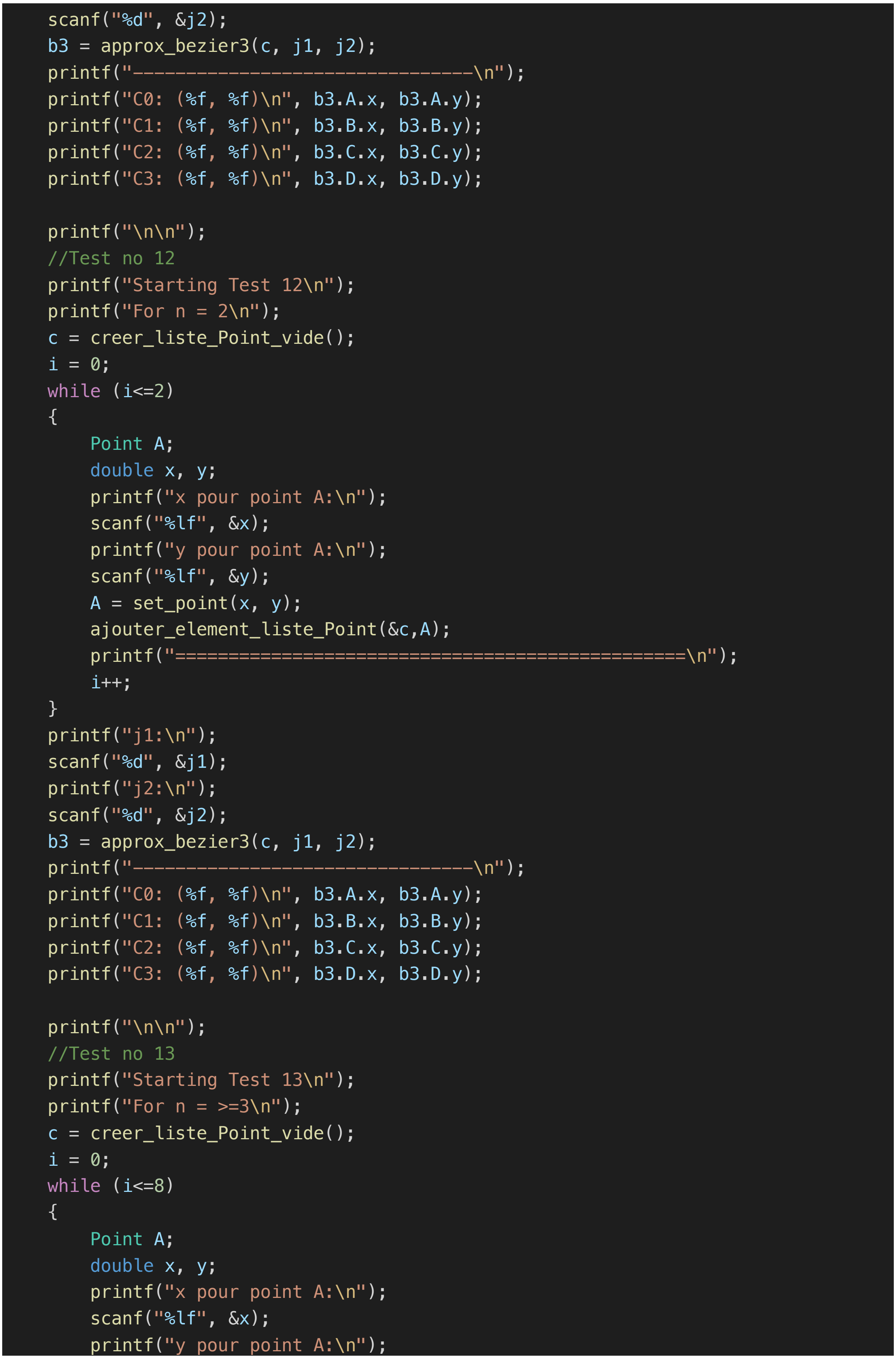
```c
            alpha = (6 * i_dbl * i_dbl * i_dbl * i_dbl) - (8 * n_double * i_dbl *
i_dbl * i_dbl) + (6 * i_dbl * i_dbl) - (4 * n_double * i_dbl) + (n_double *
n_double * n_double * n_double) - (n_double * n_double);
            //FIXED
            id = T.tab[j1 + i];
            x = x + alpha * (id.x);
            y = y + alpha * (id.y);
        }
        double res_x, res_y;
        res_x = a * ((double)C0.x) + lambda * x + b * (double)(C3.x);
        res_y = a * ((double)C0.y) + lambda * y + b * (double)(C3.y);

        Point C1, C2;
        C1 = set_point(res_x, res_y);
        x = 0;
        y = 0;
        for (int i = 1; i < n; i++)
        {
            i_dbl = n_double - (double)(i);
            alpha = (6 * i_dbl * i_dbl * i_dbl * i_dbl) - (8 * n_double * i_dbl *
i_dbl * i_dbl) + (6 * i_dbl * i_dbl) - (4 * n_double * i_dbl) + (n_double *
n_double * n_double * n_double) - (n_double * n_double);
            id = T.tab[j1 + i];
            x = x + alpha * ((double)id.x);
            y = y + alpha * ((double)id.y);
        }
        res_x = b * ((double)C0.x) + lambda * x + a * (double)(C3.x);
        res_y = b * ((double)C0.y) + lambda * y + a * (double)(C3.y);
        C2 = set_point(res_x, res_y);

        b3.A = C0;
        b3.B = C1;
        b3.C = C2;
        b3.D = C3;
        return b3;
    }
    else
    {
        printf("Error with the approximation to courbe Bezier3");
        return b3;
    }
}

double distance_point_bezier3(Point P1, Bezier3 b3, double ti)
{
    double result;
    Point A;

    A = calcul_ct_bezier3(b3, ti);
    result = distance(P1, A);
    return result;
```



```
}

Contour simplification_douglas_peucker_bezier3(Contour C, int j1, int j2, double d)
{
    int n = j2 - j1;

    // Creation de la courbe de Bezier
    Bezier3 b3;
    b3 = approx_bezier3(C, j1, j2);

    Tableau_Point T = sequence_points_liste_vers_tableau(C);

    // Variable initialisations
    double distance, ti;
    double max_distance = 0; // dmax
    int far_away, j;

    for (int i = j1 + 1; i < j2; i++)
    {
        j = i - j1;
        ti = (double)(j) / (double)(n);
        distance = distance_point_bezier3(T.tab[i], b3, ti);
        if (max_distance < distance)
        {
            max_distance = distance;
            far_away = i;
        }
    }

    if (max_distance <= d)
    {
        Contour L;
        L = creer_liste_Point_vide();
        ajouter_element_liste_Point(&L, b3.A);
        ajouter_element_liste_Point(&L, b3.B);
        ajouter_element_liste_Point(&L, b3.C);
        ajouter_element_liste_Point(&L, b3.D);
        return L;
    }
    else
    {

        Contour L1;
        L1 = creer_liste_Point_vide();
        L1 = simplification_douglas_peucker_bezier3(C, j1, far_away, d);

        Contour L2;
        L2 = creer_liste_Point_vide();
        L2 = simplification_douglas_peucker_bezier3(C, far_away, j2, d);

        return concatener_liste_Point(L1, L2);
    }
}

void create_postscript_contours_bezier3(Liste_Contours c, char *file_name, int
hauteur, int largeur) // Mode remplisage uniquement
{
    // Extension managment
    char *no_extension = strtok(file_name, ".");
    char *with_extension = malloc(strlen(no_extension) + 4);
    strcpy(with_extension, no_extension);
    strcat(with_extension, ".eps"); // concantenation

    FILE *fptr;
    fptr = fopen(with_extension, "w");
    if (fptr == NULL)
    {
        printf("EPS File Error!");
        exit(1);
    }

    fprintf(fptr, "%%!PS-Adobe-3.0 EPSF-3.0\n");
    fprintf(fptr, "%%%%BoundingBox:   %d   %d   %d   %d\n", 0, 0, largeur,
hauteur);
    fprintf(fptr, "\n");
    Cellule_Liste_Contours *al;
    al = c.first;
    while (al != NULL)
    {
        Cellule_Liste_Point *el;
        el = (al->data).first;
        Bezier3 b3;
        b3.A = el->data;
        el = el->suiv;
        b3.B = el->data;
        el = el->suiv;
        b3.C = el->data;
        el = el->suiv;
        b3.D = el->data;
        fprintf(fptr, "%.3f %.3f moveto ", b3.A.x, hauteur - b3.A.y);
        fprintf(fptr, "%.3f %.3f %.3f %.3f %.3f %.3f curveto ", b3.B.x, hauteur -
b3.B.y, b3.C.x, hauteur - b3.C.y, b3.D.x, hauteur - b3.D.y);
        el = el->suiv;
        while (el != NULL)
        {
            b3.A = el->data;
            el = el->suiv;
            b3.B = el->data;
            el = el->suiv;
            b3.C = el->data;
            el = el->suiv;
            b3.D = el->data;
            fprintf(fptr, "%.3f %.3f %.3f %.3f %.3f %.3f curveto ", b3.B.x, hauteur - b3.B.y, b3.C.x, hauteur - b3.C.y, b3.D.x, hauteur - b3.D.y);
            el = el->suiv;
        }
        fprintf(fptr, "\n 2.0 setlinewidth");
        fprintf(fptr, "\n");
        al = al->suiv;
    }
    fprintf(fptr, "fill\n");
    fprintf(fptr, "\n");
    fprintf(fptr, "\n");
    fprintf(fptr, "showpage\n");
    fclose(fptr);
    return;
}
```

Source code de partie 2.1 (test program : test_approx3.c) :

```
#include <stdint.h>
#include <string.h>
#include<stdlib.h>

#include "contour.h"
#include "image.h"

int main(int argc, char **argv)
{
    //Test no 11
    printf("Starting Test 11\n");
    printf("For n = 1\n");
    Contour c;
    c = creer_liste_Point_vide();
    int i = 0;
    while (i<=1)
    {
        Point A;
        double x, y;
        printf("x pour point A:\n");
        scanf("%lf", &x);
        printf("y pour point A:\n");
        scanf("%lf", &y);
        A = set_point(x, y);
        ajouter_element_liste_Point(&c,A);
        printf("=================================================\n");
        i++;
    }
    Bezier3 b3;
    int j1, j2;
    printf("j1:\n");
    scanf("%d", &j1);
    printf("j2:\n");
```

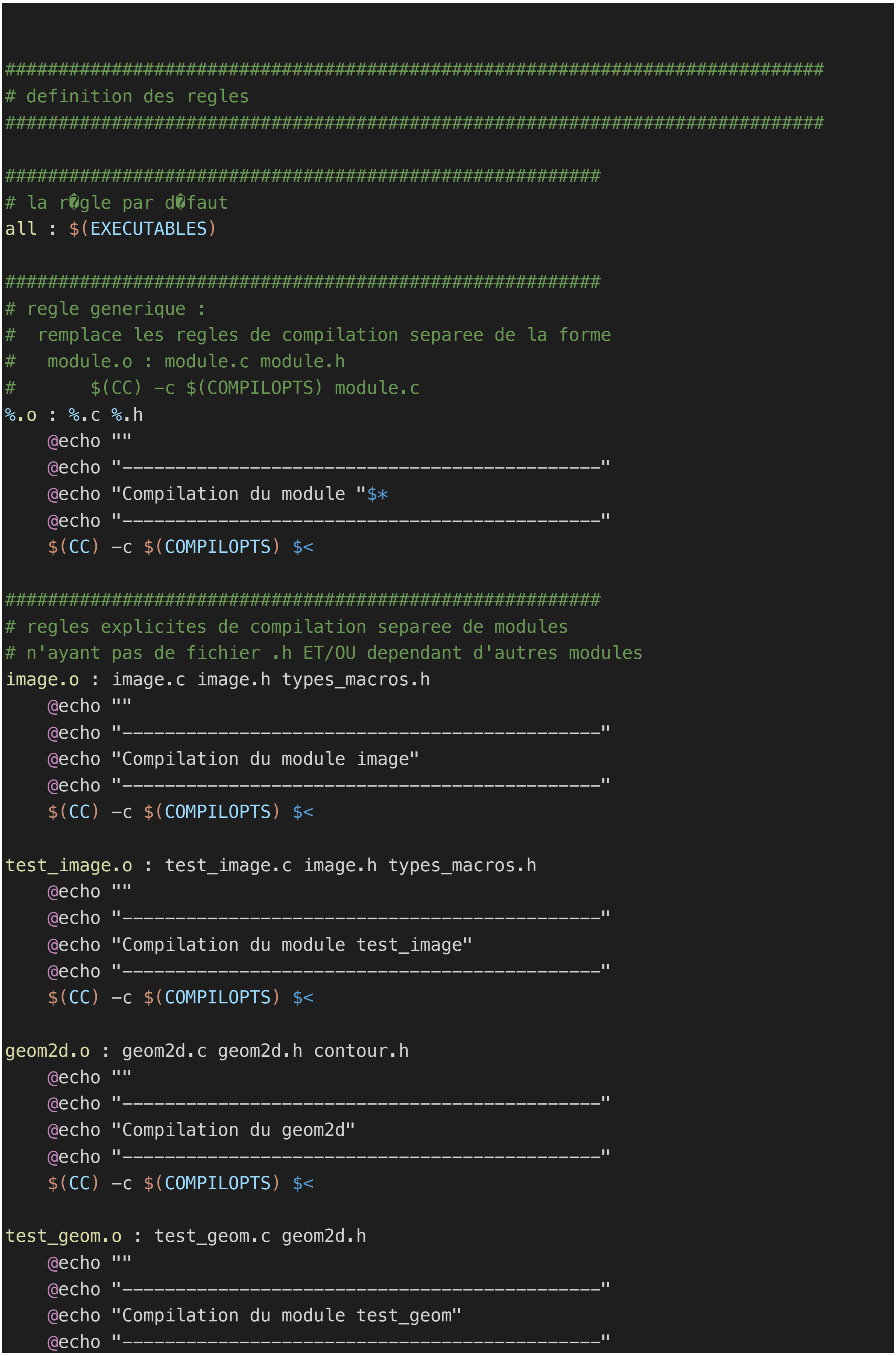


```
    scanf("%d", &j2);
    b3 = approx_bezier3(c, j1, j2);
    printf("--------------------------------\n");
    printf("C0: (%f, %f)\n", b3.A.x, b3.A.y);
    printf("C1: (%f, %f)\n", b3.B.x, b3.B.y);
    printf("C2: (%f, %f)\n", b3.C.x, b3.C.y);
    printf("C3: (%f, %f)\n", b3.D.x, b3.D.y);

    printf("\n\n");
    //Test no 12
    printf("Starting Test 12\n");
    printf("For n = 2\n");
    c = creer_liste_Point_vide();
    i = 0;
    while (i<=2)
    {
        Point A;
        double x, y;
        printf("x pour point A:\n");
        scanf("%lf", &x);
        printf("y pour point A:\n");
        scanf("%lf", &y);
        A = set_point(x, y);
        ajouter_element_liste_Point(&c,A);
        printf("================================================\n");
        i++;
    }
    printf("j1:\n");
    scanf("%d", &j1);
    printf("j2:\n");
    scanf("%d", &j2);
    b3 = approx_bezier3(c, j1, j2);
    printf("--------------------------------\n");
    printf("C0: (%f, %f)\n", b3.A.x, b3.A.y);
    printf("C1: (%f, %f)\n", b3.B.x, b3.B.y);
    printf("C2: (%f, %f)\n", b3.C.x, b3.C.y);
    printf("C3: (%f, %f)\n", b3.D.x, b3.D.y);

    printf("\n\n");
    //Test no 13
    printf("Starting Test 13\n");
    printf("For n = >=3\n");
    c = creer_liste_Point_vide();
    i = 0;
    while (i<=8)
    {
        Point A;
        double x, y;
        printf("x pour point A:\n");
        scanf("%lf", &x);
        printf("y pour point A:\n");
```

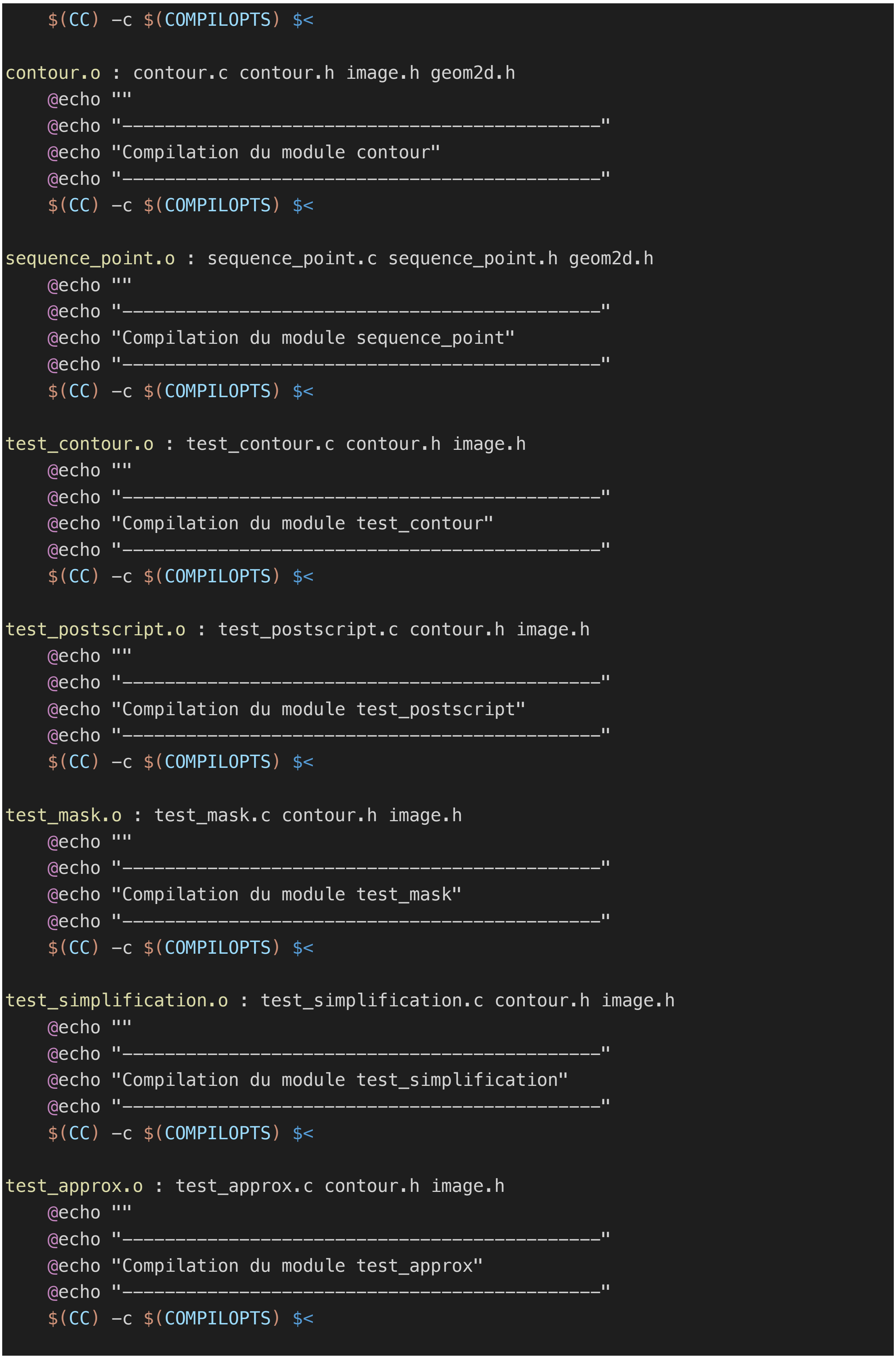
```c
        scanf("%lf", &y);
        A = set_point(x, y);
        ajouter_element_liste_Point(&c,A);
        printf("=================================================\n");
        i++;
    }
    printf("j1:\n");
    scanf("%d", &j1);
    printf("j2:\n");
    scanf("%d", &j2);
    b3 = approx_bezier3(c, j1, j2);
    printf("--------------------------------\n");
    printf("C0: (%f, %f)\n", b3.A.x, b3.A.y);
    printf("C1: (%f, %f)\n", b3.B.x, b3.B.y);
    printf("C2: (%f, %f)\n", b3.C.x, b3.C.y);
    printf("C3: (%f, %f)\n", b3.D.x, b3.D.y);

    return 0;

}
```

Nouveau Makefile :

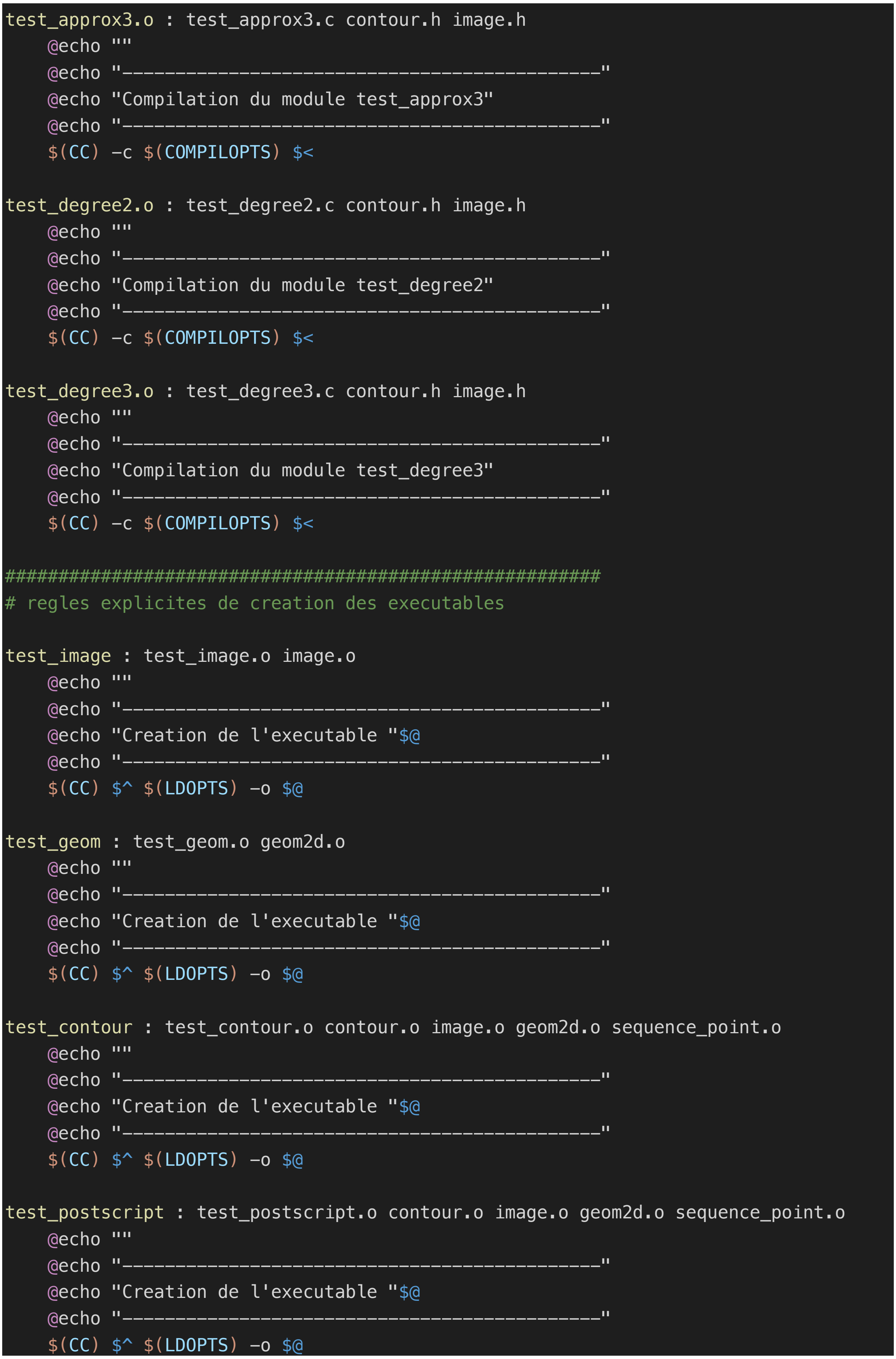
```makefile
##############################################################################
# Fichier Makefile
# UE MAP401 - DLST - UGA - 2022/2023
##############################################################################

# compilateur C
CC = clang

# chemin d'acces aux librairies (interfaces)
INCDIR = .

# chemin d'acces aux librairies (binaires)
LIBDIR = .

# options pour l'�dition des liens
LDOPTS = -L$(LIBDIR) -lm

# options pour la recherche des fichiers .o et .h
INCLUDEOPTS = -I$(INCDIR)

# options de compilation
COMPILOPTS = -g -Wall $(INCLUDEOPTS)

# liste des executables
EXECUTABLES = test_image test_geom test_contour test_postscript test_mask
test_simplification test_approx test_degree2 test_degree3 test_approx3
```



```
#############################################################################
# definition des regles
#############################################################################

########################################################
# la r�gle par d�faut
all : $(EXECUTABLES)

########################################################
# regle generique :
#  remplace les regles de compilation separee de la forme
#   module.o : module.c module.h
#       $(CC) -c $(COMPILOPTS) module.c
%.o : %.c %.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module "$*
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

########################################################
# regles explicites de compilation separee de modules
# n'ayant pas de fichier .h ET/OU dependant d'autres modules
image.o : image.c image.h types_macros.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module image"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

test_image.o : test_image.c image.h types_macros.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module test_image"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

geom2d.o : geom2d.c geom2d.h contour.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du geom2d"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

test_geom.o : test_geom.c geom2d.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module test_geom"
    @echo "---------------------------------------------"
```



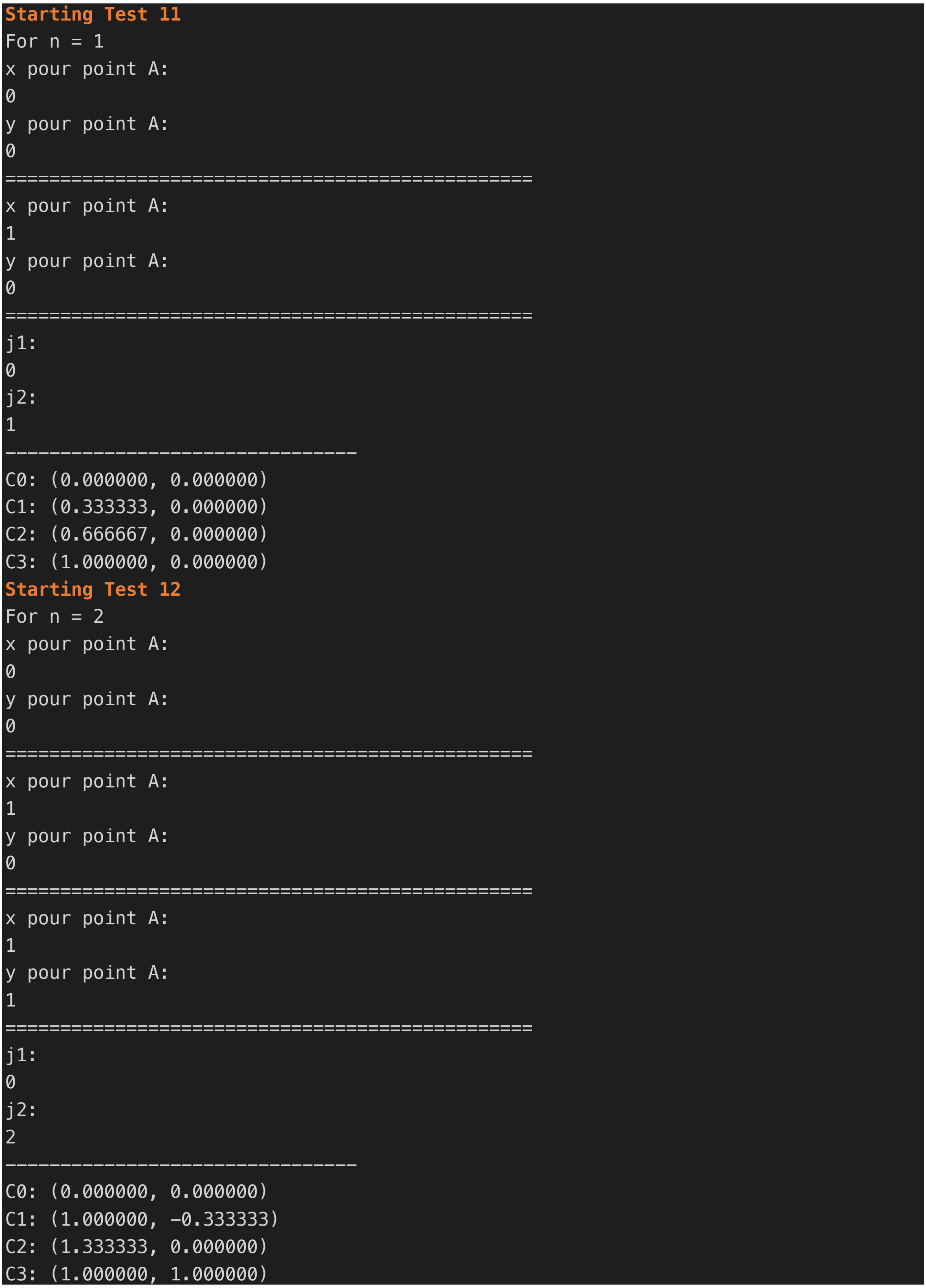
```
    $(CC) -c $(COMPILOPTS) $<

contour.o : contour.c contour.h image.h geom2d.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module contour"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

sequence_point.o : sequence_point.c sequence_point.h geom2d.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module sequence_point"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

test_contour.o : test_contour.c contour.h image.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module test_contour"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

test_postscript.o : test_postscript.c contour.h image.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module test_postscript"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

test_mask.o : test_mask.c contour.h image.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module test_mask"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

test_simplification.o : test_simplification.c contour.h image.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module test_simplification"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

test_approx.o : test_approx.c contour.h image.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module test_approx"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<
```



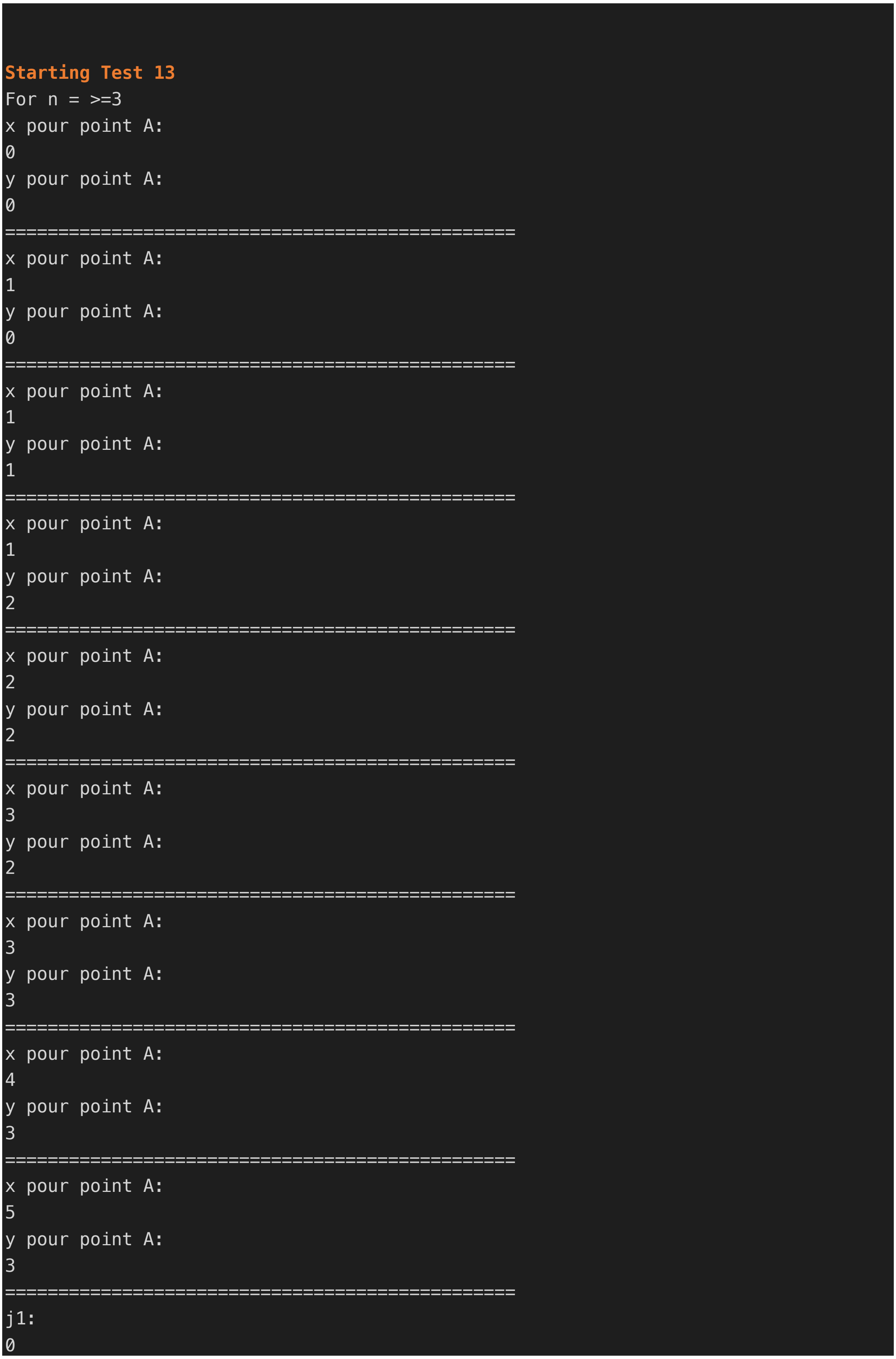
```
test_approx3.o : test_approx3.c contour.h image.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module test_approx3"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

test_degree2.o : test_degree2.c contour.h image.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module test_degree2"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

test_degree3.o : test_degree3.c contour.h image.h
    @echo ""
    @echo "---------------------------------------------"
    @echo "Compilation du module test_degree3"
    @echo "---------------------------------------------"
    $(CC) -c $(COMPILOPTS) $<

########################################################
# regles explicites de creation des executables

test_image : test_image.o image.o
    @echo ""
    @echo "---------------------------------------------"
    @echo "Creation de l'executable "$@
    @echo "---------------------------------------------"
    $(CC) $^ $(LDOPTS) -o $@

test_geom : test_geom.o geom2d.o
    @echo ""
    @echo "---------------------------------------------"
    @echo "Creation de l'executable "$@
    @echo "---------------------------------------------"
    $(CC) $^ $(LDOPTS) -o $@

test_contour : test_contour.o contour.o image.o geom2d.o sequence_point.o
    @echo ""
    @echo "---------------------------------------------"
    @echo "Creation de l'executable "$@
    @echo "---------------------------------------------"
    $(CC) $^ $(LDOPTS) -o $@

test_postscript : test_postscript.o contour.o image.o geom2d.o sequence_point.o
    @echo ""
    @echo "---------------------------------------------"
    @echo "Creation de l'executable "$@
    @echo "---------------------------------------------"
    $(CC) $^ $(LDOPTS) -o $@
```

```
test_mask : test_mask.o contour.o image.o geom2d.o sequence_point.o
    @echo ""
    @echo "---------------------------------------------"
    @echo "Creation de l'executable "$@
    @echo "---------------------------------------------"
    $(CC) $^ $(LDOPTS) -o $@

test_simplification : test_simplification.o contour.o image.o geom2d.o
sequence_point.o
    @echo ""
    @echo "---------------------------------------------"
    @echo "Creation de l'executable "$@
    @echo "---------------------------------------------"
    $(CC) $^ $(LDOPTS) -o $@

test_approx : test_approx.o contour.o image.o geom2d.o sequence_point.o
    @echo ""
    @echo "---------------------------------------------"
    @echo "Creation de l'executable "$@
    @echo "---------------------------------------------"
    $(CC) $^ $(LDOPTS) -o $@

test_approx3 : test_approx3.o contour.o image.o geom2d.o sequence_point.o
    @echo ""
    @echo "---------------------------------------------"
    @echo "Creation de l'executable "$@
    @echo "---------------------------------------------"
    $(CC) $^ $(LDOPTS) -o $@

test_degree2 : test_degree2.o contour.o image.o geom2d.o sequence_point.o
    @echo ""
    @echo "---------------------------------------------"
    @echo "Creation de l'executable "$@
    @echo "---------------------------------------------"
    $(CC) $^ $(LDOPTS) -o $@

test_degree3 : test_degree3.o contour.o image.o geom2d.o sequence_point.o
    @echo ""
    @echo "---------------------------------------------"
    @echo "Creation de l'executable "$@
    @echo "---------------------------------------------"
    $(CC) $^ $(LDOPTS) -o $@




# regle pour "nettoyer" le r�pertoire
clean:
    rm -fR $(EXECUTABLES) *.o
```

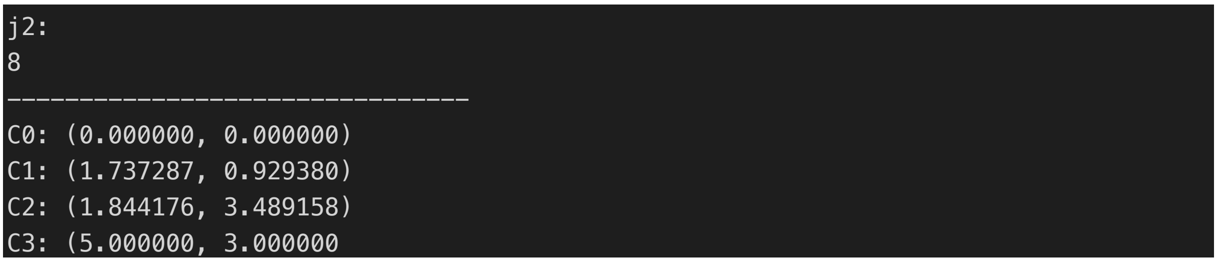
Results of the test_approx3:



```
Starting Test 11
For n = 1
x pour point A:
0
y pour point A:
0
=================================================
x pour point A:
1
y pour point A:
0
=================================================
j1:
0
j2:
1
--------------------------------
C0: (0.000000, 0.000000)
C1: (0.333333, 0.000000)
C2: (0.666667, 0.000000)
C3: (1.000000, 0.000000)
Starting Test 12
For n = 2
x pour point A:
0
y pour point A:
0
=================================================
x pour point A:
1
y pour point A:
0
=================================================
x pour point A:
1
y pour point A:
1
=================================================
j1:
0
j2:
2
--------------------------------
C0: (0.000000, 0.000000)
C1: (1.000000, -0.333333)
C2: (1.333333, 0.000000)
C3: (1.000000, 1.000000)
```



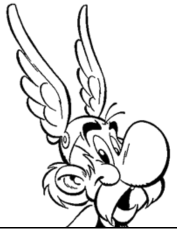
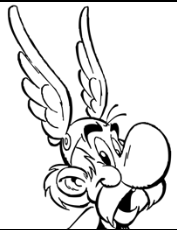
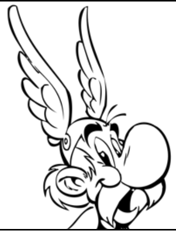
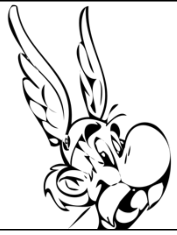
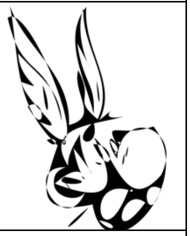
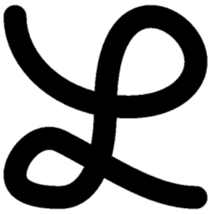
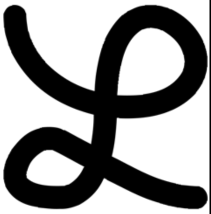
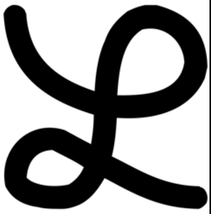
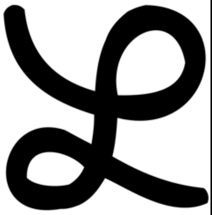
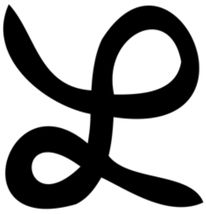
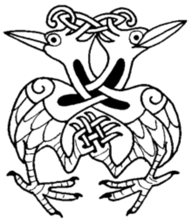
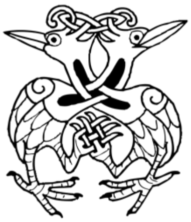
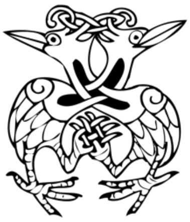
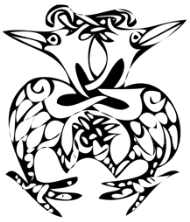
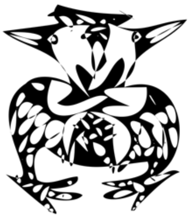
```
Starting Test 13
For n = >=3
x pour point A:
0
y pour point A:
0
================================================
x pour point A:
1
y pour point A:
0
================================================
x pour point A:
1
y pour point A:
1
================================================
x pour point A:
1
y pour point A:
2
================================================
x pour point A:
2
y pour point A:
2
================================================
x pour point A:
3
y pour point A:
2
================================================
x pour point A:
3
y pour point A:
3
================================================
x pour point A:
4
y pour point A:
3
================================================
x pour point A:
5
y pour point A:
3
================================================
j1:
0
```



```
j2:
8
--------------------------------
C0: (0.000000, 0.000000)
C1: (1.737287, 0.929380)
C2: (1.844176, 3.489158)
C3: (5.000000, 3.000000
```

Partie 2.2 :

Table:

Original	D=1	D=3	D=10	D=30
				
Asterix3 Nombre des contours: 32 Nombre des segments totals: 12926	Nombre des bezier totals: 648	Nombre des bezier totals: 242	Nombre des bezier totals: 135	Nombre des bezier totals: 58
				
lettre-L-cursive Nombre des contours: 3 Nombre des segments totals: 4228	Nombre des bezier totals: 157	Nombre des bezier totals: 32	Nombre des bezier totals: 22	Nombre des bezier totals: 13
				
ColombesDeLaPaix Nombre des contours: 106	Nombre des bezier totals: 1155	Nombre des bezier totals: 451	Nombre des bezier totals: 230	Nombre des bezier totals: 138

Nombre des segments totals: 21764				