COLEGIUL NAȚIONAL

“GHEORGHE MUNTEANU MURGOCI”

LUCRARE PRACTICĂ PENTRU OBȚINEREA ATESTATULUI PROFESIONAL LA INFORMATICĂ

Clasa a XII-a C

APLICAȚII ALE RECURSIVITĂȚII ÎN C++

Realizator: **Prof. coordonator:**

Legian Andrei-Ioan **Stanciu Gina**

An scolar 2021-2022

PREFAȚĂ

Fractalii sunt încă un domeniu relativ nou în matematică. Astfel că, în ziua de azi, mulți matematicieni încă descoperă lucruri noi în această ramură a artei în matematică, studiul fractalilor începând în secolul al XVII-lea, când matematicianul și filozoful Gottfried Leibniz s-a gândit la auto-asemănarea recursivă.

Acest program are ca scop facilitarea înțelegerii fractalilor și reprezentarea lor. De la simplul fractal de tip Square Grid, până la complexul fractal tip Hilbert, utilizatorul poate alege dintre mai multe tipuri de fractali, cât și detalierea lor.

După ce am studiat acest capitol al matematicii și al informaticii, am fost fascinate atât de ambiguitatea și complexitatea, cât și de simplitatea acestor reprezentări grafice. Astfel, m-am apucat să le aduc o oarecare reprezentare, prin transpunerea într-un program care să faciliteze înțelegerea acestui concept.

NOȚIUNI TEORETICE DESPRE RECURSIVITATE

1.1 Definire noțiunea de recursivitate

Recursivitatea apare atunci când un lucru este definit de el însuși sau de lucruri asemenea lui. Recursivitatea este folosită într-o varietate de domenii și discipline, de la cea lingvistică la cea logică. Cele mai comune aplicații ale recursivității sunt în matematică și informatică, acestea funcționând prin definirea unor reguli prin care cazurile mai complexe se reduc la cazuri mai simple, definite de aceeași funcție. În timp ce acest lucru aparent definește un număr infinit de instanțe, creând la prima vedere un cerc vicios, este adesea făcut în așa fel încât să nu aibă loc nicio buclă infinită sau lanț infinit de referințe.

1.2 Paralela cu matematica

În teoria mulțimilor, teorema recursivității garantează existența funcțiilor definite recursiv. Având în vedere o mulțime *X*, un element *a* al lui *X* și o funcție *f: X -> X*, teorema afirmă că există o funcție unică *F: N -> X*, astfel încât:

F(0) = a

F(n+1) = f(F(n))

pentru orice număr natural *n*.

Cel mai simplu exemplu de recursivitate este în definirea formală a numerelor naturale:

F(0) = 0, fiind primul element din *N*. Dacă *n* face parte din *N*, atunci și *n+1* este din *N*.

O funcție poate fi definită recursiv în termenii ei înșiși. Un exemplu celebru este șirul de numere Fibonacci: *F(n) = F(n-1) + F(n-2)*. Pentru ca o astfel de definire să fie utilă, trebuie să fie reductibilă la valori definite nerecursiv; în acest caz *F(0) = 0* și *F(1) = 1*. O altă funcție celebră este funcția Ackermann, care, spre deosebire de șirul lui Fibonacci, nu poate fi exprimată altfel decât recursiv.

1.3 Implementare

Un exemplu simplu este calculul produsului factorial al unui număr. Acest produs poate fi calculat atât nerecursit (iterativ) (fotografia 1), cât și recursiv (fotografia 2). Ambele variante returnează rezultatul corect.

```
int profFactItr(int n)
{
    int p = 1;
    for(int i = 1; i <= n; i++)
        p = p * i;
    return p;
}
```

Varianta iterativa (fotografia 1)

```
int prodFactRec(int n)
{
    if(n == 1)
        return 1;
    else
        return n * prodFactRec(n - 1);
}
```

Varianta recursiva (fotografia 2)

1.4 Tipuri de recursivitate

Recursivitatea se clasifică în 2 tipuri: cea directă și cea indirectă. O funcție este numită ca fiind recursivă directă dacă ea se autoapelează (fotografia 3). Recursivitatea indirectă apare când în definiția funcției *F* apare apelul funcției *G*, iar în definiția funcției *G* apare apelul funcției *F* (fotografia 4).

```
void directRecFun()
{
    // cod....

    directRecFun();

    // cod...
}
```

Rec ursivitate directa (fotografia 3)

```
void indirectRecFun1()
{
    // cod...

    indirectRecFun2();

    // cod...
}
void indirectRecFun2()
{
    // cod...

    indirectRecFun1();

    // cod...
}
```

Recursivitate indirecta (fotografia 4)

APLICAȚII ALE RECURSIVITĂȚII

2.1 Fractali

Cuvântul “fractal” are adesea conotații diferite pentru publicul larg, spre deosebire de matematicieni, unde publicul este mai probabil să fie familiarizat cu arta fractală, decât cu conceptul matematic. Conceptul matematic este dificil de definit formal, chiar și pentru matematicieni, dar caracteristicile cheie pot fi înțelese cu puțin fundal matematic.

Caracteristica “auto-asemănării”, de exemplu, este ușor de înțeles prin analogie cu mărirea cu un obiectiv sau cu alt dispozitiv care mărește imaginile digitale, pentru a descoperi o nouă structură mai fină, invizibilă anterior. Dacă acest lucru se face pe fractali, totuși, nu apare niciun detaliu nou; nimic nu se schimbă, iar modelul se repetă iar și iar, sau pentru unii fractali, aproape același model reapare iar și iar.

Istoria fractalilor urmărește o cale de la studii, în principal teoretice, la aplicații moderne în grafica computerizată, cu mai mulți oameni de seamă contribuind cu forme fractale canonice pe parcurs. O temă comună în arhitectura tradițională africană este utilizarea scalării fractale, prin care părțile mici ale structurii tind să semene cu părțile mari, cum ar fi un sat circular, format din case circulare. Potrivit lui Pickover, matematica din spatele fractalilor a început să prindă contur în secolul al XVII-lea, când matematicianul și filozoful Gottfried Leibniz s-a gândit la auto-asemănarea recursivă.

2.2 Grafica în C++

Biblioteca “graphics.h” este folosită datorită faptului că include și facilitează operațiile grafice în program. Funcțiile din biblioteca “graphics.h” pot să deseneze diferite forme, să afișeze text în diferite fonturi, să schimbe culori și multe altele. Utilizând funcțiile bibliotecii “graphics.h” se pot realiza programe grafice, animații, proiecte și jocuri.

Presupunând că se lucrează în modul grafic pe un ecran cu *lațime*înalțime* pixeli, coordonatele colțului stânga sus sunt (0, 0), iar ale colțului dreapta-jos sunt (*lățime*-1, *înalțime*-1). În acest mod grafic, dispuneți de mai multe primitive grafice, cum ar fi: o funcție grafică *void setcolor(unsigned int c),* care stabilește culoarea pentru a desena

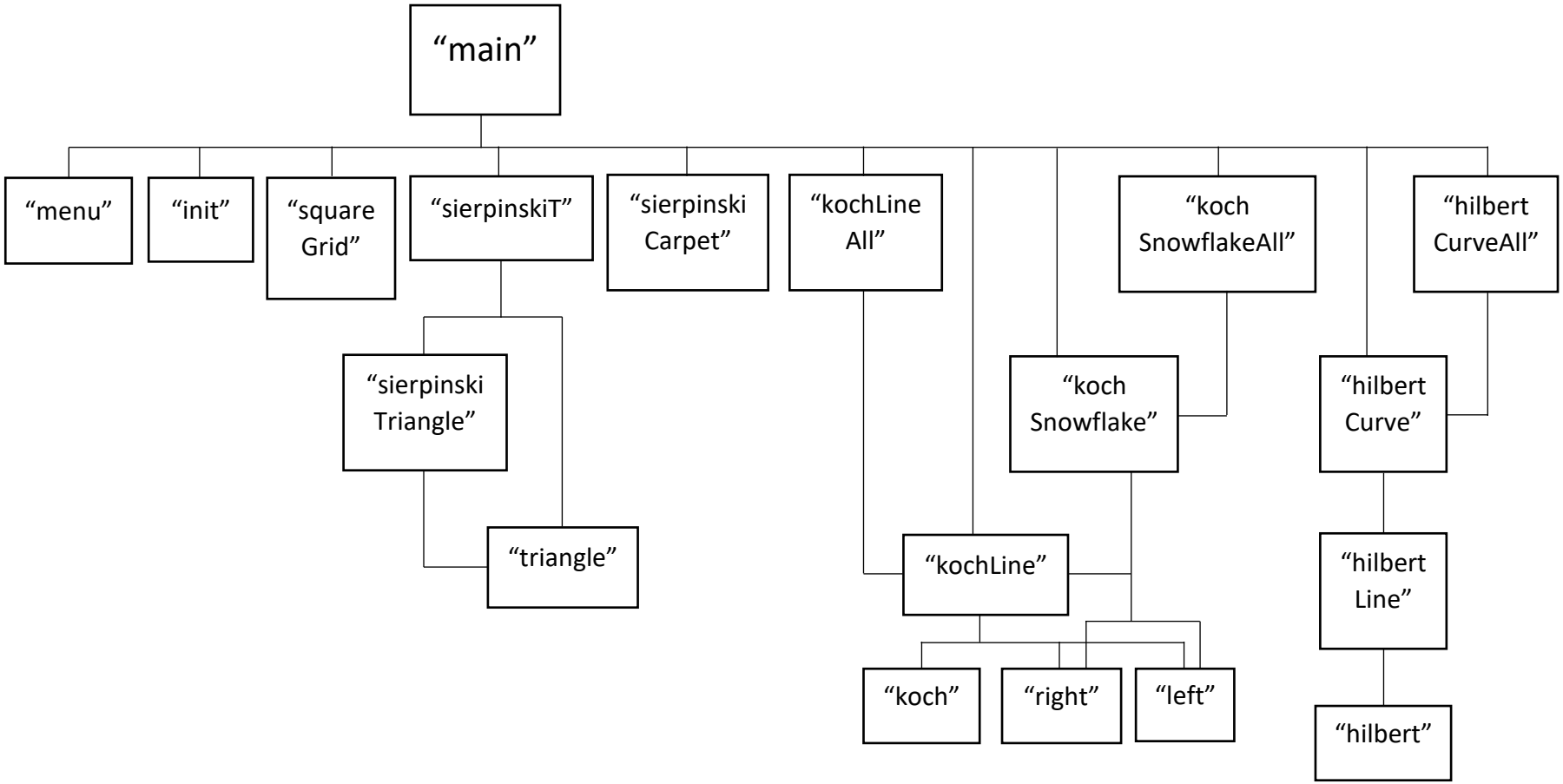
următoarele linii, până la schimbarea culorii de un alt apel al acestei funcții (inițial se consideră *c*=0, implicit); *c* este un cod de culoare, care corespunde unei valori între 0 și 15, de exemplu 0=negru, 1=albastru, 2=roșu etc; o funcție grafică *void line(unsigned int x1, unsigned int y1, unsingned int x2, unsigned int y2)* care trasează o linie între punctul de coordonate *(x1, y1)* și cel de coordonate *(x2, y2)*, în culoarea curentă, de grosimea curentă.

DESCRIEREA APLICAȚIEI

3.1 Enunț

Utilizatorul va introduce un număr specific fractalului pe care dorește să-l vadă. Programul va executa instrucțiunile și va “desena” fractalul ales, exclusiv prin metode recursive. Programul are și un meniu intuitiv, care va facilita utilizarea acestuia de către orice utilizator.

3.2 Structura programului



3.3 Sursa

3.3.1 Funcția “menu”

Această funcție reprezintă meniul pe care utilizatorul îl va folosi pentru a naviga prin opțiunile pe care programul i le oferă.

```
void menu()
{
    gotoxy(0, 0);
    cout<<"1.  Square Grid\n";
    cout<<"2.  Sierpinski Triangle\n";
    cout<<"3.  Sierpinski Carpet\n";
    cout<<"4.  Koch Line\n";
    cout<<"5.  Koch Snowflake\n";
    cout<<"6.  Hilbert\n";
    cout<<"7.  Koch Line Levels\n";
    cout<<"8.  Koch Snowflake Levels\n";
    cout<<"9.  Hilbert Levels\n";
    cout<<"10. Exit\n";
    cout<<"Your option:";
    cin>>op;
}
```

3.3.2 Funcția “init”

Această funcție deschide o nouă fereastră, unde vor apărea reprezentările fractalilor și oferă câte un titlu fiecărei ferestre, în funcție de opțiunea din meniu aleasă.

```
void init(int fractalType = 0)
{
    const char *titleWindow;

    switch (fractalType)
    {
        case 1: titleWindow = "S Q U A R E     G R I D";                    break;
        case 2: titleWindow = "S I E R P I N S K I     T R I A N G L E";    break;
        case 3: titleWindow = "S I E R P I N S K I     C A R P E T";        break;
        case 4: titleWindow = "K O C H     L I N E";                        break;
        case 5: titleWindow = "K O C H     S N O W F L A K E";              break;
        case 6: titleWindow = "H I L B E R T      C U R V E";               break;
        case 7: titleWindow = "K O C H     L I N E";                        break;
        case 8: titleWindow = "K O C H     S N O W F L A K E";              break;
        case 9: titleWindow = "H I L B E R T      C U R V E";               break;
        default: titleWindow = "F R A C T A L S";
    }

    initwindow(WIDTH, HEIGHT, titleWindow, 400, 100);
    center.x = getmaxx() / 2;
    center.y = getmaxy() / 2;
}
```

3.3.3 Funcția “squareGrid”

Această funcție trasează câte un pătrat, urmând să se autoapeleze pentru fiecare colț al pătratului, formând astfel fractalul de tip “Square Grid”.

```
//Square Grid Fractal
void squareGrid(point p, int L)
{
    point u;
    if (L > 10)
    {
        delay(10);
        srand (time(0));
        setcolor(rand()%15 + 1);

        p.x = p.x - L / 2;
        p.y = p.y - L / 2;
        u.x = p.x + L;
        u.y = p.y + L;
        rectangle(p.x, p.y, u.x, u.y);

        u.x = p.x;      u.y = p.y;      squareGrid(u, L / 2);
        u.x = p.x + L;  u.y = p.y;      squareGrid(u, L / 2);
        u.x = p.x + L;  u.y = p.y + L;  squareGrid(u, L / 2);
        u.x = p.x;      u.y = p.y + L;  squareGrid(u, L / 2);
    }
}
```

3.3.4 Funcția “sierpinskiT”

Această funcție determină coordonatele celor 3 vârfuri are triunghiului mare, urmând să-l deseneze și să apeleze funcția “sierpinskiTriangle”.

```
void sierpinskiT(point p, int L)
{
    int l = sqrt(3) * L / 2;
    point p1, p2, p3;

    p1.x = p.x - L / 2;
    p1.y = p.y + l / 2;
    p2.x = p.x;
    p2.y = p.y - l / 2;
    p3.x = p.x + L / 2;
    p3.y = p1.y;

    triangle(p1, p2, p3);
    sierpinskiTriangle(p1, p2, p3, L);
}
```

3.3.5 Funcția “sierpinskiTriangle”

Această funcție determină mijloacele fiecărei laturi, urmând să deseneze câte un triunghi cu vârfurile în acele puncte. Funcția se autoapelează pentru fiecare triunghi care urmează să fie desenat.

```
void sierpinskiTriangle(point p1, point p2, point p3, int L)
{
    point u1, u2, u3;
    if (L > 10)
    {
        delay(10);
        srand (time(0));
        setcolor(rand()%15 + 1);

        u1.x = (p1.x + p3.x) / 2;
        u1.y = (p1.y + p3.y) / 2;
        u2.x = (p1.x + p2.x) / 2;
        u2.y = (p1.y + p2.y) / 2;
        u3.x = (p3.x + p2.x) / 2;
        u3.y = (p3.y + p2.y) / 2;

        triangle(u1, u2, u3);

        sierpinskiTriangle(p1, u2, u1, L / 2);
        sierpinskiTriangle(u2, p2, u3, L / 2);
        sierpinskiTriangle(u1, u3, p3, L / 2);
    }
}
```

3.3.6 Funcția “triangle”

Biblioteca “graphics.h” nu include o funcție predefinită pentru a desena un triunghi, iar această funcție rezolvă respectiva problemă. Trasează câte o linie între punctele transmise ca parametru.

```
void triangle(point p1, point p2, point p3)
{
    line(p1.x, p1.y, p2.x, p2.y);
    line(p1.x, p1.y, p3.x, p3.y);
    line(p3.x, p3.y, p2.x, p2.y);
}
```

3.3.7 Funcția “sierpinskiCarpet”

Funcția “sierpinskiCarpet” calculează punctele care determină pătratul, pe care ulterior îl desenează și se autoapelează pentru pozițiile de jur-împrejur.

```
void sierpinskiCarpet(point r, int L)
{
    point p, u;
    if (L > 10)
    {
        delay(5);

        p.x = r.x - L / 2;
        p.y = r.y - L / 2;
        u.x = p.x + L;
        u.y = p.y + L;

        setfillstyle(SOLID_FILL, 15);
        bar(p.x, p.y, u.x, u.y);

        u.x = r.x - L; u.y = r.y - L; sierpinskiCarpet(u, L / 3);
        u.x = r.x    ; u.y = r.y - L; sierpinskiCarpet(u, L / 3);
        u.x = r.x + L; u.y = r.y - L; sierpinskiCarpet(u, L / 3);

        u.x = r.x + L; u.y = r.y;     sierpinskiCarpet(u, L / 3);
        u.x = r.x - L; u.y = r.y ;    sierpinskiCarpet(u, L / 3);

        u.x = r.x - L; u.y = r.y + L; sierpinskiCarpet(u, L / 3);
        u.x = r.x    ; u.y = r.y + L; sierpinskiCarpet(u, L / 3);
        u.x = r.x + L; u.y = r.y + L; sierpinskiCarpet(u, L / 3);
    }
}
```

3.3.8 Funcția “kochLineAll”

Această funcție facilitează înțelegerea dezvoltării fractalului de tip linie Koch.

```
void kochLineAll(float L)
{
    for (int i = 0; i < 5; i++)
    {
        cleardevice();
        x = 150; y = 150; alfa = 0;
        kochLine(i, L);
        outtextxy(150, HEIGHT - 50, "Press any key...");
        getch();
    }
}
```

3.3.9 Funcția “kochLine”

Funcția “kochLine” se autoapelează pentru a desena linia de tip Koch.

```
void kochLine(int n, float L)
{
    if (n == 0)
        koch (L);
    else
    {
        kochLine(n - 1, L / 3); right(60);   delay(3);
        kochLine(n - 1, L / 3); left(120);   delay(3);
        kochLine(n - 1, L / 3); right(60);   delay(3);
        kochLine(n - 1, L / 3);              delay(3);
    }
}
```

3.3.10 Funcția “koch”

Această funcție este cea care determină coordonatele punctelor, în funcție de unghiul alfa dintre drepte.

```
void koch(float L)
{
    int x1 = x;
    int y1 = y;
    x += (int) (L * cos(alfa));
    y += (int) (L * sin(alfa));
    line (x1, y1, x, y);
}
```

3.3.11 Funcția “right”

Această funcție incrementează unghiul alfa cu o valoare dată.

```
void right(float angle)
{
    alfa -= angle * PI / 180.;
}
```

3.3.12 Funcția “left”

Această funcție decrementează unghiul alfa cu o valoare dată.

```
void left(float angle)
{
    alfa += angle * PI / 180.;
}
```

3.3.13 Funcția “kochSnowflakeAll”

Această funcție facilitează înțelegerea dezvoltării fractalului de tip fulg de zăpadă Koch.

```
void kochSnowflakeAll(float L)
{
    for (int i = 0; i < 5; i++)
    {
        cleardevice();
        x = 150; y = 150; alfa = 0;
        kochSnowflake(i, L);
        outtextxy(150, HEIGHT - 50, "Press any key...");
        getch();
    }
}
```

3.3.14 Funcția “kochSnowflake”

Această funcție apelează funcția “kochLine” de 3 ori, înclinând linia de tip Koch spre stânga de 2 ori cu 120 grade, formând astfel un fulg de zăpadă.

```
void kochSnowflake(int n, float L)
{
    kochLine(n, L); left(120);
    kochLine(n, L); left(120);
    kochLine(n, L);
}
```

3.3.15 Funcția “hilbertCurveAll”

Această funcție facilitează înțelegerea dezvoltării fractalului de tip Hilbert.

```
void hilbertCurveAll()
{
    for (int i = 1; i < 8; i++)
    {
        cleardevice();
        hilbertCurve(i);
        outtextxy(150, HEIGHT - 20, "Press any key...");
        getch();
    }
}
```

3.3.16 Funcția “hilbertCurve”

Această funcție determină punctul de plecare și lungimile pentru fiecare nivel al fractalului de tip Hilbert, apelând ulterior funcția “hilbertLine”.

```
void hilbertCurve(int n)
{
    float totalLength, startX, startY, startLength;

    if (HEIGHT < WIDTH)
        totalLength = (float)(0.9 * HEIGHT);
    else
        totalLength = (float)(0.9 * WIDTH);

    startX = (WIDTH - totalLength) / 2;
    startY = (HEIGHT - totalLength) / 2;
    startLength = (float)(totalLength / ((1<<n) - 1));

    x = (int)startX;
    y = (int)startY;
    hilbertLine(n, 0, (int)startLength);
}
```

3.3.17 Funcția “hilbertLine”

Funcția “hilbertLine” trasează liniile pentru fiecare nivel al fractalului Hilbert, autoapelându-se și apelând funcția “hilbert”.

```
void hilbertLine(int n, float dx, float dy)
{
    if (n > 1)
        hilbertLine(n - 1, dy, dx);

    hilbert(dx, dy);

    if (n > 1)
        hilbertLine(n - 1, dx, dy);

    hilbert(dy, dx);

    if (n > 1)
        hilbertLine(n - 1, dx, dy);

    hilbert(-dx, -dy);

    if (n > 1)
        hilbertLine(n - 1, -dy, -dx);
}
```

3.3.18 Funcția “hilbert”

Această funcție determină poziția noului punct, unde trebuie trasată drepta pentru a desena fractalul Hilbert, ulterior trasând-o.

```
void hilbert(float dx, float dy)
{
    int x1 = x + dx;
    int y1 = y + dy;

    line (x, y, x1, y1);
    x = x1;
    y = y1;
}
```

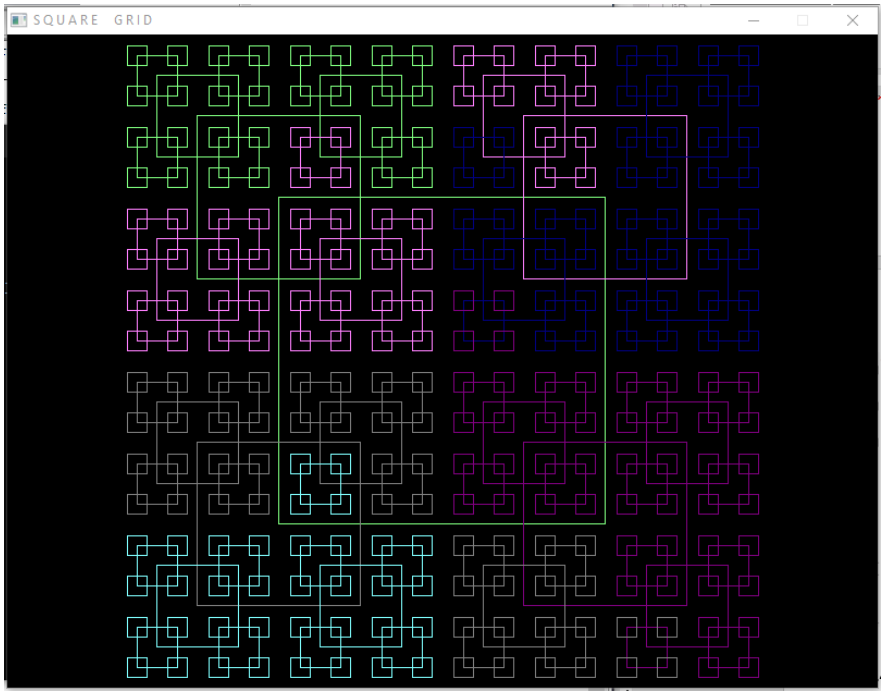
3.3.19 Funcția “main”

Funcția “main” apelează funcțiile necesare în funcție de alegerea făcută în meniu.

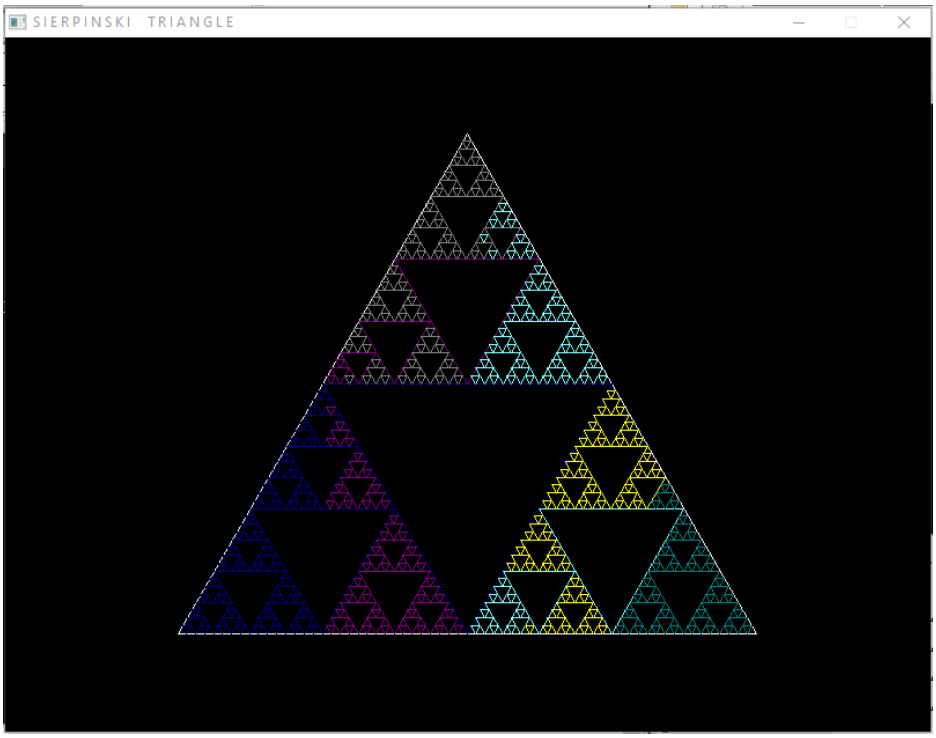
```
int main()
{
    SetConsoleTitle("F R A C T A L S");

    do{
        menu();
        switch (op)
        {
            case 1: init(1); squareGrid(center, 300);                    close(); break;
            case 2: init(2); sierpinskiT(center, LENGTH);                close(); break;
            case 3: init(3); sierpinskiCarpet(center, 200);              close(); break;
            case 4: init(4); x = 150; y = 150; kochLine(3, LENGTH);      close(); break;
            case 5: init(5); x = 150; y = 150; kochSnowflake(3, LENGTH);close(); break;
            case 6: init(6); hilbertCurve(4);                            close(); break;
            case 7: init(4); x = 150; y = 150; kochLineAll(LENGTH);      close(); break;
            case 8: init(5); x = 150; y = 150; kochSnowflakeAll(LENGTH);close(); break;
            case 9: init(6); hilbertCurveAll();                          close(); break;
        }
    } while (op != 10);

    return 0;
}
```

3.4 Rezultate

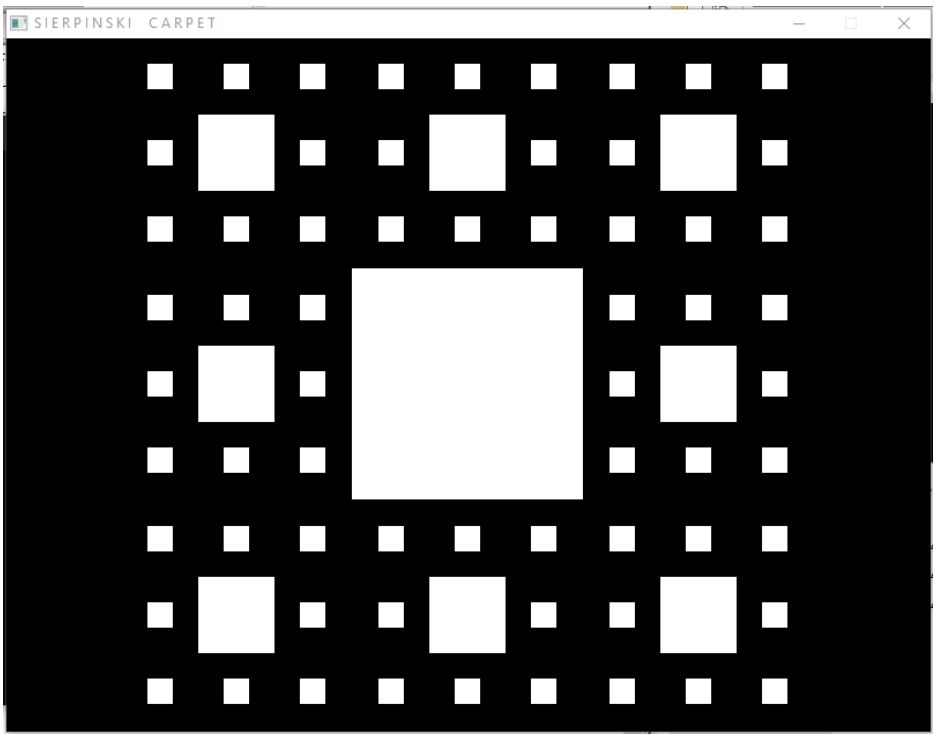
3.4.1 Fractalul Square Grid



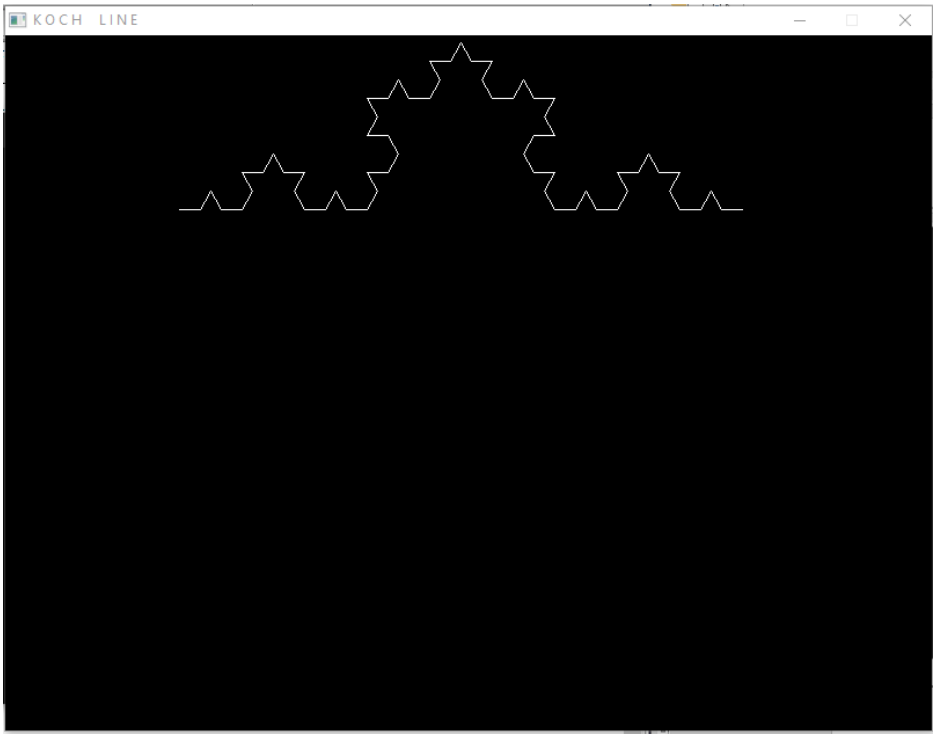
3.4.2 Triunghiul Sierpinski



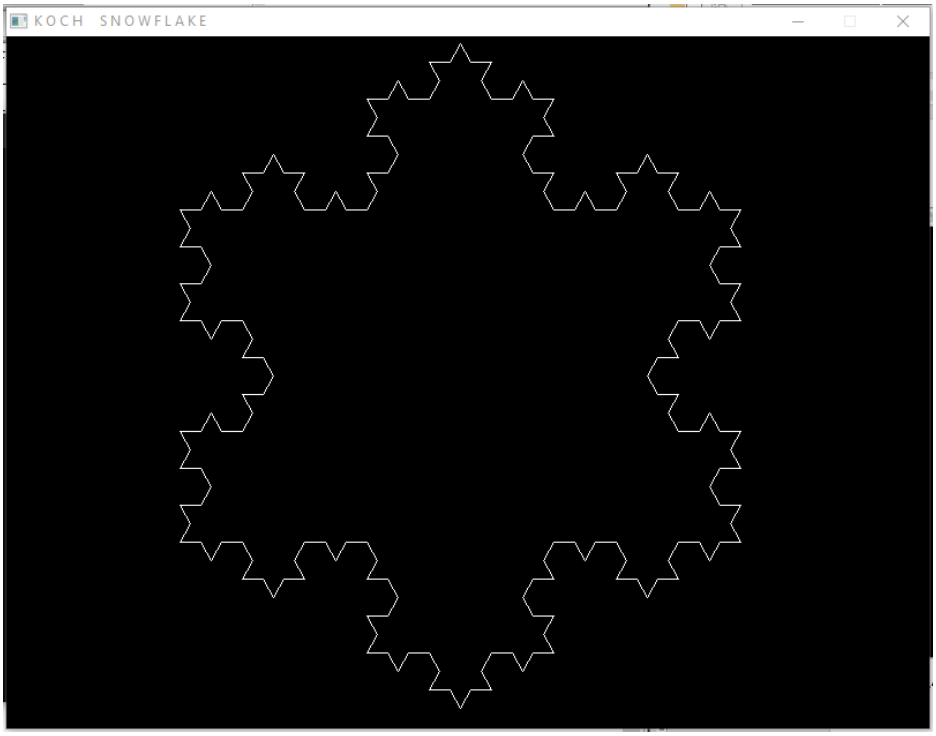
3.4.3 “Covorul” Sierpinski



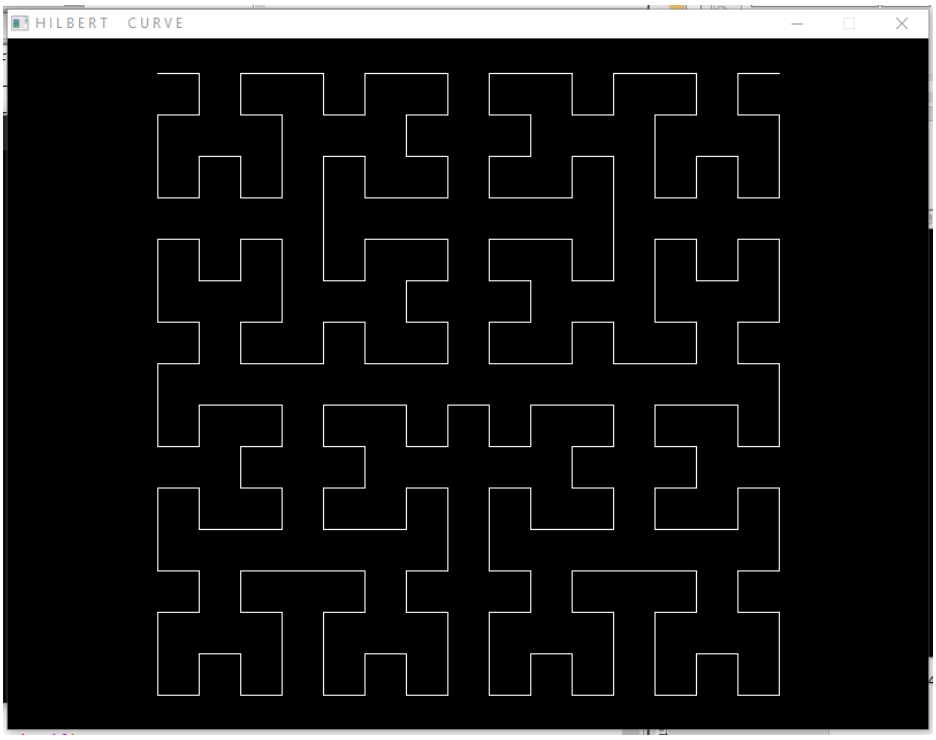
3.4.4 Linia Koch



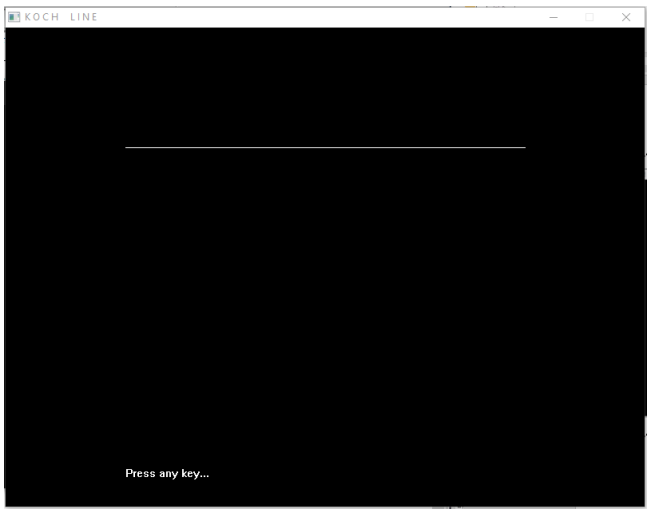
3.4.5 Fulgul de zăpadă Koch



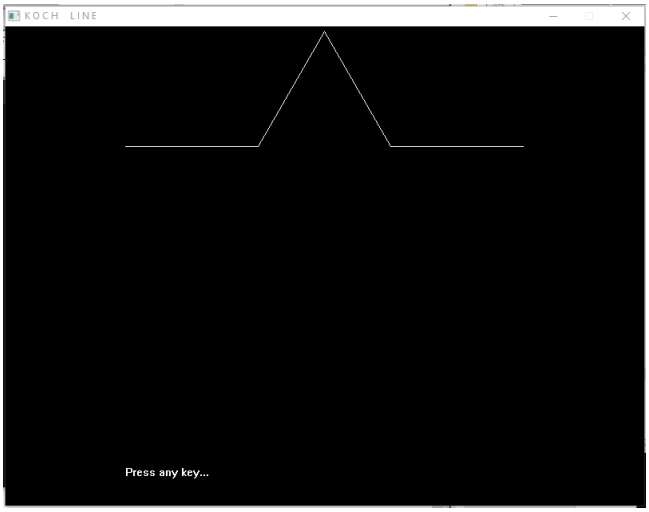
3.4.6 Fractalul Hilbert



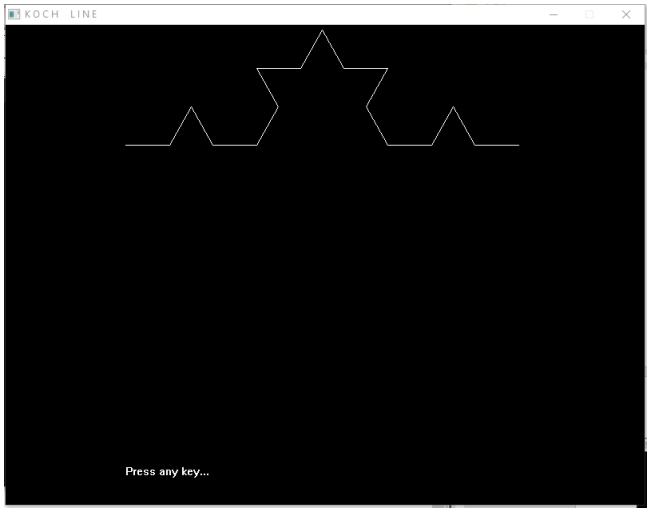
3.4.7 Detalierea liniei Koch



fotografia 1



fotografia 2

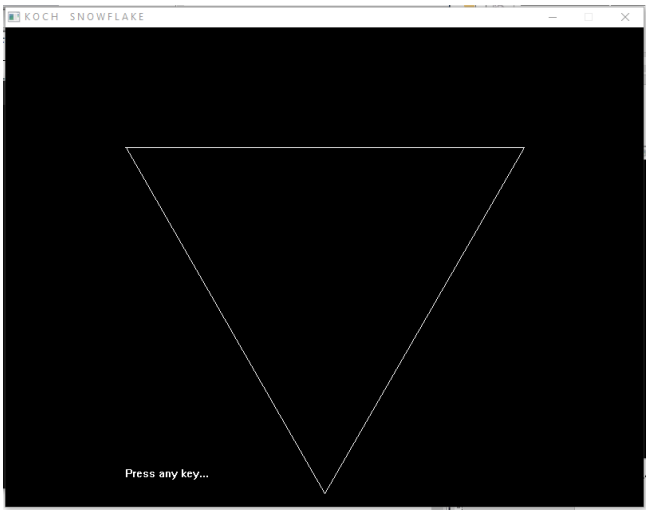


fotografia 3

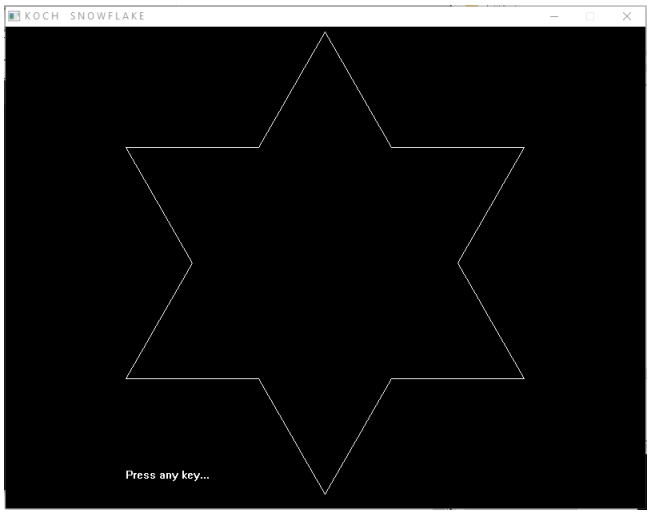
fotografia 4

fotografia 5

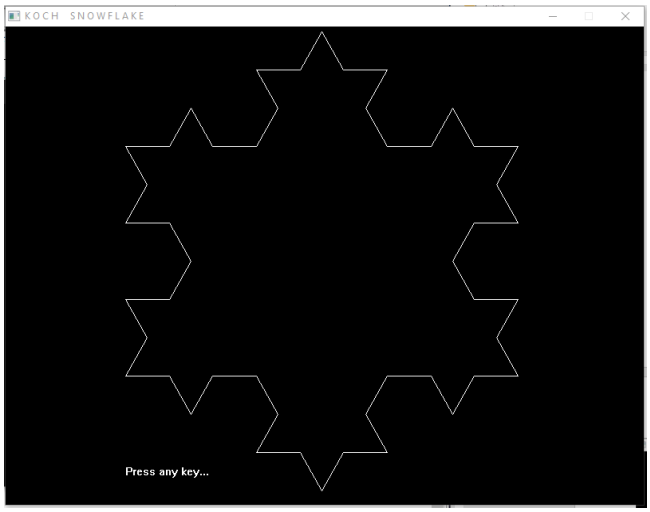
3.4.8 Detalierea fulgului de zăpadă Koch



fotografia 1

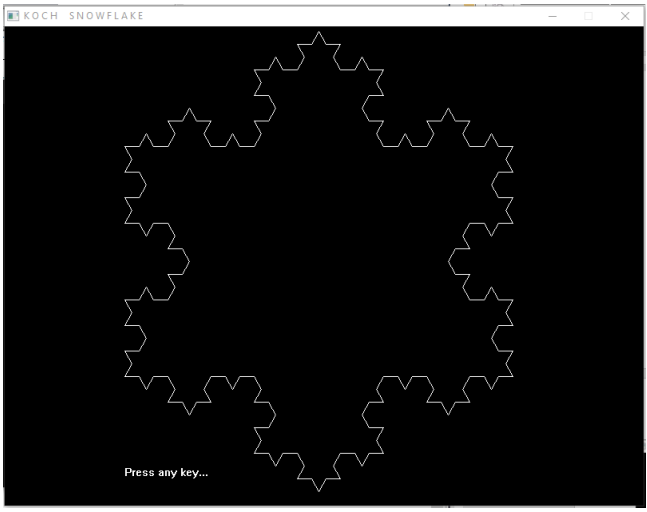


fotografia 2

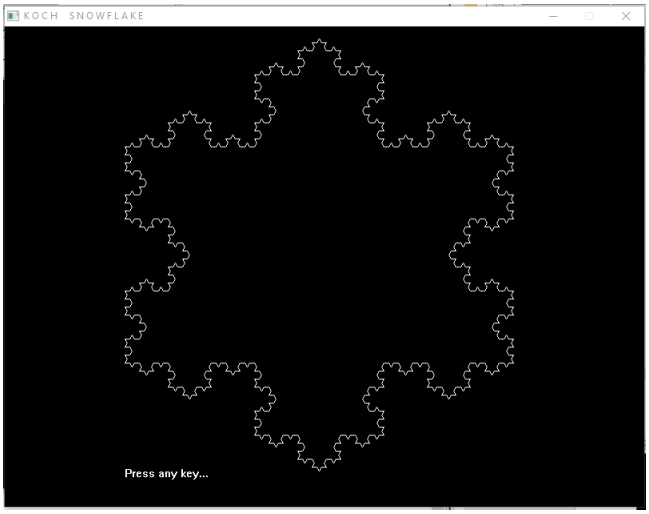


ografia 4

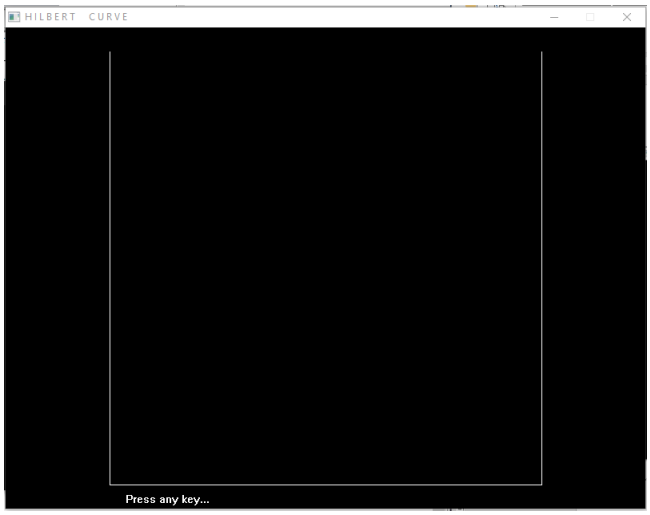
fot ografia 3

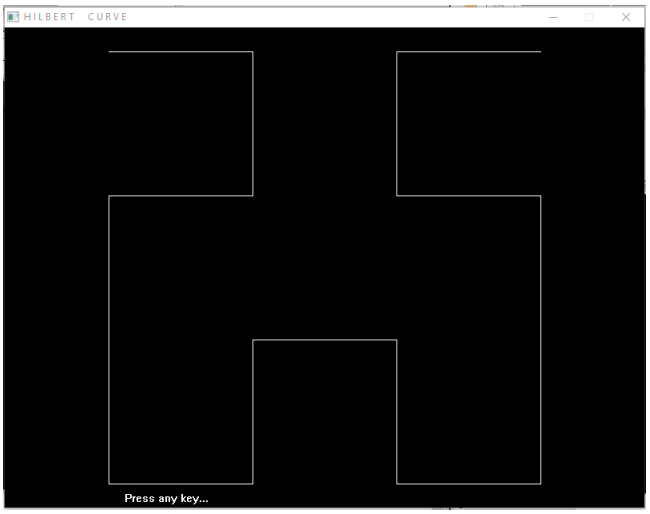


fot

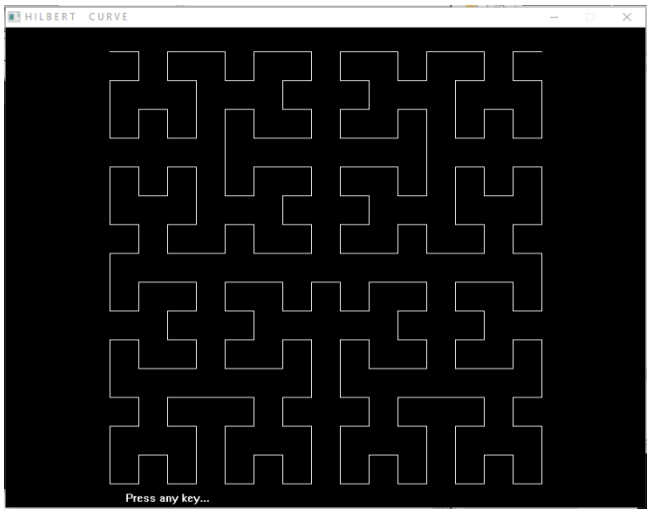


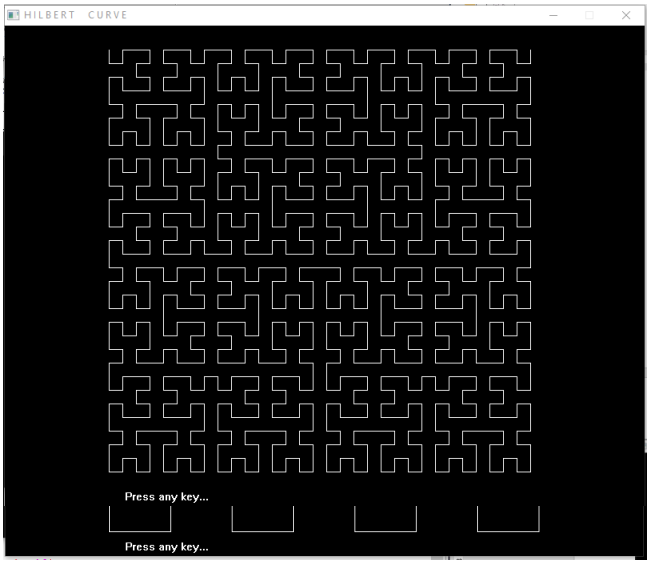
fotografia 5

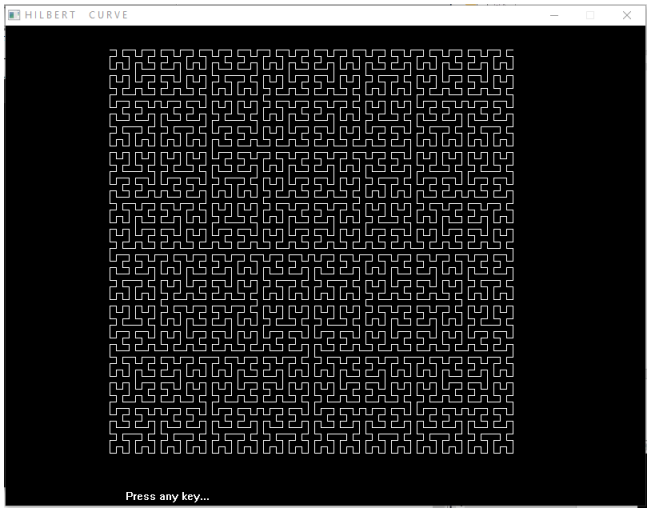




3.4.9 Detalierea fractalului Hilbert







WEBOGRAFIE

https://en.wikipedia.org/wiki/Recursion

https://en.wikipedia.org/wiki/Fractal

https://www.geeksforgeeks.org/recursion/

https://www.geeksforgeeks.org/fractals-in-cc/

https://www.youtube.com/watch?v=IJDJ0kBx2LM

https://www.w3schools.com

https://programarecurabdare.ro/2018/11/29/grafica-in-codeblocks/

CUPRINS