**Aim:**  To implement Character Generation.

**Objective:**

Identify the different Methods for Character Generation and generate the character using Stroke

**Theory:**

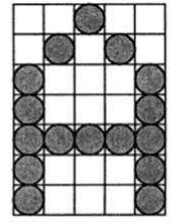
**Bit map method –**

Bitmap method is a called dot-matrix method as the name suggests this method use array of bits for generating a character. These dots are the points for array whose size is fixed.

∙ In bit matrix method when the dots are stored in the form of array the value 1 in array represent the characters i.e. where the dots appear we represent that position with numerical value 1 and the value where dots are not present is represented by 0 in array.

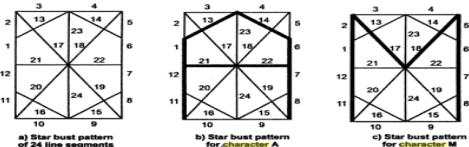
∙ It is also called dot matrix because in this method characters are represented by an array of dots in the matrix form. It is a two-dimensional array having columns and rows.

A 5x7 array is commonly used to represent characters. However, 7x9 and 9x13 arrays are also used. Higher resolution devices such as inkjet printer or laser printer may use character arrays that are over 100x100.



**Starburst method –**

In this method a fix pattern of line segments is used to generate characters. Out of these 24-line segments, segments required to display for particular character are highlighted. This method of character generation is called starburst method because of its characteristic appearance. The starburst patterns for characters A and M. the patterns for particular characters are stored in the form of 24 bit code, each bit representing one line segment. The bit is set to one to highlight the line segment; otherwise, it is set to zero. For example, 24-bit code for Character A is 0011 0000 0011 1100 1110 0001 and for character M is 0000 0011 0000 1100 1111 0011.



**Program:**

**#include <stdio.h>**

**#include <conio.h>**

**#include <graphics.h>**

**void main() {**

**int gd = DETECT, gm;**

**int i, j, k;**

**int a [10][10] = {**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1}**

**};**

**int b [10][10] = {**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 1, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 1, 1, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 1, 1, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 1, 1, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 1, 1, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 1, 1, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 1, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 1, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1}**

**};**

**int c [10][10] = {**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 0, 0},**

**{1, 1, 0, 0, 0, 0, 0, 0, 0, 0},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{0, 0, 0, 0, 0, 0, 0, 0, 1, 1},**

**{0, 0, 0, 0, 0, 0, 0, 0, 0, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1}**

**};**

**int d [10][10] = {**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1}**

**};**

**int e [10][10] = {**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1}**

**};**

**int f [10][10] = {**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 1, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 1, 0, 0, 0, 0, 1, 1, 1},**

**{1, 1, 0, 1, 0, 0, 1, 1, 1, 1},**

**{1, 1, 0, 1, 1, 1, 1, 0, 1, 1},**

**{1, 1, 0, 0, 1, 1, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1}**

**};**

**int g [10][10] = {**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1}**

**};**

**int h [10][10] = {**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 1, 0, 0, 0, 0, 0, 1, 1},**

**{1, 1, 1, 1, 0, 0, 0, 0, 1, 1},**

**{1, 1, 0, 1, 1, 0, 0, 0, 1, 1},**

**{1, 1, 0, 0, 1, 1, 0, 0, 1, 1},**

**{1, 1, 0, 0, 0, 1, 1, 0, 1, 1},**

**{1, 1, 0, 0, 0, 0, 1, 1, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 1, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 1, 1, 1},**

**{1, 1, 0, 0, 0, 0, 0, 0, 1, 1}**

**};**

**initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");**

**for (k = 0; k < 8; k++) {**

**for (i = 0; i < 10; i++) {**

**for (j = 0; j < 10; j++) {**

**if (a[i][j]==1) {**

**putpixel(200+j, 200+i, 15);**

**delay(200);**

**}**

**}**

**}**

**for (i = 0; i < 10; i++) {**

**for (j = 0; j < 10; j++) {**

**if (b[i][j]==1) {**

**putpixel(220+j, 200+i, 15);**

**delay(200);**

**}**

**}**

**}**

**for (i = 0; i < 10; i++) {**

**for (j = 0; j < 10; j++) {**

**if (c[i][j]==1) {**

**putpixel(240+j, 200+i, 15);**

**delay(200);**

**}**

**}**

**}**

**for (i = 0; i < 10; i++) {**

**for (j = 0; j < 10; j++) {**

**if (d[i][j]==1) {**

**putpixel(260+j, 200+i, 15);**

**delay(200);**

**}**

**}**

**for (i = 0; i < 10; i++) {**

**for (j = 0; j < 10; j++) {**

**if (e[i][j]==1) {**

**putpixel(280+j, 200+i, 15);**

**delay(200);**

**}**

**}**

**}**

**for (i = 0; i < 10; i++) {**

**for (j = 0; j < 10; j++) {**

**if (f[i][j]==1) {**

**putpixel(300+j, 200+i, 15);**

**delay(200);**

**}**

**}**

**}**

**for (i = 0; i < 10; i++) {**

**for (j = 0; j < 10; j++) {**

**if (g[i][j]==1) {**

**putpixel(320+j, 200+i, 15);**

**delay(200);**

**}**

**}**

**}**

**for (i = 0; i < 10; i++) {**

**for (j = 0; j < 10; j++) {**

**if (h[i][j]==1) {**

**putpixel(340+j, 200+i, 15);**

**delay(200);**

**}**

**}**

**}**

**}**

**}**

**getch();**

**closegraph();**

**}**

**Output -**



**Conclusion: Comment on**

**different methods :-**

●Bitmap Method: In this method, characters are represented as collections of pixels (dots). Each character is defined by a grid of pixels, which can be easily displayed on a screen. This method is simple and efficient but can result in pixelation when scaled.

●Vector Method: Characters are represented as mathematical curves, lines, and shapes in this method. Vector graphics allow for smooth scaling and high-quality printing. Fonts based on vector graphics are commonly used in design and printing.

●Stroke Method: The Stroke Method is a subset of vector graphics. In this method, characters are defined by a series of strokes or pen movements. Characters are constructed by sequentially drawing strokes, and these strokes can be stored as a series of points or curves. It's often used in handwriting recognition and digital calligraphy.

●Outline Method: Similar to the Stroke Method, characters are represented as outlines consisting of curves and straight lines. These outlines can be filled to create solid characters. Outline fonts are versatile and can be scaled without losing quality.

**advantage of stroke method** : - Natural Appearance: The Stroke Method can produce characters that closely resemble handwritten or calligraphic text. This natural appearance is particularly useful for artistic or creative applications.

Pen Pressure Sensitivity: When using digital tools like stylus pens and graphics tablets, the Stroke Method can capture pen pressure sensitivity. This means that characters can vary in thickness and opacity, creating a more realistic and expressive look.

Artistic Freedom: Stroke-based character generation provides artists with significant creative freedom. Artists can experiment with different strokes, styles, and flourishes to create unique characters and fonts.

**one limitation** :-Complexity :- Creating characters with the Stroke Method can be more complex than other methods, especially for intricate or highly detailed characters. Artists must define each stroke, which can be time-consuming.