Image Processing in C

Group –6 Members:

- Arnab Nath(2021CSB054)
- Sk Fardeen Hossain(2021CSB023)
- Avishek Rana(2021CSB056)
- Ankita Das(2021CSB055)
- Sujit Pradipkumar Jaiswal(2021CSB086)

PROJECT ADVISOR:

PROF. DR. SEKHAR MANDAL

Index

Introduction to Image Processing

- Image processing in general refers to the manipulation of digital/analog images through some algorithms. In our course of work we are working with digital images, thus doing digital image processing which has advantages over analog image processing in terms of wide range of algorithms and the ability to process out the noise and distortions during the build-up of the image.
- Images are formatted in different file formats like TIFF, BMP, PGM, PNG, JPG, etc. In our course work we will be studying and dealing with PGM images.
- Image processing requires prerequisite knowledge regarding handling with the given file format in the given language for development and basic programming skills.
- Our development language will be C, but one can use any other language to do image processing.
- At the end of the course work we will be able to appreciate the need for image processing.

Concept of Gray-Scale Images

- Gray-Scale images are those image where the value of each pixel is a single sample representing the amount of intensity of light it contains, thereby giving various shades of gray. The contrast ranges from black at the weakest intensity to white at the strongest.
- The minimum intensity value of each pixel is 0(black) and the maximum intensity value depends on the number of bits each pixel is composed of(if it is n bits the maximum value is 2^n-1). For our course work we will be dealing with 8 bit images so the maximum value of each pixel is 255(100% white). Thus we are expressing information in 256 intensity levels for each pixels.
- The concept of gray-scale images can be expanded to get coloured images in RGB format where each pixel contains information regarding the intensity value in Channel-Red, Channel-Green and Channel-Blue.
- Let us now delve deep into PGM file formatting and handling PGM files in C.

Handling with PGM Files

- **PGM** stands for **Portable Gray Map**. Saving a 2D array in C as images in PNG, JPG or other formats would need a lot of effort to encode the data in the specified format before writing to a file.
- Each file starts with a two-byte magic number (in ASCII) that identifies the type of file it is (PBM, PGM, and PPM) and its encoding (ASCII or binary). The magic number is a capital P followed by a single-digit number.
- The ASCII formats allow for human readability and easy transfer to other platforms; the binary formats are more efficient in file size but may have native byte-order issues. In the binary formats, PGM uses 8 bits per pixel.
- We will be using GIMP as our image viewer for PGM files.

Type	Magic		Extension	Colors
	ASCII	Binary		
Portable BitMap	P1	P4	.pbm	0-1(white and black)
Portable GrayMap	P2	P5	.pgm	0-255(grayscale)
Portable PixMap	P3	P6	.ppm	0-255(RGB)

Contents in PGM File

The File format is as follows:

1. Magic Number "**P2**"
2. Whitespace (blanks, TABs, CRs, LFs).
3. A **width**, formatted as ASCII characters in decimal.
4. Whitespace.
5. A **height**, again in ASCII decimal.
6. Whitespace.
7. The maximum gray value, again in ASCII decimal.
8. Whitespace.
9. **width** X **height** gray values, each in ASCII decimal, between 0 and the specified maximum value, separated by whitespace, in raster format, from top to bottom.

Implementing a read and write subroutine in C

1. Structure for handling the PGM file

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

typedef struct PGM
{
  char type[3];
  unsigned char **data;
  unsigned int width;
  unsigned int height;
  unsigned int grayvalue;
} PGM;
```

Storing the type of file("Px\n").

2D array storing the intensity value of each pixel.

Storing width of the file in pixels.

Storing height of the file in pixels.

Storing the maximum grayscale value of all the pixels in the file.

2. Function for ignoring comments in the file

```c
void ignoreComments(FILE *fp)
{
  int ch;
  char line[100];
  // Ignore any blank lines
  while ((ch = fgetc(fp)) != EOF && isspace(ch))
    ;
  // Recursively ignore comments in a PGM image
  // commented lines start with a '#'
  if (ch == '#')
  {
    fgets(line, sizeof(line), fp);
    // To get cursor to next line
    ignoreComments(fp);
  }
  // check if anymore comments available
  else
  {
    fseek(fp, -1, SEEK_CUR);
  }
  // Beginning of current line
}
```

3. Function for reading the PGM file

```c
int openPGM(PGM *pgm, char fname[])
{
  FILE *pgmfile = fopen(fname, "rb");
  if (pgmfile == NULL)
  {
    printf("File does not exist\n");
    return 0;
  }

  ignoreComments(pgmfile);
  fscanf(pgmfile, "%s", pgm->type);
  ignoreComments(pgmfile);

  fscanf(pgmfile, "%d %d", &(pgm->width),
         &(pgm->height));
  ignoreComments(pgmfile);

  fscanf(pgmfile, "%d", &(pgm->grayvalue));
  ignoreComments(pgmfile);

  pgm->data = malloc(pgm->height * sizeof(unsigned char *));

  for (int i = 0; i < pgm->height; i++)
  {
    pgm->data[i] = malloc(pgm->width *
                          sizeof(unsigned char));
    fread(pgm->data[i], pgm->width * sizeof(unsigned char), 1, pgmfile);
  }

  fclose(pgmfile);
  return 1;
}
```

4. Function for writing contents into PGM File

```
void saveImage(PGM *pgm, char fname[])
{
  FILE *fp = fopen(fname, "wb");

  fprintf(fp, "%s\n", pgm->type);
  fprintf(fp, "%d %d\n", pgm->width, pgm->height);
  fprintf(fp, "%d\n", pgm->grayvalue);

  for (int i = 0; i < pgm->height; i++)
  {
    for (int j = 0; j < pgm->width; j++)
    {
      fprintf(fp, "%c", pgm->data[i][j]);
    }
  }

  fclose(fp);
}
```

```
void printImageDetails(PGM *pgm, char filename[])
{
  FILE *pgmfile = fopen(filename, "rb");

  // searches the occurrence of '.'
  char *ext = strrchr(filename, '.');

  if (!ext)
    printf("No extension found in file %s", filename);
  else // portion after .
    printf("File format     : %s\n", ext + 1);

  printf("PGM File type   : %s\n", pgm->type);
  printf("Width of img    : %d px\n", pgm->width);
  printf("Height of img   : %d px\n", pgm->height);
  printf("Max Gray value  : %d\n", pgm->grayvalue);

  fclose(pgmfile);
}
```

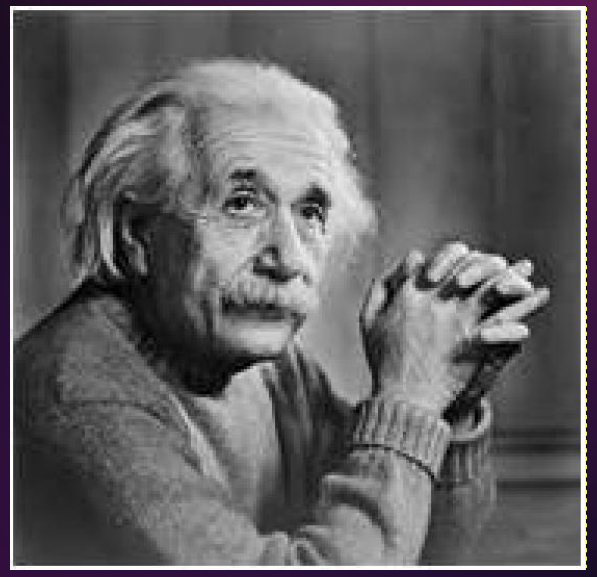
5. Printing Image Details of the given File.

6. Main Function

```
int main()
{
  PGM *pgm = malloc(sizeof(PGM));
  if (openPGM(pgm, "image01.pgm"))
  {
    printImageDetails(pgm, "img.pgm");
    saveImage(pgm, "image.pgm");
  }
  free(pgm->data);
  free(pgm);
  return 0;
}
```

Testcase 01:

Input File



Output File

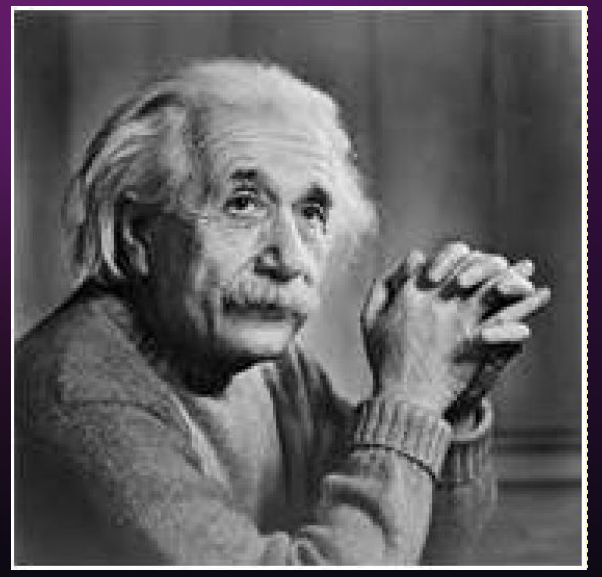


Image Details:

```
File format     : pgm
PGM File type   : P5
Width of img    : 186 px
Height of img   : 182 px
Max Gray value  : 255
```

Testcase 02:

Input File

Output File

Image Details:

```
File format     : pgm
PGM File type   : P5
Width of img    : 766 px
Height of img   : 511 px
Max Gray value  : 255
```

Testcase 03:

Input File

Output File

Image Details:

```
File format     : pgm
PGM File type   : P5
Width of img    : 275 px
Height of img   : 183 px
Max Gray value  : 255
```

Concept of kernels and filters in image processing

- In image processing, a **kernel**, **convolution matrix**, or **mask** is a small matrix used for blurring, sharpening, embossing, edge detection, and more. This is accomplished by doing a convolution between the kernel and an image.
- Kernels are widely used in image-processing works while doing filtering, object-detection, smoothing, sharpening, morphological operations like dilation, erosion, opening, closing and also helps in OCR related works.

- Any pixel in the output image which would require values from beyond the edge is skipped and the convolution is done only for the existing neighbouring pixels for the current pixel.
- We have implemented the following brute force algorithm for filtering without any optimisation:

```
for each image row in input image:
    for each pixel in image row:

        set accumulator to zero

        for each kernel row in kernel:
            for each element in kernel row:

                if element position  corresponding* to pixel position then
                    multiply element value  corresponding* to pixel value
                    add result to accumulator
                endif

         set output image pixel to accumulator
```

- In the course-work, we have implemented the average kernel and median kernel filter for various types of images.

Developing an Average Filter Kernel in C

- We have maintained the same read and write subroutines as mentioned in the previous code snippets, and have added a new function named filter() which accepts the PGM structure pointer and the size of filter window; and also tweaked the main function to some extent.

1. Main Function

```c
int main()
{
  PGM *pgm = malloc(sizeof(PGM));
  printf("Enter the PGM file name with extension\n");
  char ch[30];
  scanf("%s", ch);
  if (openPGM(pgm, ch))
  {
    printImageDetails(pgm, ch);
    int choice;
    printf("Enter the size of the filter window\n");
    scanf("%d", &choice);
    if (choice && 1)
    {
      PGM *filtered = filter(pgm, choice);
      saveImage(filtered, "filtered.pgm");
      printf("The image file has been filtered\n");
      free(filtered->data);
      free(filtered);
    }
    else
    {
      printf("Wrong size for the filter window\n");
      exit(EXIT_FAILURE);
    }
  }
  free(pgm->data);
  free(pgm);
  return 0;
}
```

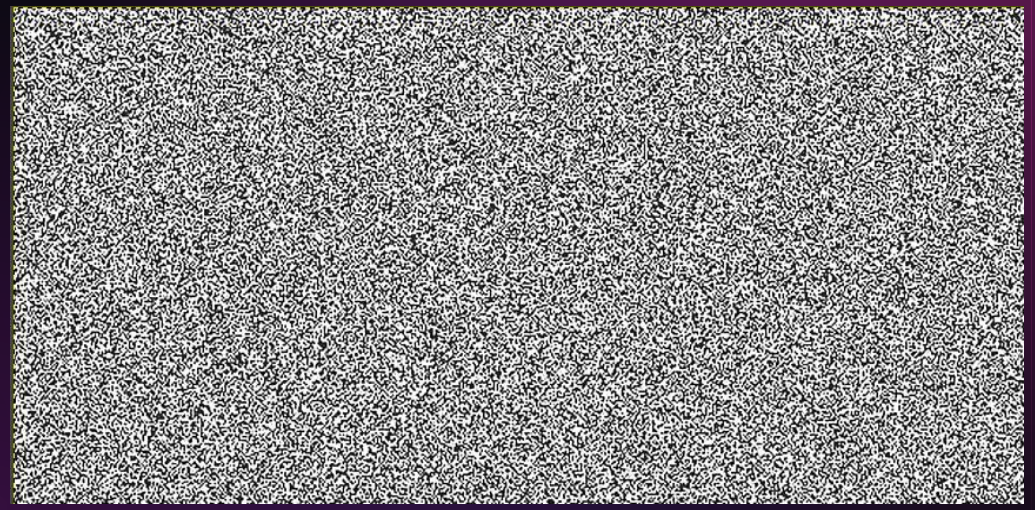
Filter Function

```
PGM *filter(PGM *pgm, int filt_size)
{
  PGM *pgm1 = malloc(sizeof(PGM));
  int s = filt_size / 2;

  strcpy(pgm1->type, pgm->type);
  pgm1->height = pgm->height;
  pgm1->width = pgm->width;
  pgm1->grayvalue = pgm->grayvalue;
  pgm1->data = malloc(pgm->height * sizeof(unsigned char *));
  for (int i = 0; i < pgm->height; i++)
  {
    pgm1->data[i] = malloc(pgm->width *
                           sizeof(unsigned char));
  }
  for (int i = 0; i < pgm->height; i++)
  {
    for (int j = 0; j < pgm->width; j++)
    {
      int d = 0;
      int e = 0;
      for (int k = -s; k <= s; k++)
      {
        for (int l = -s; l <= s; l++)
        {
          if (i + k > 0 && j + l > 0 && i + k < pgm->height && j + l < pgm->width)
          {
            d++;
            e = e + (int)pgm->data[i + k][j + l];
          }
        }
      }
      e = e / d;
      pgm1->data[i][j] = (char)e;
    }
  }
  return pgm1;
}
```

Testcase 01:

Input File



Output File

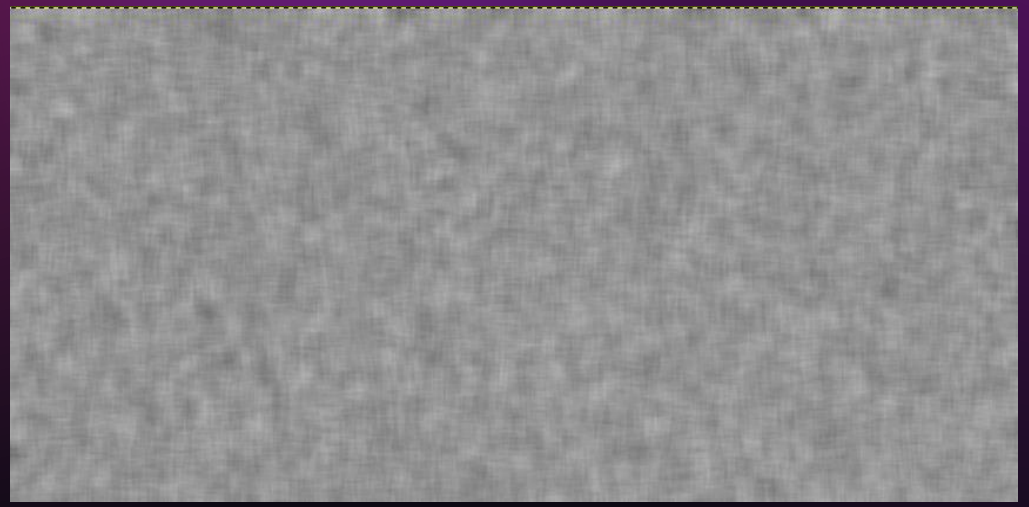


Image Details:

```
Enter the PGM file name with extension
noise.pgm
File format      : pgm
PGM File type    : P5
Width of img     : 862 px
Height of img    : 404 px
Max Gray value   : 255
Enter the size of the filter window
15
The image file has been filtered
```

Testcase 02:

Input File

Output File

Image Details:

```
Enter the PGM file name with extension
noise.pgm
File format      : pgm
PGM File type    : P5
Width of img     : 862 px
Height of img    : 404 px
Max Gray value   : 255
Enter the size of the filter window
15
The image file has been filtered
```

Testcase 03:

Input File

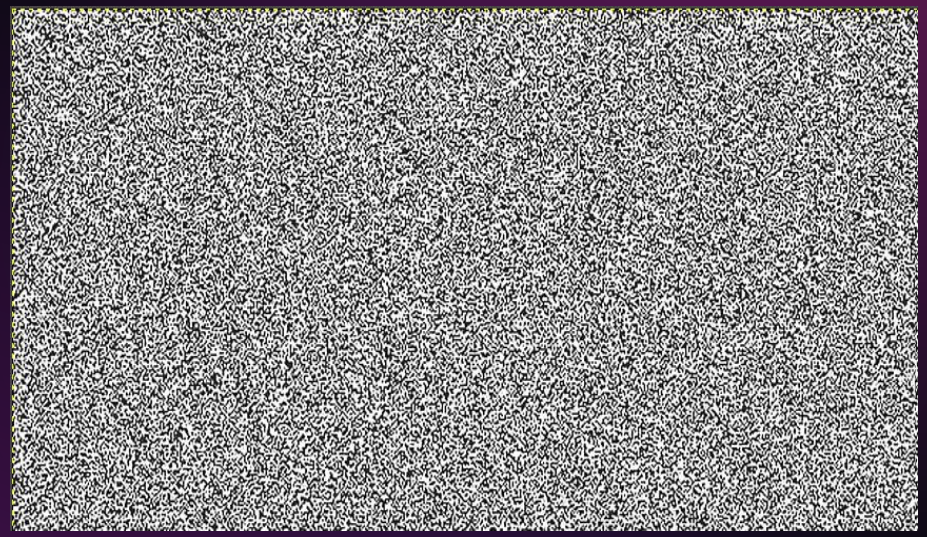
Output File

Image Details:

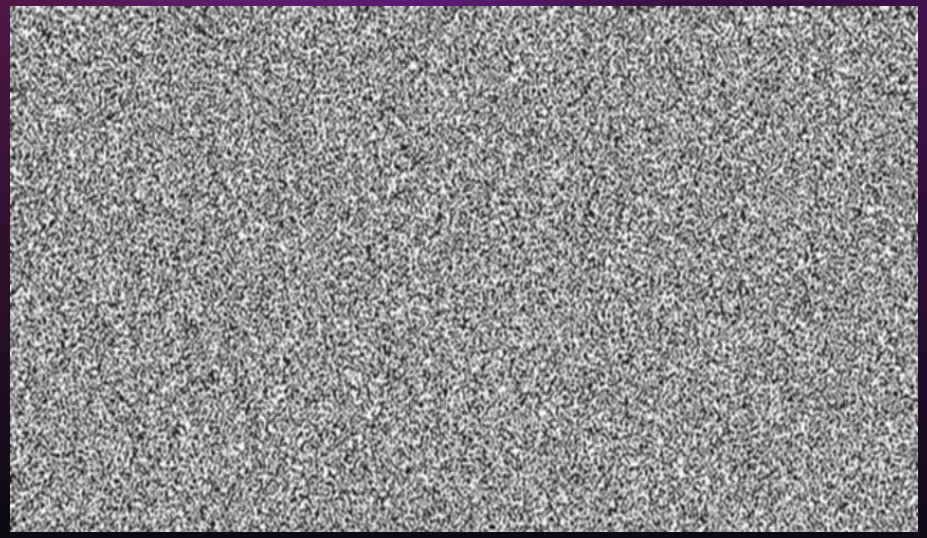
```
Enter the PGM file name with extension
SaltAndPepperNoise.pgm
File format      : pgm
PGM File type    : P5
Width of img     : 256 px
Height of img    : 256 px
Max Gray value   : 255
Enter the size of the filter window
11
The image file has been filtered
```

3x3 Filter kernel:

Input File

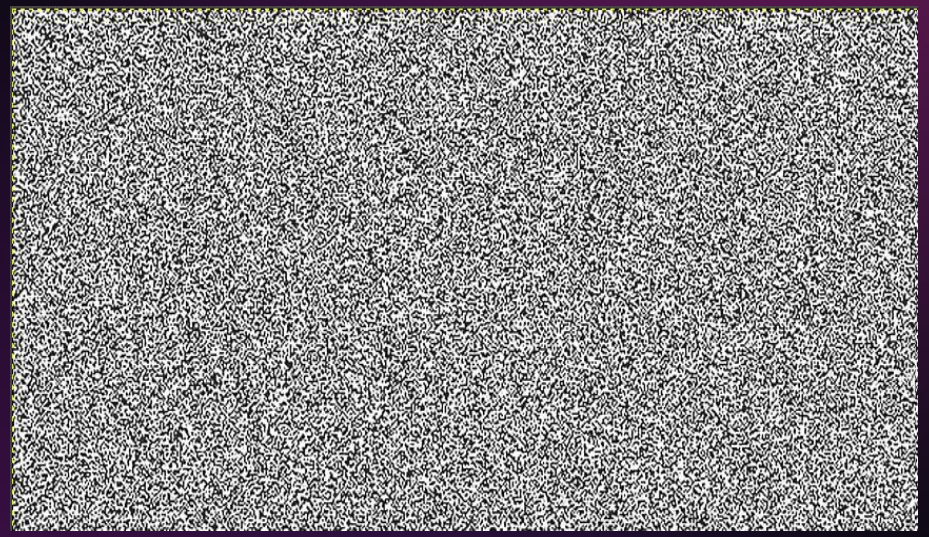


Output File

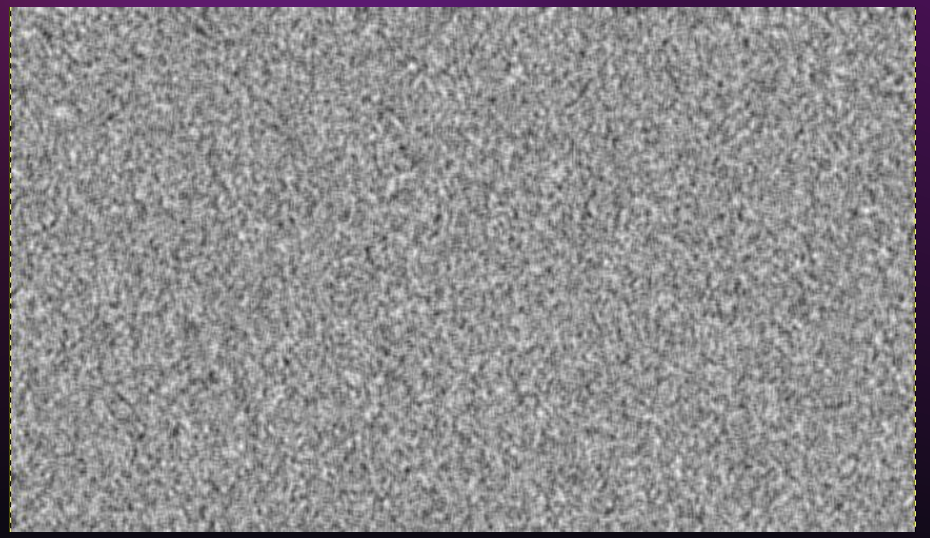


5x5 Filter kernel:

Input File

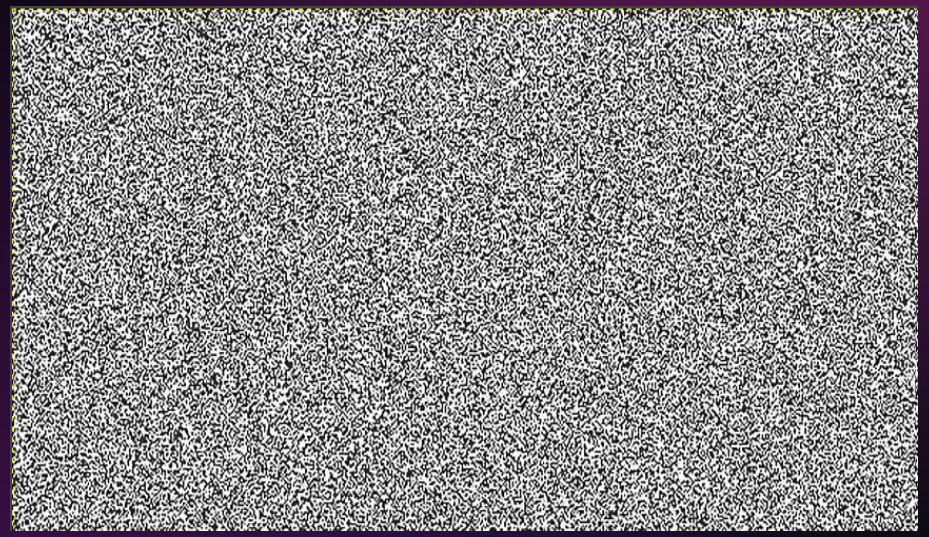


Output File



7x7 Filter kernel:

Input File



Output File

The Blurring effect increases with increase in size of filter kernel

Developing a Median Filter Kernel in C

- We have maintained the same read and write subroutines as mentioned in the previous code snippets, and have added a new function named filter() which accepts the PGM structure pointer and the size of filter window; and also tweaked the main function to some extent.

1. Main Function

```c
int main()
{
  PGM *pgm = malloc(sizeof(PGM));
  printf("Enter the PGM file name with extension\n");
  char ch[30];
  scanf("%s", ch);
  if (openPGM(pgm, ch))
  {
    printImageDetails(pgm, ch);
    int choice;
    printf("Enter the size of the filter window\n");
    scanf("%d", &choice);
    if (choice && 1)
    {
      PGM *filtered = filter(pgm, choice);
      saveImage(filtered, "filtered.pgm");
      printf("The image file has been filtered\n");
      free(filtered->data);
      free(filtered);
    }
    else
    {
      printf("Wrong size for the filter window\n");
      exit(EXIT_FAILURE);
    }
  }
  free(pgm->data);
  free(pgm);
  return 0;
}
```

```c
PGM *filter(PGM *pgm, int filt_size)
{
  PGM *pgm1 = malloc(sizeof(PGM));
  int s = filt_size / 2;
  int sq = filt_size * filt_size;
  int *a = (int *)malloc(sq * sizeof(int));
  strcpy(pgm1->type, pgm->type);
  pgm1->height = pgm->height;
  pgm1->width = pgm->width;
  pgm1->grayvalue = pgm->grayvalue;
  pgm1->data = malloc(pgm->height * sizeof(unsigned char *));
  for (int i = 0; i < pgm->height; i++)
  {
    pgm1->data[i] = malloc(pgm->width *
                           sizeof(unsigned char));
  }
  for (int i = 0; i < pgm->height; i++)
  {
    for (int j = 0; j < pgm->width; j++)
    {
      int d = 0;
      int e = 0;
      int m = 0;
      for (int k = -s; k <= s; k++)
      {
        for (int l = -s; l <= s; l++)
        {
          if (i + k > 0 && j + l > 0 && i + k < pgm->height && j + l < pgm->width)
          {
            a[m] = (int)pgm->data[i + k][j + l];
            m++;
          }
        }
      }
      qsort(a, m + 1, sizeof(int), cmp);
      int p = a[filt_size * filt_size / 2];
      // e = e / d;
      pgm1->data[i][j] = (char)p;
      free(a);
    }
  }
  return pgm1;
}
```

Filter Function

Testcase 01:

Input File

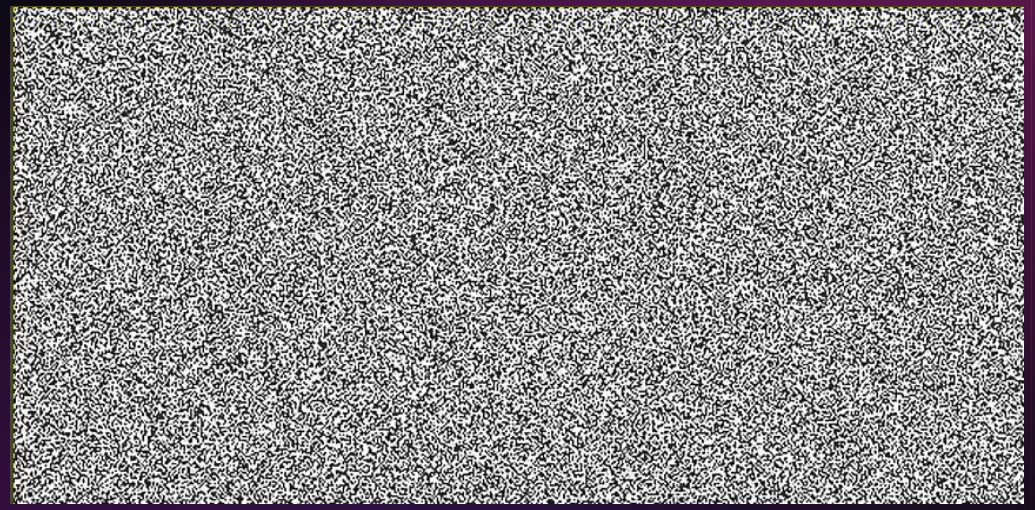
Output File

Image Details:

```
Enter the PGM file name with extension
SaltAndPepperNoise.pgm
File format      : pgm
PGM File type    : P5
Width of img     : 256 px
Height of img    : 256 px
Max Gray value   : 255
Enter the size of the filter window
3
The image file has been filtered
```

Testcase 02:

Input File



Output File

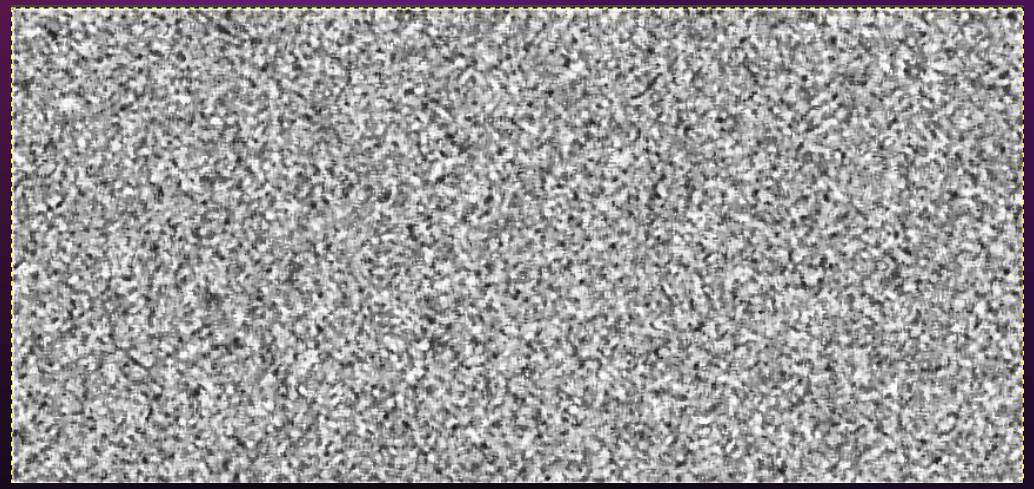


Image Details:

```
Enter the PGM file name with extension
noise.pgm
File format      : pgm
PGM File type    : P5
Width of img     : 862 px
Height of img    : 404 px
Max Gray value   : 255
Enter the size of the filter window
5
The image file has been filtered
```

Testcase 03:

Input File

Output File

Image Details:

```
Enter the PGM file name with extension
Horse.pgm
File format      : pgm
PGM File type    : P5
Width of img     : 766 px
Height of img    : 511 px
Max Gray value   : 255
Enter the size of the filter window
13
```

3x3 Filter kernel:

Input File

Output File

5x5 Filter kernel:

Input File

Output File

7x7 Filter kernel:

Input File

Output File

The image becomes smoother with increase in size of filter kernel

References:

Links:-

- https://www.geeksforgeeks.org/c-program-to-write-an-image-in-pgm-format/
- https://www.geeksforgeeks.org/how-to-read-a-pgmb-format-image-in-c/
- https://en.wikipedia.org/wiki/Kernel_(image_processing)
- https://setosa.io/ev/image-kernels/

Books:-

- Image Processing in C by Dwayne Phillips