```c
/* C code for ECE 6310 Lab 1: Convolution, Separable filters, sliding windows */
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<time.h>

int main(int argc, char* argv[])
{
    FILE            *fpt,*fpt_1,*fpt_2,*fpt_3;
    unsigned char   *image;
    unsigned char   *smoothed,*smoothed_1,*smoothed_final, *sep_smoothed_1,
    *sep_smoothed_final;
    char            header[320];
    int             ROWS,COLS,BYTES;
    int             r,c,r2,c2,sum,prev_val;
    struct timespec tp1,tp2;

    // Read and check original image:
    if ((fpt=fopen("bridge.ppm","rb"))==NULL)
    {
        printf("Unable to open bridge.ppm for reading.\n");
        exit(0);
    }
    fscanf(fpt,"%s %d %d %d",header,&COLS,&ROWS,&BYTES);

    if(strcmp(header,"P5")!=0 || BYTES!=255)
    {
        printf("Not a greyscale 8-bit PPM image.\n");
        exit(0);
    }
    image=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
    header[0]=fgetc(fpt);   //read whitespace char that separates header
    fread(image,1,COLS*ROWS,fpt);
    fclose(fpt);

    //allocate memory for smoothed versions:
    smoothed=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
    smoothed_1=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
    smoothed_final=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
    sep_smoothed_1=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
    sep_smoothed_final=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));

    // Assign a 0(black) to all border points in image:
    for(r=0;r<3;r++)
    {
        for(c=0;c<COLS;c++)
        {
            image[r*COLS+c]=0;
        }
    }
    for(r=ROWS-3;r<ROWS-1;r++)
    {
        for(c=0;c<COLS;c++)
        {
            image[r*COLS+c]=0;
        }
    }
    for(r=3;r<ROWS-3;r++)
    {
        for(c=0;c<3;c++)
        {
            image[r*COLS+c]=0;
        }
    }
    for(r=3;r<ROWS-3;r++)
    {
        for(c=COLS-3;c<COLS;c++)
        {
            image[r*COLS+c]=0;
        }
    }
    smoothed=image;         // smoothed image for 2D convolution
    smoothed_1=image;       // intermediary smoothed image after sliding window
    smoothed_final=image;   // final smoothed image after sliding window
    sep_smoothed_1=image;   // intermediary smoothed image after separable filters
    sep_smoothed_final=image; // final smoothed image after separable filters

    clock_gettime(CLOCK_REALTIME,&tp1); // Query timer for 2D convolution
    printf("Total time for 2D convolution:\t");
    for (r=3;r<ROWS-3;r++)  // Excluding border row points
    {
        for (c=3;c<COLS-3;c++)      // Excluding border column points
        {
            sum=0;
            for (r2=-3;r2<=3;r2++)
            {
                for(c2=-3;c2<=3;c2++)
                {
                    sum+= image[(r+r2)*COLS+(c+c2)]; // 2D filter of size 7*7
                }
            }
                smoothed[r*COLS+c]=sum/49;
        }
    }
    clock_gettime(CLOCK_REALTIME,&tp2); // Query timer for 2D convolution
    printf("%ld\n",tp2.tv_nsec-tp1.tv_nsec);// Report time to smooth using 2D convolution

    clock_gettime(CLOCK_REALTIME,&tp1); // Query timer for separable filters
    printf("Total time for sep filters:\t");
    for(r=3;r<ROWS-3;r++)
    {
        for(c=3;c<COLS-3;c++)
        {
            sum=0;
            for(c2=-3;c2<=3;c2++)   // 1D filter of size 1*7
            {
                sum+= image[r*COLS+(c+c2)]; // Taking sum across column values
            }
            sep_smoothed_1[r*COLS+c]=sum/7;  // Intermediary result image
        }
    }
    for(r=3;r<ROWS-3;r++)
    {
        for(c=3;c<COLS-3;c++)
        {
            sum=0;
            for(r2=-3;r2<=3;r2++)   // 1D filter of size 7*1
            {
                sum+= sep_smoothed_1[(r+r2)*COLS+c];// Taking sum across row values
            }
            sep_smoothed_final[r*COLS+c]=sum/7; // Final result image
        }
    }
    clock_gettime(CLOCK_REALTIME,&tp2); // Query timer for separable filters
    printf("%ld\n",tp2.tv_nsec-tp1.tv_nsec); // report time to smooth using separable filters

    clock_gettime(CLOCK_REALTIME,&tp1); // Query timer for sliding window
    printf("Total time for sep filters and sliding window:\t");
    sum=0;
    prev_val=0;
    for(r=3;r<ROWS-3;r++)
    {
        for(c=3;c<COLS-3;c++)
        {
            if(r==c==3)          // Calculating sum for first 1D sliding window
            {
                for(c2=-3;c2<=3;c2++)        // Separable 1D filter of size 1*7
                {
                    sum+= image[r*COLS+(c+c2)];
                    if (!prev_val)
                    {
                        prev_val=sum;        // Updating value for oldest column
                    }
                }
                smoothed_1[r*COLS+c]=sum/7; // Intermediary result image
            }
            else
            {
                sum-=prev_val;              // Subtracting oldest column value from sum
                prev_val=image[r*COLS+c-3]; // Updating oldest column for new window
                sum+= image[r*COLS+(c+3)];  // adding last column value to the existing sum
                smoothed_1[r*COLS+c]=sum/7; // Intermediary result image
            }
        }
    }
    sum=0;
    prev_val=0;
    for(c=3;c<COLS-3;c++)
    {
        for(r=3;r<ROWS-3;r++)
        {
            if(r==c==3) // Calculating sum for first 1D sliding window
            {
                for(r2=-3;r2<=3;r2++)   // Separable 1D filter of size 7*1
                {
                    sum+= smoothed_1[(r+r2)*COLS+c];
                    if (!prev_val)
                    {
                        prev_val=sum;// Updating value for oldest row
                    }
                }
                smoothed_final[r*COLS+c]=sum/7; // Final result image
            }
            else
            {
                sum-=prev_val;  // Subtracting oldest row value from sum
                prev_val=smoothed_1[(r-3)*COLS+c];// Updating oldest row for new window
                sum+= smoothed_1[(r+3)*COLS+c]; // Adding last row value to sum
                smoothed_final[r*COLS+c]=sum/7; // Final result image
            }
        }
    }
    clock_gettime(CLOCK_REALTIME,&tp2); // Query timer for sliding window
    printf("%ld\n",tp2.tv_nsec-tp1.tv_nsec);// report time to smooth using sliding window

    // write out smoothed images to see result:
    fpt_1=fopen("smoothed_by_convolution_1.ppm","wb");
    fpt_2=fopen("smoothed_by_sep_filters_1.ppm","wb");
    fpt_3=fopen("smoothed_by_sliding_window_1.ppm","wb");

    fprintf(fpt_1,"P5 %d %d 255\n",COLS,ROWS);
    fprintf(fpt_2,"P5 %d %d 255\n",COLS,ROWS);
    fprintf(fpt_3,"P5 %d %d 255\n",COLS,ROWS);

    fwrite(smoothed,COLS*ROWS,1,fpt_1); // final image after 2D convolution version
    fwrite(sep_smoothed_final,COLS*ROWS,1,fpt_2);// final image after separable filters version
    fwrite(smoothed_final,COLS*ROWS,1,fpt_3);// final image after sliding window version

    fclose(fpt_1);
    fclose(fpt_2);
    fclose(fpt_3);
}
```