```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>
#define PRINTF 0
//Thomas Gibbons

//prototypes for files.h
//{
int* readFile(int* sampleCount,int* sampleMax,char* filename);
double* offsetFile(int* sampleCount,int* sampleArray, double offset);
double* scaleFile(int* sampleCount,int* sampleArray, double scale);
void printArray(int Count,double* Array);
void outputFile(double count, double value, double* data, char* outputFile);
int copyFile(char* sourceFile, char* targetFile);
//}

//prototypes for calculations.h
//{
double mean(int* data,int count);
int maxValue(int* data, int count);
//}
int main(int argc, char *argv[])
{
    //if Lab5 is only argument
    if(argc==1)
    {
        printf("Usage for help:\t\tLab5\t-h\n");
        return 1;
    }
    //define variables and flags
        int c=1;
        int inputFile,renameLength;
        double offsetVal,scaleVal;
        int nn=0,oo=0,ss=0,SS=0,CC=0,NN=0,rr=0,hh=0;
        char** rename;
    //go through input arguments
        while(c<argc)
        {
            /*checks the next arguement for file # and
            sets nn flag if  # is there and bumps count
            so value isn't seen as an invalid option*/
            if(strcmp(argv[c],"-n")==0)
            {
                if(c+1>=argc)
                {
                    printf("\nInvalid input: No file selected\n");
                    printf("Usage for help:\t\tLab5\t-h\n");
                    return 1;
                }
                else
                {
                    if(argv[c+1][0]>='0' && argv[c+1][0]<='9')
```

```
            {
                inputFile=atoi(argv[c+1]);
                if(inputFile<0 || inputFile>99)
                {
                    printf("Invalid input: File number must be between 0 and 99");
                    return 1;
                }
                nn++;
                c++;
            }
            else
            {
                printf("\nInvalid input: No file selected\n");
                printf("Usage for help:\t\tLab5\t-h\n");
                return 1;
            }

        }
    }
    /*checks the next arguement for offset value and
    sets oo flag if  # is there and bumps count
    so value isn't seen as an invalid option*/
    else if(strcmp(argv[c],"-o")==0)
    {
        if(c+1>=argc)
        {
            printf("\nInvalid input: No offset selected\n");
        }
        else
        {
            if(argv[c+1][0]>=48 && argv[c+1][0]<=57)
            {
                offsetVal=strtod(argv[c+1],NULL);
                oo++;
                c++;
            }
            else
            {
                printf("\nInvalid input: Offset not specified");
            }
        }
    }
    /*checks the next arguement for scale value and
    sets ss flag if  # is there and bumps count
    so value isn't seen as an invalid option*/
    else if(strcmp(argv[c],"-s")==0)
    {
        if(c+1>=argc)
        {
            printf("\nInvalid input: No scale selected\n");
        }
        else
        {
```

```
            if(argv[c+1][0]>=48 && argv[c+1][0]<=57)
            {
                scaleVal=strtod(argv[c+1],NULL);
                ss++;
                c++;
            }
            else
            {
                printf("\nInvalid input: Scale not specified");
            }
        }
    }
    //checks if argument are there and sets flag accordingly
    else if(strcmp(argv[c],"-S")==0)
        SS++;
    else if(strcmp(argv[c],"-C")==0)
        CC++;
    else if(strcmp(argv[c],"-N")==0)
        NN++;
    /*checks the next arguement for scale value and
    sets ss flag if  # is there and bumps count
    so value isn't seen as an invalid option*/
    else if(strcmp(argv[c],"-r")==0)
    {
        if(c+1>=argc)
        {
            printf("\nInvalid input: No name change selected\n");
        }
        else
        {
            rr=c+1;
            while(argv[c+1][renameLength]!='\o')
                renameLength++;
            c++;
        }
    }
    //checks for argument, displays help, and exits
    else if(strcmp(argv[c],"-h")==0)
    {
        printf("Program can be run with the following options:\n\n\t");
        printf("-n:\tFile number(value needed)\n\t");
        printf("-o:\tOffset value(value needed)\n\t");
        printf("-s:\tScale factor(value needed)\n\t");
        printf("-S:\tGet statistics\n\t");
        printf("-C:\tCenter the signal\n\t");
        printf("-N:\tNormalize the signal\n\t");
        printf("-r:\tRename files(name needed)\n\t");
        printf("-h:\tHelp\n\n");
        return 1;
    }
    //other options are considered invalid
    else
    {
```

```
                printf("\nOption %s is not valid",argv[c]);
            }
            //bump count to next arguement
            c++;
        }
    //file number necessary to continue so terminate if it wasn't found
    if(nn==0)
    {
        printf("\nInvalid input: File number not included\n");
        printf("Usage for help:\t\tLab5\t-h\n");
        return 1;
    }
    //something to do needed to do or terminate
    if((rr+oo+ss+CC+NN)<1)
    {
        printf("\nInvalid input: No tasks to perform\n");
        printf("Usage for help:\t\tLab5\t-h\n");
        return 1;
    }
    //creates string of filename user selects
    //{
        char* filename=malloc(15*sizeof(char));
        if(inputFile<10)
            sprintf(filename,"Raw_data_0%d.txt",inputFile);
        else
            sprintf(filename,"Raw_data_%d.txt",inputFile);
    //}
    //read data and store integers in array
    //{
        int Count, Max;
        int* Array=readFile(&Count,&Max,filename);
    //}
    //ends program if input file invalid
    //{
        if (Array==NULL)
        {
            printf("%s could not be accessed\n",filename);
            free(filename);
            return 1;
        }
    //}

    if(rr>0)
    //copy raw data to new file
    {
        char* newName=malloc((renameLength+4)*sizeof(char));
        sprintf(newName,"%s.txt",argv[rr]);

        copyFile(filename,newName);
        printf("\n%s copied to %s\n",filename,newName);
        free(newName);
    }
    free(filename);
```

```
if(oo>0)
//operations for offsetting data
{
    if(rr>0)
    {
    //same amount of space for both scaled and offset
        char* outFile11=malloc((renameLength+11)*sizeof(char));

    //creates string of output file name user selected
        double* offset11=offsetFile(&Count,Array,offsetVal);
        sprintf(outFile11,"%s_Offset.txt",argv[rr]);

    //print offsetted data to output file
        outputFile(Count,offsetVal,offset11,outFile11);

    //free memory allocated
        free(offset11);
        free(outFile11);
    }
    else
    {
    //same amount of space for both scaled and offset
        char* outFile12=malloc(18*sizeof(char));

    //creates string of output file name user selected
        double* offset12=offsetFile(&Count,Array,offsetVal);
        if(inputFile<10)
            sprintf(outFile12,"Offset_data_0%d.txt",inputFile);
        else
            sprintf(outFile12,"Offset_data_%d.txt",inputFile);

    //print offsetted data to output file
        outputFile(Count,offsetVal,offset12,outFile12);

    //free memory allocated
        free(offset12);
        free(outFile12);
    }
}

if(ss>0)
//operations for scaling data
{
    if(rr>0)
    {
    //same amount of space for both scaled and offset
        char* outFile21=malloc((renameLength+11)*sizeof(char));

    //creates string of output file name user selected
        double* scale21=scaleFile(&Count,Array,scaleVal);
        sprintf(outFile21,"%s_Scaled.txt",argv[rr]);
```

```
        //print scaled data to output file
            outputFile(Count,scaleVal,scale21,outFile21);
        //free memory allocated
            free(scale21);
            free(outFile21);
        }
        else
        {
        //same amount of space for both scaled and offset
            char* outFile22=malloc(18*sizeof(char));

        //creates string of output file name user selected
            double* scale22=scaleFile(&Count,Array,scaleVal);
            if(inputFile<10)
                sprintf(outFile22,"Scaled_data_0%d.txt",inputFile);
            else
                sprintf(outFile22,"Scaled_data_%d.txt",inputFile);

        //print scaled data to output file
            outputFile(Count,scaleVal,scale22,outFile22);
        //free memory allocated
            free(scale22);
            free(outFile22);
        }
    }

    int maxData=maxValue(Array,Count);
    double ave=mean(Array,Count);

    //stats data output
    if(SS>0)
    {
        if(rr>0)
        {
        char* statFile11=malloc((renameLength+15)*sizeof(char));
        sprintf(statFile11,"%s_Statistics.txt",argv[rr]);

        outputFile(ave,maxData,NULL,statFile11);
        free(statFile11);
        }
        else
        {
        char* statFile12=malloc(22*sizeof(char));
        if(inputFile<10)
            sprintf(statFile12,"Statistics_data_0%d.txt",inputFile);
        else
            sprintf(statFile12,"Statistics_data_%d.txt",inputFile);

        outputFile(ave,maxData,NULL,statFile12);
        free(statFile12);
        }
    }
```

```
    //centered data output
    if(CC>0)
    {
        if(rr>0)
        {
        double* centered11=offsetFile(&Count,Array,ave*-1);
        char* centeredFile11=malloc((renameLength+13)*sizeof(char));
        sprintf(centeredFile11,"%s_Centered.txt",argv[rr]);

        outputFile(Count,ave*-1,centered11,centeredFile11);
        free(centered11);
        free(centeredFile11);
        }
        else
        {
        double* centered12=offsetFile(&Count,Array,ave*-1);
        char* centeredFile12=malloc(20*sizeof(char));
        if(inputFile<10)
            sprintf(centeredFile12,"Centered_data_0%d.txt",inputFile);
        else
            sprintf(centeredFile12,"Centered_data_%d.txt",inputFile);

        outputFile(Count,ave*-1,centered12,centeredFile12);
        free(centered12);
        free(centeredFile12);
        }
    }

    //normalized data output
    if(NN>0)
    {
        if(rr>0)
        {
        double* normalized11=scaleFile(&Count,Array,1.0/Max);
        char* normalizedFile11=malloc((renameLength+15)*sizeof(char));
        sprintf(normalizedFile11,"%s_Normalized.txt",argv[rr]);

        outputFile(Count,1.0/Max,normalized11,normalizedFile11);
        free(normalized11);
        free(normalizedFile11);
        }
        else
        {
        double* normalized12=scaleFile(&Count,Array,1.0/Max);
        char* normalizedFile12=malloc(22*sizeof(char));
        if(inputFile<10)
            sprintf(normalizedFile12,"Normalized_data_0%d.txt",inputFile);
        else
            sprintf(normalizedFile12,"Normalized_data_%d.txt",inputFile);

        outputFile(Count,1.0/Max,normalized12,normalizedFile12);
        free(normalized12);
        free(normalizedFile12);
```

```
        }
    }

    //free allocated memory
        free(Array);

    //end succesfully
        printf("\n");
        return 0;
}
//functions for files.c
//{
int copyFile(char* sourceFile, char* targetFile)
/*  input:  name of source file
            name of file it should be copied to*/
{
    FILE *fp1;
    fp1=fopen(sourceFile,"r");
    if(fp1==NULL)
    {
        printf("%s could not be accessed",sourceFile);
        return 1;
    }

    FILE *fp2;
    fp2=fopen(targetFile,"w");
    if(fp2==NULL)
    {
        printf("%s could not be accessed",targetFile);
        return 1;
    }

    char ch;
    while((ch=fgetc(fp1))!=EOF)
        fputc(ch,fp2);

    fclose(fp1);
    fclose(fp2);

    return 0;
}
int* readFile(int* sampleCount,int* sampleMax,char* filename)
/*  input:  address to store count
            address to store max value of data
            name of data file
    output: address of array of integer data*/
{
    FILE *fp;
    fp=fopen(filename,"r");

    if(fp==NULL)
        return NULL;
```

```
    fscanf(fp,"%d %d",sampleCount,sampleMax);
    int count=*(sampleCount);

    int* sampleArray;
    sampleArray=malloc(sizeof(int)*count);
    int x=0;

    while(count>0)
    {
        fscanf(fp,"%d", sampleArray+x);
        x++;
        count--;
    }

    fclose(fp);
    return sampleArray;
}

double* offsetFile(int* sampleCount,int* sampleArray, double offset)
/*  input:  address of count
            address of array of integer data
            value of offset
    output: address of array of double off-setted data*/
{
    double* offsetArray=malloc(*(sampleCount)*sizeof(double));
    int x=0;
    int count=*(sampleCount);

    while (count>0)
    {
        *(offsetArray+x)=*(sampleArray+x)+offset;
        x++;
        count--;
    }
    return offsetArray;
}

double* scaleFile(int* sampleCount,int* sampleArray, double scale)
/*  input:  address of count
            address of array of integer data
            value of scale
    output: address of array of double scaled data*/
{
    double* scaleArray=malloc(*(sampleCount)*sizeof(double));
    int x=0;
    int count=*(sampleCount);

    while (count>0)
    {
        *(scaleArray+x)=*(sampleArray+x)*scale;
        x++;
        count--;
    }
```

```
    return scaleArray;
}

void printArray(int Count,double* Array)
/*  input:  value of count
            address of array of double data
    output: displays double data*/
{
    int x=0;
    while(Count>0)
    {
        printf("%.4f ", (float)*(Array+x));
        x++;
        Count--;
    }
}

void outputFile(double count, double value, double* data, char* outputFile)
/*  input:  amount of data
            value(offset or scale) to be put in file
            double array to be printed
            name of file to output to*/
{
    FILE *write;

    write=fopen(outputFile,"w");

    fprintf(write,"%lf %lf\n",count, value);

    int x=0;
    if(data!=NULL)
        while(count>0)
        {
            fprintf(write,"%.4f\n",*(data+x));
            x++;
            count--;
        }

    fclose(write);

    printf("\n%s is loaded",outputFile);
}
//}

//functions for calculations.c
//{
double mean(int* data, int count)
/*  input:  integer array
            number of integers in array
    output: average of integers*/
{
    int total=0;
    int tempCount=count;
```

```
    while(tempCount>0)
    {
        total+=*(data+count-tempCount);
        tempCount--;
    }

    return (double) total/count;
}

int maxValue(int* data,int count)
/*  input:  integer array
            number of integers in array
    output: maximum value in array*/
{
    int tempCount=count;
    int maxValue=INT_MIN;
    while(tempCount>0)
    {
        maxValue=(maxValue>*(data+count-tempCount))? maxValue:*(data+count-tempCount);
        tempCount--;
    }

    return maxValue;
}


//}
```