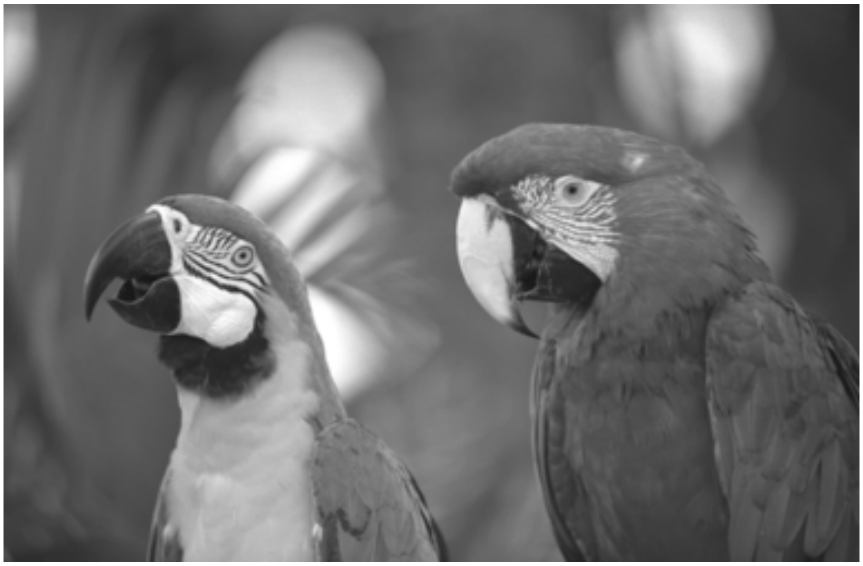
Report Lab 3 – Misha Tsysin, 0033922418

Area fill

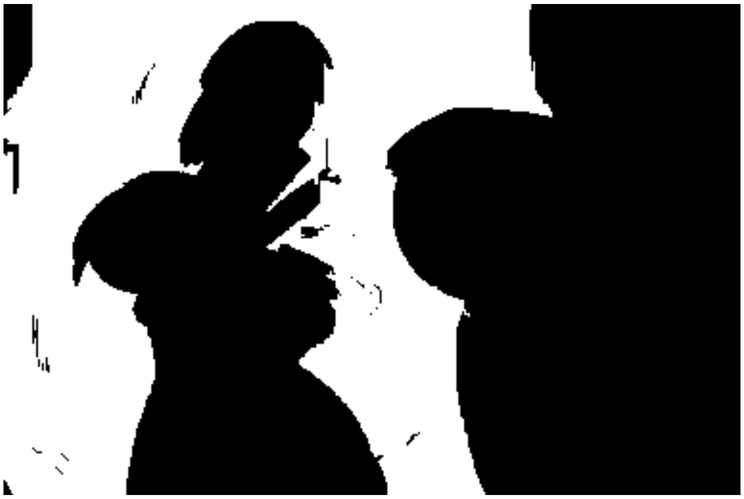
Essentially in this part, we need to write a simple graph traversal algorithm – breadth-first search. The idea is to maintain a list of nodes (pixels) that we want to visit and pop them from the queue as we go. The simple implementation of a queue in C is a linked list. The models based on dynamic arrays are too complicated and don't provide any improved performance.

The original image is:

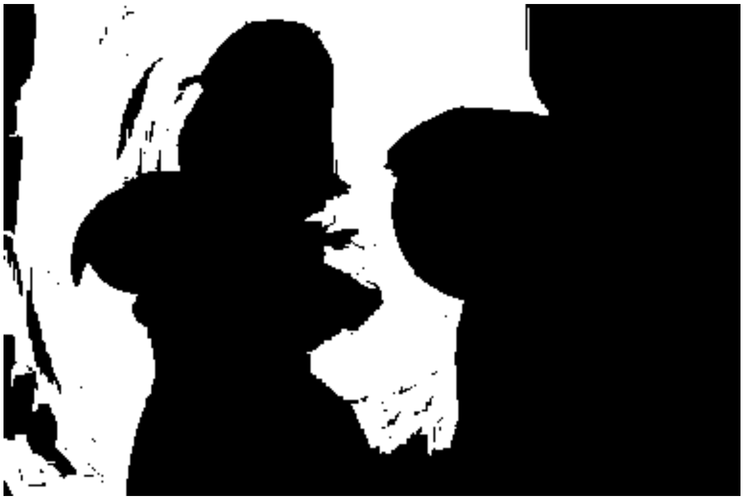


Now, the resulting segments for different starting points and thresholds:

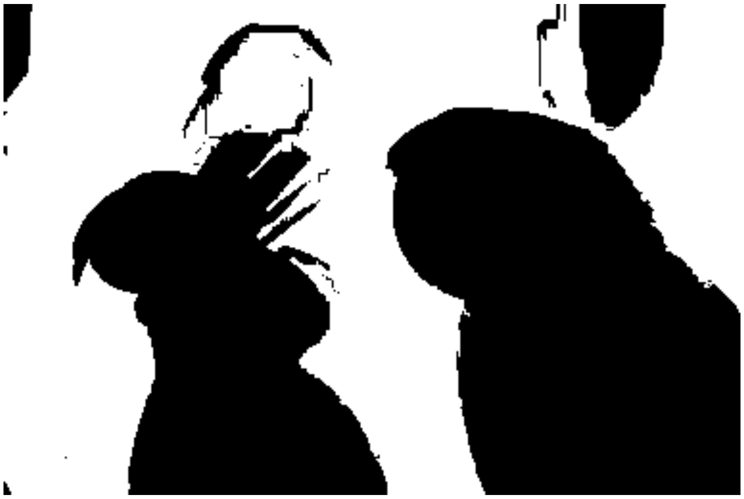
s = (67, 45), and T = 2:



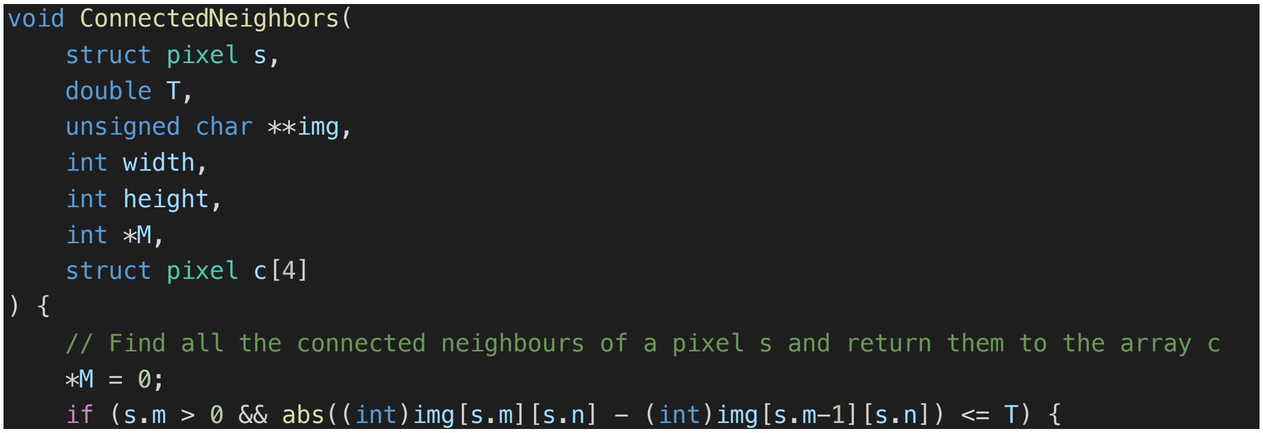
s = (67, 45), and T = 1:



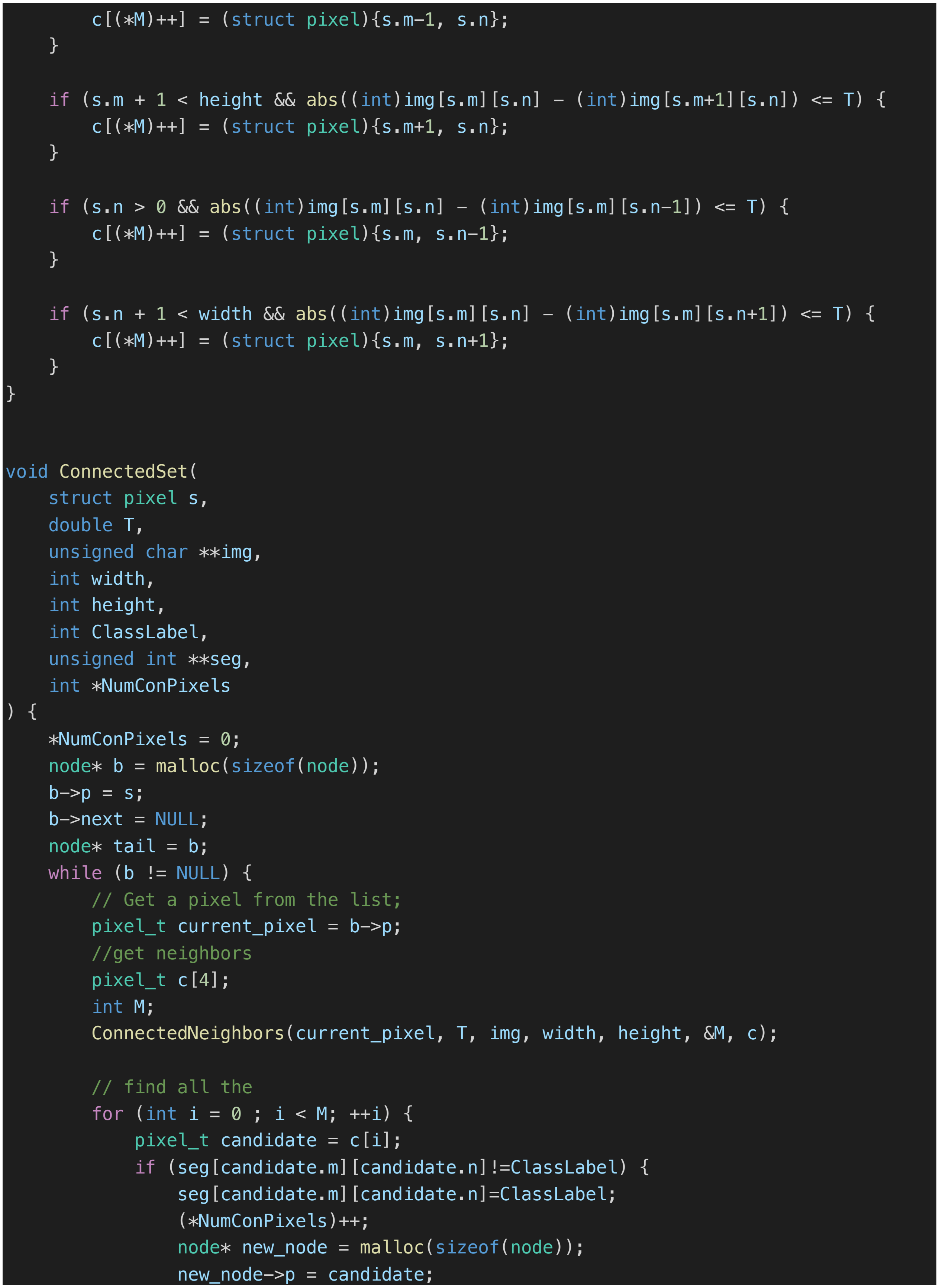
s = (67, 45), and T = 3:



The code responsible for finding the area fill is presented below. The logic can be summarized in the two functions:



```
void ConnectedNeighbors(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int *M,
    struct pixel c[4]
) {
    // Find all the connected neighbours of a pixel s and return them to the array c
    *M = 0;
    if (s.m > 0 && abs((int)img[s.m][s.n] - (int)img[s.m-1][s.n]) <= T) {
```

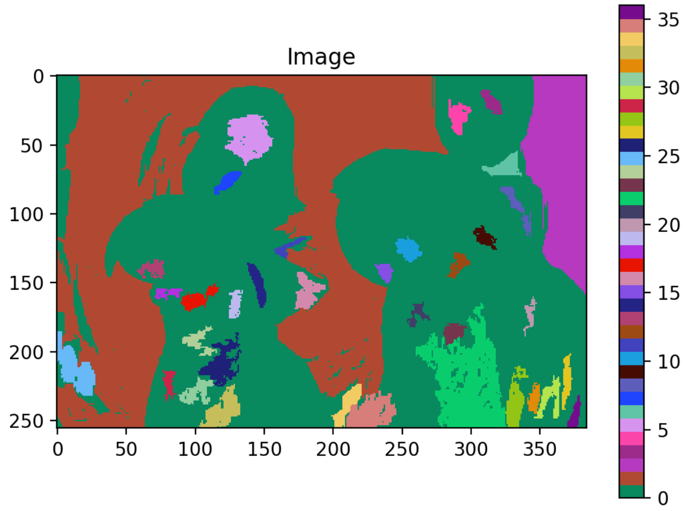


```
        c[(*M)++] = (struct pixel){s.m-1, s.n};
    }

    if (s.m + 1 < height && abs((int)img[s.m][s.n] - (int)img[s.m+1][s.n]) <= T) {
        c[(*M)++] = (struct pixel){s.m+1, s.n};
    }

    if (s.n > 0 && abs((int)img[s.m][s.n] - (int)img[s.m][s.n-1]) <= T) {
        c[(*M)++] = (struct pixel){s.m, s.n-1};
    }

    if (s.n + 1 < width && abs((int)img[s.m][s.n] - (int)img[s.m][s.n+1]) <= T) {
        c[(*M)++] = (struct pixel){s.m, s.n+1};
    }
}


void ConnectedSet(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int ClassLabel,
    unsigned int **seg,
    int *NumConPixels
) {
    *NumConPixels = 0;
    node* b = malloc(sizeof(node));
    b->p = s;
    b->next = NULL;
    node* tail = b;
    while (b != NULL) {
        // Get a pixel from the list;
        pixel_t current_pixel = b->p;
        //get neighbors
        pixel_t c[4];
        int M;
        ConnectedNeighbors(current_pixel, T, img, width, height, &M, c);

        // find all the
        for (int i = 0 ; i < M; ++i) {
            pixel_t candidate = c[i];
            if (seg[candidate.m][candidate.n]!=ClassLabel) {
                seg[candidate.m][candidate.n]=ClassLabel;
                (*NumConPixels)++;
                node* new_node = malloc(sizeof(node));
                new_node->p = candidate;
                new_node->next = NULL;
                tail->next = new_node;
                tail = new_node;
            }
        }
        node* old_b = b;
        b = b->next;
        free(old_b);
    }
}
```

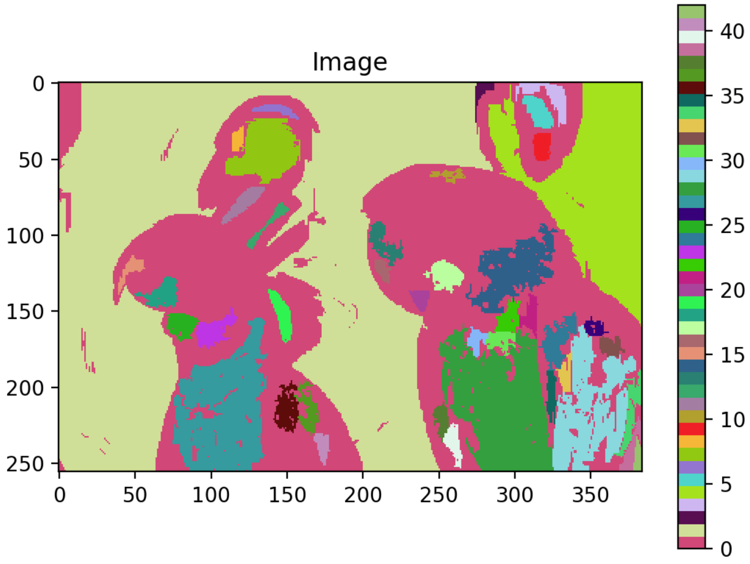
Image Segmentation

Now we will essentially run the previous algorithm on every single pixel to generate the segmentation.

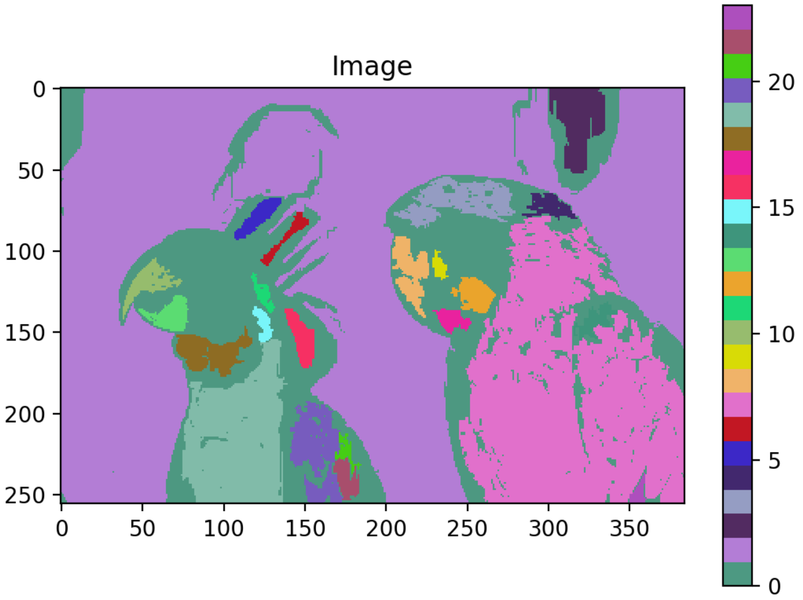
For T = 1:



T = 2:



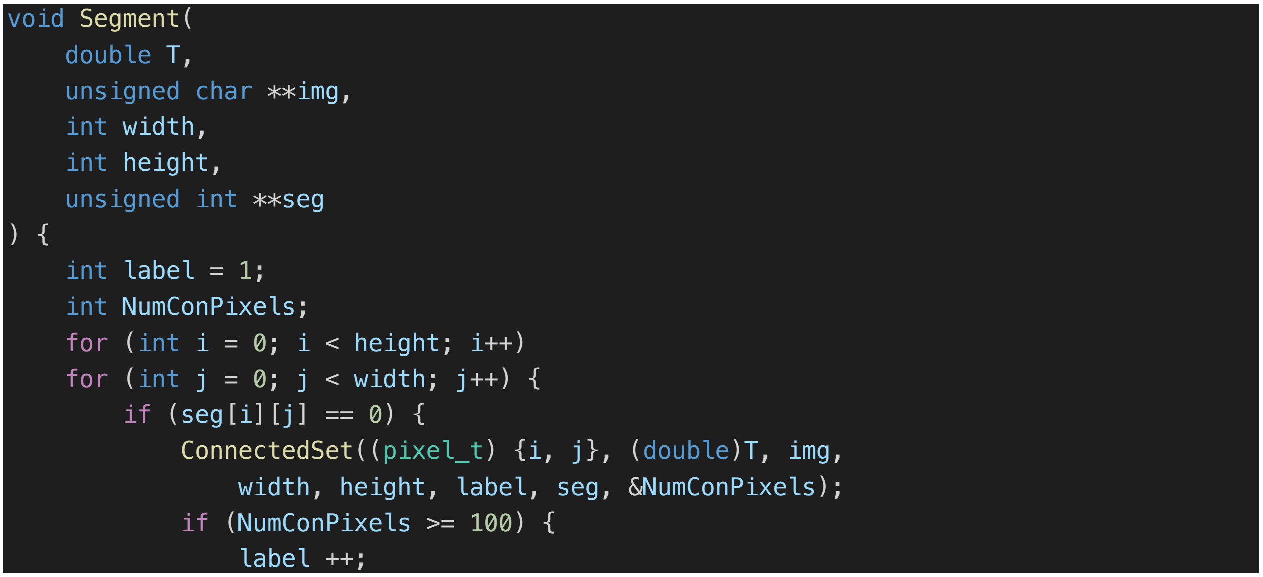
T = 3:



The number of segments is:

T	# Segments
1	36
2	41
3	23

The code for this part is essentially using the previous routine for each pixel:

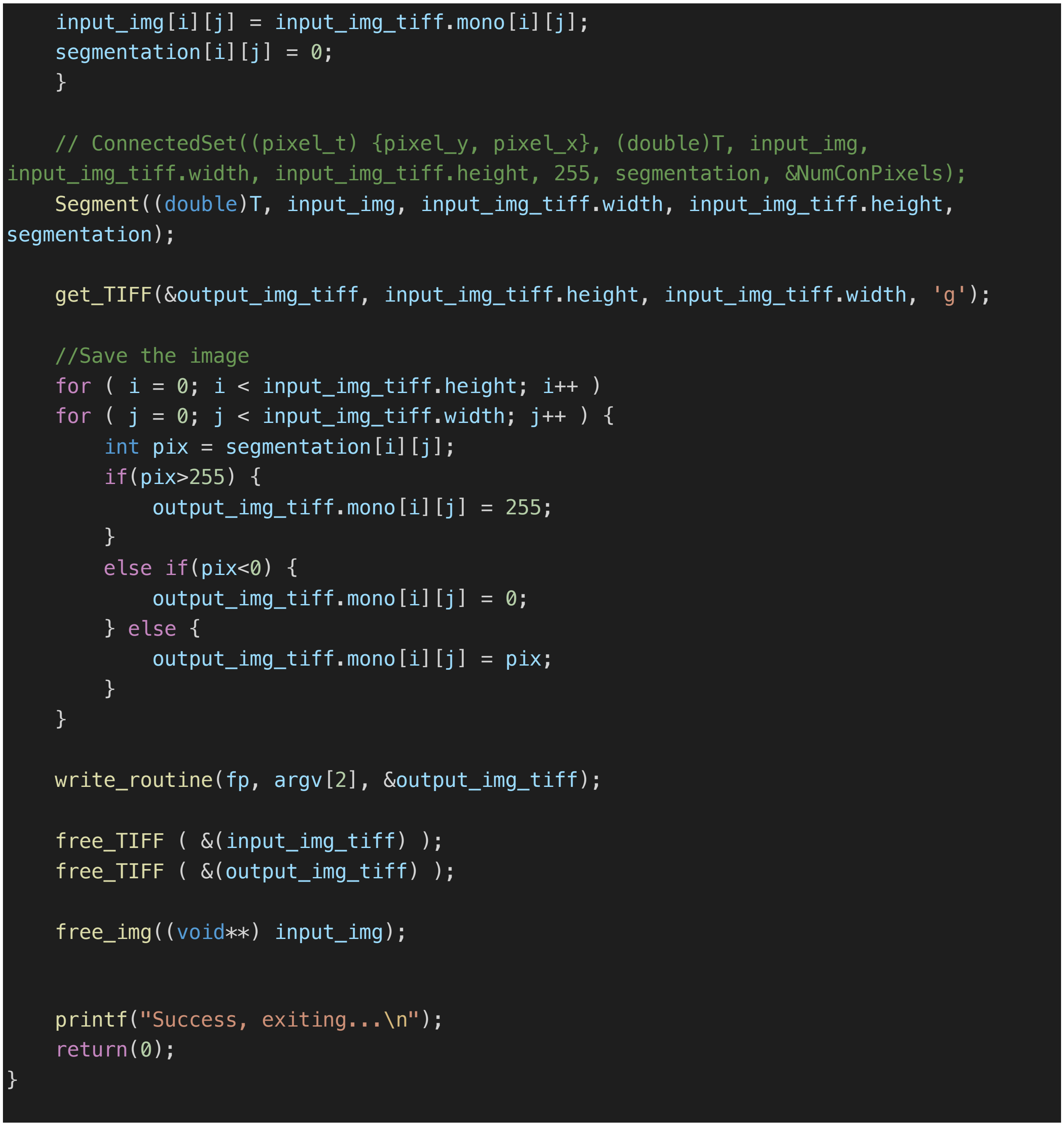


```
void Segment(
    double T,
    unsigned char **img,
    int width,
    int height,
    unsigned int **seg
) {
    int label = 1;
    int NumConPixels;
    for (int i = 0; i < height; i++)
    for (int j = 0; j < width; j++) {
        if (seg[i][j] == 0) {
            ConnectedSet((pixel_t) {i, j}, (double)T, img,
                width, height, label, seg, &NumConPixels);
            if (NumConPixels >= 100) {
                label ++;
            } else {
                ConnectedSet((pixel_t) {i, j}, (double)T, img,
                    width, height, 0, seg, &NumConPixels);
            }
        }
    }
}
```

Full Code With Driver:

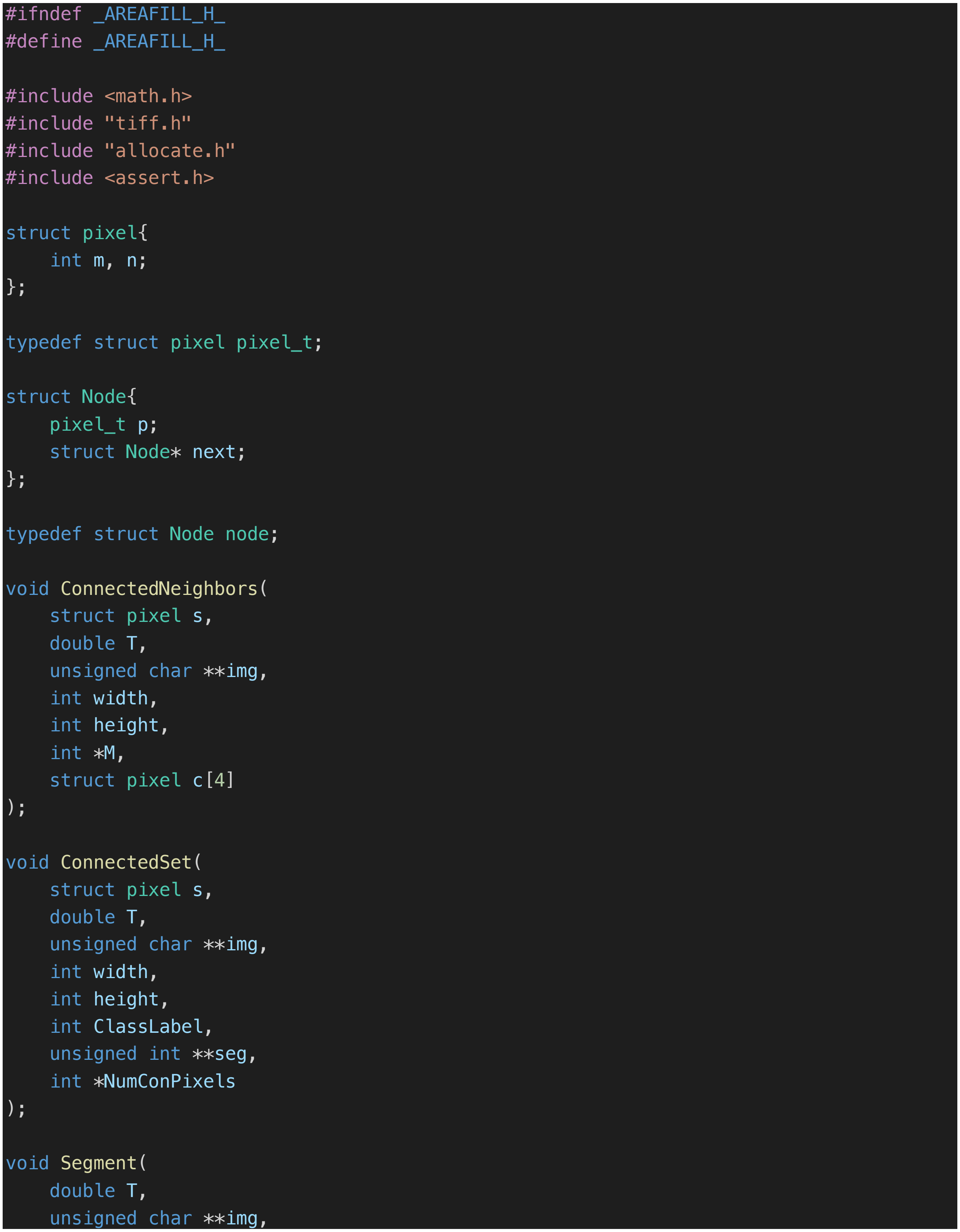
Main.c:

```
#include "tiff.h"
#include "allocate.h"
#include <assert.h>
#include "filter.h"
#include "areafill.h"

void error(char *name)
{
    printf("usage: %s input.tiff output.tiff pixel_y, pixel_x, T \n\n", name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("and a customfilter image.\n");
    printf("It then performs neighbourhood search\n");
    exit(1);
}

int main (int argc, char **argv)
{
    FILE *fp = 0;
    struct TIFF_img input_img_tiff, output_img_tiff;
    unsigned char **input_img;
    unsigned int **segmentation;
    int32_t i,j, pixel_y, pixel_x, T, NumConPixels;

    // Parse args:
    if (argc != 6) {
        error(argv[0]);
    }
    pixel_y = atoi(argv[3]);
    pixel_x = atoi(argv[4]);
    T = atoi(argv[5]);

    open_routine(fp, argv[1], &input_img_tiff, 'g');

    /* Allocate image of double precision floats */

    input_img = (unsigned char **)get_img(input_img_tiff.width, input_img_tiff.height,
sizeof(unsigned char));
    segmentation = (unsigned int **)get_img(input_img_tiff.width,
input_img_tiff.height, sizeof(unsigned int));

    /* copy all components */
    for ( i = 0; i < input_img_tiff.height; i++ )
    for ( j = 0; j < input_img_tiff.width; j++ ) {
```



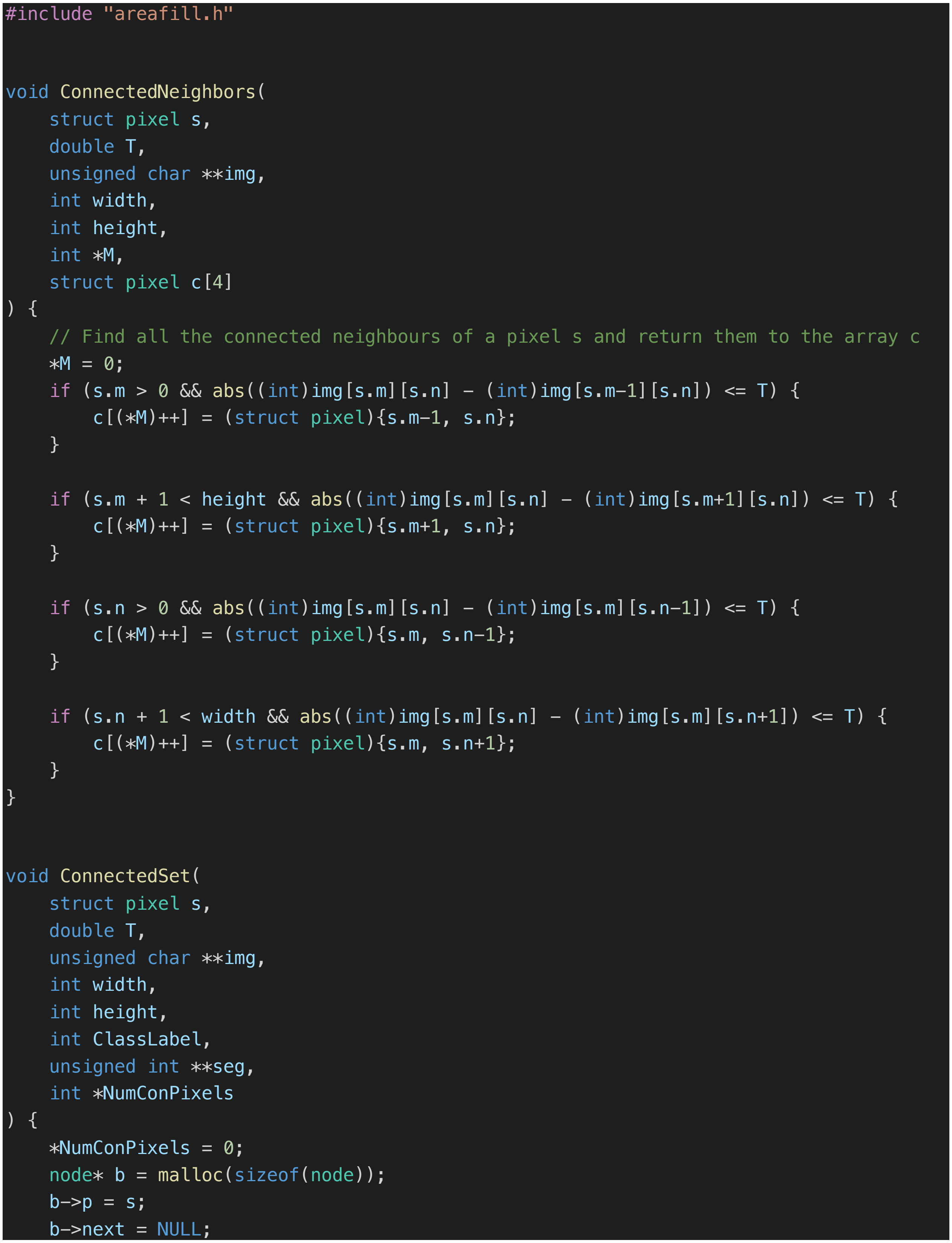
```
    input_img[i][j] = input_img_tiff.mono[i][j];
    segmentation[i][j] = 0;
    }

    // ConnectedSet((pixel_t) {pixel_y, pixel_x}, (double)T, input_img,
input_img_tiff.width, input_img_tiff.height, 255, segmentation, &NumConPixels);
    Segment((double)T, input_img, input_img_tiff.width, input_img_tiff.height,
segmentation);

    get_TIFF(&output_img_tiff, input_img_tiff.height, input_img_tiff.width, 'g');

    //Save the image
    for ( i = 0; i < input_img_tiff.height; i++ )
    for ( j = 0; j < input_img_tiff.width; j++ ) {
        int pix = segmentation[i][j];
        if(pix>255) {
            output_img_tiff.mono[i][j] = 255;
        }
        else if(pix<0) {
            output_img_tiff.mono[i][j] = 0;
        } else {
            output_img_tiff.mono[i][j] = pix;
        }
    }

    write_routine(fp, argv[2], &output_img_tiff);

    free_TIFF ( &(input_img_tiff) );
    free_TIFF ( &(output_img_tiff) );

    free_img((void**) input_img);


    printf("Success, exiting...\n");
    return(0);
}
```

Areafill.h

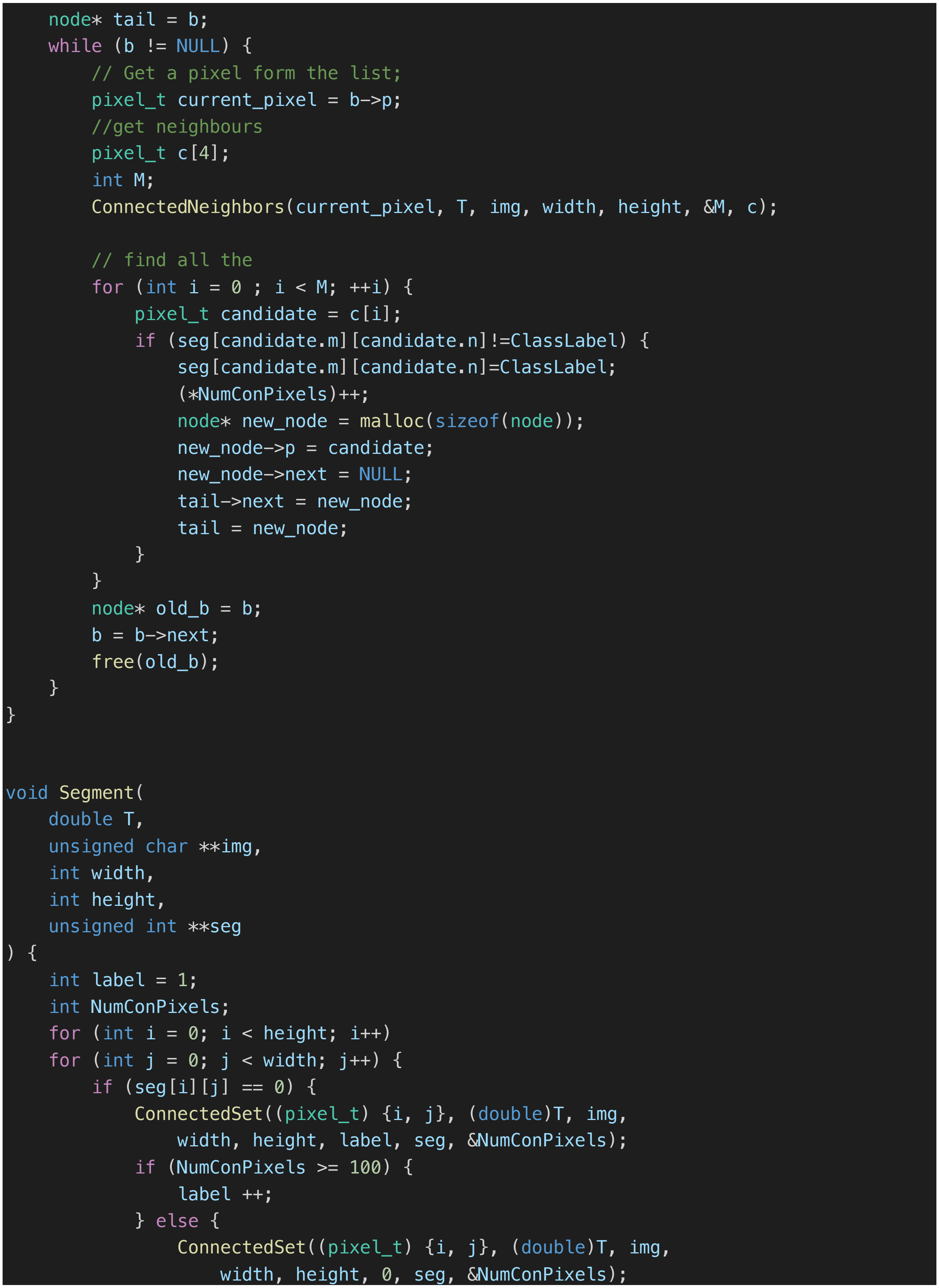


```
#ifndef _AREAFILL_H_
#define _AREAFILL_H_

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include <assert.h>

struct pixel{
    int m, n;
};

typedef struct pixel pixel_t;

struct Node{
    pixel_t p;
    struct Node* next;
};

typedef struct Node node;

void ConnectedNeighbors(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int *M,
    struct pixel c[4]
);

void ConnectedSet(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int ClassLabel,
    unsigned int **seg,
    int *NumConPixels
);

void Segment(
    double T,
    unsigned char **img,
    int width,
    int height,
    unsigned int **seg
);

#endif //_AREAFILL_H_
```

Areafill.c



```c
#include "areafill.h"


void ConnectedNeighbors(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int *M,
    struct pixel c[4]
) {
    // Find all the connected neighbours of a pixel s and return them to the array c
    *M = 0;
    if (s.m > 0 && abs((int)img[s.m][s.n] - (int)img[s.m-1][s.n]) <= T) {
        c[(*M)++] = (struct pixel){s.m-1, s.n};
    }

    if (s.m + 1 < height && abs((int)img[s.m][s.n] - (int)img[s.m+1][s.n]) <= T) {
        c[(*M)++] = (struct pixel){s.m+1, s.n};
    }

    if (s.n > 0 && abs((int)img[s.m][s.n] - (int)img[s.m][s.n-1]) <= T) {
        c[(*M)++] = (struct pixel){s.m, s.n-1};
    }

    if (s.n + 1 < width && abs((int)img[s.m][s.n] - (int)img[s.m][s.n+1]) <= T) {
        c[(*M)++] = (struct pixel){s.m, s.n+1};
    }
}


void ConnectedSet(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int ClassLabel,
    unsigned int **seg,
    int *NumConPixels
) {
    *NumConPixels = 0;
    node* b = malloc(sizeof(node));
    b->p = s;
    b->next = NULL;
```



```
    node* tail = b;
    while (b != NULL) {
        // Get a pixel form the list;
        pixel_t current_pixel = b->p;
        //get neighbours
        pixel_t c[4];
        int M;
        ConnectedNeighbors(current_pixel, T, img, width, height, &M, c);

        // find all the
        for (int i = 0 ; i < M; ++i) {
            pixel_t candidate = c[i];
            if (seg[candidate.m][candidate.n]!=ClassLabel) {
                seg[candidate.m][candidate.n]=ClassLabel;
                (*NumConPixels)++;
                node* new_node = malloc(sizeof(node));
                new_node->p = candidate;
                new_node->next = NULL;
                tail->next = new_node;
                tail = new_node;
            }
        }
        node* old_b = b;
        b = b->next;
        free(old_b);
    }
}


void Segment(
    double T,
    unsigned char **img,
    int width,
    int height,
    unsigned int **seg
) {
    int label = 1;
    int NumConPixels;
    for (int i = 0; i < height; i++)
    for (int j = 0; j < width; j++) {
        if (seg[i][j] == 0) {
            ConnectedSet((pixel_t) {i, j}, (double)T, img,
                width, height, label, seg, &NumConPixels);
            if (NumConPixels >= 100) {
                label ++;
            } else {
                ConnectedSet((pixel_t) {i, j}, (double)T, img,
                    width, height, 0, seg, &NumConPixels);
            }
        }
    }
}
```