```c
/**
 * binary.c
 *
 * week 3 section
 * fall 2013
 *
 * iterative binary search implementation
 */

#include <stdbool.h>

bool binary_search(int value, int values[], int n)
{
    // save bounds of list
    int beginning = 0;
    int end = n - 1;

    // while length of list > 0
    while (beginning <= end)
    {
        // look at middle of list
        int middle = (beginning + end) / 2;
        if (values[middle] == value)
        {
            // if number found, return true
            return true;
        }

        // else if middle higher, search left
        else if (values[middle] > value)
        {
            end = middle - 1;
        }

        // else if middle lower, search right
        else
        {
            beginning = middle + 1;
        }
    }

    // return false
    return false;
}
```

```c
/**
 * caesar.c
 *
 * week 3 section
 * fall 2013
 *
 * h/t Mike Rizzo
 *
 * practice using GDB on buggy code
 */

#include <cs50.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, string argv[])
{
    // ensure user entered a valid key
    if (argc != 2 || atoi(argv[0]) < 0)
    {
        printf("Usage: ./caesar key\n");
        return 1;
    }

    // save the key
    int key = atoi(argv[1]);

    // get the plaintext from the user
    string plaintext = GetString();

    // encrypt the plaintext
    for (int i = 0, n = strlen(plaintext); i < n; i++)
    {
        if (plaintext[i] > 'A' && plaintext[i] <= 'Z')
        {
            plaintext[i] = (plaintext[i] - 'A' + key) % 26;
        }
        else if (plaintext[i] >= 'a' && plaintext[i] < 'z')
        {
            plaintext[i] = (plaintext[i] - 'a' + key) % 26;
        }
    }

    // print out the encrypted message
    printf("%s\n", plaintext);

```

```
49.     return 0;
50. }
```

```c
/**
 * helpers.h
 *
 * week 3 section
 * fall 2013
 *
 * function declarations for searching and sorting algorithms
 */

#include <cs50.h>

// binary search
bool binary_search(int value, int values[], int n);

// bubble sort
void bubble_sort(int values[], int n);

// selection sort
void selection_sort(int values[], int n);
```

```
#
# Makefile
#
# week 3 section
# fall 2013
#

all: find generate

find: find.c binary.c bubble.c selection.c helpers.h
    clang -ggdb3 -O0 -std=c99 -Wall -Werror -o find find.c binary.c bubble.c selection.c -lcs50 -lm

generate: generate.c
    clang -ggdb3 -O0 -std=c99 -Wall -Werror -o generate generate.c

clean:
    rm -f *.o a.out core find generate
```

```c
/**
 * selection.c
 *
 * week 3 section
 * fall 2013
 *
 * selection sort implementation
 */

void selection_sort(int values[], int n)
{
    // run the sort n times
    for (int i = 0; i < n; i++)
    {
        // remember the smallest value
        int smallest_index = i;

        // make comparisons throughout the rest of the list
        for (int j = i + 1; j < n; j++)
        {
            // find the smallest number
            if (values[j] < values[smallest_index])
            {
                smallest_index = j;
            }
        }

        // put the beginning of the list where the smallest number was
        int tmp = values[smallest_index];
        values[smallest_index] = values[i];

        // place it in the beginning of the list
        values[i] = tmp;
    }
}
```