```c
/****************************************************************************
 * compare1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Tries (and fails) to compare two strings.
 *
 * Demonstrates strings as pointers to arrays.
 ***************************************************************************/

#include <cs50.h>
#include <stdio.h>


int
main(void)
{
    // get line of text
    printf("Say something: ");
    string s1 = GetString();

    // get another line of text
    printf("Say something: ");
    string s2 = GetString();

    // try (and fail) to compare strings
    if (s1 == s2)
        printf("You typed the same thing!\n");
    else
        printf("You typed different things!\n");
}
```

```c
/****************************************************************************
 * compare2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Compares two strings.
 *
 * Demonstrates strings as pointers to arrays.
 ***************************************************************************/

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


int
main(void)
{
    // get line of text
    printf("Say something: ");
    char *s1 = GetString();

    // get another line of text
    printf("Say something: ");
    char *s2 = GetString();

    // try to compare strings
    if (s1 != NULL && s2 != NULL)
    {
        if (strcmp(s1, s2) == 0)
            printf("You typed the same thing!\n");
        else
            printf("You typed different things!\n");
    }
}
```

```c
/****************************************************************************
 * copy1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Tries and fails to copy two strings.
 *
 * Demonstrates strings as pointers to arrays.
 ***************************************************************************/

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


int
main(void)
{
    // get line of text
    printf("Say something: ");
    char *s1 = GetString();
    if (s1 == NULL)
        return 1;

    // try (and fail) to copy string
    char *s2 = s1;

    // change "copy"
    printf("Capitalizing copy...\n");
    if (strlen(s2) > 0)
        s2[0] = toupper(s2[0]);

    // print original and "copy"
    printf("Original: %s\n", s1);
    printf("Copy:     %s\n", s2);

    // free memory
    free(s1);
}
```

```c
/****************************************************************************
 * copy2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Copies a string.
 *
 * Demonstrates strings as pointers to arrays.
 ***************************************************************************/

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


int
main(void)
{
    // get line of text
    printf("Say something: ");
    char *s1 = GetString();
    if (s1 == NULL)
        return 1;

    // allocate enough space for copy
    char *s2 = malloc((strlen(s1) + 1) * sizeof(char));
    if (s2 == NULL)
        return 1;

    // copy string
    int n = strlen(s1);
    for (int i = 0; i < n; i++)
        s2[i] = s1[i];
    s2[n] = '\0';

    // change copy
    printf("Capitalizing copy...\n");
    if (strlen(s2) > 0)
        s2[0] = toupper(s2[0]);

    // print original and copy
    printf("Original: %s\n", s1);
    printf("Copy:     %s\n", s2);

    // free memory
    free(s1);
    free(s2);
}
```

```c
/****************************************************************************
 * CS50 Library 3.0
 *
 * https://manual.cs50.net/Library
 *
 * Glenn Holloway <holloway@eecs.harvard.edu>
 * David J. Malan <malan@harvard.edu>
 *
 * Based on Eric Roberts' genlib.c and simpio.c.
 *
 * Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
 * http://creativecommons.org/licenses/by-nc-sa/3.0/
 ***************************************************************************/

#ifndef _CS50_H
#define _CS50_H

#include <float.h>
#include <limits.h>
#include <stdbool.h>
#include <stdlib.h>


/*
 * Our own data type for string variables.
 */

typedef char *string;


/*
 * Reads a line of text from standard input and returns the equivalent
 * char; if text does not represent a char, user is prompted to retry.
 * Leading and trailing whitespace is ignored.  If line can't be read,
 * returns CHAR_MAX.
 */

char
GetChar(void);


/*
 * Reads a line of text from standard input and returns the equivalent
 * double as precisely as possible; if text does not represent a
 * double, user is prompted to retry.  Leading and trailing whitespace
 * is ignored.  For simplicity, overflow and underflow are not detected.
 * If line can't be read, returns DBL_MAX.
 */
```

```

double
GetDouble(void);


/*
 * Reads a line of text from standard input and returns the equivalent
 * float as precisely as possible; if text does not represent a float,
 * user is prompted to retry.  Leading and trailing whitespace is ignored.
 * For simplicity, overflow and underflow are not detected.  If line can't
 * be read, returns FLT_MAX.
 */

float
GetFloat(void);


/*
 * Reads a line of text from standard input and returns it as an
 * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
 * does not represent such an int, user is prompted to retry.  Leading
 * and trailing whitespace is ignored.  For simplicity, overflow is not
 * detected.  If line can't be read, returns INT_MAX.
 */

int
GetInt(void);


/*
 * Reads a line of text from standard input and returns an equivalent
 * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
 * does not represent such a long long, user is prompted to retry.
 * Leading and trailing whitespace is ignored.  For simplicity, overflow
 * is not detected.  If line can't be read, returns LLONG_MAX.
 */

long long
GetLongLong(void);


/*
 * Reads a line of text from standard input and returns it as a
 * string (char *), sans trailing newline character.  (Ergo, if
 * user inputs only "\n", returns "" not NULL.)  Returns NULL
 * upon error or no input whatsoever (i.e., just EOF).  Leading
 * and trailing whitespace is not ignored.  Stores string on heap
 * (via malloc); memory must be freed by caller to avoid leak.
 */

string GetString(void);



#endif
```

```c
/****************************************************************************
 * sigma1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Adds the numbers 1 through n.
 *
 * Demonstrates iteration.
 ***************************************************************************/

#include <cs50.h>
#include <stdio.h>


// prototype
int sigma(int);


int
main(void)
{
    // ask user for a positive int
    int n;
    do
    {
        printf("Positive integer please: ");
        n = GetInt();
    }
    while (n < 1);

    // compute sum of 1 through n
    int answer = sigma(n);

    // report answer
    printf("%d\n", answer);
}


/*
 * Returns sum of 1 through m; returns 0 if m is not positive.
 */

int
sigma(int m)
{
    // avoid risk of infinite loop
    if (m < 1)
```

```c
        return 0;

    // return sum of 1 through m
    int sum = 0;
    for (int i = 1; i <= m; i++)
        sum += i;
    return sum;
}
```

```c
/****************************************************************************
 * sigma2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Adds the numbers 1 through n.
 *
 * Demonstrates recursion.
 ***************************************************************************/

#include <cs50.h>
#include <stdio.h>


// prototype
int sigma(int);


int
main(void)
{
    // ask user for a positive int
    int n;
    do
    {
        printf("Positive integer please: ");
        n = GetInt();
    }
    while (n < 1);

    // compute sum of 1 through n
    int answer = sigma(n);

    // report answer
    printf("%d\n", answer);
}


/*
 * Returns sum of 1 through m; returns 0 if m is not positive.
 */

int
sigma(int m)
{
    // base case
    if (m <= 0)
```

```
        return 0;

    // recursive case
    else
        return (m + sigma(m-1));
}
```