```c
/**
 * copy-2.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Copies a string.
 *
 * Demonstrates pointer arithmetic.
 */

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

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

    // allocate enough space for copy
    char* t = malloc((strlen(s) + 1) * sizeof(char));
    if (t == NULL)
    {
        return 1;
    }

    // copy string, including '\0' at end
    for (int i = 0, n = strlen(s); i <= n; i++)
    {
        *(t + i) = *(s + i);
    }

    // change copy
    printf("Capitalizing copy...\n");
    if (strlen(t) > 0)
    {
        *t = toupper(*t);
    }

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

    // success
    return 0;
}
```

```c
/****************************************************************************
 * CS50 Library 5
 * https://manual.cs50.net/library/
 *
 * Based on Eric Roberts' genlib.c and simpio.c.
 *
 * Copyright (c) 2013,
 * Glenn Holloway <holloway@eecs.harvard.edu>
 * David J. Malan <malan@harvard.edu>
 * All rights reserved.
 *
 * BSD 3-Clause License
 * http://www.opensource.org/licenses/BSD-3-Clause
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 * * Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 * * Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 * * Neither the name of CS50 nor the names of its contributors may be used
 *   to endorse or promote products derived from this software without
 *   specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
 * TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 ***************************************************************************/

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

#include "cs50.h"

/**
 * 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)
{
    // try to get a char from user
    while (true)
    {
        // get line of text, returning CHAR_MAX on failure
        string line = GetString();
        if (line == NULL)
        {
            return CHAR_MAX;
        }

        // return a char if only a char (possibly with
        // leading and/or trailing whitespace) was provided
        char c1, c2;
        if (sscanf(line, " %c %c", &c1, &c2) == 1)
        {
            free(line);
            return c1;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}

/**
 * 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)
{
    // try to get a double from user
    while (true)
    {
        // get line of text, returning DBL_MAX on failure
        string line = GetString();
        if (line == NULL)
        {
            return DBL_MAX;
        }

        // return a double if only a double (possibly with
        // leading and/or trailing whitespace) was provided
        double d; char c;
        if (sscanf(line, " %lf %c", &d, &c) == 1)
        {
            free(line);
            return d;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}

/**
 * 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)
{
    // try to get a float from user
    while (true)
    {
        // get line of text, returning FLT_MAX on failure
        string line = GetString();
        if (line == NULL)
        {
            return FLT_MAX;
        }

        // return a float if only a float (possibly with
        // leading and/or trailing whitespace) was provided
        char c; float f;
        if (sscanf(line, " %f %c", &f, &c) == 1)
        {
            free(line);
            return f;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}

/**
 * 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)
{
    // try to get an int from user
    while (true)
    {
        // get line of text, returning INT_MAX on failure
        string line = GetString();
        if (line == NULL)
        {
            return INT_MAX;
        }

        // return an int if only an int (possibly with
        // leading and/or trailing whitespace) was provided
        int n; char c;
        if (sscanf(line, " %d %c", &n, &c) == 1)
        {
            free(line);
            return n;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}

/**
 * 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.
 */
```

```c

long long GetLongLong(void)
{
    // try to get a long long from user
    while (true)
    {
        // get line of text, returning LLONG_MAX on failure
        string line = GetString();
        if (line == NULL)
        {
            return LLONG_MAX;
        }

        // return a long long if only a long long (possibly with
        // leading and/or trailing whitespace) was provided
        long long n; char c;
        if (sscanf(line, " %lld %c", &n, &c) == 1)
        {
            free(line);
            return n;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}

/**
 * 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)
{
    // growable buffer for chars
    string buffer = NULL;

    // capacity of buffer
    unsigned int capacity = 0;

    // number of chars actually in buffer
    unsigned int n = 0;

    // character read or EOF
    int c;

    // iteratively get chars from standard input
    while ((c = fgetc(stdin)) != '\n' && c != EOF)
    {
        // grow buffer if necessary
        if (n + 1 > capacity)
        {
            // determine new capacity: start at 32 then double
            if (capacity == 0)
            {
                capacity = 32;
            }
            else if (capacity <= (UINT_MAX / 2))
            {
                capacity *= 2;
            }
            else
            {
                free(buffer);
                return NULL;
            }

            // extend buffer's capacity
            string temp = realloc(buffer, capacity * sizeof(char));
            if (temp == NULL)
            {
                free(buffer);
                return NULL;
            }
            buffer = temp;
        }

        // append current character to buffer
        buffer[n++] = c;
    }

    // return NULL if user provided no input
    if (n == 0 && c == EOF)
    {
        return NULL;
    }

    // minimize buffer
    string minimal = malloc((n + 1) * sizeof(char));
    strncpy(minimal, buffer, n);
    free(buffer);
```

```c

    // terminate string
    minimal[n] = '\0';

    // return string
    return minimal;
}
```

```c
/****************************************************************************
 * CS50 Library 5
 * https://manual.cs50.net/library/
 *
 * Based on Eric Roberts' genlib.c and simpio.c.
 *
 * Copyright (c) 2013,
 * Glenn Holloway <holloway@eecs.harvard.edu>
 * David J. Malan <malan@harvard.edu>
 * All rights reserved.
 *
 * BSD 3-Clause License
 * http://www.opensource.org/licenses/BSD-3-Clause
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 * * Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 * * Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 * * Neither the name of CS50 nor the names of its contributors may be used
 *   to endorse or promote products derived from this software without
 *   specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
 * TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 ***************************************************************************/

#ifndef _CS50_H
#define _CS50_H

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

```

```c
/**
 * 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
/**
 * hello-2.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Says hello to whomever.
 *
 * Demonstrates use of CS50's library and standard input.
 */

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

int main(void)
{
    printf("State your name: ");
    string name = GetString();
    printf("hello, %s\n", name);
}
```