```c
/****************************************************************************
 * argv1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Prints command-line arguments, one per line.
 *
 * Demonstrates use of argv.
 ***************************************************************************/

#include <stdio.h>


int
main(int argc, char *argv[])
{
    // print arguments
    printf("\n");
    for (int i = 0; i < argc; i++)
        printf("%s\n", argv[i]);
    printf("\n");
}
```

```c
/****************************************************************************
 * bar.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Offers opportunities to play with pointers with GDB.
 ***************************************************************************/

#include <stdio.h>


int foo(int n);
void bar(int m);

int
main(void)
{
    int a;
    char * s = "hello, world";
    printf("%s\n", &s[7]);
    a = 5;
    foo(a);
    return 0;
}

int
foo(int n)
{
    int b;
    b = n;
    b *= 2;
    bar(b);
    return b;
}

void
bar(int m)
{
    printf("Hi, I'm bar!\n");
}
```

```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/
 ***************************************************************************/

#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.
 */

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);

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

    // return string
    return minimal;
}
```

```c
/****************************************************************************
 * pointers.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Prints a given string one character per line.
 *
 * Demonstrates pointer arithmetic.
 ***************************************************************************/

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


int
main(void)
{
    // get line of text
   char *s = GetString();
    if (s == NULL)
        return 1;

    // print string, one character per line
    for (int i = 0, n = strlen(s); i < n; i++)
        printf("%c\n", *(s+i));

    // free string
    free(s);
}
```

```c
/****************************************************************************
 * scanf1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Reads a number from the user into an int.
 *
 * Demonstrates scanf and address-of operator.
 ***************************************************************************/

#include <stdio.h>


int
main(void)
{
    int x;
    printf("Number please: ");
    scanf("%d", &x);
    printf("Thanks for the %d!\n", x);
}
```

```c
/****************************************************************************
 * scanf2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Reads a string from the user into memory it shouldn't.
 *
 * Demonstrates possible attack!
 ***************************************************************************/

#include <stdio.h>


int
main(void)
{
    char *buffer;
    printf("String please: ");
    scanf("%s", buffer);
    printf("Thanks for the \"%s\"!\n", buffer);
}
```

```c
/****************************************************************************
 * scanf3.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Reads a string from the user into an array (dangerously).
 *
 * Demonstrates potential buffer overflow!
 ***************************************************************************/

#include <stdio.h>


int
main(void)
{
    char buffer[16];
    printf("String please: ");
    scanf("%s", buffer);
    printf("Thanks for the \"%s\"!\n", buffer);
}
```

```c
/****************************************************************************
 * structs.h
 *
 * Computer Science 50
 * David J. Malan
 *
 * Defines a student for structs{1,2}.c.
 ***************************************************************************/


// structure representing a student
typedef struct
{
    int id;
    char *name;
    char *house;
}
student;
```

```c
/****************************************************************************
 * structs1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Demonstrates use of structs.
 ***************************************************************************/

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

#include "structs.h"


// class size
#define STUDENTS 3


int
main(void)
{
    // declare class
    student class[STUDENTS];

    // populate class with user's input
    for (int i = 0; i < STUDENTS; i++)
    {
        printf("Student's ID: ");
        class[i].id = GetInt();

        printf("Student's name: ");
        class[i].name = GetString();

        printf("Student's house: ");
        class[i].house = GetString();
        printf("\n");
    }

    // now print anyone in Mather
    for (int i = 0; i < STUDENTS; i++)
        if (strcmp(class[i].house, "Mather") == 0)
            printf("%s is in Mather!\n\n", class[i].name);

    // free memory
    for (int i = 0; i < STUDENTS; i++)
    {
        free(class[i].name);
        free(class[i].house);
    }
}
```

```c
/****************************************************************************
 * swap.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Swaps two variables' values.
 *
 * Demonstrates passing by reference.
 ***************************************************************************/

#include <stdio.h>


// function prototype
void swap(int *a, int *b);


int
main(void)
{
    int x = 1;
    int y = 2;

    printf("x is %d\n", x);
    printf("y is %d\n", y);
    printf("Swapping...\n");
    swap(&x, &y);
    printf("Swapped!\n");
    printf("x is %d\n", x);
    printf("y is %d\n", y);
}


/*
 * Swap arguments' values.
 */

void
swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```