Hai.java

1/1

lectures/2/src/

```
class Hai
{
    public static void main(String [] args)
    {
        System.out.println("O hai, world!");
    }
}
```

argv1.c

1/1

lectures/2/src/

```
/*****************************************************************************
 * 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[])
{
    int i;

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

```
/*****************************************************************************
 * argv2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Prints command-line arguments, one character per line.
 *
 * Demonstrates argv as a two-dimensional array.
 *****************************************************************************/

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


int
main(int argc, char *argv[])
{
    int i, j, n;

    // print arguments
    printf("\n");
    for (i = 0; i < argc; i++)
    {
        for (j = 0, n = strlen(argv[i]); j < n; j++)
            printf("%c\n", argv[i][j]);
        printf("\n");
    }
}
```



```
/*****************************************************************************
 * array.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Computes a student's average across 3 quizzes (without dropping lowest).
 *
 * Demonstrates use of an array, a constant, a multiline string, and
 * rounding.
 *****************************************************************************/

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

/* number of quizzes per term */
#define QUIZZES 3


int
main(int argc, char * argv[])
{
    float grades[QUIZZES];
    int average, i, sum;

    /* ask user for grades */
    printf("\nWhat were your quiz scores?\n\n");
    for (i = 0; i < QUIZZES; i++)
    {
        printf("Quiz #%d of %d: ", i+1, QUIZZES);
        grades[i] = GetFloat();
    }

    /* compute average */
    sum = 0;
    for (i = 0; i < QUIZZES; i++)
        sum += grades[i];
    average = (int) (sum / (float) QUIZZES + 0.5);

    /* report average */
    printf("\nWithout dropping your lowest score, "
           "your average is: %d\n\n", average);
}
```

```
/***************************************************************************
 * array1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Computes a student’s average across 2 quizzes.
 *
 * Demonstrates use of an array, a constant, and rounding.
 ***************************************************************************/

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


// number of quizzes per term
#define QUIZZES 2


int
main(int argc, char *argv[])
{
    float grades[QUIZZES], sum;
    int average, i;

    // ask user for grades
    printf("\nWhat were your quiz scores?\n\n");
    for (i = 0; i < QUIZZES; i++)
    {
        printf("Quiz #%d of %d: ", i+1, QUIZZES);
        grades[i] = GetFloat();
    }

    // compute average
    sum = 0;
    for (i = 0; i < QUIZZES; i++)
        sum += grades[i];
    average = (int) (sum / QUIZZES + 0.5);

    // report average
    printf("\nYour average is: %d\n\n", average);
}
```



```
/***************************************************************************
 * array2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Computes a student’s average across 2 quizzes.
 *
 * Demonstrates C’s math library.
 ***************************************************************************/

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


// number of quizzes per term
#define QUIZZES 2


int
main(int argc, char *argv[])
{
    float grades[QUIZZES], sum;
    int average, i;

    // ask user for grades
    printf("\nWhat were your quiz scores?\n\n");
    for (i = 0; i < QUIZZES; i++)
    {
        printf("Quiz #%d of %d: ", i+1, QUIZZES);
        grades[i] = GetFloat();
    }

    // compute average
    sum = 0;
    for (i = 0; i < QUIZZES; i++)
        sum += grades[i];
    average = (int) round(sum / QUIZZES);

    // report average
    printf("\nYour average is: %d\n\n", average);
}
```

```
/*****************************************************************************
 * ascii1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Displays the mapping between alphabetical ASCII characters and
 * their decimal equivalents using one column.
 *
 * Demonstrates casting from int to char.
 *****************************************************************************/

#include <stdio.h>


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

    // display mapping for uppercase letters
    for (i = 65; i < 65 + 26; i++)
        printf("%c: %d\n", (char) i, i);

    // separate uppercase from lowercase
    printf("\n");

    // display mapping for lowercase letters
    for (i = 97; i < 97 + 26; i++)
        printf("%c: %d\n", (char) i, i);
}

```



```
/*****************************************************************************
 * ascii2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Displays the mapping between alphabetical ASCII characters and
 * their decimal equivalents using two columns.
 *
 * Demonstrates specification of width in format string.
 *****************************************************************************/

#include <stdio.h>


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

    // display mapping for uppercase letters
    for (i = 65; i < 65 + 26; i++)
        printf("%c  %d    %3d  %c\n", (char) i, i, i + 32, (char) (i + 32));
}

```

```
/*****************************************************************************
 * ascii3.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Displays the mapping between alphabetical ASCII characters and
 * their decimal equivalents.
 *
 * Demonstrates iteration with a char.
 *****************************************************************************/

#include <stdio.h>


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

    // display mapping for uppercase letters
    for (c = 'A'; c <= 'Z'; c = (char) ((int) c + 1))
        printf("%c: %d\n", c, (int) c);
}
```



```
/*****************************************************************************
 * battleship.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Prints a Battleship board.
 *
 * Demonstrates nested loop.
 *****************************************************************************/

#include <stdio.h>


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

    // print top row of numbers
    printf("\n   ");
    for (i = 1; i <= 10; i++)
        printf("%d  ", i);
    printf("\n");

    // print rows of holes, with letters in leftmost column
    for (i = 0; i < 10; i++)
    {
        printf("%c  ", 'A' + i);
        for (j = 1; j <= 10; j++)
            printf("o  ");
        printf("\n");
    }
    printf("\n");
}
```

```
/*****************************************************************************
 * beer1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Sings "99 Bottles of Beer on the Wall."
 *
 * Demonstrates a for loop (and an opportunity for hierarchical
 * decomposition).
 *****************************************************************************/

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


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

    // ask user for number
    printf("How many bottles will there be? ");
    n = GetInt();

    // exit upon invalid input
    if (n < 1)
    {
        printf("Sorry, that makes no sense.\n");
        return 1;
    }

    // sing the annoying song
    printf("\n");
    for (i = n; i > 0; i--)
    {
        printf("%d bottle(s) of beer on the wall,\n", i);
        printf("%d bottle(s) of beer,\n", i);
        printf("Take one down, pass it around,\n");
        printf("%d bottle(s) of beer on the wall.\n\n", i - 1);
    }

    // exit when song is over
    printf("Wow, that’s annoying.\n");
    return 0;
}
```



```
/*****************************************************************************
 * beer2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Sings "99 Bottles of Beer on the Wall."
 *
 * Demonstrates a while loop (and an opportunity for hierarchical
 * decomposition).
 *****************************************************************************/

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


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

    // ask user for number
    printf("How many bottles will there be? ");
    n = GetInt();

    // exit upon invalid input
    if (n < 1)
    {
        printf("Sorry, that makes no sense.\n");
        return 1;
    }

    // sing the annoying song
    printf("\n");
    while (n > 0)
    {
        printf("%d bottle(s) of beer on the wall,\n", n);
        printf("%d bottle(s) of beer,\n", n);
        printf("Take one down, pass it around,\n");
        printf("%d bottle(s) of beer on the wall.\n\n", n - 1);
        n--;
    }

    // exit when song is over
    printf("Wow, that’s annoying.\n");
    return 0;
}
```

```
/*****************************************************************************
 * beer3.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Sings "99 Bottles of Beer on the Wall."
 *
 * Demonstrates a condition within a for loop.
 *****************************************************************************/

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


int
main(int argc, char *argv[])
{
    int i, n;
    string s1, s2;

    // ask user for number
    printf("How many bottles will there be? ");
    n = GetInt();

    // exit upon invalid input
    if (n < 1)
    {
        printf("Sorry, that makes no sense.\n");
        return 1;
    }

    // sing the annoying song
    printf("\n");
    for (i = n; i > 0; i--)
    {
        // use proper grammar
        s1 = (i == 1) ? "bottle" : "bottles";
        s2 = (i == 2) ? "bottle" : "bottles";

        // sing verses
        printf("%d %s of beer on the wall,\n", i, s1);
        printf("%d %s of beer,\n", i, s1);
        printf("Take one down, pass it around,\n");
        printf("%d %s of beer on the wall.\n\n", i - 1, s2);
    }

    // exit when song is over
    printf("Wow, that's annoying.\n");
    return 0;
}
```



```
/*****************************************************************************
 * beer4.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Sings "99 Bottles of Beer on the Wall."
 *
 * Demonstrates hierarchical decomposition and parameter passing.
 *****************************************************************************/

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


// function prototype
void chorus(int);


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

    // ask user for number
    printf("How many bottles will there be? ");
    n = GetInt();

    // exit upon invalid input
    if (n < 1)
    {
        printf("Sorry, that makes no sense.\n");
        return 1;
    }

    // sing the annoying song
    printf("\n");
    while (n)
        chorus(n--);

    // exit when song is over
    printf("Wow, that's annoying.\n");
    return 0;
}


/*
 * void
 * chorus(int bottles)
 *
 * Sings about specified number of bottles.
 */

void
chorus(int b)
{
    string s1, s2;

    // use proper grammar
    s1 = (b == 1) ? "bottle" : "bottles";
    s2 = (b == 2) ? "bottle" : "bottles";

    // sing verses
    printf("%d %s of beer on the wall,\n", b, s1);
```

```
    printf("%d %s of beer,\n", b, s1);
    printf("Take one down, pass it around,\n");
    printf("%d %s of beer on the wall.\n\n", b - 1, s2);
}
```



```
/*****************************************************************************
 * buggy1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Should print 10 asterisks but doesn't!
 * Can you find the bug?
 *****************************************************************************/

#include <stdio.h>

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

    for (i = 0; i <= 10; i++)
        printf("*");
}
```

buggy2.c

lectures/2/src/


```
/*****************************************************************************
 * buggy2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Should print 10 asterisks, one per line, but doesn't!
 * Can you find the bug?
 *****************************************************************************/

#include <stdio.h>

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

    for (i = 0; i <= 10; i++)
        printf("*");
        printf("\n");
}
```

buggy3.c

lectures/2/src/


```
/*****************************************************************************
 * buggy3.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Should swap two variables' values, but doesn't!
 * Can you find the bug?
 *****************************************************************************/

#include <stdio.h>


// function prototype
void swap(int, int);


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


/*
 * void
 * swap(int a, int b)
 *
 * Swap arguments' values.
 */

void
swap(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
}
```

```
 1: /*****************************************************************************
 2:  * buggy4.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Should increment a variable, but doesn't!
 8:  * Can you find the bug?
 9:  *****************************************************************************/
10:
11: #include <stdio.h>
12:
13:
14: // function prototype
15: void increment();
16:
17:
18: int
19: main(int argc, char *argv[])
20: {
21:     int x = 1;
22:     printf("x is now %d\n", x);
23:     printf("Incrementing...\n");
24:     increment();
25:     printf("Incremented!\n");
26:     printf("x is now %d\n", x);
27: }
28:
29:
30: /*
31:  * void
32:  * increment()
33:  *
34:  * Tries to increment x.
35:  */
36:
37: void
38: increment()
39: {
40:     x++;
41: }
```



```
 1: /*****************************************************************************
 2:  * buggy5.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Should increment a variable, but doesn't!
 8:  * Can you find the bug?
 9:  *****************************************************************************/
10:
11: #include <stdio.h>
12:
13:
14: // global variable
15: int x;
16:
17: // function prototype
18: void increment();
19:
20:
21: int
22: main(int argc, char *argv[])
23: {
24:     printf("x is now %d\n", x);
25:     printf("Initializing...\n");
26:     x = 1;
27:     printf("Initialized!\n");
28:     printf("x is now %d\n", x);
29:     printf("Incrementing...\n");
30:     increment();
31:     printf("Incremented!\n");
32:     printf("x is now %d\n", x);
33: }
34:
35:
36: /*
37:  * void
38:  * increment()
39:  *
40:  * Increments x.
41:  */
42:
43: void
44: increment()
45: {
46:     int x = 10;
47:     x++;
48: }
```

```c
/*****************************************************************************
 * buggy6.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Asks student for their grades but prints too many!
 * Can you find the bug?
 *
 * Demonstrates accidental use of a "magic number."
 *****************************************************************************/

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


// number of quizzes per term
#define QUIZZES 2


int
main(int argc, char *argv[])
{
    float grades[QUIZZES];
    int i;

    // ask user for scores
    printf("\nWhat were your quiz scores?\n\n");
    for (i = 0; i < QUIZZES; i++)
    {
        printf("Quiz #%d of %d: ", i+1, QUIZZES);
        grades[i] = GetFloat();
    }

    // print scores
    for (i = 0; i < 3; i++)
        printf("%.2f\n", grades[i]);
}
```



```c
/*****************************************************************************
 * capitalize.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Capitalizes a given string.
 *
 * Demonstrates casting and iteration over strings as arrays of chars.
 *****************************************************************************/

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


int
main(int argc, char *argv[])
{
    int i, n;
    string s;

    // get line of text
    s = GetString();

    // capitalize text
    for (i = 0, n = strlen(s); i < n; i++)
    {
        if (s[i] >= 'a' && s[i] <= 'z')
            printf("%c", s[i] - ('a' - 'A'));
        else
            printf("%c", s[i]);
    }
    printf("\n");
}
```

```
/*****************************************************************************
 * cs50.c
 *
 * version 1.1.2
 *
 * Computer Science 50
 * Glenn Holloway
 * David J. Malan
 *
 * Definitions for CS 50's library.
 * Based on Eric Roberts' genlib.c and simpio.c.
 *
 * The latest version of this file can be found at
 * http://cs50.net/pub/releases/cs50/cs50.c.
 *
 * To compile as a static library on your own system:
 * % gcc -c -ggdb -std=c99 cs50.c -o cs50.o
 * % ar rcs libcs50.a cs50.o
 * % rm -f cs50.o
 * % cp cs50.h /usr/local/include
 * % cp libcs50.a /usr/local/lib
 *****************************************************************************/

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

#include "cs50.h"


/*
 * Default capacity of buffer for standard input.
 */

#define CAPACITY 128


/*
 * char
 * GetChar()
 *
 * 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()
{
    // 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: ");
        }
    }
}


/*
 * double
 * GetDouble()
 *
 * 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()
{
    // 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: ");
        }
    }
}


/*
 * float
 * GetFloat()
 *
 * 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()
{
```

```
    // 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: ");
        }
    }
}


/*
 * int
 * GetInt()
 *
 * 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()
{
    // 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: ");
        }
    }
}


```



```
/*
 * long long
 * GetLongLong()
 *
 * 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()
{
    // 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: ");
        }
    }
}


/*
 * string
 * GetString()
 *
 * Reads a line of text from standard input and returns it as a string,
 * sans trailing newline character.  (Ergo, if user inputs only "\n",
 * returns "" not NULL.)  Leading and trailing whitespace is not ignored.
 * Returns NULL upon error or no input whatsoever (i.e., just EOF).
 */

string
GetString()
{
    // 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 CAPACITY then double
            if (capacity == 0)
                capacity = CAPACITY;
            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;
}

```



```
/*****************************************************************************
 * cs50.h
 *
 * version 1.1.2
 *
 * Computer Science 50
 * Glenn Holloway
 * David J. Malan
 *
 * Declarations for CS 50's library.
 * Based on Eric Roberts' genlib.h and simpio.h.
 *
 * The latest version of this file can be found at
 * http://cs50.net/pub/releases/cs50/cs50.h.
 *
 * To compile as a static library on your own system:
 * % gcc -c -ggdb -std=c99 cs50.c -o cs50.o
 * % ar rcs libcs50.a cs50.o
 * % rm -f cs50.o
 * % cp cs50.h /usr/local/include
 * % cp libcs50.a /usr/local/lib
 *****************************************************************************/

#ifndef _CS50_H
#define _CS50_H

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


/*
 * bool
 *
 * Borrow the standard library's data type for Boolean variables whose
 * values must be (true|false).
 */

#include <stdbool.h>


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

typedef char *string;


/*
 * char
 * GetChar()
 *
 * 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();


/*
 * double
```

```
 * GetDouble()
 *
 * 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();


/*
 * float
 * GetFloat()
 *
 * 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();


/*
 * int
 * GetInt()
 *
 * 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();


/*
 * long long
 * GetLongLong()
 *
 * 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();


/*
 * string
 * GetString()
 *
 * Reads a line of text from standard input and returns it as a string,
 * sans trailing newline character.  (Ergo, if user inputs only "\n",
 * returns "" not NULL.)  Leading and trailing whitespace is not ignored.
 * Returns NULL upon error or no input whatsoever (i.e., just EOF).
 */

```



```
string GetString();


#endif

```

global.c

lectures/2/src/

```
/*****************************************************************************
 * global.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Increments variables.
 *
 * Demonstrates use of global variable and issue of scope.
 *****************************************************************************/

#include <stdio.h>


// global variable
int x;

// function prototype
void increment();


int
main(int argc, char *argv[])
{
    printf("x is now %d\n", x);
    printf("Initializing...\n");
    x = 1;
    printf("Initialized!\n");
    printf("x is now %d\n", x);
    printf("Incrementing...\n");
    increment();
    printf("Incremented!\n");
    printf("x is now %d\n", x);
}


/*
 * void
 * increment()
 *
 * Increments x.
 */

void
increment()
{
    x++;
}
```

hai.cc

lectures/2/src/

```
#include <iostream>

using namespace std;

int
main(int argc, char * argv[])
{
    cout << "O hai, world!" << endl;
}
```

hai.lisp

1/1

lectures/2/src/

```
1: (print "O hai, world!")
```

hai.php

1/1

lectures/2/src/

```
1: <?
2:     echo "O hai, world!\n";
3: ?>
```

hai.pl

lectures/2/src/


```
MAIN:
{
    print "O hai, world!\n";
}
```

return1.c

lectures/2/src/


```
/****************************************************************************
 * return1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Increments a variable.
 *
 * Demonstrates use of parameter and return value.
 ****************************************************************************/

#include <stdio.h>


// function prototype
int increment(int);


int
main(int argc, char *argv[])
{
    int x = 1;
    printf("x is now %d\n", x);
    printf("Incrementing...\n");
    x = increment(x);
    printf("Incremented!\n");
    printf("x is now %d\n", x);
}


/*
 * int
 * increment(int a)
 *
 * Returns argument plus one.
 */

int
increment(int a)
{
    return a + 1;
}
```

```
/*****************************************************************************
 * return2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Cubes a variable.
 *
 * Demonstrates use of parameter and return value.
 *****************************************************************************/

#include <stdio.h>


// function prototype
int cube(int);


int
main(int argc, char *argv[])
{
    int x = 1;
    printf("x is now %d\n", x);
    printf("Cubing...\n");
    x = cube(x);
    printf("Cubed!\n");
    printf("x is now %d\n", x);
}


/*
 * int
 * cube(int a)
 *
 * Cubes argument.
 */

int
cube(int a)
{
    return a * a * a;
}
```



```
/*****************************************************************************
 * string1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Prints a given string one character per line.
 *
 * Demonstrates strings as arrays of chars and use of strlen.
 *****************************************************************************/

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


int
main(int argc, char *argv[])
{
    char c;
    int i;
    string s;

    // get line of text
    s = GetString();

    // print string, one character per line
    if (s != NULL)
    {
        for (i = 0; i < strlen(s); i++)
        {
            c = s[i];
            printf("%c\n", c);
        }
    }
}
```

```
/*****************************************************************************
 * string2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Prints a given string one character per line.
 *
 * Demonstrates strings as arrays of chars with slight optimization.
 *****************************************************************************/

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


int
main(int argc, char *argv[])
{
    int i, n;
    string s;

    // get line of text
    s = GetString();

    // print string, one character per line
    if (s != NULL)
    {
        for (i = 0, n = strlen(s); i < n; i++)
        {
            printf("%c\n", s[i]);
        }
    }
}
```