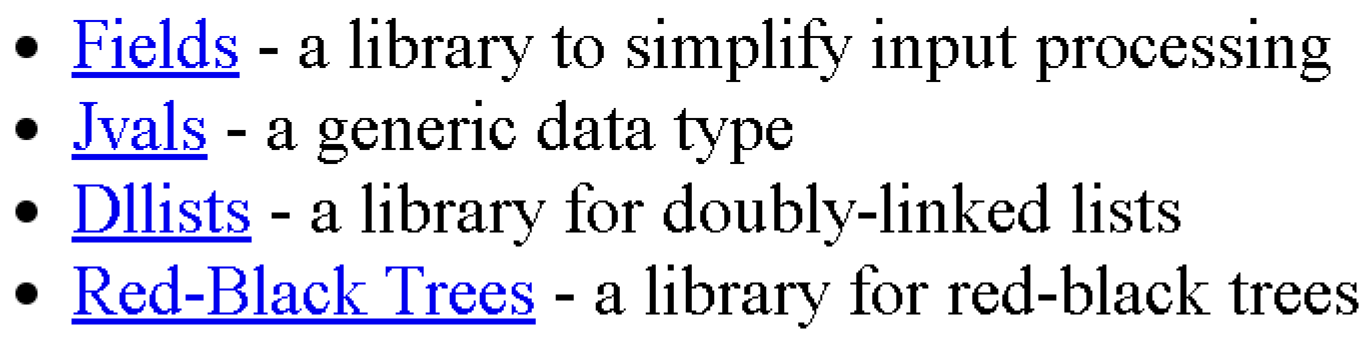
BSM-308System Programming

Sakarya University-Computer Engineering

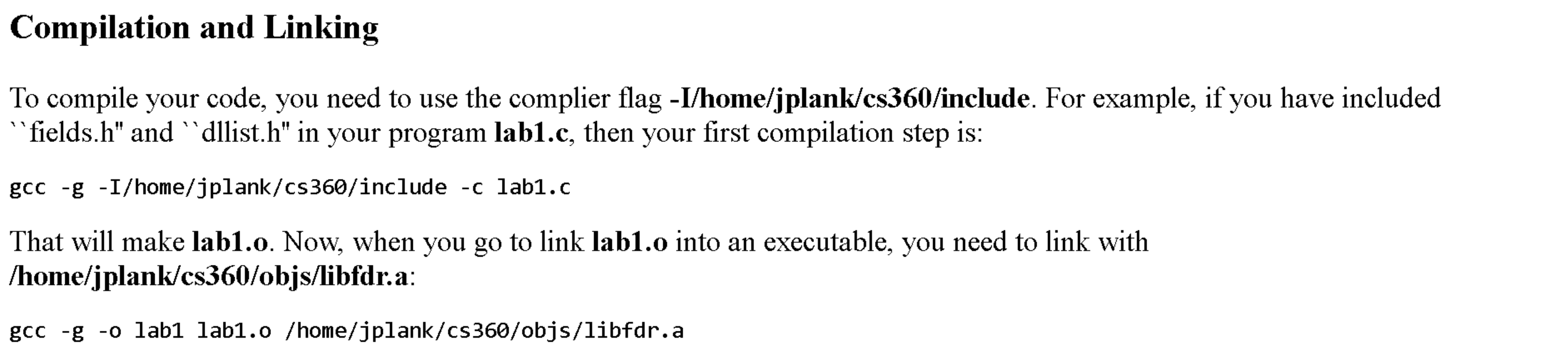
Contents



- Fields - a library to simplify input processing
- Jvals - a generic data type
- Dllists - a library for doubly-linked lists
- Red-Black Trees - a library for red-black trees

- http://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/Libfdr/index.html
- http://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/Fields/index.html

Jvals , Fields, Dllists , Red-Black Trees



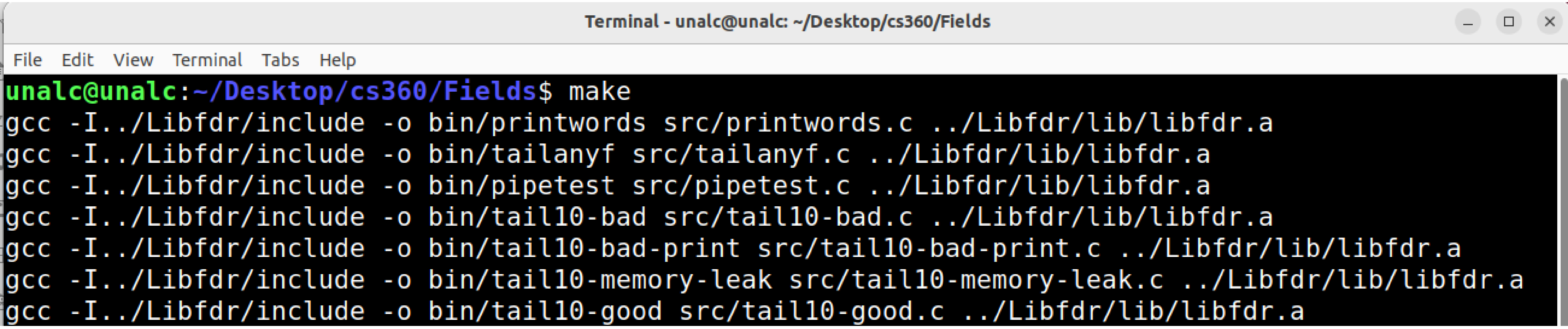
Compilation and Linking

To compile your code, you need to use the complier flag **-I/home/jplank/cs360/include**. For example, if you have included ``fields.h" and ``dllist.h" in your program **lab1.c**, then your first compilation step is:

```
gcc -g -I/home/jplank/cs360/include -c lab1.c
```

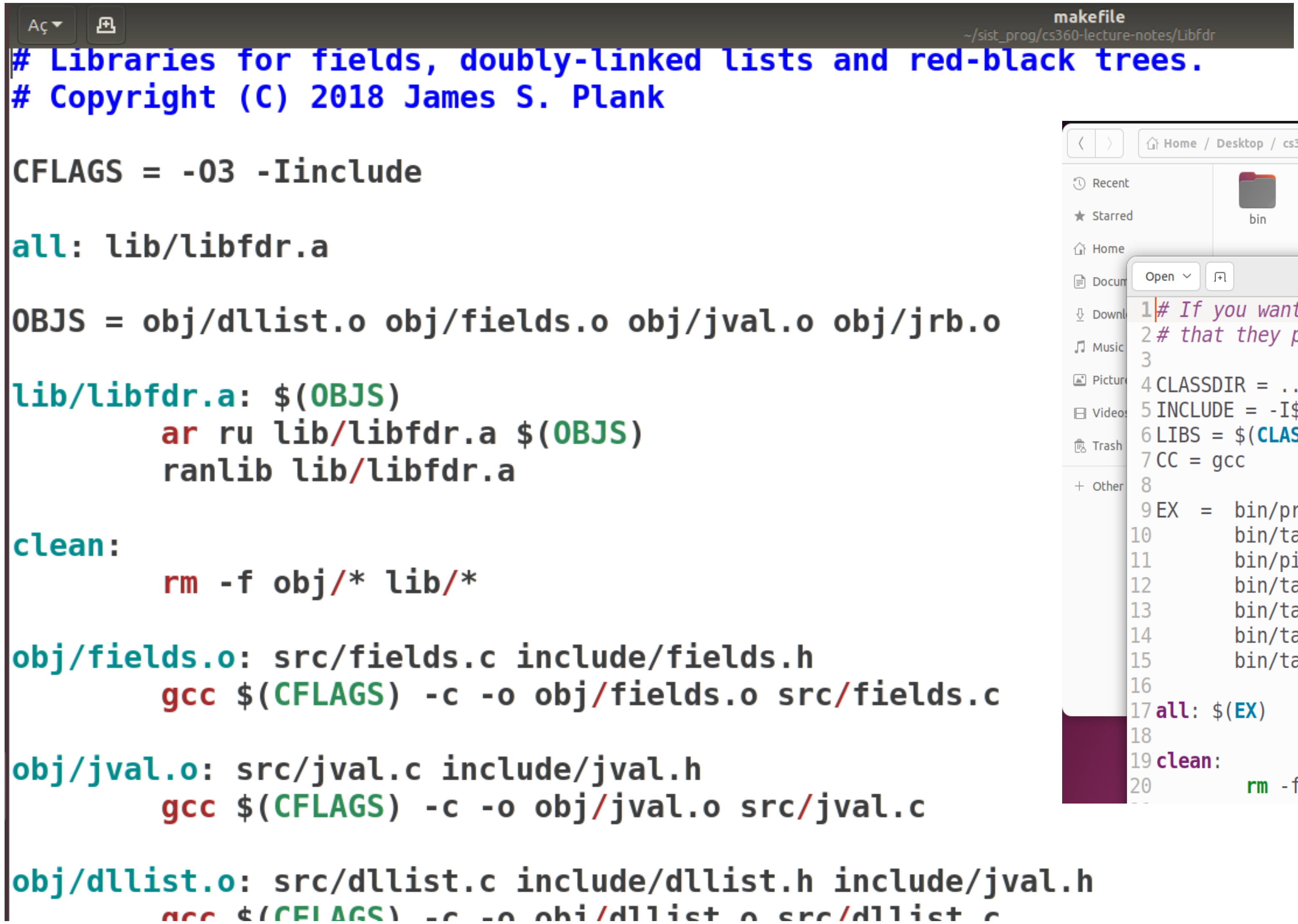
That will make **lab1.o**. Now, when you go to link **lab1.o** into an executable, you need to link with **/home/jplank/cs360/objs/libfdr.a**:

```
gcc -g -o lab1 lab1.o /home/jplank/cs360/objs/libfdr.a
```

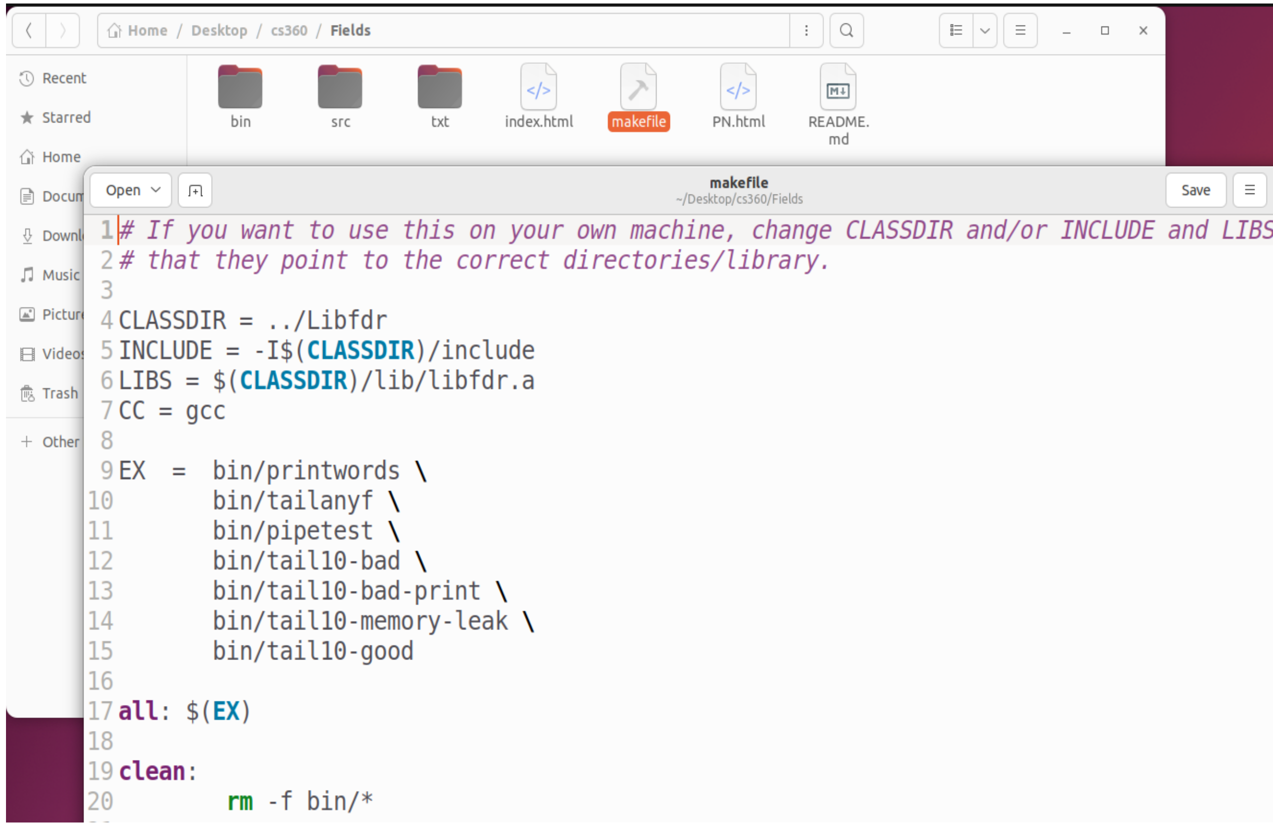

Terminal - unalc@unalc: ~/Desktop/cs360/Fields
File Edit View Terminal Tabs Help

```
unalc@unalc:~/Desktop/cs360/Fields$ make
gcc -I../Libfdr/include -o bin/printwords src/printwords.c ../Libfdr/lib/libfdr.a
gcc -I../Libfdr/include -o bin/tailanyf src/tailanyf.c ../Libfdr/lib/libfdr.a
gcc -I../Libfdr/include -o bin/pipetest src/pipetest.c ../Libfdr/lib/libfdr.a
gcc -I../Libfdr/include -o bin/tail10-bad src/tail10-bad.c ../Libfdr/lib/libfdr.a
gcc -I../Libfdr/include -o bin/tail10-bad-print src/tail10-bad-print.c ../Libfdr/lib/libfdr.a
gcc -I../Libfdr/include -o bin/tail10-memory-leak src/tail10-memory-leak.c ../Libfdr/lib/libfdr.a
gcc -I../Libfdr/include -o bin/tail10-good src/tail10-good.c ../Libfdr/lib/libfdr.a
```

makefile



```
# Libraries for fields, doubly-linked lists and red-black trees.
# Copyright (C) 2018 James S. Plank

CFLAGS = -O3 -Iinclude

all: lib/libfdr.a

OBJS = obj/dllist.o obj/fields.o obj/jval.o obj/jrb.o

lib/libfdr.a: $(OBJS)
        ar ru lib/libfdr.a $(OBJS)
        ranlib lib/libfdr.a

clean:
        rm -f obj/* lib/*

obj/fields.o: src/fields.c include/fields.h
        gcc $(CFLAGS) -c -o obj/fields.o src/fields.c

obj/jval.o: src/jval.c include/jval.h
        gcc $(CFLAGS) -c -o obj/jval.o src/jval.c

obj/dllist.o: src/dllist.c include/dllist.h include/jval.h
        gcc $(CFLAGS) -c -o obj/dllist.o src/dllist.c
```



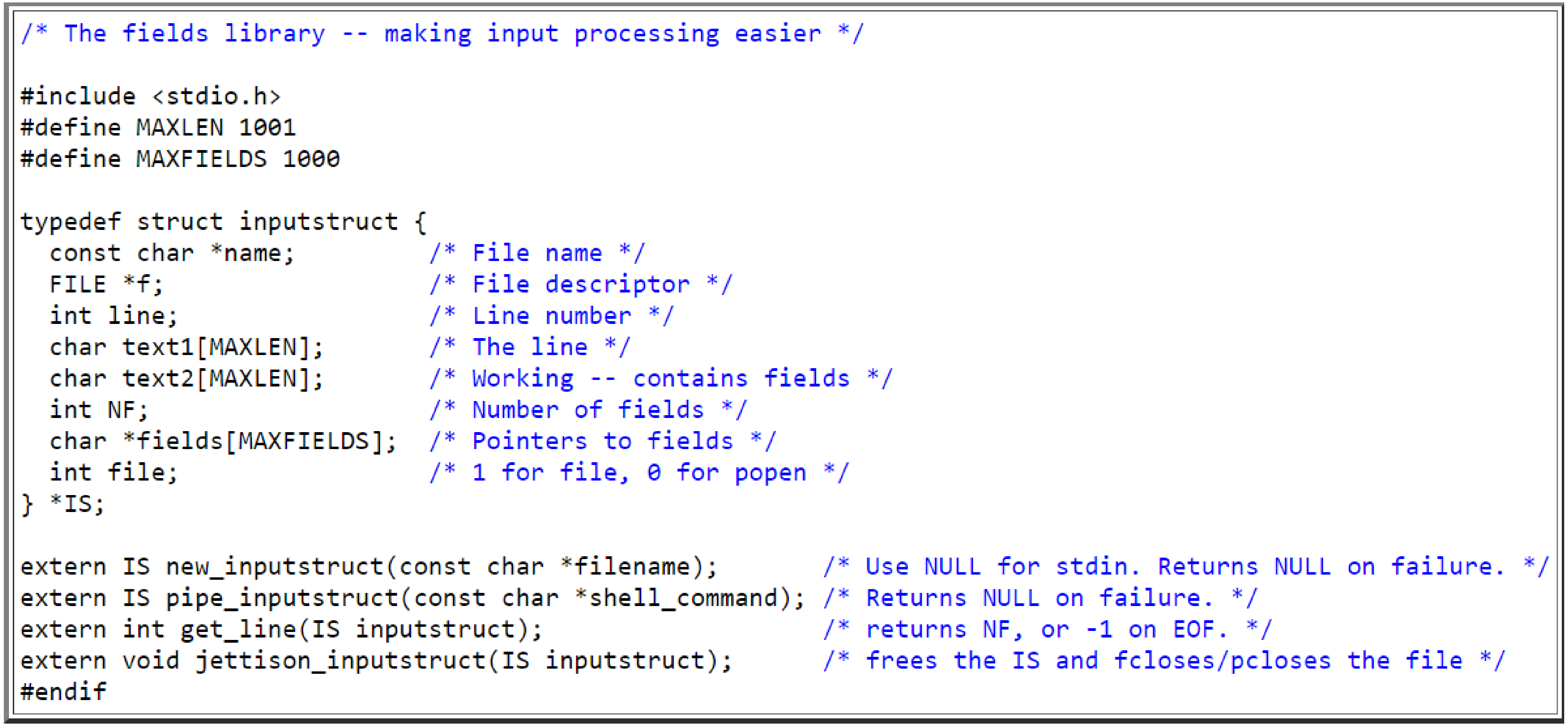
```
# If you want to use this on your own machine, change CLASSDIR and/or INCLUDE and LIBS
# that they point to the correct directories/library.

CLASSDIR = ../Libfdr
INCLUDE = -I$(CLASSDIR)/include
LIBS = $(CLASSDIR)/lib/libfdr.a
CC = gcc

EX  =  bin/printwords \
       bin/tailanyf \
       bin/pipetest \
       bin/tail10-bad \
       bin/tail10-bad-print \
       bin/tail10-memory-leak \
       bin/tail10-good

all: $(EX)

clean:
        rm -f bin/*
```

fields

- ❑**The Fields** library is a suite of routines that make reading input easier than using getchar (), scanf (), or fgets ().
- ❑A library written by the authors from the source -- not standard on Unix, but should work with any C compiler (this includes DOS/Windows).
- ❑The source code is in this repository, in the " Libfdr " directory.
- ❑To use **Fields procedures** in this class , you must include the fields.h file.
- ❑Instead of including the full pathname in your C file, do this:

```
#include "fields.h"
```

fields.h



```c
/* The fields library -- making input processing easier */

#include <stdio.h>
#define MAXLEN 1001
#define MAXFIELDS 1000

typedef struct inputstruct {
  const char *name;          /* File name */
  FILE *f;                   /* File descriptor */
  int line;                  /* Line number */
  char text1[MAXLEN];        /* The line */
  char text2[MAXLEN];        /* Working -- contains fields */
  int NF;                    /* Number of fields */
  char *fields[MAXFIELDS];   /* Pointers to fields */
  int file;                  /* 1 for file, 0 for popen */
} *IS;

extern IS new_inputstruct(const char *filename);       /* Use NULL for stdin. Returns NULL on failure. */
extern IS pipe_inputstruct(const char *shell_command); /* Returns NULL on failure. */
extern int get_line(IS inputstruct);                   /* returns NF, or -1 on EOF. */
extern void jettison_inputstruct(IS inputstruct);      /* frees the IS and fcloses/pcloses the file */
#endif
```

fields

- ❑To read a file with the **Fields library, you call** new_inputstruct () .
- ❑New_inputstruct () takes the filename as the input parameter (NULL for standard input) and returns an IS as a result.
- ❑ A struct of IS Notice that there is a pointer to the inputstruct .
- ❑This definition is available in the new_inputstruct () call malloc ()'.

```
#define talloc(ty, sz) (ty *) malloc (sz * sizeof(ty))
```

- ❑new_inputstruct () cannot open the file, it will return NULL and you can call perror () to print the reason for the error (see perror () if you want to know about this read the man page).

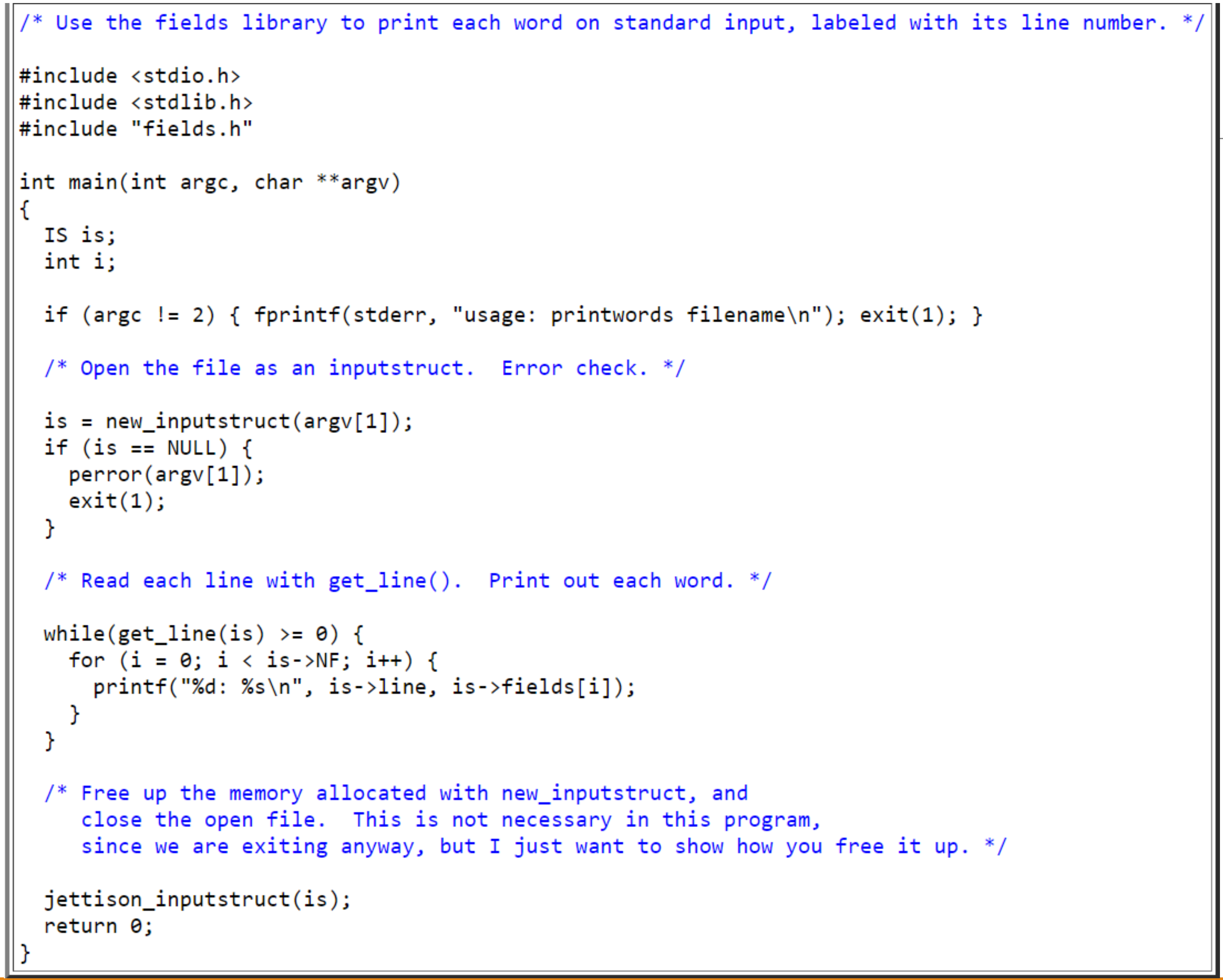
fields

- get_line () to read a line . Get_line () changes the state of the IS to reflect the reading of the line . especially:

1. Puts the contents of the line into text1.
2. Divides the line into words. NF field contains the number of words in the field . The NF slots of the Fields array refer to each of the NF words (and these words are null-terminated).
3. line field contains the line number of the row.
4. Get_line () returns NF field as return value.
5. Returns -1 when it reaches the end of the file.

- Jettison_inputstruct () closes the file associated with the IS and releases (releases) the IS .

src / printwords.c



```c
/* Use the fields library to print each word on standard input, labeled with its line number. */

#include <stdio.h>
#include <stdlib.h>
#include "fields.h"

int main(int argc, char **argv)
{
  IS is;
  int i;

  if (argc != 2) { fprintf(stderr, "usage: printwords filename\n"); exit(1); }

  /* Open the file as an inputstruct.  Error check. */

  is = new_inputstruct(argv[1]);
  if (is == NULL) {
    perror(argv[1]);
    exit(1);
  }

  /* Read each line with get_line().  Print out each word. */

  while(get_line(is) >= 0) {
    for (i = 0; i < is->NF; i++) {
      printf("%d: %s\n", is->line, is->fields[i]);
    }
  }

  /* Free up the memory allocated with new_inputstruct, and
     close the open file.  This is not necessary in this program,
     since we are exiting anyway, but I just want to show how you free it up. */

  jettison_inputstruct(is);
  return 0;
}
```

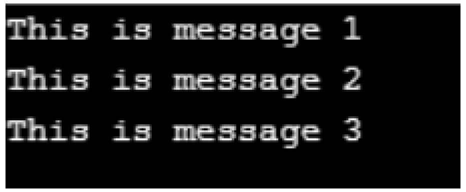
- These procedures are well suited for processing input files. For example, the following program (in src / printwords.c) prints each word in an input file, preceded by its line number.

```
UNIX> bin/printwords txt/rex-1.txt
1: June:
1: Hi
1: ...
1: I
1: missed
1: you!
2: Rex:
2: Same
2: here!
2: You're
2: all
2: I
2: could
2: think
2: about!
3: June:
3: I
3: was?
UNIX>
```

stdderr

- In the C programming language, there are different file descriptors, also known as standard output.
- stdin for standard input ,
- stdout for standard output,
- There are 3 standard I/O methods, including stderr for error message output .

```
# include < stdio.h > int main () { fprintf (
stdout , "This is message 1\n" ); fprintf (
stderr , "This is message 2\n" ); fprintf (
stdout , "This is message 3\n" ); return ( 0 );
}
```



```
This is message 1
This is message 2
This is message 3
```

stdin

```
#include < stdio.h >
int main( ) {

   char a[ 100 ];

   fprintf ( "Enter a string :" );
   fscanf ( stdin, "%s " , a);

   fprintf ( stdout , "\ nYou entered the following string: %s " , a);

   fprintf ( stdout , “\n” );
   return 0 ;

}
```

fgets

```
char * fgets ( char * str , int n , FILE * stream )
```

Parameters

- **str** - This is a pointer to a character array where the read string is stored.
- **n** – This is the maximum number of characters to read (including the last null character). Usually the length of the array passed as str is used.
- **stream** – This is a pointer to a FILE object that defines the stream from which characters are read.

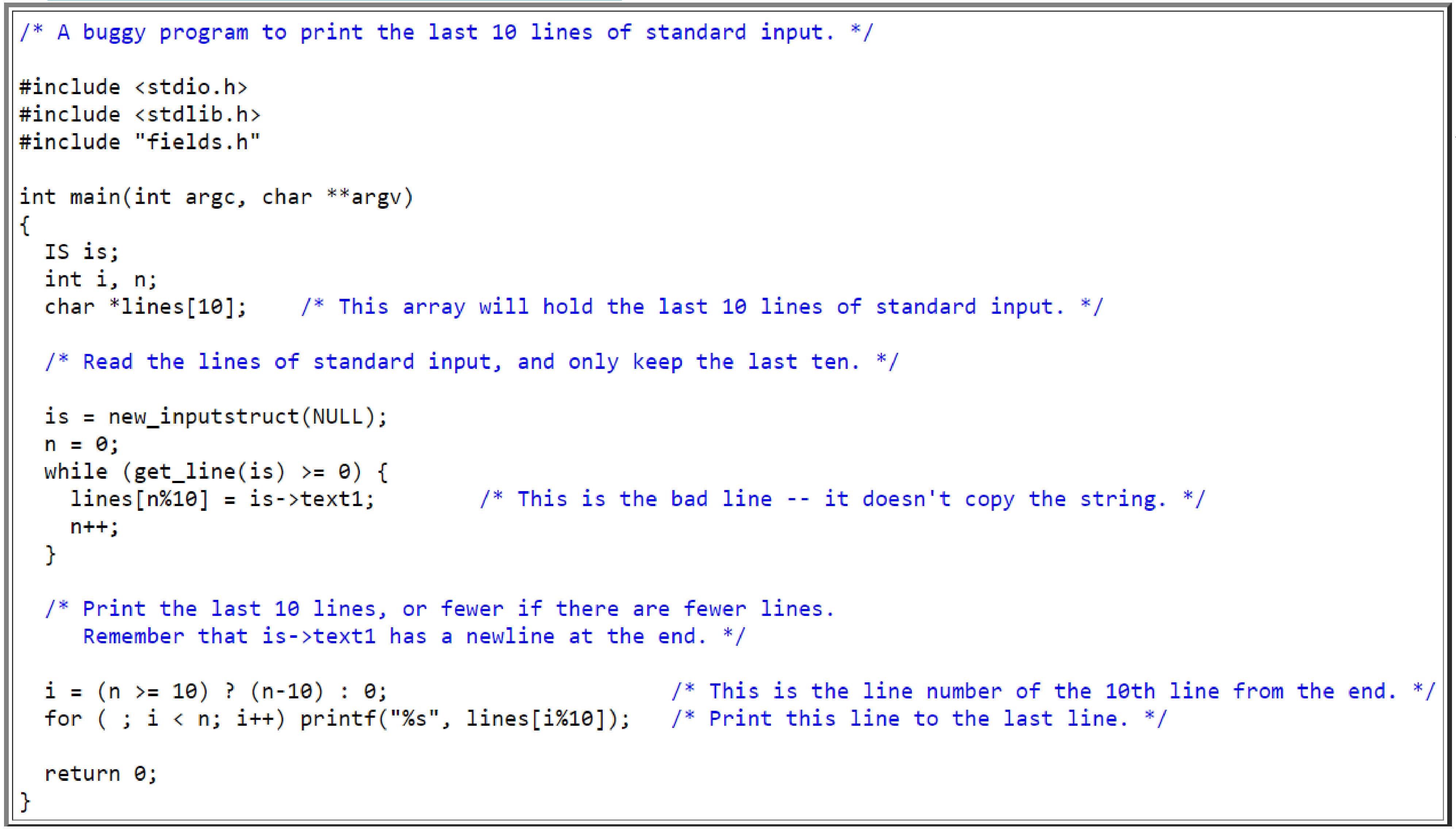
fields

- An important thing to note about the Fields library is that the only time malloc () is called is during new_inputstruct ().
- Get_line () just populates the fields of the IS structure --- it does not perform memory allocation.
- means that if you want to store a line or field and not want it to be overwritten by the next call to get_line (), you usually need to create a copy of it with strdup ().
- This is very important, in general the most common mistake made with fgets () is not making a copy when they need one.
- We will show you this error here to help you with pointers and malloc ().

fields

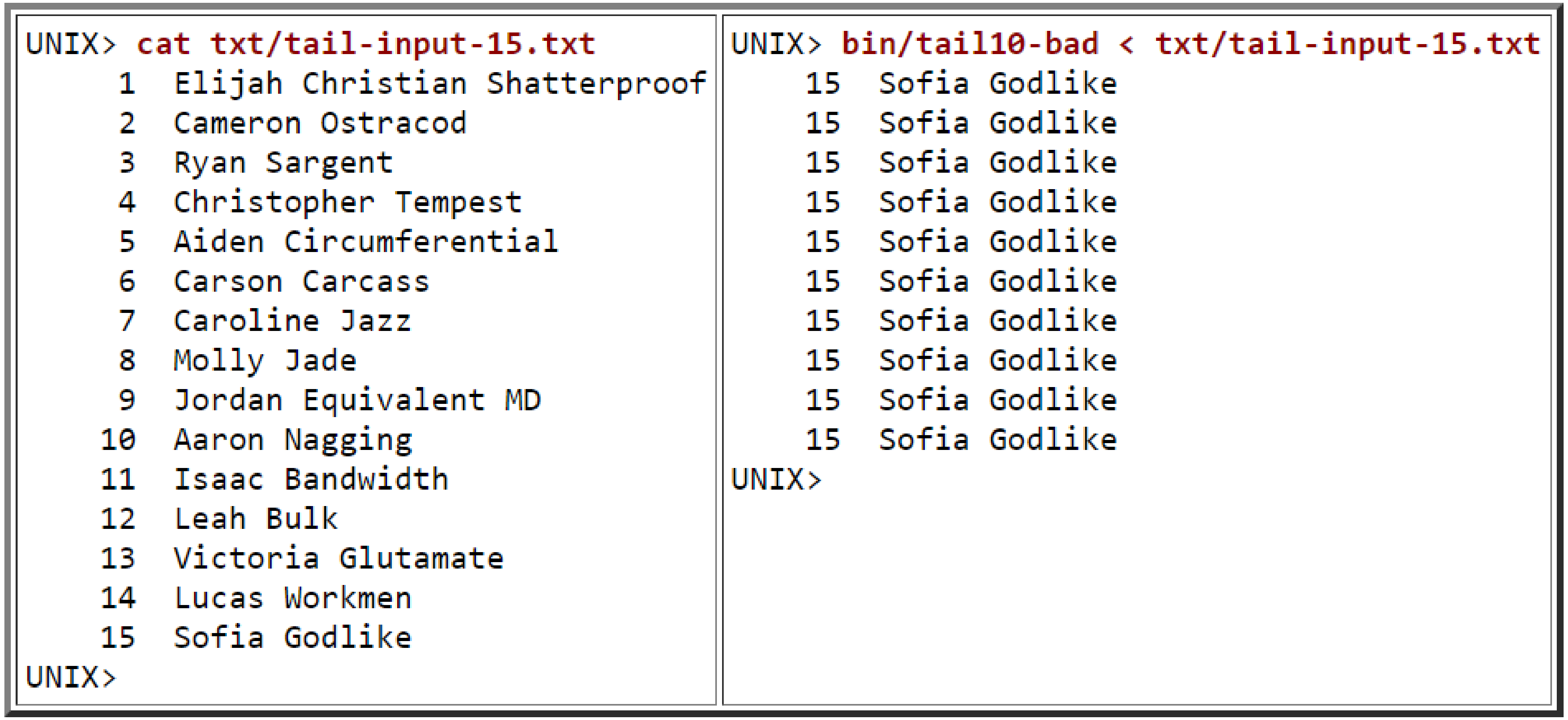
- Our goal will be to write a program queue that prints the last n lines of standard input.
- The value of n is 10 by default, but you must be able to specify this on the command line.
- src /tail10-bad.c , which will try to print the last 10 lines using the Fields library .
- This will show the common error mentioned above.
- Here is the code which is pretty simple. We will have a string of 10 characters *, we will simply set it to is->text1 as we read each line:

src /tail10-bad.c ,



```
/* A buggy program to print the last 10 lines of standard input. */

#include <stdio.h>
#include <stdlib.h>
#include "fields.h"

int main(int argc, char **argv)
{
  IS is;
  int i, n;
  char *lines[10];    /* This array will hold the last 10 lines of standard input. */

  /* Read the lines of standard input, and only keep the last ten. */

  is = new_inputstruct(NULL);
  n = 0;
  while (get_line(is) >= 0) {
    lines[n%10] = is->text1;        /* This is the bad line -- it doesn't copy the string. */
    n++;
  }

  /* Print the last 10 lines, or fewer if there are fewer lines.
     Remember that is->text1 has a newline at the end. */

  i = (n >= 10) ? (n-10) : 0;                    /* This is the line number of the 10th line from the end. */
  for ( ; i < n; i++) printf("%s", lines[i%10]);   /* Print this line to the last line. */

  return 0;
}
```

src /tail10-bad.c ,



```
UNIX> cat txt/tail-input-15.txt
     1  Elijah Christian Shatterproof
     2  Cameron Ostracod
     3  Ryan Sargent
     4  Christopher Tempest
     5  Aiden Circumferential
     6  Carson Carcass
     7  Caroline Jazz
     8  Molly Jade
     9  Jordan Equivalent MD
    10  Aaron Nagging
    11  Isaac Bandwidth
    12  Leah Bulk
    13  Victoria Glutamate
    14  Lucas Workmen
    15  Sofia Godlike
UNIX>
```

```
UNIX> bin/tail10-bad < txt/tail-input-15.txt
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
UNIX>
```

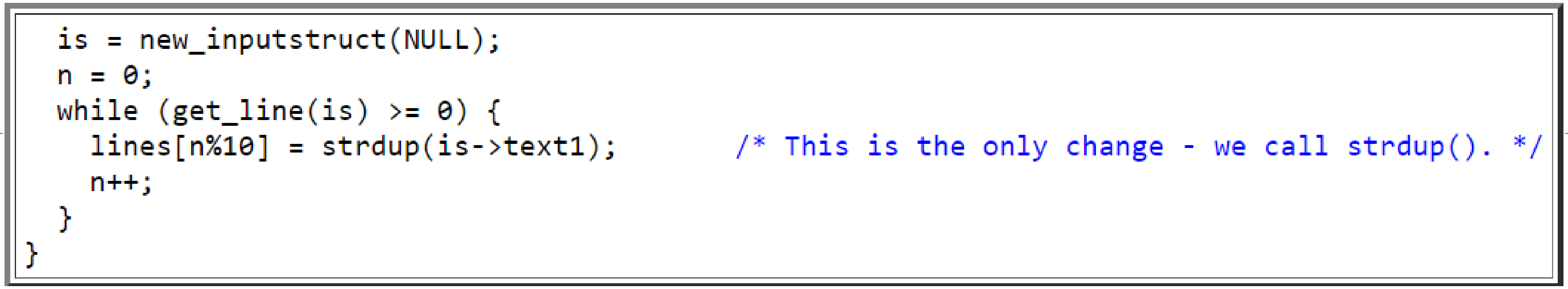
- Cat command, Any One choice without specifying your command itself One of the file their contents will read and on console will show .
- Cat command, standard output on the screen folders create , combine or to suppress permission gives

src /tail10-bad-print.c ,

```
UNIX> bin/tail10-bad-print < txt/tail-input-15.txt
I have set lines[0] to 0x7fc014002614, which is currently      1       Elijah Christian Shatterproof
I have set lines[1] to 0x7fc014002614, which is currently      2       Cameron Ostracod
I have set lines[2] to 0x7fc014002614, which is currently      3       Ryan Sargent
I have set lines[3] to 0x7fc014002614, which is currently      4       Christopher Tempest
I have set lines[4] to 0x7fc014002614, which is currently      5       Aiden Circumferential
I have set lines[5] to 0x7fc014002614, which is currently      6       Carson Carcass
I have set lines[6] to 0x7fc014002614, which is currently      7       Caroline Jazz
I have set lines[7] to 0x7fc014002614, which is currently      8       Molly Jade
I have set lines[8] to 0x7fc014002614, which is currently      9       Jordan Equivalent MD
I have set lines[9] to 0x7fc014002614, which is currently     10       Aaron Nagging
I have set lines[0] to 0x7fc014002614, which is currently     11       Isaac Bandwidth
I have set lines[1] to 0x7fc014002614, which is currently     12       Leah Bulk
I have set lines[2] to 0x7fc014002614, which is currently     13       Victoria Glutamate
I have set lines[3] to 0x7fc014002614, which is currently     14       Lucas Workmen
I have set lines[4] to 0x7fc014002614, which is currently     15       Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
    15  Sofia Godlike
UNIX>
```

```
    printf("I have set lines[%d] to 0x%lx, which is currently %s",
           n%10, (unsigned long) (lines[n%10]), lines[n%10]);
```

src /tail10-memory-leak.c ,



```
  is = new_inputstruct(NULL);
  n = 0;
  while (get_line(is) >= 0) {
    lines[n%10] = strdup(is->text1);       /* This is the only change - we call strdup(). */
    n++;
  }
}
```

```
UNIX> bin/tail10-memory-leak < txt/tail-input-15.txt
      6  Carson Carcass
      7  Caroline Jazz
      8  Molly Jade
      9  Jordan Equivalent MD
     10  Aaron Nagging
     11  Isaac Bandwidth
     12  Leah Bulk
     13  Victoria Glutamate
     14  Lucas Workmen
     15  Sofia Godlike
UNIX>
```

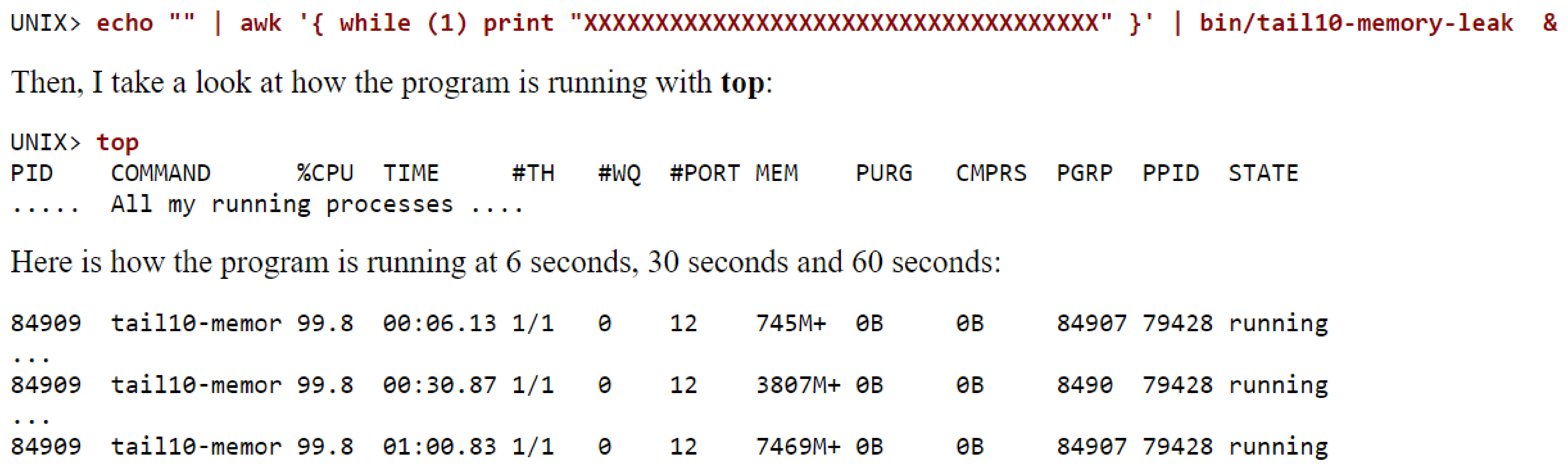
- Simple fix is to use strdup ().
- This will allocate memory for a copy of the row and then copy the row.
- The code is in src /tail10-memory-leak.c, which, as you can tell from the name, will have some problems of its own. The only change is that we no longer assign lines[n%10] to is->text1, but instead create a copy with strdup ().

src /tail10-memory-leak.c ,

- ❑In fact, it works just fine on most inputs.
- ❑But as the name suggests, there is a memory leak.
- ❑When n is greater than or equal to 10, the strdup () line overwrites the pointer currently located on lines [n%10] and the pointer is lost forever.
- ❑However, the memory it points to is still allocated and will not be reallocated until the program exits.
- ❑This is the very definition of a memory leak.
- ❑If we run this on input with a large number of lines, the program's memory usage will explode and eventually your machine will grind to a halt and/or terminate when strdup () fails.

src /tail10-memory-leak.c

- UNIX> **echo "" | awk '{ while (1) print "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" }' | bin/tail10-memory-leak &**
- awk script Prints an infinite number of lines



```
UNIX> echo "" | awk '{ while (1) print "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" }' | bin/tail10-memory-leak  &
```

Then, I take a look at how the program is running with **top**:

```
UNIX> top
PID    COMMAND      %CPU  TIME     #TH   #WQ  #PORT MEM    PURG   CMPRS  PGRP  PPID  STATE
.....  All my running processes ....
```

Here is how the program is running at 6 seconds, 30 seconds and 60 seconds:

```
84909  tail10-memor 99.8  00:06.13 1/1   0    12    745M+  0B     0B     84907 79428 running
...
84909  tail10-memor 99.8  00:30.87 1/1   0    12    3807M+ 0B     0B     8490  79428 running
...
84909  tail10-memor 99.8  01:00.83 1/1   0    12    7469M+ 0B     0B     84907 79428 running
```

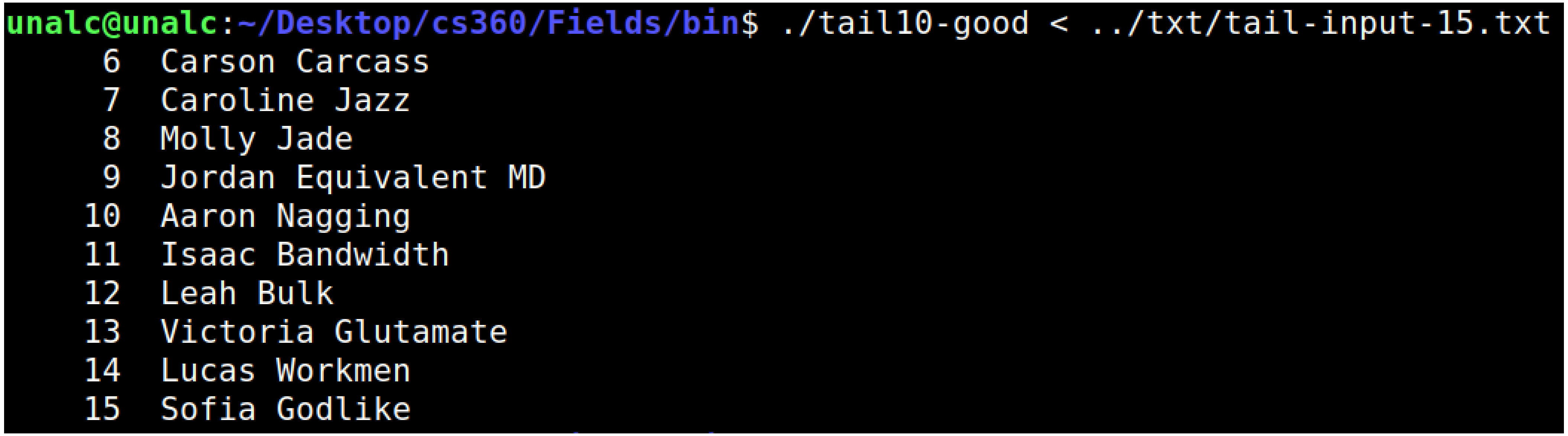
src /tail10-good.c

- So let's fix it. We call strdup () malloc () so that we free the string when we no longer need the string and are about to overwrite the pointer.

```
is = new_inputstruct(NULL);
n = 0;
while (get_line(is) >= 0) {
  if (n >= 10) free(lines[n%10]);    /* This line prevents the memory leak. */
  lines[n%10] = strdup(is->text1);
  n++;
}
```

```
PID    COMMAND     %CPU  TIME     #TH   #WQ  #PORT MEM   PURG  CMPRS  PGRP  PPID  STATE
85101  tail10-good 99.7  00:07.44 1/1   0    12    492K  0B    0B     85099 79428 running
...
85101  tail10-good 99.9  00:30.15 1/1   0    12    492K  0B    0B     85099 79428 running
...
85101  tail10-good 99.9  01:00.11 1/1   0    12    492K  0B    0B     85099 79428 running
```

tail10-good print



```
unalc@unalc:~/Desktop/cs360/Fields/bin$ ./tail10-good < ../txt/tail-input-15.txt
     6  Carson Carcass
     7  Caroline Jazz
     8  Molly Jade
     9  Jordan Equivalent MD
    10  Aaron Nagging
    11  Isaac Bandwidth
    12  Leah Bulk
    13  Victoria Glutamate
    14  Lucas Workmen
    15  Sofia Godlike
```

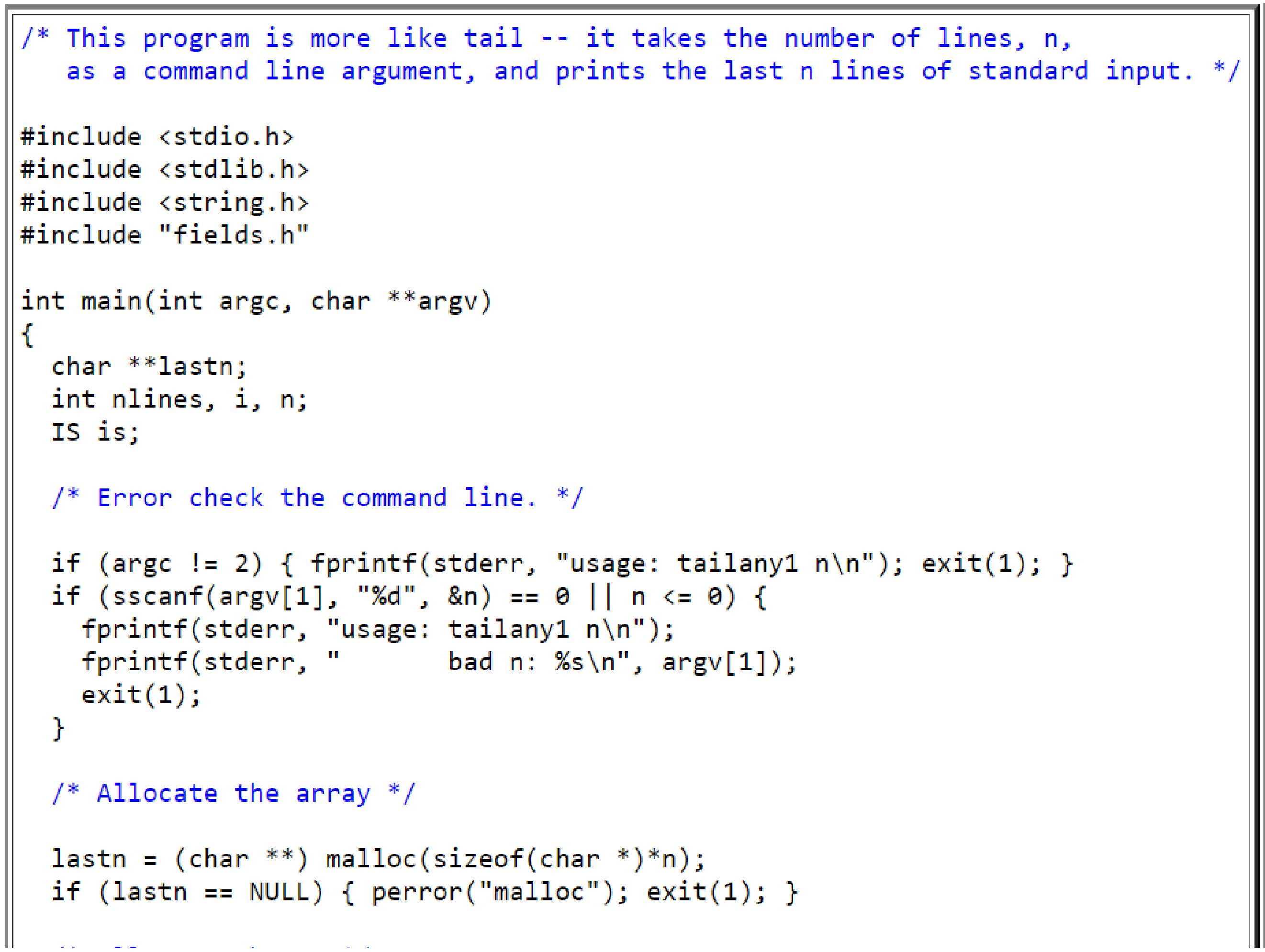
tailanyf

of tail where you specify the number of lines on the command line .

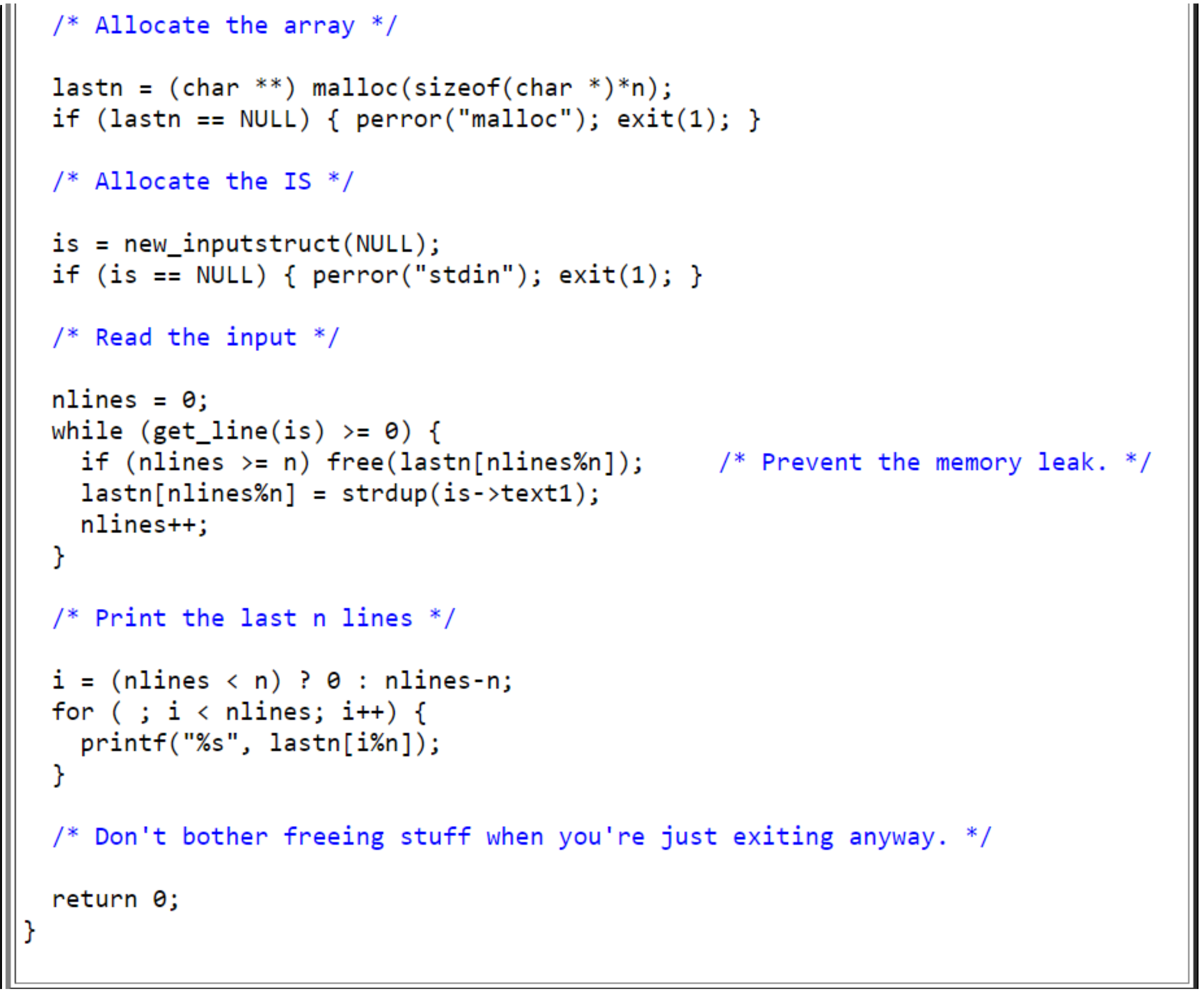
This program shows a few things you'll need to get used to:

1. Using sscanf () to convert a string to an integer and test if it works .
2. malloc () to allocate an array whose size is unknown until runtime .
3. Fields library and using strdup () to store a copy of a string.

tailanyf

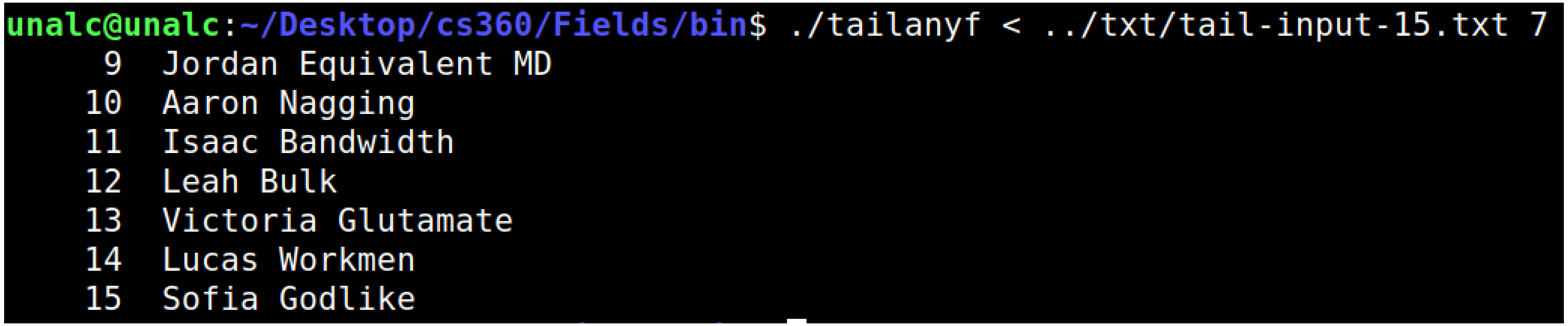


```
/* This program is more like tail -- it takes the number of lines, n,
   as a command line argument, and prints the last n lines of standard input. */

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

int main(int argc, char **argv)
{
  char **lastn;
  int nlines, i, n;
  IS is;

  /* Error check the command line. */

  if (argc != 2) { fprintf(stderr, "usage: tailany1 n\n"); exit(1); }
  if (sscanf(argv[1], "%d", &n) == 0 || n <= 0) {
    fprintf(stderr, "usage: tailany1 n\n");
    fprintf(stderr, "       bad n: %s\n", argv[1]);
    exit(1);
  }

  /* Allocate the array */

  lastn = (char **) malloc(sizeof(char *)*n);
  if (lastn == NULL) { perror("malloc"); exit(1); }
```



```
  /* Allocate the array */

  lastn = (char **) malloc(sizeof(char *)*n);
  if (lastn == NULL) { perror("malloc"); exit(1); }

  /* Allocate the IS */

  is = new_inputstruct(NULL);
  if (is == NULL) { perror("stdin"); exit(1); }

  /* Read the input */

  nlines = 0;
  while (get_line(is) >= 0) {
    if (nlines >= n) free(lastn[nlines%n]);     /* Prevent the memory leak. */
    lastn[nlines%n] = strdup(is->text1);
    nlines++;
  }

  /* Print the last n lines */

  i = (nlines < n) ? 0 : nlines-n;
  for ( ; i < nlines; i++) {
    printf("%s", lastn[i%n]);
  }

  /* Don't bother freeing stuff when you're just exiting anyway. */

  return 0;
}
```

tailanyf print

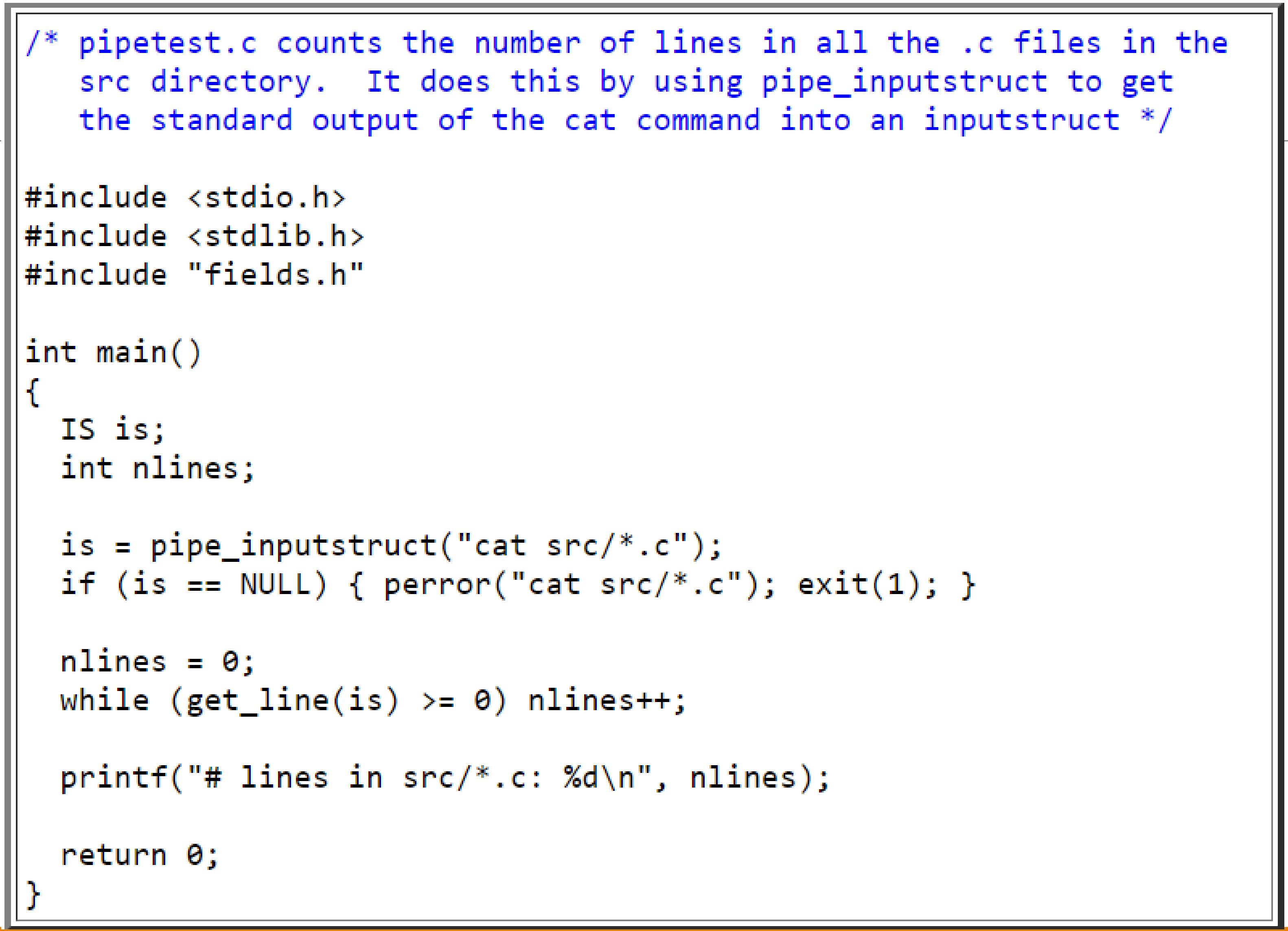


```
unalc@unalc:~/Desktop/cs360/Fields/bin$ ./tailanyf < ../txt/tail-input-15.txt 7
     9  Jordan Equivalent MD
    10  Aaron Nagging
    11  Isaac Bandwidth
    12  Leah Bulk
    13  Victoria Glutamate
    14  Lucas Workmen
    15  Sofia Godlike
```

pipe_inputstruct ()

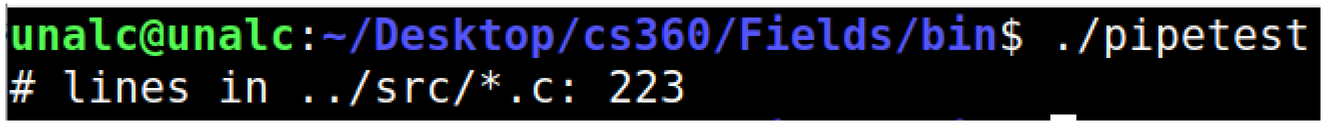
- ❑This allows you to read from a pipe opened with popen () .
- ❑The src / pipetest.c program uses the pipe_inputstruct () method to count the number of lines in all .c files in the src directory.
- ❑Do this, " cat src /*.c" by using pipe_inputstruct () to get its standard output into an inputstruct

pipe_inputstruct ()

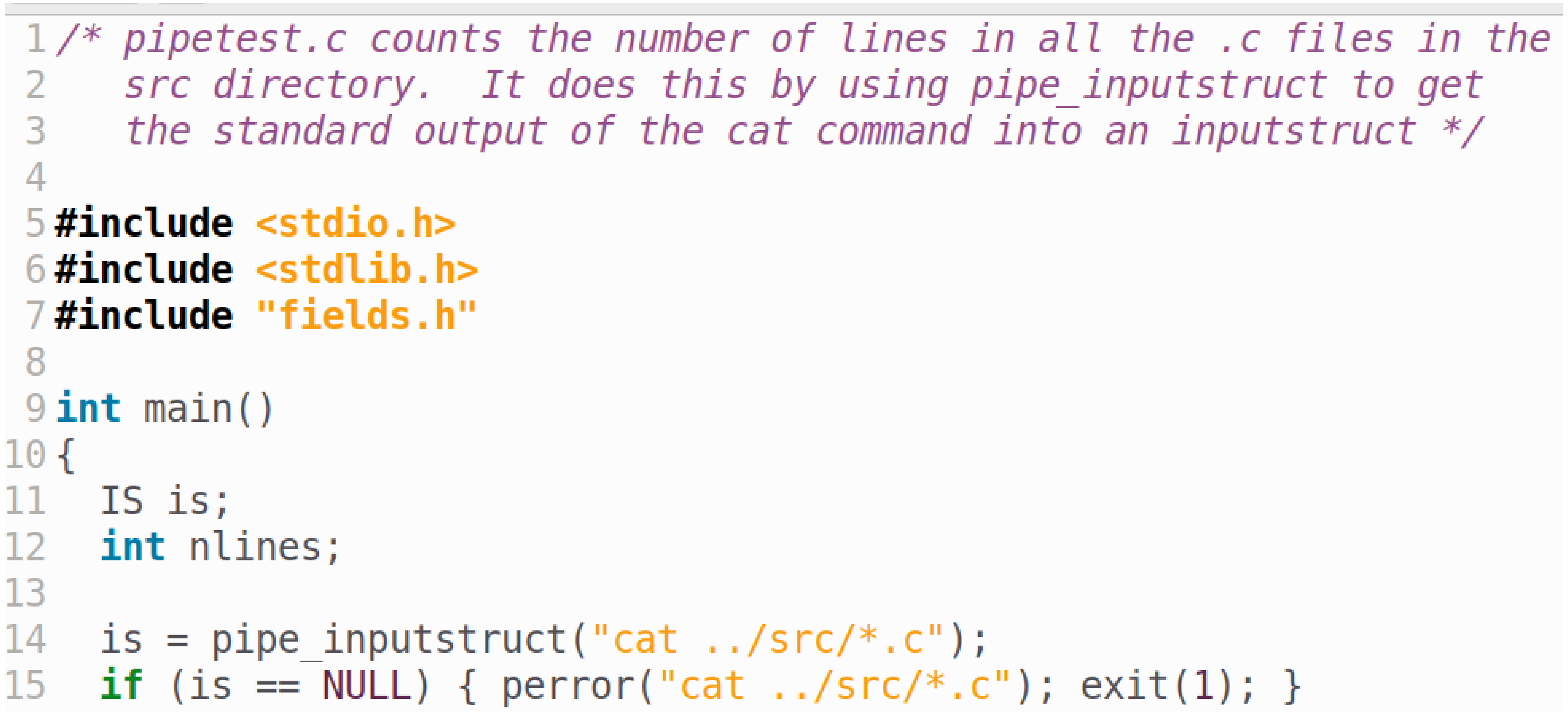


```
/* pipetest.c counts the number of lines in all the .c files in the
   src directory.  It does this by using pipe_inputstruct to get
   the standard output of the cat command into an inputstruct */

#include <stdio.h>
#include <stdlib.h>
#include "fields.h"

int main()
{
  IS is;
  int nlines;

  is = pipe_inputstruct("cat src/*.c");
  if (is == NULL) { perror("cat src/*.c"); exit(1); }

  nlines = 0;
  while (get_line(is) >= 0) nlines++;

  printf("# lines in src/*.c: %d\n", nlines);

  return 0;
}
```

pipetest print



```
unalc@unalc:~/Desktop/cs360/Fields/bin$ ./pipetest
# lines in ../src/*.c: 223
```



```c
/* pipetest.c counts the number of lines in all the .c files in the
   src directory.  It does this by using pipe_inputstruct to get
   the standard output of the cat command into an inputstruct */

#include <stdio.h>
#include <stdlib.h>
#include "fields.h"

int main()
{
  IS is;
  int nlines;

  is = pipe_inputstruct("cat ../src/*.c");
  if (is == NULL) { perror("cat ../src/*.c"); exit(1); }
```