Fall 2004 Handout 3

Examine the symbol table of a C program

```
 1 /* This is file1.c. */
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4
 5 void f1(void);
 6 static void f2(void);
 7 void f3(void);                /* external */
 8
 9 int g1;                      /* implicitly initialized */
10 int g2 = 10;                 /* explicitly initialized */
11
12 extern int e;
13
14 int main()
15 {
16     int i;                   /* Doesn't appear in symbol table. */
17
18     f1();
19     f2();
20     f3();
21     printf ("%d %d\n", g1, e);
22     return EXIT_SUCCESS;
23 }
24
25 void f1(void)
26 {
27 }
28
29 static void f2(void)
30 {
31 }
```

```
 1 /* This is file2.c. */
 2 #include <stdio.h>
 3 int e = 10;
 4
 5 void f3(void)
 6 {
 7 }
```

A **.c** file whose symbol table has **U**'s or **V**'s is not a complete program. See KP p. 106 for the arguments of **sort**.

```
1$ gcc -c file1.c                                          Create file1.o
2$ nm file1.o | awk 'NR >= 3 {print $5, $1}' | \
    sort +0f -1 +0 -1 +1df +1d | more


 23468.o:
Bind [Index]
|GLOB [10]
|GLOB [11]
|GLOB [12]
|GLOB [13]
|GLOB [7]
|GLOB [8]
|GLOB [9]
|LOCL [1]
|LOCL [2]
|LOCL [3]
|LOCL [4]
|LOCL [5]
|LOCL [6]

3$ gcc -c file2.c                                          Create file2.o
4$ nm file2.o | awk 'NR >= 3 {print $5, $1}' | \
    sort +0f -1 +0 -1 +1df +1d | more


 23468file2.o:
Bind [Index]
|GLOB [5]
|GLOB [6]
|LOCL [1]
|LOCL [2]
|LOCL [3]
|LOCL [4]
```

You can even see the symbol table of a successfully linked executable. In this case all the **U**'s and **V**'s have changed to other letters:


```
5$ gcc -o prog file1.o file2.o                          Create prog
6$ nm prog | awk 'NR >= 3 {print $5, $1}' | \
    sort +0f -1 +0 -1 +1df +1d | \
    pr -2 -l29 -t | more                                minus lowercase L 29
                                          |LOCL [15]
Bind [Index]                              |LOCL [16]
|GLOB [59]                                |LOCL [17]
|GLOB [61]                                |LOCL [18]
|GLOB [62]                                |LOCL [19]
|GLOB [63]                                |LOCL [2]
|GLOB [64]                                |LOCL [20]
|GLOB [65]                                |LOCL [21]
|GLOB [66]                                |LOCL [22]
|GLOB [68]                                |LOCL [23]
|GLOB [69]                                |LOCL [24]
|GLOB [70]                                |LOCL [25]
|GLOB [72]                                |LOCL [26]
|GLOB [73]                                |LOCL [27]
|GLOB [74]                                |LOCL [28]
|GLOB [75]                                |LOCL [29]
|GLOB [76]                                |LOCL [3]
|GLOB [77]                                |LOCL [30]
|GLOB [78]                                |LOCL [31]
|GLOB [79]                                |LOCL [32]
|GLOB [80]                                |LOCL [33]
|GLOB [81]                                |LOCL [34]
|GLOB [82]                                |LOCL [35]
|LOCL [1]                                 |LOCL [36]
|LOCL [10]                                |LOCL [37]
|LOCL [11]                                |LOCL [38]
|LOCL [12]                                |LOCL [39]
|LOCL [13]                                |LOCL [4]
|LOCL [14]                                |LOCL [40]
|LOCL [41]                                |LOCL [53]
|LOCL [42]                                |LOCL [54]
|LOCL [43]                                |LOCL [55]
|LOCL [44]                                |LOCL [56]
|LOCL [45]                                |LOCL [57]
|LOCL [46]                                |LOCL [58]
|LOCL [47]                                |LOCL [6]
|LOCL [48]                                |LOCL [7]
|LOCL [49]                                |LOCL [8]
|LOCL [5]                                 |LOCL [9]
|LOCL [50]                                |WEAK [60]
|LOCL [51]                                |WEAK [67]
|LOCL [52]                                |WEAK [71]
```

▼ Homework 3.1: which .o file is trying to use a missing function or variable?

Try to make an executable file out of three object files that I've already made for you:

```
1$ cd $d46
2$ gcc -o ~/bin/prog undefined1.o undefined2.o undefined3.o
3$ gcc -o ~/bin/prog undefined*.o
4$ gcc -o ~/bin/prog undefined[123].o
5$ gcc -o ~/bin/prog undefined[1-3].o
```

The error messages will be

```
cc: undefined[1-3].o: No such file or directory
cc: No input files
```

Which of these three object files (maybe more than one) are trying to use these undefined functions or variables?

▲

▼ Homework 3.2: which library do you have to link in?

Write a C program that calls an undefined function. Then write a shellscript with a **`for`** loop to look for that function in all the libraries in the **`/usr/lib`** directory.

▲

strip

You can **`strip`** the symbol table from an executable file but not from a **`.o`** file. This makes the executable file smaller, but has no effect on how much memory it uses when it runs.

```
1$ ls -l prog                          See how big prog is.
-rwx------   1 abc1234  users       6508 Jan  9 00:36 prog

2$ strip prog                          Remove the symbol table from prog.
3$ ls -l prog
-rwx------   1 abc1234  users       4104 Jan  9 00:36 prog

4$ nm prog                             ...but you can no longer see prog's symbol table.


prog:
5$ prog                                prog still runs.

6$ gcc -s -o prog file1.o file2.o      Create a stripped executable.

7$ bc
2^13
8192

8$ bc
2 * (2^13)
16384

9$ bc
3 * (2^13)
24576
```

Create a library of text files

Print **`ar`**(1). Create four text files and archive them in an file named **`libdate.a`**. The name of a library will always begin with **`lib`** and end with **`.a`**. Some versions of **`ar`** allow an optional dash at the start of the first argument.

```
1$ cd
2$ date > date1
3$ date > date2
4$ date > date3
5$ date > date4

    6$ ls -l date[1-4]
    -rw-------   1 mm64     users         29 Jan  9 00:36 date1
    -rw-------   1 mm64     users         29 Jan  9 00:36 date2
    -rw-------   1 mm64     users         29 Jan  9 00:36 date3
    -rw-------   1 mm64     users         29 Jan  9 00:36 date4

7$ ar crv libdate.a date1 date2 date3                    Create libdate.a
ar: writing libdate.a
a - date1
a - date2
a - date3

8$ ls -l libdate.a
-rw-------   1 abc1234  users        278 Jan  9 00:36 libdate.a

9$ ar tv libdate.a                    Display the library's table of contents.
rw------- 50766/    15     29 Jan  9 00:36 2004 date1
rw------- 50766/    15     29 Jan  9 00:36 2004 date2
rw------- 50766/    15     29 Jan  9 00:36 2004 date3

10$ rm date[1-3]
11$ ar pv libdate.a date1             Print date1 on the standard output.

<date1>

Fri Jan  9 00:36:22 EST 2004

12$ ar xv libdate.a date1             Create ("extract") a new copy of date1 from the library.
x - date1

13$ ls -l date1                       Make sure the file date1 reappeared.
-rw-------   1 abc1234  users          29 Jan  9 00:36 date1

14$ ar dv libdate.a date1             Delete date1 from the library.
ar: writing libdate.a
d - date1

15$ ar tv libdate.a                   Make sure date1 is deleted from the library.
rw------- 50766/    15     29 Jan  9 00:36 2004 date2
rw------- 50766/    15     29 Jan  9 00:36 2004 date3

16$ ar rv libdate.a date1             Put date1 back into the library.
ar: writing libdate.a
a - date1
```

```
17$ ar tv libdate.a                    Make sure date1 is back in the library.
rw------- 50766/    15     29 Jan  9 00:36 2004 date2
rw------- 50766/    15     29 Jan  9 00:36 2004 date3
rw------- 50766/    15     29 Jan  9 00:36 2004 date1

18$ vi date1                           Create a new version of date1.
19$ ar rv libdate.a date1              Replace the date1 in the library with the new date1.
ar: writing libdate.a
r - date1

20$ ar rv libdate.a date4              Add a new file to the library.
ar: writing libdate.a
a - date4

21$ ar tv libdate.a                    Make sure the new file is in the library.
rw------- 50766/    15     29 Jan  9 00:36 2004 date2
rw------- 50766/    15     29 Jan  9 00:36 2004 date3
rw------- 50766/    15     29 Jan  9 00:36 2004 date1
rw------- 50766/    15     29 Jan  9 00:36 2004 date4

22$ more libdate.a                     Don't try this with any other library.
```

You can **more libdate.a** because it contains only text files. See if you can decipher the header information at the start of each text file. **ar**(4) will confirm your guesses.

▼ Homework 3.3: create a library

Create and play with a little library of text files.

▲

A library of .o files

The real purpose of **ar** is to build a library (e.g., **/usr/lib/libc.a**, **/usr/lib/libm.a**) containing **.o** files that were created with **gcc -c**. These **.o** files do not constitute a complete C program: for example, they have no **main** function. They contain individual functions and variables (e.g., **printf** and **_iob**) that other people will use as part of their C programs.

Every library has a **.h** file (e.g., **/usr/include/stdio.h**, **/usr/include/math.h**) declaring all the functions and variables in the library. The people who use the library must **#include** the **.h** file in their C programs. Note that the library contains only the **.o** files; the **.h** file and **.c** files are not stored in the library.

When creating a library of **.o** files or adding extra **.o** files to it, always give the **s** option along with the **r** option (e.g., **crvs** or **rvs**). This will add a new member to the start of the library named **_____64ELEL_** (eight leading underscores) containing an index of the functions and variables in all the **.o** files. In some versions of Unix, this index has a different name (e.g., **SYMDEF**) and is created by a separate **ranlib** command after the **ar** command instead of the **s** option to **ar**.

```
/* This file is astor.h. */

void myprintf(void);
void myscanf(void);
```



```
/* This file is myprintf.c. */
#include <astor.h>

void myprintf(void)
{
}
```

```
/* This file is myscanf.c. */
#include <astor.h>

void myscanf(void)
{
}
```

```
1$ gcc -I. -c myprintf.c myscanf.c
2$ -s -l myprintf.o myscanf.o
-rw-------   1 mm64     users        536 Jan  9 00:36 myprintf.o
-rw-------   1 mm64     users        536 Jan  9 00:36 myscanf.o

3$ ar crvs libastor.a myprintf.o myscanf.o
ar: writing libastor.a
a - myprintf.o
a - myscanf.o

4$ ar tv libastor.a
rw------- 50766/    15    536 Jan  9 00:36 2004 myprintf.o
rw------- 50766/    15    536 Jan  9 00:36 2004 myscanf.o
```

The standard library **/usr/lib/libc.a** (which contains all the functions declared in **stdio.h** and **stdlib.h**) is automatically linked to every C program. To link another library to your program, **#include** the **.h** file for that library and specify the name of the library with the **-l** (minus lowercase L) option of **gcc**; see **ld**(1). The **-l** option must come *after* the names of the **.c**, **.s**, and **.o** files. Put no space after the lowercase **l**, and chop off the leading **lib** and the trailing **.a** from the name of the library. For example, if your program calls functions in the math library **/usr/lib/libm.a**,

```
5$ cd ~mm64/46/moon
6$ gcc -o ~/moon moonmain.c moonphase.c moondraw.c -lm
7$ gcc -o ~/prog prog.o -lm -lX11                    two libraries
```

gcc assumes that libraries are in the **/usr/lib** directory. Use the **-L** option before the **-l** option to change this assumption; see **ld**(1). For example, to use the library **/usr/libby/libastor.a**,

```
8$ gcc -o ~/prog prog.o -L/usr/libby -lastor
```

You can even see the symbol table of each **.o** file in a library with the **-r** option of **nm**:

```
9$ nm -r libastor.a | \
    awk '$1 ~ /^libastor\.a/ || $1 == "[Index]" || /\|FUNC/'
libastor.a[myprintf.o]:
[Index]   Value      Size    Type  Bind  Other Shndx   Name
[4] |        0|      12|FUNC |GLOB |0    |2      |myprintf.o:myprintf
libastor.a[myscanf.o]:
[Index]   Value      Size    Type  Bind  Other Shndx   Name
[4] |        0|      12|FUNC |GLOB |0    |2      |myscanf.o:myscanf
```

When linking in two or more libraries, (i.e., when giving two or more **-l** options to **gcc**), you may get an error message saying that a symbol is undefined even though the symbol is present in one of the libraries.

If this happens, put the **`-l`** option for that library last.

▼ Homework 3.4: examine some libraries of .o files with ar tv and nm -A

Does the C library **`/usr/lib/libc.a`** really contain the file **`printf.o`**? Does that file **`printf.o`** really contain the definition of the function **`printf`**?

Does the math library **`/usr/lib/libm.a`** really contain the file **`sqrt.o`**? Does that file **`sqrt.o`** really contain the definition of the function **`sqrt`**?

Does the X Window library **`/usr/lib/libX11.a`** really contain the file **`XOpenDis.o`**? Does that file **`XOpenDis.o`** really contain the definition of the function **`XOpenDisplay`**?

Does each library begin with **`_____64ELEL_`**? Does any library contain two files with the same name?

▲

Digitized images

A picture is stored in a file with one of the following suffixes:

`.gif`	*CompuServe graphic interchange format*
`.jpg` *or* **`.jpeg`**	*Joint Photographic Experts Group*
`.ppm`	*Jef Poskanzer's portable pixmap*
`.ps`	*Adobe PostScript*

```
1$ cd
2$ cd public_html
3$ lynx -source http://www.nyu.edu/images/torch1.gif > torch1.gif
4$ ls -l torch1.gif
5$ chmod 444 torch1.gif
```

Now you can point your browser at **`http://i5.nyu.edu/~abc1234/torch1.gif`**.

```
6$ head -c8 torch1.gif
GIF87a2
7$ od -c torch1.gif | more

8$ giftoppm torch1.gif > torch1.ppm
9$ lynx -source http://www.nyu.edu/images/torch1.gif | giftoppm > torch1.ppm
10$ ls -l torch1.ppm
11$ head -c2 torch1.ppm
P6
12$ od -c torch1.ppm | more
```

Conventions to prevent bugs

We adopt three conventions in code to manipulate **`.ppm`** files:

(1) Whenever we talk about the coördinates of a point, the **`x`** coördinate will come before the **`y`** coordinate.

(2) **`x`** and **`y`** coördinates will be zero-based. For example, if the image is 600 columns wide, the **`x`** coördinate will go from 0 to 599, not from 1 to 600. The origin **`(0, 0)`** will be in the upper left corner.

(3) The name of each function will begin with **`ppm_`**. The return value will be of type **`int`**: −1 for failure, 0 for success.

Generate a test .ppm file: http://i5.nyu.edu/~mm64/x52.9546/little.gif

Your **ppm_** function will be easier to debug if you feed it a tiny **.ppm** file instead of one containing thousands of pixels. You can't type a little **.ppm** file with **vi**, however—it contains binary data. Run the following C program instead:

```
 1 /* This file is testgen.c. */
 2
 3 #include <stdio.h>
 4 #include <stdlib.h>
 5
 6 main()
 7 {
 8     printf ("P6\n");                  /* format of .ppm file */
 9     printf ("3 2\n");                 /* width and height in pixels */
10     printf ("255\n");                 /* maximum color value */
11
12     printf ("%c%c%c", 0, 0, 0);       /* x=0, y=0 (upper left): black */
13     printf ("%c%c%c", 128, 128, 128); /* x=1, y=0 (top center): gray */
14     printf ("%c%c%c", 255, 255, 255); /* x=2, y=0 (upper right): white */
15
16     printf ("%c%c%c", 255, 0, 0);     /* x=0, y=1 (lower left): red */
17     printf ("%c%c%c", 0, 255, 0);     /* x=1, y=1 (bottom center): green */
18     printf ("%c%c%c", 0, 0, 255);     /* x=2, y=1 (lower right): blue */
19
20     exit (0);
21 }
```

Each of the **printf**'s in lines 12−18 above could have been written as three **putchar**'s. For example, line 18 could have been written as

```
18     putchar (0);       /* red */
19     putchar (0);       /* green */
20     putchar (255);     /* blue */
```

```
1$ gcc -o testgen testgen.c                          create testgen
2$ rehash
3$ testgen > little.ppm
4$ ls -l little.ppm
-rw-------   1 abc1234  users         29 Jan  9 00:36 little.ppm

5$ head -3 little.ppm
P6
3 2
255
```

```
6$ tail +4 little.ppm | od -An -v -tu1 | \
    tr -cs '[:digit:]' '[\n*]' | tail +2 | more
000
000
000
128
128
128
255
255
255
255
000
000
000
255
000
000
000
255
```

```
#!/bin/sh

tail +4 little.ppm |
od -Ad -v -tu1 |
tr -cs '[:digit:]' '[\n*]' |
awk '
    BEGIN  {n = 3}                  #colors per pixel
           {
        p = NR / n                  #divide by number of primary colors
                                    #to get pixel number
        if (p > 0 && p % n == 0) {
            print ""                #output empty line between rows
        }
        printf "%3d (%d, %d) %3d %3d %3d\n",
            NR, p % n, int(p / n), $1, $2, $3
    }
'
```

```
 1 (0, 0)   0   0   0
 2 (0, 0)   0   0   0
 3 (1, 0)   0   0   0
 4 (1, 0)   0   0   0
 5 (1, 0) 128   0   0
 6 (2, 0) 128   0   0
 7 (2, 0) 128   0   0
 8 (2, 0) 255   0   0

 9 (0, 1) 255   0   0
10 (0, 1) 255   0   0
11 (0, 1) 255   0   0
12 (1, 1)   0   0   0
13 (1, 1)   0   0   0
14 (1, 1)   0   0   0
15 (2, 1) 255   0   0
16 (2, 1)   0   0   0
17 (2, 1)   0   0   0

18 (0, 2)  16   0   0
19 (0, 2)   0   0   0
20 (0, 2) 255   0   0
21 (1, 2)  18   0   0
```

Could you make the above program also print out the name of the closest color to each pixel in the file **`/usr/lib/X11/rgb.txt`**?

Make a negative

Get a **`.gif`** or **`.jpeg`** file and change it into a **`.ppm`** file. Input the **`.ppm`** file into the following C program which will output a new **`.ppm`** file. Then convert the new **`.ppm`** file back to **`.gif`** with **`ppmtogif`**.

The **`scanf`** assignment suppression characater **`*`** in line 18 lets you avoid having to create a dummy variable:

```
char dummy;      /* input the newline after the maxcolor */
scanf("%s%d%d%d%c", format, &width, &height, &maxcolor, &dummy);
```

See pp. 157, 245 in the second edition of the K&R C book.

```
 1 /* This file is ppm_negative1.c.  Read in a P6 pgm file from the standard input,
 2 and write to the standard output another P6 format ppm file which is a negative
 3 of the first file. */
 4
 5 #include <stdio.h>
 6 #include <stdlib.h>
 7 #include <string.h>
 8
 9 int main(int argc, char **argv)
10 {
11     char format[256];
12     int width, height;
13     int maxcolor;        /* this number means full color; 0 means no color */
14     int x, y;
15     unsigned char red, green, blue;      /* unsigned to avoid sign extension */
16
```



```
17      /* %*c will input (and discard) the newline after the maxcolor. */
18      scanf("%s%d%d%d%*c", format, &width, &height, &maxcolor);
19
20      if (strcmp(format, "P6") != 0) {
21          fprintf (stderr, "%s: input must be P6 format ppm\n", argv[0]);
22          return 1;
23      }
24
25      printf("P6\n%d %d\n%d\n", width, height, maxcolor);
26
27      for (y = 0; y < height; ++y) {
28          for (x = 0; x < width; ++x) {
29              if (scanf("%c%c%c", &red, &green, &blue) != 3) {
30                  fprintf(stderr, "%s: EOF when x == %d, y == %d\n",
31                      argv[0], x, y);
32                  return 2;
33              }
34
35              printf("%c%c%c",
36                  maxcolor - red,
37                  maxcolor - green,
38                  maxcolor - blue);
39          }
40      }
41
42      return 0;
43  }
```

```
1$ gcc -o $HOME/bin/ppm_negative1 ppm_negative1.c
2$ rehash
3$ ppm_negative1 < original.ppm > negative.ppm
4$ giftoppm original.gif | ppm_negative1 | ppmtogif > negative.gif
5$ chmod 444 negative.gif
6$ mv negative.gif $HOME/public_html
```

Write a main function and ppm_ function in separate files

The **ppm_** function should no longer output error messages or call **exit**:

```
 1 /* This file is main_negative.c. */
 2
 3 #include <stdio.h>
 4 #include <stdlib.h>
 5 #include <ppm.h>
 6
 7 int main(int argc, char **argv)
 8 {
 9     const int r = ppm_negative(stdin, stdout);
10
11     if (r != 0) {
12         fprintf(stderr, "%s: ppm_negative function returned %d\n", argv[0], r);
13         return 1;
14     }
15
16     return 0;
```

```
17 }
```

```
 1 /* This file is ppm_negative.c.  Input a P6 ppm file from in, and write to out
 2 another P6 ppm file which is a negative of the first file. */
 3
 4 #include <stdio.h>
 5 #include <string.h>
 6 #include <ppm.h>
 7
 8 int ppm_negative(FILE *in, FILE *out)
 9 {
10     char format[256];
11     int width, height;
12     int maxcolor;        /* this number means full color; 0 means no color */
13     int x, y;
14     unsigned char red, green, blue;
15
16     /* %*c will input the newline after the maxcolor. */
17     fscanf(in, "%s%d%d%d%*c", format, &width, &height, &maxcolor);
18
19     if (strcmp(format, "P6") != 0) {
20         return -1;
21     }
22
23     fprintf(out, "P6\n%d %d\n%d\n", width, height, maxcolor);
24     for (y = 0; y < height; ++y) {
25         for (x = 0; x < width; ++x) {
26             if (fscanf(in, "%c%c%c", &red, &green, &blue) != 3) {
27                 return -1;
28             }
29
30             fprintf(out, "%c%c%c",
31                 maxcolor - red,
32                 maxcolor - green,
33                 maxcolor - blue);
34         }
35     }
36
37     return 0;
38 }
```

```
1$ gcc -I$m46/ppm/include -o $HOME/bin/ppm_negative \
    main_negative.c \
    ppm_negative.c
2$ rehash
```

Let the user specify an input file as a command line argument: pp. 130−131

Most Unix programs assume that a command line argument that doesn't start with a dash is the name of an input file. If there are no command line arguments that do not start with a dash, then the input is taken from the standard input.

```
1$ sort -n file
2$ sort -n
```


Here is an improved **main_negative.c** file for the above program. A command line argument is assumed to be the name of an input file. Otherwise, input is taken from the standard input.

The **main** function should do nothing except process the command line arguments and call your **ppm_** function. If your **ppm_** function requires a **FILE *in**, have **main fopen** a file or give it **stdin**. If your **ppm_** function requires a **FILE *out**, have **main** give it **stdout**.

Before **main** uses any command line argument (i.e., any member of the array **argv**), it must make sure that the argument exists (i.e., that **argc** is big enough). Also make sure that there are not too many arguments.

```
 1 /* This file is main_negative.c. */
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <ppm.h>
 5
 6 int main(int argc, char **argv)
 7 {
 8     FILE *in;
 9     int r;
10
11     if (argc > 2) {
12         fprintf(stderr, "%s: at most one command line argument\n", argv[0]);
13         return 1;
14     }
15
16     if (argc == 1) {
17         in = stdin;
18     } else if ((in = fopen(argv[1], "r")) == NULL) {
19         fprintf(stderr, "%s: couldn't open input file %s\n", argv[0], argv[1]);
20         return 2;
21     }
22
23     r = ppm_negative(in, stdout);
24     if (r != 0) {
25         fprintf(stderr, "%s: ppm_negative function returned %d\n", argv[0], r);
26         return 3;
27     }
28
29     return 0;
30 }
```

```
3$ gcc -I$m46/ppm/include -o $HOME/bin/ppm_negative \
    main_negative.c \
    ppm_negative.c
4$ rehash
```

You can now use **ppm_negative** in any of the following ways:

```
5$ ppm_negative < original.ppm > negative.ppm
6$ ppm_negative original.ppm > negative.ppm
7$ giftoppm original.gif | ppm_negative | ppmtogif > negative.gif
8$ chmod 444 negative.gif
9$ mv negative.gif $HOME/public_html
```


Pass a numeric command line argument to a C program

If your **ppm_** function requires a numeric argument, let it be the first command line argument. See p. 251 in the K&R C book, second edition, for **atoi**, **atol** and **atof**.

```
 1 /* This file is main_hstretch.c. */
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <ppm.h>
 5
 6 int main(int argc, char **argv)
 7 {
 8     FILE *in;
 9     int r;
10
11     if (argc < 2 || argc > 3) {
12         fprintf(stderr, "%s: requires 1 or 2 command line arguments\n", argv[0]);
13         return 1;
14     }
15
16     if (argc == 2) {
17         in = stdin;
18     } else if ((in = fopen(argv[2], "r")) == NULL) {
19         fprintf(stderr, "%s: couldn't open input file %s\n", argv[0], argv[2]);
20         return 2;
21     }
22
23     r = ppm_hstretch(in, stdout, atoi(argv[1]));
24     if (r < 0) {
25         fprintf(stderr, "%s: ppm_hstretch function returned %d\n", argv[0], r);
26         return 3;
27     }
28
29     return 0;
30 }
```

```
1$ ppm_hstretch 2 original.ppm | ppm_vstretch 2 > stretched.ppm
```

▼ Homework 3.5: write a ppm_ function and a main function to call it

Write one or more of the **ppm_** functions declared in **$m46/ppm/include/ppm.h**, or invent your own **ppm_** function and send me **mail** describing its arguments and what it does. Also write a **main** function to call your **ppm_** function.

Common bugs:

(1) If you read the input pixels in nested **for** loops, be sure to keep the **y** loop on the outside and the **x** loop on the inside, as in Handout 3, p. 9, lines 27−28.

(2) Make sure that every number that you store in an **unsigned char** variable is in the range zero to 255 inclusive.

(3) Make sure that every color that you output is in the range zero to **maxcolor** inclusive.

Write the two functions in separate **.c** files named **main_negative.c** and **ppm_negative.c** (for example). The two **.c** files must both **#include <ppm.h>**. Since **ppm.h** is in the directory **$m46/ppm/include**, you'll have to compile the **.c** files with the **-I$m46/ppm/include** option of **gcc**.

Hand in your two **.c** files. Use **ppmtogif** to change the **.ppm** output of your C program back to a **.gif**, and link your home page to the original and processed **.gif**'s.

Copy the two **.c** files into the **$m46/ppm/src** directory and **chmod** them to **r--r--r--**. (First make sure that files with these names do not already exist there.) **chmod** the **.o** file that contains your **ppm_** function to **r--r--r--** and archive it into the library **$m46/ppm/lib/libppm.a**:

```
1$ cd to the directory where your ppm_negative.c file is
2$ gcc -c -I$m46/ppm/include ppm_negative.c       create ppm_negative.o
3$ ls -l
4$ chmod 444 ppm_negative.o
5$ ls -l
6$ ar rvs $m46/ppm/lib/libppm.a ppm_negative.o
7$ ar tv $m46/ppm/lib/libppm.a | more
8$ rm -f ppm_negative.o
```

(First make sure that a file with the same name is not already in the library.) Copy the executable C program into the **$m46/ppm/bin** directory and **chmod** it to **r-xr-xr-x**. (First make sure that a file with the same name does not already exist there.) Edit your **.login** file to insert the directory **/home1/m/mm64/46/ppm/bin** into your **$PATH** just before the **.** at the end of the path. (You can't use the variable **$m46** in your **.login** file before you **setenv** it.) Then log out and log in.

If your **ppm_** function doesn't work, debug it with the little 3×2 **.ppm** file shown above instead of a **.ppm** file of normal size. Or dispense with the input and output files and take the rows and columns of pixels from a small two-dimensional array which you initialize yourself:

```
 1 typedef struct {
 2     unsigned char red;
 3     unsigned char green;
 4     unsigned char blue;
 5 } pixel;
 6
 7 pixel input[2][3] = {
 8     {{  0,   0,   0},  {128, 128, 128},   {255, 255, 255}},  /* y = 0 */
 9     {{255,   0,   0},  {  0, 255,   0},   {  0,   0, 255}}   /* y = 1 */
10 };
```

Deposit the output image in another two-dimensional array of **struct pixel**'s which you can then print as rows and columns of numbers. Take the data from an input file only after you can successfully process the pixels in an array.

▲

Call a function from a C program or from the Unix command line

```
1$ rm myfile
```

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <unistd.h>
 4
 5 int main(int argc, char **argv)
 6 {
 7     if (unlink(argv[1]) != 0) {
 8         fprintf(stderr, "%s: couldn't remove the file %s.\n", argv[0], argv[1]);
 9         return 1;
10     }
11
12     return 0;
```



```
13 }
```

The **rm** command is merely a program that calls the **unlink** function. See

```
2$ man -t rm | grops | lpr                    the command
3$ man -t 2 unlink | grops | lpr              the C function

4$ chmod 644 myfile
```

The following **strtol** is like **atoi** or **atol**, except that the input number is in base 8.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 #include <sys/mode.h>
 5 #include <sys/types.h>
 6 #include <sys/stat.h>
 7
 8 int main(int argc, char **argv)
 9 {
10     const mode_t mode = strtol(argv[1], NULL, 8);
11
12     if (chmod(argv[2], mode) != 0) {
13         fprintf(stderr, "%s: couldn't chmod the files %s.\n", argv[0], argv[2]);
14         return 1;
15     }
16     return 0;
17 }
```

(By the way, instead of writing **0644** in your C program, you should take advantage of the macros that are **#define**'d in the file **/usr/include/sys/mode.h** and write **S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH**.)

The **chmod** command is merely a program that calls the **chmod** function. See

```
5$ man -t   chmod | grops | lpr                the command
6$ man -t 2 chmod | grops | lpr                the C function
```

Suggested ppm_ functions

```
 1 /* This file is $m46/ppm/include/ppm.h, which must be #include'd by any .c file
 2 that calls a function in $m46/ppm/lib/libppm.a.  When compiling the .c file,
 3 give the -I$m46/ppm/include option to cc.

 4 Since ppm.h uses the data type FILE which is defined in stdio.h,
 5 you must #include <ppm.h> after you #include <stdio.h>.
 6 */

 7 /* Make a negative of the image.  This function is shown in the Handout. */
 8 int ppm_negative(FILE *in, FILE *out);

 9 /* Change the green and blue components to zero; leave the red untouched. */
10 int ppm_tored(FILE *in, FILE *out);

11 /* Change the red and blue components to zero; leave the green untouched. */
12 int ppm_togreen(FILE *in, FILE *out);
```

```
13 /* Change the red and green components to zero; leave the blue untouched. */
14 int ppm_toblue(FILE *in, FILE *out);

15 /* Convert to black and white using this formula:
16 gray = .299 * red + .587 * green + .114 * blue */
17 int ppm_bw(FILE *in, FILE *out);

18 /* If a pixel would be lighter than or equal to the cutoff value when converted
19 to black and white, change it to pure white.  Otherwise, leave it unchanged. */
20 int ppm_cutoff(FILE *in, FILE *out, int cutoff);

21 /* If a pixel would be lighter than or equal to the cutoff value when converted
22 to black and white, change it to pure white.  Otherwise, change it to pure
23 black. */
24 int ppm_kodalith(FILE *in, FILE *out, int cutoff);

25 /* Change rows interval-1, 2*interval-1, 3*interval-1, etc., to the specified
26 color.  Leave the other rows unchanged. */
27 int ppm_hgraph(FILE *in, FILE *out, int interval, char red, char green, char blue);

28 /* Change columns interval-1, 2*interval-1, 3*interval-1, etc., to the specified
29 color.  Leave the other columns unchanged. */
30 int ppm_vgraph(FILE *in, FILE *out, int interval, char red, char green, char blue);

31 /* Add this number to the value of each color of each pixel.  Positive will
32 lighten, negative will darken.  Make sure that each color output remains in
33 the range 0 to maxcolor inclusive. */
34 int ppm_darken(FILE *in, FILE *out, int darker);

35 /* Multiply the value of each pixel by this number.  A number greater than
36 1.0 will lighten, less than 1.0 will darken. */
37 int ppm_mdarken(FILE *in, FILE *out, double darker);

38 /* Add the last three arguments to the red, green, and blue values
39 (respectively) of each pixel.  Positive arguments will therefore make each
40 pixel brighter, negative will make each pixel darker.  If the pixel's value
41 plus the argument is greater than maxval, make the pixel maxval.  If the pixel's
42 value plus the argument is less than zero, make the pixel zero.  For example,
43 the three arguments 0, -255, -255 will do the same thing as ppm_tored. */
44 int ppm_adjust(FILE *in, FILE *out, int red, int green, int blue);

45 /* Turn down the contrast, with 0 <= contrast <= 1.  If contrast == 0, leave the
46 picture unchanged.  If contrast == 1, change each pixel to maxcolor/2.
47 The exact formula is: change each pixel to
48 (1 - contrast) * pixel + contrast * (maxcolor/2) */
49 int ppm_downcontrast(FILE *in, FILE *out, double contrast);

50 /* Turn up the contrast, with 0 <= contrast <= 1.  If contrast == 0, leave the
51 picture unchanged.  If contrast == 1, change each pixel that is lighter than or
52 equal to maxcolor / 2 to pure white, and change each pixel that is darker than
53 maxcolor / 2 to pure black.  The exact formula is: change each pixel that is
54 lighter than or equal to maxcolor/2
55 to (1 - contrast) * pixel + contrast * maxcolor, and change each
56 pixel that is darker than maxcolor/2 to (1 - contrast) * pixel. */
```



```
 57 int ppm_upcontrast(FILE *in, FILE *out, double contrast);

 58 /* Horizontal mirror image (left to right), and vertical mirror image (top to
 59 bottom. */
 60 int ppm_hmirror(FILE *in, FILE *out);
 61 int ppm_vmirror(FILE *in, FILE *out);

 62 /* Rotate the image counterclockwise 90*direction degrees.  If direction < 0,
 63 the rotation will be clockwise.  If direction is even, the resulting image will
 64 have the same width and height as the original.  If direction is odd, the width
 65 of the result will be equal to the height of the original, and the height of the
 66 result will be equal to the width of the original.  Any int is a legal value for
 67 the third argument, even 0. */
 68 int ppm_rotate(FILE *in, FILE *out, int direction);

 69 /* Superimpose two images by outputting their average, not their sum.  They must
 70 have the same height, width, and maximum color value. */
 71 int ppm_superimpose(FILE *in1, FILE *in2, FILE *out);

 72 /* Weighted superimposition.  The two input files must have the same height,
 73 width, and maximum color value.  The first file makes 100 * (1 - weight) percent
 74 of the contribution; the second file makes 100 * weight percent of the
 75 contribution.  For example, if weight == .5 they make equal contributions. */
 76 int ppm_wsuperimpose(FILE *in1, FILE *in2, FILE *out, double weight);

 77 /* Concatenate two images side by side (or one above another).  They must have
 78 the same height (or width) and maximum color value. */
 79 int ppm_hcat (FILE *inleft, FILE *inright, FILE *out);
 80 int ppm_vcat (FILE *intop, FILE *inbottom, FILE *out);

 81 /* Stretch horizontally (vertically) by a factor of n. */
 82 int ppm_hstretch(FILE *in, FILE *out, int n);
 83 int ppm_vstretch(FILE *in, FILE *out, int n);

 84 /* Squeeze horizontally (vertically) by a factor of n, which must be a divisor
 85 of the width (height).  Each output pixel will have the average color of n
 86 consecutive pixels. */
 87 int ppm_hsqueeze(FILE *in, FILE *out, int n);
 88 int ppm_vsqueeze(FILE *in, FILE *out, int n);

 89 /* Add a right (left, top, bottom) margin of a given width and color. */
 90 int ppm_rmargin(FILE *in, FILE *out, int width, char red, char green,char blue);
 91 int ppm_lmargin(FILE *in, FILE *out, int width, char red, char green,char blue);
 92 int ppm_tmargin(FILE *in, FILE *out, int width, char red, char green,char blue);
 93 int ppm_bmargin(FILE *in, FILE *out, int width, char red, char green,char blue);

 94 /* Add right, left, top, and bottom margins of the specified color to make the
 95 image the desired height and width.  The right and left margins must be of equal
 96 width, and the top and bottom margins must be of equal width. */
 97 int ppm_center(FILE *in, FILE *out, int width, int height,
 98     char red, char green, char blue);

 99 /* Output only the subrectangle lying within the starting and ending positions,
100 inclusive.  Must have start <= end.  0 is the top row of pixels and height-1 is
```

```
101 the bottom row. The subrectangle has the same width as the original image, but
102 its height is only end-start+1 pixels.  In other words, chop off the top and
103 bottom of the image. */
104 int ppm_hcrop(FILE *in, FILE *out, int start, int end);

105 /* Output only the subrectangle lying within the starting and ending positions,
106 inclusive.  Must have start <= end.  0 is the leftmost column of pixels and
107 width-1 is the rightmost column. The subrectangle has the same height as the
108 original image, but its width is only end-start+1 pixels.  In other words, chop
109 off the left and right ends of the image. */
110 int ppm_vcrop(FILE *in, FILE *out, int start, int end);

111 /* Replace the indicated subrectangle by one of the specified color.
112 Must have
113     0 <= top <= bottom < height
114     0 <= left <= right < width
115 */
116 int ppm_censor(FILE *in, FILE *out,
117     int top, int bottom, int left, int right,
118     char red, char green, char blue);

119 /* Move each pixel in row r to the right [slant*r] places, where [] is the
120 greatest integer function.  The pixels in row 0 will therefore never be moved,
121 and if slant == 0 this function will have no effect.  If slant < 0, the pixels
122 will be moved to the left.  Add two right triangles of the specified color to
123 the image, so that the resulting image will be a rectangle.  The width of the
124 resulting image will be original_width + fabs(slant) * original_height.  The
125 height of the resulting image will be the same as the original height.
126 +-------+               +-----------+                 +-----------+
127 |       |               |\       \  |                 |  /       /|
128 |       | original      | \       \ | slant > 0       | /       / | slant < 0
129 |       |               |  \       \|                 |/       /  |
130 +-------+               +-----------+                 +-----------+
131 */
132 int ppm_slant(FILE *in, FILE *out, double slant,
133     char red, char green, char blue);

134 /* Divide the image into rows and columns of little rectangles all of the same
135 width and height.  The width of each rectangle must be a divisor of the width of
136 the image; the height of each rectangle must be a divisor of the height of the
137 image.  Replace each rectangle by a solid color which is the average color of
138 all pixels in the rectangle.  You need to malloc only enough memory to hold a
139 two-dimensional array of long unsigned's, one for each little rectangle. */
140 int ppm_digitize(FILE *in, FILE *out, int width, int height);

141 /* Divide the image into n by n squares.  The width and height of the image
142 must be multiples of n.  Replace each square by a black circle on a white
143 background.  The center of the circle should be at the center of the square.
144 The diameter of the circle depends on the average color of the square; the exact
145 formula is: diameter in pixels = factor * n * average_color_value / maxcolor.
146 Let factor = 1.0 for the initial experiments; then adjust. */
147 int ppm_lichtenstein(FILE *in, FILE *out, int n, double factor);

148 /* Change the picture to horizontal scan lines to make it look like a photo of a
```



```
149 black and white video screen.  The scan lines are all parallel and are all the
150 same length, which is equal to the width of the image.  The scan lines are all
151 pure black, with pure white between them.  n is the number of scan lines, which
152 must be a divisor of the height of the image.  The width of each scan line will
153 almost always vary from left to right.  Therefore the width of the white
154 horizontal bands between the scan lines will also vary.  The minimum width of a
155 scan line is one pixel, and the maximum width is (height/n)-2 pixels. */
156 int ppm_video(FILE *in, FILE *out, int n);

157 /* The output has the same width and height as the input, but is cropped to one
158 of the following shapes. */
159 int ppm_binoculars(FILE *in, FILE *out, char red, char green, char blue);
160 int ppm_keyhole   (FILE *in, FILE *out, char red, char green, char blue);
161 int ppm_heart     (FILE *in, FILE *out, char red, char green, char blue);
162 int ppm_diamond   (FILE *in, FILE *out, char red, char green, char blue);

163 /* Apply a function to the y coordinates of the picture to give it a "melted"
164 effect. */
165 int ppm_melt(FILE *in, FILE *out);

166 /* Draw horizontal white lines. */
167 int ppm_4h_split (FILE *in, FILE *out);
```

□