READ THIS FIRST

Read this page now but do not turn this page until you are told to do so. Fill in your last name in the MIDDLE box above, and get the names of your classmates, if any, on each side of you, too.

This is a 100-minute exam with a total of 100 points of regular questions and an extra credit section.

To avoid distractions for those who are still working, I ask that nobody leaves their seat during the last five minutes of the exam. If you finish during that last five minutes, please remain quietly seated—no "packing up", checking phones, etc.—until the exam is over.

You are allowed no reference materials whatsoever.

If you have a question, raise your hand. We will come to you. DO NOT leave your seat.

If you have a question that can be safely resolved with a minor assumption, such as the name of an option for a UNIX command, or the order of arguments for a function, state the assumption and proceed.

Don't make problems hard by assuming that they need to do more than is specifically mentioned in the write-up, or by assuming that the solution that comes to mind is "too easy."

If you're stuck on a problem, please ask for a hint. Try to avoid leaving a problem completely blank—that's a sure zero.

It is better to put forth a solution that violates stated restrictions than to leave a problem blank—a solution with violations may still be worth partial credit.

When told to begin, scribble your initials in the right hand corner of the top side of each sheet, checking to be sure you have all six sheets. (Exams are scanned, to inhibit the temptation of altering graded work, and once in a while the sheets of two exams become intermixed when scanning.)

Remember that we'll have a sign-out process when then exam is over. If you have a tight schedule, make that known, and we'll get you to the front of a line.
Problem 1: (30 points)

For this problem you are to write a **C program** named `showcols` that reads lines from standard input and writes to standard output the characters on each line that appear in column ranges specified as command-line arguments.

Here's an example that prints the first three columns of each line:

```
% cat sc.1
Line one
Second line
The third line of input

% showcols 1-3 < sc.1
Lin
Sec
The
```

Any number of column range specifications may appear. If non-existent columns are specified, no output is produced for those columns. In the following example, note that the 300-400 produces no output for any line and that the 16-50 only produces output for the third line (which is 23 characters long).

```
% showcols 2-2 300-400 16-50 < sc.1
i
e
hof input
```

One more example, using my `ruler` command to show column numbers:

```
% ruler
  1  2  3  4  5  6
1234567890123456789012345678901234567890123456789012345678901234567

% ls -l
-rw-r--r-- 1 whm staff 1009 Dec 13 17:47 mkstrs.c
-rwxr-xr-x 1 whm staff 9264 Dec 14 01:26 showcols
-rw-r--r-- 1 whm staff  709 Dec 14 01:26 showcols.c

% ls -l | showcols 2-4 41-100 35-40
rw-  mkstrs.c 17:47
rwx  showcols 01:26
rw-  showcols.c 01:26
```

Assume ranges are well-formed—always **START-END**, with START \( \leq \) END, and START \( \geq \) 1 There may be any number of ranges. Assume lines are less than 1000 characters in length; read them with

```
char *gets(char *line).
```

**Your solution must be 100% C**—you might be tempted to write a hybrid solution that uses `cut (1)`, but that is not allowed.
It's fine to use `sscanf(3)` to parse ranges. Use the format "%d-%d":

```c
sscanf(spec, "%d-%d", ...);
```
Problem 2: (30 points)

For this problem you are to write a **C function** named `mkstrs`:

```c
String *mkstrs(char *text, int *lengths)
```

It returns a pointer to an allocated array of `String` structures whose contents are based on `text` and `lengths`.

Instances of `String` represent character strings with a length and a pointer to a sequence of characters:

```c
typedef struct String {
    int length;
    char *bytes;
} String;
```

**The sequence of characters pointed to by bytes is NOT zero-terminated.** If `length` is zero, `bytes` is zero. `length` is never negative.

Here's an example with a string `s` and an array of lengths, `lens`:

```c
char *s = "testing";
int lens[] = {3, 0, 1, 2, -1};

String *strp = mkstrs(s, lens);
```

`lens` specifies that four Strings are to be created from "testing", with lengths 3, 0, 1, and 2, respectively. The -1 is a sentinel value that indicates the end of the lengths.

Here's a diagram that shows the result of the `mkstrs` call above. Note that four allocations were made: One for an array of four Strings, and one for each of the three non-zero length byte sequences. Note that there are no terminating zeroes on the byte sequences; do not allocate space for a terminating zero.
Here's an example of using the result produced by `mkstrs`:

```c
String *mkstrs(char *text, int *lengths);

int main()
{
    char *s = "testing";
    int lens[] = {3, 0, 1, 2, -1};

    String *strp = mkstrs(s, lens);
    for (int i = 0; i < 4; i++) {
        int len = strp->length;
        printf("String %d (%d bytes): ", i, len);
        char *bytes = strp->bytes;
        while (len--) 
            putchar(*bytes++);
        puts("\n");
        strp++;
    }
}
```

Output:

```
% a.out
String 0 (3 bytes): 'tes'
String 1 (0 bytes): ''
String 2 (1 bytes): 't'
String 3 (2 bytes): 'in'
```

For reference:

```c
String *mkstrs(char *text, int *lengths);
```

There may be any number of lengths in the array referenced by `lengths` but assume that there is always at least one length; that is, `*lengths` won't be `-1`. No negative values will appear except for the `-1` sentinel.

Assume that `strlen(text)` is greater than or equal to the sum of the lengths.

Assume that `malloc` never fails to allocate the requested amount of memory.

**There is space for your solution on the next page.**
For reference:

typedef struct String {
    int length;
    char *bytes;
} String;

String *mkstrs(char *text, int *lengths);

char *s = "testing";
int lens[] = {3, 0, 1, 2, -1};
String *strp = mkstrs(s, lens);
Problem 3: (12 points)

On assignment 12 you implemented a vector datatype using an interface style shown in the book *C Interfaces and Implementations*. For this problem you are to implement a rectangle datatype using the same techniques.

Here is the interface, in `rect.h`, that describes the four functions you are to implement.

```c
#ifndef rect_h_
#define rect_h_

typedef struct Rect *Rect_T;

Rect_T Rect_new(double w, double h);
    // Create a new rectangle with the given width and height.

double Rect_area(Rect_T r);
    // Return the area of the rectangle.

void Rect_get_wh(Rect_T r, double *result);
    // result points to an array of two doubles. The width of the rectangle is written to the first element; the height is written into the second.

void Rect_free(Rect_T *rp);
    // free all memory associated with the rectangle AND zero the pointer referenced by rp.
#endif
```

Here is a test program:

```c
int main()
{
    Rect_T r1 = Rect_new(3, 4);

double dims[2];
Rect_get_wh(r1, dims);
printf("r1 is %g x %g, with an area of %g\n",
         dims[0], dims[1], Rect_area(r1));

    Rect_free(&r1);
}
```

Execution:

```bash
% a.out
r1 is 3 x 4, with an area of 12
```
Here again are the four functions you are to implement:

```c
Rect_T Rect_new(double w, double h);
double Rect_area(Rect_T r);
void Rect_get_wh(Rect_T r, double *result);
void Rect_free(Rect_T *rp);
```

Below is a skeleton for `Rect_T`. **Start by filling in whatever members you'd like `Rect` to have.**

```c
typedef struct Rect {

} *Rect_T;
```
Problem 4: (1 point each; 11 points total)

Answer the following questions, all related to C.

(1) In as few characters as possible, what is the value of x++?

(2) What is an L-value?

(3) Imagine a C function void printInts(int a[]) that prints the integers in a. What's problematic about writing printInts?

(4) What is the value of sizeof("ab")?

(5) What does int (*f)(char); declare?

(6) Add parentheses to show the order of operations in *p++.

(7) If sizeof(int) is 3 and given int *p = (int*)100, what is p+5?

(8) If the type of p is int *, what does *p += 1 do?

(9) What's wrong with the following code?

```c
char *p;
strcpy(p, "xyz");
```

(10) Write a function void zero(int *p) such that given int n, after the call zero(&n), the value of n is zero.

(11) Write the int strlen(char *s) function.
Problem 5: (1 point each; 6 points total)

For each of the following multiple choice questions, circle the best answer.

(1) Which of the following words does "valgrind" rhyme with?
   (a) grand
   (b) grinned
   (c) grind
   (d) greened

(2) Despite its name, the C function `getchar()` returns an int. Why is that?
   (a) It's more efficient to return an int value than a char value.
   (b) If `getchar()` returned a char, there would be no good sentinel value.
   (c) It avoids the need to cast the return value to an int.
   (d) EOF is -1, and a char can't have the value -1.

(3) With regards to UNIX paths, a relative path is a path that
   (a) starts with a /
   (b) starts with a dot
   (c) does not start with a /
   (d) does not start with a dot

(4) On a bash command line, where can I/O redirections appear?
   (a) Only before the command
   (b) Only after the command
   (c) Anywhere on the line, but input redirection must precede output redirection
   (d) Anywhere on the line

(5) The term "translation unit" is related to
   (a) gdb
   (b) C
   (c) Makefiles
   (d) bash scripts

(6) Why is the command "cd ~" a poor way to switch to your home directory?
   (a) It should be "cd ~/" instead.
   (b) It should "cd $HOME" instead.
   (c) `popd` returns to your home directory, no matter where you are.
   (d) None of the above are true.
   (e) All of (a), (b), and (c) are true.
Problem 6: (5 points)

This question tests your knowledge of bash command-line anatomy. Draw a line connecting each of the following terms with a circled example of the term that appears in the command line below. Note that terms appear both above and below the command line.

command argument input redirection output redirection option piping

fgrep -v <(cat $words) TEST | cut $(pick 2 < x*y ) | wc -l > out

wildcard command substitution process substitution variable

Problem 7: (1 point each; 6 points total)

Answer the following questions, all related to UNIX.

(1) What's the difference between the man command and the help command?

(2) Write a one-line bash script count_hits STRING FILE that prints the number of lines in FILE in which STRING appears. Don't worry about any error handling. Example of use:

```
% count_hits char mkstrs.c
6
```

(3) In a UNIX path, a dot (.) is recognized as the name of the current directory. What's a case in which it's useful to use dot?

(4) Consider the following commands and their output:

```
% wc evalit.c
 92  209 1841 evalit.c
% wc < evalit.c
 92  209 1841
```

Briefly explain why wc prints the file name, evalit.c, in the first case but not the second.
Imagine you're struggling with a command line that uses wildcards and command substitution—it seems like the command isn't getting the arguments you expect it to be getting. How could you find out what arguments the command is really being invoked with?

What does the following command do?

```bash
% wc ~/352/????*.c
```

Extra Credit Section (½ point each unless otherwise noted)

1. I asked the TAs to learn all your names. That had mixed results but let's see if you've learned their names! For one point total, what is the first and last name of each of the TAs?

2. Assignment 12's `vector.c` was based on techniques from the book *C Interfaces and Implementations*. Who wrote that book?

3. A focus group has determined that the name of this class, *Systems Programming & UNIX*, is dry and unappealing. What's a better name for this class?

4. For a quarter-point each, write the character and code for four non-alphanumeric ASCII characters with a code >= 32.

5. What does `typedef char *string;` declare? Show a good example of it being used.

6. With regards to the errors that `valgrind` can detect, what's a downside of using `calloc` instead of `malloc` when the memory doesn't need to be zeroed?

7. Based on the amount of the work you've been asked to do this semester, how many units should this class be worth?

8. For up to two points, what did you find hardest to "get" about C, and why?