

**SHOW ALL WORK FOR PARTIAL CREDIT! IF YOU DON'T SHOW ME WHAT YOU'RE THINKING AND THEN MAKE A MINOR ERROR, I HAVE TO TAKE OFF ALL CREDIT!**

**THERE WAS MORE SPACE ON ORIGINAL EXAMS**

NAME \_\_\_\_\_

1. (20 points) Show the values printed by the following code fragment. **Be careful:** one of these statements **might have** an unusual effect on a later statement.

```
int a = 2, b = 3;
struct point {
    int x;
    int y;
} *pp, parr[ ] = { {12, 3}, {10, 7}, {8, 11}, {6, 15}, {4, 19} }; /* Note: 5 array elements */

pp = &parr[2];

a = pp[2].x;

printf ("%d, %d", a, (pp+1)->y ); ANSWER:

b = (*(++pp)).y +2;

printf ("%d", (++(pp+1)->x)); ANSWER:

pp = &((parr+1)[3]);

printf ("%d, %d", pp->x, b); ANSWER:
```

(2) (20 points) Write a function memdup with the following functional prototype:

```
void *memdup(void *xyp, int cnt);
```

The argument xyp is a pointer to some object, probably a struct or an array, and the argument cnt is the count in bytes of the object size. The function memdup should malloc space for a new copy of the object, and copy the object pointed to by xyp to the newly malloced space, returning the pointer to the new copy. Note that memdup() should return NULL if malloc returns an error. You need the library function memcpy() to copy the object; strcpy() won't work!

3. (25 pts) Write a main program noteerr.c to be compiled as: gcc noteerr.c -o noteerror, with the rule that noteerror should be invoked with either one or zero arguments (more than one will be ignored). The program will write the words: "NOTE ERROR" to the file named in the single argument (if it can be opened), appending these words to any prior content of the file; if there is no argument, the words will be written to standard output. If can't open file with arg1 name, return 1. HINT: see the cat program, pg 162 of K&R, and handle failure to open a file the same way. The noteerr.c program requires no more lines than the main of cat (no function needed).

II. (35 points, pts as marked) Write a program called "search.c" that will be compiled to the executable "search" and then executed at the Unix prompt with the statement:

```
%search pattern fnamefr fnameto
```

Here pattern is a search string and fnamefr and fnameto are files you need to open. The program you write should read a line at a time from the file fnamefr and write to fnameto exactly those lines that contain pattern as a substring. You will be led through this below.

(a) (2 pts) The program search.c consists of a single main function that provides for exactly three arguments. Write the introductory line of a main function (int main . . .) to accept these arguments.

Next, assume your program has the following declarations:

```
char line[1000]; FILE *r, *s;
```

Assume line[ ] is large enough to hold any line from the file fnamefr; in the program that follows, **DO NOT COPY** the command line arguments pattern, fnamefr, and fnameto to local arrays: instead always use argv to reference the strings that were stored as command line arguments.

(b) (3 pts) Main should now determine the number of arguments of the search command and return 1 if there are not the expected number (**DON'T** put out an error message -- just return 1).

(c) (7 pts) Open the fnamefr and fnameto files in the proper modes (writing to fnameto should not destroy old contents), setting r and s to the values returned, and returning 2 and 3 if there are errors opening. (Use single if statements with the fopen evaluated inside the tests. Remember to reference the fnamefr and fnameto character string in the command line argument.)

(d) (7 pts) Perform a while loop with a loop test containing an fgets function to read in line[ ] from r and a loop body specified below (just write <body> for now). The loop should terminate when fgets indicates there are no more lines to read in from r (assume this is not an error).

(e) (10 pts) In the body of the loop of question (d), test if the string in the pattern array appears in the line array (**use an appropriate string function for the test**), and if it does, write the line out to fnameto (use fputs or fprintf). If there is an error in writing the line, return 4.

(f) (3 Points) Close the files fnamefr and fname2; return 5 if there is an error closing either one of the files; else return 0.

(g) (3 pts) There are two #include statements you should have at the top of this program. Write them here and **LIST ALL FUNCTIONS** that you use from each header file you #include!

[At this point, you might want to put all these parts together on the back of some page to ensure you have a program that makes sense. I am not requiring this because you might be short of time, but you should at least THINK of how the parts fit together.]

## CS240 Exam 2 Practice 1, Solutions

```
(1) int a = 2, b = 3;
struct point {
    int x;
    int y;
} *pp, parr[] = {{12,3}, {10,7}, {8,11}, {6,15}, {4,19}}; /* 5 array elements */
                0         1         2         3         4
pp = &parr[2]; NOTE: pp points to parr[2]: (8, 11)
a = pp[2].x;   NOTE: pp[2] is parr[4], so a = 4
printf ("%d, %d", a, (pp+1)->y ); ANSWER: 4, 15 (NOTE: pp+1 points to (6,15))
b = (*(++pp)).y +2; NOTE: pp now points to parr[3], so b = 15 + 2 = 17
printf ("%d", (++(pp+1)->x)); ANSWER: 5 NOTE: pp+1 points to parr[4], x++ = 5
pp = &((parr+1)[3]); NOTE: pp points to parr[4], so pp->x below is 5 (set above)
printf ("%d, %d", pp->x, b); ANSWER: 5, 17
```

```
(2) void memdup(void *xyp, int cnt) {
    void * temp;
    if((temp = malloc(cnt)) != NULL)
        memcpy(temp, xyp, cnt);
    return temp;
}
```

Most common errors: act like xyp is some known struct, write struct xyp \* temp; as a declaration. We DON'T KNOW the type xyp points to, it's just a void \*.

(3) Use cat on pg 162 as a model, but of course no filecopy is being performed.

```
#include <stdio.h>
int main (int argc, char *argv[])
{
    FILE *fp;
    if (argc == 1) /* No args, then write to stdout */
        printf("%s", "NOTE ERROR"); /* "%s" not required, could just give string */
    else
        if((fp = fopen(argv[1], "a")) == NULL) { /* Only 1 arg counts */
            return 1;
        }
        else /* fopen worked fine */
            fputs("NOTE ERROR", fp); /* fclose is automatic at end of main() */
    return 0; /* even if no return type for main(), int return OK */
}
```

```
II.
#include <stdio.h> /* for fopen, fclose, fputs, fgets */
#include <string.h> /* for strstr */
```

```
main(int argc, char *argv[])
{
    char line[1000]; FILE *r, *s;

    if (argc != 4) return 1;
    if((r = fopen(argv[2], "r")) == NULL) return 2;
    if((s = fopen(argv[3], "a")) == NULL) return 3;
    while(fgets(line, 1000, r)) {
        if(strstr(line, argv[1]))
            if(fputs(line, s) == EOF)
                return 4;
    }
    if(fclose(r) == EOF || fclose(s) == EOF)
        return 5;
    return 0;
}
```

\*\* OPEN BOOK -- OPEN NOTES \*\*

**SHOW ALL WORK FOR PARTIAL CREDIT! IF YOU DON'T SHOW ME WHAT YOU'RE THINKING AND THEN MAKE A MINOR ERROR, I HAVE TO TAKE OFF ALL CREDIT!**

THERE WAS MORE SPACE ON ORIGINAL EXAMS

NAME \_\_\_\_\_

(1) (16 points) Say what is printed out by the following code fragment. As always, show work for partial credit.

```
int *pa, a[] = {2,4,6,8,10,12,14,16,18,20}, b[5];  
  
b[1] = *(a+2)+1;  
  
pa = &a[2] + 2;  
  
b[2] = *(pa+1);  
  
b[3] = (&(a[1])+1)++;  
  
b[4] = (a[2] ^ a[5]);  
  
printf("%d, %d, %d, %d", b[1], b[2], b[3], b[4]);
```

Answer:

(2) (16 points) Consider the following declaration that creates a struct for a *counted string*, a different concept than a *null-terminated* string:

```
typedef struct {  
    int count; /* count of chars in character string following */  
    char * cp; /* pointer to non null-terminated character string */  
} Cntstr;
```

Write a function (it should only require a few lines) with functional prototype:

```
Cntstr *ntstocs(Cntstr *csp, char *ntsp);
```

that will fill in the members of a counted string pointed to by *csp*, from a null-terminated character string pointed to by *ntsp*; your function should malloc the necessary space for the *cp* member and return the pointer *csp* or null if malloc fails. Use appropriate library functions.

(3) (18 points) Write a full main program (with #include statements, etc.) to be compiled as an executable file named "prsubstr" and executed with two command-line arguments, as follows:

```
%prsubstr token1 token2, i.e.: %prsubstr Hello,world! 7
```

Both token1 and token2 are character strings, but token2 represents an integer, k; use atoi( ) function to transform the character string token2 to int k. Your program should print out the first k characters of token1 (or if the number of characters of token 1 is less than k, then print all of token1). You are allowed to modify characters of the array in which token1 will appear. If there are **not exactly two** tokens, the program should print ERROR.

As an example, given: %prsubstr Hello,world! 7, the text printed out should be: Hello,w

(4) (20 points) You are given an array students[ ] of struct sttype below, and want to sort them using qsort, in order by grade.

```
struct sttype {
    char name[20];
    int grade;
} students[100];
```

(a) (10 points) Write the function: int cmpgr(struct sttype \*, struct sttype \*); to compare two struct sttype variables and return < 0, = 0, or > 0, according to grade values of the two.

(b) (10 points) Write the line you would use to call the C library qsort() function. Careful! Remember to cast appropriately; what is the size; give a real number for the argument n.

II. (30 points) Write a main program with arguments argc and argv, which will read a line from stdin and print out to the line "They match!" to standard output if all the argv[ ] values starting at argv[1] exactly match successive tokens on the line read from stdin (you will use strtok( ) as explained below), and "They don't match!" otherwise. You should use fgets to bring the stdin line into the char array line[ ] which you declare with dimension MAXLINE, where MAXLINE is a symbolic constant you have defined as 1000. Then use strtok() to retrieve successive tokens in the array line[ ], with space as the only separator character. DON'T WORRY ABOUT HANDLING ERRORS IN THE CODE; just keep it simple. YOU CAN ASSUME the number of tokens in the line is the same as the number you read from argv[ ], so just loop on argv[ ].

## cs240 Exam 2 Practice 2, Solutions

```
(1) int *pa, a[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20}, b[5];
           0  1  2  3  4  5  6  7  8  9
b[1] = *(a+2)+1; same as a[2]+1 = 7      b[1] = 7
pa = &a[2] + 2; pa points at a[4]
b[2] = *(pa+1); b[2] = a[5]              b[2] = 12
b[3] = *(&a[1])+1++; b[3] = a[2]++       b[3] = 6, a[2] = 7 afterward
b[4] = (a[2] ^ a[5]); This is XOR, 7^12, 0111
                                           ^ 1100
                                           1011 b[4] = 11
printf("%d, %d, %d, %d", b[1], b[2], b[3], b[4]); 7, 12, 6, 11
```

```
(2) Cntstr *ntstocs(Cntstr *csp, char *ntsp) {
    csp -> count = strlen(ntsp);           /* set count member          */
    csp -> cp = (char *) malloc(csp -> count + 1); /* cp member with space for '\0' */
    strcpy(csp->cp, ntsp);                 /* space is important          */
    return csp;
}
```

```
(3)
#include <stdio.h>
#include <string.h>

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

    if (argc != 3) {                       /* 3 for two arguments and program name */
        printf("Error\n");
        return 1;                           /* specific return value not important */
    }
    k = atoi(argv[2]);
    if (strlen(argv[1]) > k)
        argv[1][k] = '\0';                 /* terminate string at length k        */
    printf("%s\n", argv[1]);
    return 0;
}
```

```
(4) (a) int cmpgr(struct sttype *s1, struct sttype *s2) {
        return (s1 -> grade - s2 -> grade);
    }
    (b) qsort((void *) students, 100, sizeof(struct sttype),
              (int (*)(const void *, const void *)) cmpgr);
```

```

II. #define MAXLINE 1000    /* I didn't require #include <string.h>, <stdio.h>    */

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

    fgets(line, MAXLINE, stdin);
    for (i=1; i < argc; i++) {
        if (i == 1)
            tok = strtok(line, " ");
        else
            tok = strtok(NULL, " ");
        if (strcmp(tok, argv[i]) {
            printf("They don't match");
            return i;          /* not required to return i, just an idea    */
        }
    }
    printf("They match!");
    return 0;
}

```