CS 240, Comp. Arch. I — Fall, 2006
Exam 1 — **FIRST PRACTICE EXAM**F05 — P. O'Neil

NAME ____

I. (66 pts.) **<u>**SHOW ALL WORK FOR PARTIAL CREDIT**</u>**

(1) (16 pts) What is printed out by the following code fragment? <u>Start by showing how you would parenthesize these expressions to AGREE with the precedence table on page 53 of K&R, then</u> **<u>show your calculations of intermediate results</u>**.

```
unsigned char c;
. . .
c = '\136';


c = c % 0X20 ^ '\x28' >> 1;



printf("Value one is %o, Value two is %d, Value three is %x\n", c, 2*c, 3*c);
```

Answer:

(2) (12 pts) Say what is printed out by the following recursive function called with input integer n = 192 **<u>SHOW ALL SUCCESSIVE CALLS AND VALUES OF n FOR EACH</u>**!

```
void func1(unsigned int n)
{
   if (n / 3)
      func1(n / 3);
   putchar(n % 3 + '0');
}
```

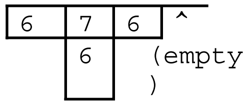
(a) (2 pts) Value of n for first call to func1? _192_ Value for successive recursive calls 2, 3, . . .

____ , _____ , _____, _____ , _____, . . .

(b) (6 pts) Values printed out by recursive calls? Give them in order printed: ___________

(c) (4 pts) Explain in a short English sentence what func1 is printing out for an arbitrary non-negative int n as an argument. (DON'T JUST PARAPHRASE THE PROGRAM CODE!)

(3) (12 pts) From homework on the calculator, show successive values pushed and popped on the stack to perform the calculation: "6 7 + 4 - 1 1 + * \n". Be careful to be specific: I want a solution showing successive contents of the stack as the calculation is performed (push gives deeper stack, pop hives shallower stack). Here is the start (^ means empty):



(Note new push goes on TOP of old.)

Answer (use back of prior sheet if you need extra space)

(4) (12 pts) Consider the following declarations in two different source files. First in source file main.c:

```
static int y;
int x;

main ( )
{
   int b;
   static int a;
}
```

In a different source file, func.c, we have declarations outside any function (these don't all work):

```
extern int x;
extern int y;
extern int a;
extern int b;
```

(a) Of the above variables, name those that are <u>automatic</u>. _________

(b) Name those that have a permanent value (storage location not on stack). __________

(c) Name those where the extern declarations in func.c will work. ________

(5) (12 pts) Consider the makefile with a single Macro and a single rule.

```
zzz =    cat

file3:   file1   file2
         $(zzz)  file2   file1  >  file3
```

Assume file1 is a file with the single line: "to come to the aid of the party.\n", while file2 has the single line: "Now is the time for all good men\n". Finally assume that file3 does not exist.
(a) After typing "make" at the Unix prompt, will file3 exist? If so, what will it contain?

(b) If you now edit file1 to change the period after party to an exclamation point, !, and then type "make" again, what will happen? What change in file contents (if any) will there be?

(c) If you now type "make" a third time without doing anything between the two "make"s, what will happen? What response will you get? What change in file contents (if any) will there be?

II. (36 pts) Write a program with main and a function explained below named "strindx", with functional prototype:

```
int strindx(char s[ ], char t[ ])
```

The function int strindx(char s[ ], char t[ ]) should return the **beginning** subscript position of the **leftmost** occurrence of the **entire** character string s ("time") in the character string t ("Now is the time for all good men to time races."), NOT counting the terminal '\0' in the string s (which need not be matched). If there is no occurrence of s in t, strindx( ) should return -1.

The main routine should declare two character strings named s1[ ] and s2[ ] with the same size, a symbolic constant set to 1000. The string s1[ ] should be initialized in its declaration to the char string "Now is the time for all good men to time races.", and s2[ ] should be initialized to "time" The main function should then call strindx() with the right arguments, print out both strings and then the return value, and terminate.

The function does **NOT** know the size of the strings in advance (they must be arguments), and thus cannot define a local array to hold either string (they might be longer strings than any guess you can make in writing main). You need to do everything one character at a time using the string array arguments passed. Use the facing sheet if you need extra space to write out the program.

WRITE EVERYTHING NEEDED FOR A PROGRAM IN A SOURCE FILE, all #declares and #defines and #includes, functional prototypes, etc.

cs240 F06, First Practice Exam 1, Solutions

(1) We want to calculate (proper parentheses): c = (c % 020) ^ ('\x2C' >> 1); So start with c % 0x20 where c = '\136' = (in 8 bits) '01'011'110' and 0x20 = '0010'0000'. When we divide '0101'1110' by the divisor '0010'0000', it should be obvious that the remainder will have its initial three bits zero, and all bits after that the same, i.e. the remainer is '0001'1110'. (We could also do this by calculating '\136' = 64 + 24 + 6 = 94, and 0x20 as 32, so the remainder after dividing is 30 = 16 + 8 + 6 = '0001'1110'. Now for the right hand ('\x28' >> 1), we get '\x28' = '0010'1000' and right shifting that 1 we get: '0001'0100'. Now we have to xor ''0001'1110'' and '0001'0100'. It is important to be neat and write one directly under another:

```
Here is the 28 from the left-hand term,                      '0001'1110'
and now here is the right hand term ('\x2C' >> 1) =        ^ '0001'0100'
                                                           -------------
Now xoring the two we get the final value for c:             '0000'1010'
```

In octal (our printf statement prints out c in %o), this number is '00'001'010' = 012. In decimal, this is 10, and two times this (we printing out for 2*c) is 20. Finally, 3*c in hex can be calculated by adding c = '0000'1010' and 2*c = '0001'0100':

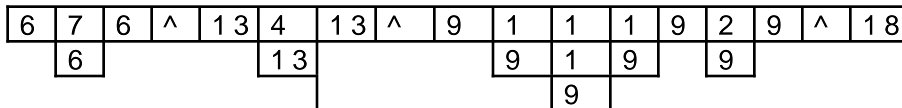
```
    '0000'1010'
+   '0001'0100'
---------------
    '0001'1110'
```

or 0x1E. So the Answer (exact wording, no leading 0 for octal or 0x for hexedecimal) is:

Value one is 12, Value two is 20, Value three is 1E

(2) (a) Successive recursive calls: 192/3 = 37, 64/3 = 21, 21/3 = 7, 7/3 = 2; (b) Values printed out by recursive calls in order: 2%3 = 2, 7%3 = 1, 21%3 = 0, 64%3 = 1, 192%3 = 0; (c) func1 is printing the value of n = 192 in base 3 representation: e.g., 12020: $2*3^4$ + $1*3^3$ + $0*3^2$ + 1*3 + 0*1 = 2*81 + 1*27 + 1*3= 162 + 27 +3 =192.

(3) I show the stack contents for "6 7 + 4 - 1 1 + * \n":



(4) (a) automatic: b (not a because it's static inside a function, thus permanent); (b) permanent values: x, y, a (x and y are globally defined and a is static inside a function); (c) extern will work: x (not y, because that's static outside a function, and thus local to the file, and a and b are of course local to main, not accessible anywhere outside).

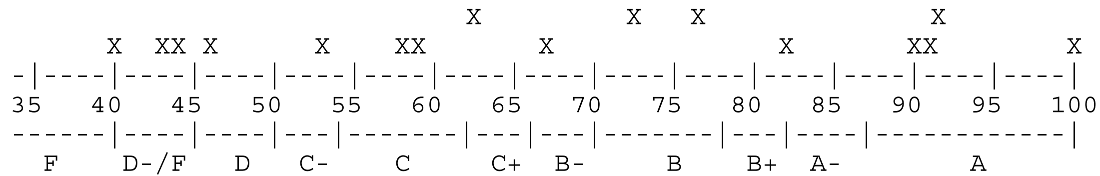
(5) (a) Yes, file 3 will exist and contain"Now is the time for all good men\nto come to the aid of the party.\n" (b) A new file3 will be created with ". . . party!\n" at the end. (c) There will be no change to any file. Response will be something like "file3 is up to date."

```c
#include <stdio.h>
#define LEN 1000
int strindx(char [ ], char [ ]);

int main()                      /* OK if no int return, but warning if gcc -Wall    * /
{
  char s1[LEN] = "Now is the time for all good men to time races.";
  char s2[LEN] = "time";
  printf("%s, %s, %d", s1, s2, strindx(s2, s1));
  return 0;                     /* OK if no return 0, but warning if gcc -Wall      * /
}
strindx(char s[ ], char t[ ])
{
  int i, j;

  for(i = 0; s[i]; i++)         /* loop through string s                            * /
   for(j = 0; t[j]; j++) {      /* loop through string t                            * /
     if(t[j]  !=  s[i+j])
        break;                  /* if string t not matched at s[i], forget it       * /
   if(t[j]  ==  '\0')           /* if left loop after full match . . .              * /
     return i;                  /* . . . found a match                              * /
  }                             /* keep looping until find a match                  * /
  return -1;                    /* failed to find a match                           * /
}
```

Letter Grade Scaling of Numeric Grades



```
                             X         X   X              X
      X  XX X      X    XX       X              X       XX        X
-|----|----|----|----|----|----|----|----|----|----|----|----|----|
35   40   45   50   55   60   65   70   75   80   85   90   95   100
------|----|----|---|-------|---|---|-------|---|----|------------|
  F    D-/F   D   C-    C     C+  B-     B    B+  A-        A
```