ECS 30 **Homework #3 (49 points)** **Winter 2013**

Due: Wednesday, January 30th. Written: 4pm in 2131 Kemper. Programs: 11:55pm using handin to cs30, p3 directory. Filenames: wind.c, taxes.c, grades.c, roman.c, number.c, pointers.c

Written (2 points): pp. 229-230: 5, 6

#5. Write an `if` statement that displays an acceptance message for an astronaut candidate if the person's weight is between the values of **`opt_min`** and **`opt_max`** inclusive, the person's age is between **`age_min`** and **`age_max`** inclusive, and the person is a nonsmoker (**`smoker`** is false).

#6 Implement the flow diagram in Fig. 4.14 using a nested `if` structure.

Zyante: 3.1-3.5, 6.8-6.10, 6.12

Programming (47 points)

Beware! The last program, pointers.c, is very difficult. Do not wait until Wednesday to tackle it!

All programs should be able to compile with no warnings when compiled with the –Wall option, e.g. gcc –Wall taxes.c You should put your name(s) in a comment on the first line of each file. Unless specified otherwise, your program need only contain a main() function. When you are to write your own functions, since main() should be the first function in each file, you will need to provide a prototype above main() for each function you write. You are not to use loops, nor arrays for any program. You are never allowed to use global variables in this course.

You will find my executables in ~ssdavis/30/p3, as well testing files in the CSIF. The prompts, and output format of each program must match the examples exactly. User inputs are in **`bold`**.

#1. Filename: wind.c (5 points) (5 minutes) Write a program that asks the user to enter a wind velocity (in knots), and then displays the corresponding description. Your program may only contain if statements (not if-else), and no other selection statements. Note that calm, and light air are adjectives, but breeze, gale, storm, and hurricane are nouns, which affects the grammar of the description.

Velocity (knots)	Description
Less than 1	Calm
1-3	Light air
4-27	Breeze
28-47	Gale
48-63	Storm
Above 63	Hurricane

```
[ssdavis@lect1 p3]$ wind.out
Please enter the wind velocity in knots: 2
That is light air.
[ssdavis@lect1 p3]$ wind.out
Please enter the wind velocity in knots: 28
That is a gale.
[ssdavis@lect1 p3]$
```

#2. Filename: taxes.c (7 points) (8 minutes) Write a program that prompts the user for their 2012 gross income, and then prints the federal taxes they owe (assuming they are an unmarried individual). Taxable income is the gross income minus the standard deduction, $5,950. You must use a single nested if-else statement for this program and no other selection statements.

Taxable Income	Amount of Tax
Not over $8,700	10% of the taxable income
$8,700 - $35,350	$870 plus 15% of the excess over $8,700
$35,350 - $85,650	$4,867.50 plus 25% of the excess over $35,350
$85650 - $178,650	$17,422.50 plus 28% of the excess over $85,650
$178,650 - $388,350	$43,482.50 plus 33% of the excess over $178,650
Over $388,350	$112,683.50 plus 35% of the excess over $388,350

The style format for if else is:

```
if(…)
  …
else // describe what remains
  if(…)
    …
  else  // describe what remains
    if(…)
      …
```

```
[ssdavis@lect1 p3]$ taxes.out
Please enter your gross taxable_income for 2012: 4987.37
The tax on $4987.37 is $0.00.
[ssdavis@lect1 p3]$ taxes.out
Please enter your gross taxable_income for 2012: 111829.15
The tax on $111829.15 is $23086.66.
[ssdavis@lect1 p3]$ taxes.out
Please enter your gross taxable_income for 2012: 450000.00
The tax on $450000.00 is $132178.50.
[ssdavis@lect1 p3]$
```

#3. Filename: grades.c (5 points) (3 minutes) Write that a program that converts a numerical test score to a letter grade. Use the following grading scale: A = 90-100, B = 80 – 89, C = 70 – 79, D = 60 – 69, F = 0 – 59. You must only use a switch statement, and no other selection statements. Hint: use only the tens digit of the score.

```
[ssdavis@lect1 p3]$ grades.out
Please enter a test score: 89
Letter grade: B
[ssdavis@lect1 p3]$ grades.out
Please enter a test score: 35
Letter grade: F
[ssdavis@lect1 p3]$ grades.out
Please enter a test score: 100
Letter grade: A
[ssdavis@lect1 p3]$
```

#4. Filename: roman.c (8 points) (10 minutes) Write a program that converts an integer value between 1 and 3999 into Roman numerals. The Roman system had I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, and M = 1000. You can preface a symbol with the next lower valued symbol to subtract the lower value from the larger value, e.g. CM = 900. Therefore there can never be more than three consecutive instances of any numeral.

```
[ssdavis@lect1 p3]$ roman.out
Please enter a number between 1 and 3999: 1
In Roman numerals that is I.
[ssdavis@lect1 p3]$ roman.out
Please enter a number between 1 and 3999: 3999
In Roman numerals that is MMMCMXCIX.
[ssdavis@lect1 p3]$ roman.out
Please enter a number between 1 and 3999: 1237
In Roman numerals that is MCCXXXVII.
[ssdavis@lect1 p3]$
```

#5. Filename: number.c (10 points) (15 minutes) Write a program the reads in a number between 0 and 99, and then writes out the number in English. Each switch statement must be in its own function separate from main().

```
[ssdavis@lect1 p3]$ number.out
Please enter a number between 0 and 99: 0
In English, that is zero.
[ssdavis@lect1 p3]$ number.out
Please enter a number between 0 and 99: 15
In English, that is fifteen.
[ssdavis@lect1 p3]$ number.out
Please enter a number between 0 and 99: 87
In English, that is eighty-seven.
[ssdavis@lect1 p3]$
```

#6. Filename: pointers.c (12 points) (20 minutes) Write a program that

1) Prints out the addresses of three ints you have declared in main(). Use %u in printf() and cast the addresses to (unsigned). For example if you have `int num;`, then you would `printf("%u", (unsigned) &num);`
2) Reads from the user the address of an int. Use %u in scanf(), and case the &int* to an (unsigned*). For example, if you have `int *num_ptr;`, then you would `scanf("%u", (unsigned*) &num_ptr);`
3) Checks that if this first address matches one of the int addresses presented in step 1 and indicates its validity;
4) If the first address is valid, then reads from the user the address of a second int.
5) If the second address was read, then checks that if the second address matches one of the int addresses presented in step 1 and indicates its validity;
6) If both addresses are valid, then reads from the user an int, named operand, and a char named operator. The operand is guaranteed to be valid.
7) If both addresses are valid, then tries to execute the specified operation. The operations are listed below. If the operator is invalid indicates so, otherwise prints the values of the two ints specified by the addresses,
8) Repeats steps 2 through 7 two more times.

Operator	Operation
'+'	Add the operand to the int of the first address, and store the result in the second addressed int.
'*'	Multiply the operand by the int of the first address, and store the result in the second addressed int.
'='	Assign both addressed ints with the value of the operand.
'%'	Divide the operand by the int of the first address, and store the remainder in the second addressed int.

Additional specifications

1. main() will have eight lines in its body.
   1.1. Three int declarations;
   1.2. The second line must be "`setvbuf(stdout, NULL, _IONBF, 0);`" to allow my testing shell script to function properly;
   1.3. A printf() that displays the addresses of the three ints
   1.4. Three calls to a function named operate().
   1.5. A final printf("Done\n")
   1.6. A return statement.
2. The operate() function may not contain any printfs().
3. The only place the '&' operator may be used as a parameter to one of your functions is in main() for the parameters to operate(), so most of your other functions will have to rely on their return statements to communicate with their calling functions.
4. No function may contain more than one "if" and/or one "switch". Therefore you cannot have an if-else-if… statement. Hint: Some of your functions will return a Boolean value as an int so a single "if" expression may call multiple functions, and rely on short circuiting to stop at the appropriate time.
5. By calling some functions more than once, I was able to write the program with a total of seven functions. You are not required to do that, but it will make it easier.

Note that there is no way to test this program with a standard input file because the addresses of the three variables will vary from run to run. So I have written a shell script called testPointers.sh that will test your program against mine for

you. You are welcome to look at the contents of testPointers.sh if you wish. It is written in the BASH shell language. testPointers.sh assumes that your executable is named a.out, and mine is named pointers.out, and that both are located in the current directory. testPointers.sh takes one command line parameter, the name of a special input file that I have written. I have demonstrated the use of testPointers.sh below.

The names of those input files are pointers*.txt, where '*" is replaced with a number, e.g. pointers2.txt. The tested input file must also be in the current directory. The format of the input files has the first three lines specifying the input for the three calls to operate(): <first int position> <second int position> <operand int> <operator>. By "position," I mean the order in which they are displayed to the user by the printf in main() described in 1.3 above. The last line is a bogus address for error checking, and is used when an address position is zero. If an error is expected (when an address position is 0), then the balance of the line is not provided since the program should move on to the next operate() call.

```
[ssdavis@lect1 p3]$ pointers.out
The addresses are num: 3215130060, num2:
3215130056, num3: 3215130052
Please enter int address #1: 3215130059
That is an invalid address.
Please enter int address #1: 3215130052
That is a valid address.
Please enter int address #2: 3215130000
That is an invalid address.
Please enter int address #1: 3215130056
That is a valid address.
Please enter int address #2: 3215130052
That is a valid address.
Please enter an integer: 5
Please enter an operator: -
- is an invalid operator.
Done
[ssdavis@lect1 p3]$
[ssdavis@lect1 p3]$ pointers.out
The addresses are num: 3214501324, num2:
3214501320, num3: 3214501316
Please enter int address #1: 3214501320
That is a valid address.
Please enter int address #2: 3214501316
That is a valid address.
Please enter an integer: 43
Please enter an operator: =
First int: 43, second int: 43.
Please enter int address #1: 3214501316
That is a valid address.
Please enter int address #2: 3214501324
That is a valid address.
Please enter an integer: 78
Please enter an operator: +
First int: 43, second int: 121.
Please enter int address #1: 3214501324
That is a valid address.
Please enter int address #2: 3214501320
That is a valid address.
Please enter an integer: 253
Please enter an operator: %
First int: 121, second int: 11.
Done
[ssdavis@lect1 p3]$

[ssdavis@lect1 p3]$ cat pointers2.txt
1 2 18 =
1 3 13 +
2 3 19 *
2888889990
[ssdavis@lect1 p3]$
```

```
[ssdavis@lect1 p3]$ testPointers.sh
Usage: testPointers.sh pointers#.txt
[ssdavis@lect1 p3]$ testPointers.sh pointers2.txt
Yours:
command: 3217919228 3217919224 18=
command: 3217919228 3217919220 13+
command: 3217919224 3217919220 19*
Sean's:
command: 3217731516 3217731512 18=
command: 3217731516 3217731508 13+
command: 3217731512 3217731508 19*
diff results:
6c6
< Please enter an integer: Please enter an operator: First int: 18, second int: 31.
---
> Please enter an integer: Please enter an operator: First int: 18, second int: 32.
done
[ssdavis@lect1 p3]$
```