QUIZ **AMAT 2120** **December 1, 2004**

NAME	ID

1. Is the assignment `n=(1==0);` correct?

✓ (a) yes

☐ (b) no, because of a wrong syntax

☐ (c) no, because 1 is not equal to 0

☐ (d) no, because 1 is not a variable and the assignment 1==0 is not valid.

Comments. The expression 1 == 0 is a relational expression (§ 4.1) which evaluates to FALSE, so its numeric value is 0. Thus, the given assignment is equivalent to `n=0;`. Choices (b) and (d) would be correct for the assignment `n=(1=0);` with single `=`.

2. Evaluate:

a) `2||0` Ans: 1	e) `1%2` Ans: 1
b) `2&&0` Ans: 0	f) `1/2` Ans: 0
c1) `2|1` Ans: 3	g) `(x+5>x)` Ans: 1
c2) `2&1` Ans: 0	h) `!(-3)` Ans: 0
d) `2%1` Ans: 0	

References. (a,b,g): p.125-126; (h): bottom of p. 126.
(c1,c2): p. 459; note: $2 = (10)_2$, $1 = (01)_2$, so $2|1 = (11)_2 = 3$, $2\&1 = (00)_2 = 0$.
(d,e,f): p. 44.

3. For the variables and addresses illustrated in the following Figure,

```
Variable  |  ptDay    ptYr       m     zAddr      x     ptNum
value     |    ?      8400    10000    18000      ?       ?
address   |  16000    17000   18000     8000    8400    9000
```

fill in the appropriate data as determined by the following statements:

`ptDay=zAddr;     *ptYr=1984;     ptNum=&m;`

Answer: The assigned values will be:

1) ptDay= 18000 (copied from zAddr).

2) x=1984. Comment: `*ptYr` refers to a variable at address equal to the current value of `ptYr`, which is 8400. The variable at 8400 is `x`. It will be assigned the value in the right-hand side of the statement.

3) ptNum=18000 (address of `m`).

4. A loop with header `for(;;)`

(a) is a syntax error

(b) will never terminate, therefore it is a programming error

(c) is not an error, but just useless, as it does nothing

(d) is useless, as it never comes to an end and therefore we'll never see a final result

(e) never stops, still can make sense in some programs

√ (f) may or may not terminate. Still can make sense.

Comments. Such a loop header is syntaxically correct and it make sense in some programs. In particular, in operating systems (which are also programs repeating their routine indefinitely), in such utility programs as window managers, shells, etc. (they are interrupted by the operating system). At a more down-to-earth level, there is a number of cases when such a loop can make sense. For example:

a) A program requests a number from the user, performs a calculation and prints the result. To perform a series of calculations, you can run the program several times (`a.out`) and enter different data. Alternatively, you can put the body of `main()` in the infinite loop and do the serial calculation without exiting the program. The loop never stops. To exit, press `Ctrl-C`.

b) The `for(;;)` loop may terminate if there is a `return` statement inside. For example, an input function may request data from the user until a valid number is obtained.

c) Using `break` statement (p. 181, which I didn't mention, since it can be avoided and isn't too good in terms of style), a loop with no exit condition can be stopped from inside. Such a loop can be made functionally equivalent to any of the loops we know (`for`, `while`, `do-while`).

5. How many bytes of memory does the string "bit" occupy?

(a) 0	√ (d) 4	(g) 3*`sizeof(char)`
(b) 1	(e) 8*3=24	√ (h) 4*`sizeof(char)`
(c) 3	(f) 8*4=32	(i) `sizeof(char*)`

Comment. The given string is an array of 4 characters: b,i,t, and NULL ('\0'). The answers (g) and (h) are both correct, but (h) is more fundamental, it doesn't depend on our knowledge of the number of bytes in a single character. The answer (f) is wrong; it gives the string length in *bits*. The answer (i) is also wrong; it represents the size of a pointer and not of the actual string. However, in the following assignment

```
        my_string="bit";
```

size of the variable `my_string` is `sizeof(char*)`: the variable `my_string` holds the address of the actual string. Its type must be `char*`.

6. What do you think of the following statement in a program? (It presumably implements Heron's Triangle Area formula.)

```
A= 0.25*sqrt( (sqrt(pow(x2-x1,2)+pow(y2-y1,2))+ sqrt(pow(x2-x3,2)+
pow(y2-y3,2))+sqrt(pow(x1-x3,2)+pow(y1-y3,2)))* (sqrt(pow(x2-x1,2)+pow(y2-y1,2))+
sqrt(pow(x2-x3,2)+pow(y2-y3,2))- sqrt(pow(x1-x3,2)+pow(y1-y3,2)))*
(sqrt(pow(x2-x1,2)+pow(y2-y1,2))- sqrt(pow(x2-x3,2)+pow(y2-y3,2))+
sqrt(pow(x1-x3,2)+pow(y1-y3,2)))* (sqrt(-pow(x2-x1,2)+pow(y2-y1,2))+
sqrt(pow(x2-x3,2)+pow(y2-y3,2))+ sqrt(pow(x1-x3,2)+pow(y1-y3,2))));
```

- ☐ Not optimal in terms of efficiency (some calculations are repeated), but explicit and clear. Good style.
- ✓ Poor style.

Comment. It is an example of a very poor programming style, which also hurts efficiency. The code is relatively easy to associate (vaguely) with Heron's formula, but difficult to make exact comparison, to trace all the brackets, and to debug. There are a number of way to rewrite this code in a better style. One possibility is this:

```
double vectorLength(double x, double y)
{
  return sqrt(x*x+y*y);
}

 ....
 ....
 double a,b,c,s,A;   /* sides, semiperimeter and area of triangle */
 ....
 a=vectorLength(x2-x3,y2-y3);  /* side a */
 b=vectorLength(x1-x3,y1-y3);  /* side b */
 c=vectorLength(x2-x1,y2-y1);  /* side c */
 s=(a+b+c)/2.0;                /* semiperimeter */
 A=sqrt(s*(s-a)*(s-b)*(s-c));  /* Heron's formula */
```