Eng101 Quiz 2

Sections 102 & 106

Winter 1999

You are allowed to use either one double-sided or two single-sided sheet(s) of notes.
No other assistance or references are permitted. You are not permitted to use computers.

I have neither given nor received aide on this examination, nor have I concealed any violation of the honor code.

Name (Printed): Solution Key

Section: 102 & 106

Signature: *Solution Key*

1. Short Answer / Multiple Choice / True or False (5 points each)

1A. The following is a valid statement in C.

```
*x = *y * *z;
```

a. True

b. False

1B. Which of the following loops will fill the array `numbers` with integers from 1 to N?
(Circle all that apply)?

a. `for( i = 0; i <  N; i++ ) numbers[ i ] = i;`

b. `for( i = 1; i <= N; i++ ) numbers[ i ] = i;`

c. `for( i = 0; i <  N; i++ ) numbers[ i ] = i + 1;`

d. `while( 1 <= i <= N ) numbers[ i ] = i;`

e. None of the above

1C. Given the following variable declarations:

```
int i, j, k, m[ 10 ];
```

Which of the variables **cannot** be changed by the following statement?
(Circle all that apply)

```
i = myfunction( j, &k, m );
```

a. `i`

b. `j`

c. `k`

d. `m`

e. All four of the variables could possibly be changed.

1D. Which of the following **best** describes the purpose of a prototype?

a. A prototype is an instruction for the compiler to include the stdio header file.

b. A prototype declares a function, so subsequent code can be checked for the correct numbers and types of arguments.

c. A prototype is used to call a function.

d. The prototype is the first line of a function.

e. Prototypes are not necessary in ANSI C.

1E. Given the following bit of code:

```
struct Date
{
   int Mo, Day, Yr;
};

struct Account
{
   char Name[100];
   int SSN;
   struct Date DateOpened;
   float Balance;
   int NextCheckNum;
};

int main()
{
   struct Date today={3, 17, 1999}, tomorrow;
   struct Account checking;

   /* ...rest of code */

}
```

Which of the following are **not** valid C statement(s)? (Circle all that apply)

a. struct Account savings;

b. tomorrow = today;

c. Account.SSN = 123456789;

d. checking.Balance = checking.Balance - $144.19;

e. checking.DateOpened.Mo = 3;

f. checking.DateOpened = today;

g. (All of the above are valid C statements.)

2. Code Tracing - (5 points each)

For each of the following code fragments, indicate in the center column whether or not it would compile. In the third column, explain either: (a) Why it would not compile, or (b) What would be the result of running the code. The first entry has been completed as an example.

Code	Compile?	Why not? **OR** What would be the results?

`int i = 0;`
`while( i < 5 );`
`  printf( "i = %d\n", i++ );`
	Yes	This code contains an infinite loop, because of the semicolon at the end of the while statement. Nothing will be printed.

`int i = 0;`
`while( i < 5 )`
`  printf( "i = %d\n", i++ );`
	Yes	This is the "correct" version of the example. Output:
		i = 0
		i = 1
		i = 2
		i = 3
		i = 4

`typedef struct {`
`  int x, y;`
`} double;`
	No	It is illegal to name a new type with the same name as a standard C type (double).
		
		If "double" were changed to a valid name, this would create a new type, which is a structure containing two ints x, and y.

`float func( int n[], int s ) {`
 	
`  float r = 0.0;`
 	
`  for( i = 0; i < s; i++ )`
`  r += n[ i ];`
 	
`  return r / s;`
 	
`}`
	No	The variable `"i"` has not been declared.
		
		If `i` had been declared, this function would return the **average** value of the first `s` elements in the array `n`.

`int i, nums[ 10 ];`
 	
`while( i != 10 )`
`  nums[ i++ ] = 1;`
	Yes	Because `"i"` has no initial value, the results of running this code are undefined.
		
		Most likely `"i"` will have some huge value, causing an access violation.
		
		If `"i"` were initialized to 0, the `while` loop would fill the first 10 elements of `nums` (i.e. `nums[ 0 ]` to `nums[ 9 ]` with ones. `"i"` would then be 10.

`int i = 10, j = 20, *ip, *jp;`
 	
`ip = &j;`
`jp = &i;`
 	
`*jp = 2 * *ip + *ip / *jp;`
`(*ip)++;`
 	
`printf( "i = %d\n", i );`
`printf( "j = %d\n", j );`
`printf( "*ip = %d\n", *ip );`
`printf( "*jp = %d\n", *jp );`
	Yes	The output would be:
		
		`i = 42`
		`j = 21`
		`*ip = 21`
		`*jp = 42`
		
		Note carefully that the pointer variable `"ip"` holds the address of `j`, not `i`, and similarly for `jp`. Therefore `ip` points to `j` and `jp` points to `i`.
		
		Also note that `*ip` **IS** `j`. Anything that changes one changes both.

`typedef struct {`
`  int day, month, year;`
`} date;`
 	
`date today = {17,3,1999},`
`  tomorrow;`
 	
`tomorrow = today;`
`tomorrow.day++;`
 	
`printf(`
`  "Tomorrow is %d/%d/%d",`
`  tomorrow.day,`
`  tomorrow.month,`
`  tomorrow.year );`
	Yes	The output would be:
		
		`Tomorrow is 18/3/1999`
		
		`today` and `tomorrow` are both declared as variables of type `"struct date"`. `today` is initialized with `{17,3,1999}`, and then this data is copied to `tomorrow` by the assignment operator `( = )`. The day field of `tomorrow` is incremented and the results printed.

3A. Code Creation (30 points) - Assessing the Value of a Structure (Note: You should do either this problem or 3B, not both.)

The following page contains the beginning of a rudimentary real-estate management program, which uses a defined structure type "House" to keep track of housing information. Your task is to write the function "appraiseHouse", which has been prototyped as:

```
int appraiseHouse( House thisHouse );
```

This function should accept a `House` structure as its argument, and appraise the value of the house as follows:

1. The land (`lot_size`) is valued at $8 per square foot.

2. If the `floor_space` is greater than 1500 square feet, add $50 for every square foot over 1500.

3. **OR**, if the `floor_space` is less than 1500, subtract $30 for every square foot under 1500.

4. Add $2000 for every `bedroom` and $1000 for every `bathroom`.

5. Add $2000 for every "car" worth of `garage` space. For example, a 1.5 car garage is larger than a single car garage but smaller than a two-car garage.

6. Decks, fences, paved driveways, and air-conditioning are features that are either present or not, as indicated by a 1 if present or a 0 otherwise. **IF** the house has any of the following features, add the appropriate bonus:

Deck:	$2000	Fence:	$1000
Paved driveway:	$2000	Central Air:	$1000

7. After all of the above has been totaled, the resulting sum is multiplied by two adjustment factors:

   - The `condition` is a floating point value from 0.5 to 2.0 which reflects the condition of the property.
   - The `location` is a floating point value from 0.25 to 4.0 which reflects the neighborhood in which the property is located.

8. The above calculations should be carried out using floating point mathematics, since several of the terms involve fractions. However the return value should be an int, rounded down to the nearest $100. For example, if the above calculations yield a value of $132,487.65, the function should return 132400.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>       /* ?? */

typedef struct {
        int lot_size, floor_space, bedrooms;    /* ft^2, ft^2, # of rooms */
        float bathrooms, garage_size;           /* # of rooms, # of cars */
        int deck, fence, paved, AC;             /* 1 or 0, Is it there? */
        float condition, location;              /* Multiplier adjustments */
} House;

#define MAX_HOUSES      1000

/* Function Prototypes */

int loadInventory( House houses[], int arraysize, char *filename );
int updateInventory( House houses[], int arraysize, int nhouses );
void saveInventory( House houses[], int nhouses, char *filename );
void printHouse( House thisHouse );
int appraiseHouse( House thisHouse ); /* Needs to be written! */

/* And now for main.  This program loads up the existing inventory from
   a datafile, updates the inventory by querying the user, and saves
   the updated inventory back to the file.  Then it loops through the
   array of houses, calling appraiseHouse for each house, and finally
   prints out the most expensive house. */

int main() {

        int i, nhouses, max = 0, imax = 0, value;
        House houses[ MAX_HOUSES ];

        /* First to update the inventory */

        nhouses = loadInventory( houses, MAX_HOUSES, "myHouses.dat" );
        nhouses = updateInventory( houses, MAX_HOUSES, nhouses );
        saveInventory( houses, nhouses, "myHouses.dat" );

        /* Then to find the most valuable house */

        for( i = 0; i < nhouses; i++ ) {
                value = appraiseHouse( houses[ i ] );
                if( value > max ) {
                        max = value;
                        imax = i;
                }
        }

        /* And print it out */
        printf( "The most valuable house is worth $%d\n", max );
        printf( "Here are it's specifications:\n\n" );
        printHouse( houses[ imax ] );

        return 0;
}

/* Function appraiseHouse.  This function will take in a variable
   "thisHouse" of type "House" ( a structure ), and appraise the
   value of the house using some simple formulas.  The result is
   returned as an integer rounded DOWN to the nearest $100.

   ( It should be noted that the real formulas used by housing
   appraisers involve many more factors than those described
   here, and generally start with the selling price of similar
```

```
   houses in the neighborhood, rather than starting with the
   land value. */

int appraiseHouse( House thisHouse ) {

     int ivalue, area;
     float value;

     /* First start with the land value */

     value = 8.0 * thisHouse.lot_size;

     /* Then add the floor space of the house */

     area = thisHouse.floor_space  - 1500;
     if( area > 0 )
          value += 50.0 * area;
     else
          value += 30.0 * area;    /* area is negative  here */

     /* Now add for bedrooms, bathrooms, and garage space */

     value += 2000.0 * thisHouse.bedrooms;
     value += 1000.0 * thisHouse.bathrooms;
     value += 2000.0 * thisHouse.garage_size;

     /* Deck, etc. are 0 or 1 variables */

     value += 2000.0 * ( thisHouse.deck + thisHouse.paved );
     value += 1000.0 * ( thisHouse.fence + thisHouse.AC );

     /* Finally multiply by condition and location */

     value *= thisHouse.condition * thisHouse.location;

     /* And convert the result to an integer, rounded down */

     ivalue = value / 100;
     ivalue *= 100;

     /* Don't forget to return the result! */

     return ivalue;

}
```

3B. Code Creation (30 points) - Maximizing the Value of an Array
(Note: You should do <u>either</u> this problem or 3A, not both.)

Write a function which takes a *10 x 10* array of ints, each of which is either 1 or 0, and returns the maximum number of 1's found in any single row or column. For example, if your function were passed an array containing

1	0	0	1	0	1	0	0	0	1
0	0	0	1	1	0	1	0	0	0
0	1	1	1	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1
0	0	0	1	1	0	1	1	0	0
0	1	1	0	0	1	0	0	0	1
1	0	0	0	0	1	0	1	0	0
0	0	0	0	1	1	0	1	1	0
0	0	0	0	0	1	0	0	1	0
1	1	1	0	1	0	0	0	0	0

it should return a 5 (the number of ones in the sixth column).

Note: This solution depends upon the array being **<u>square</u>** of size 10. If the array is not square, then two sets of nested loops would be required.

```
int maxones( int a[ ][ 10 ] ) {
     int i, j, rowOnes, colOnes, max = 0;
     for( i = 0; i < 10; i++ ) {
          rowOnes = colOnes = 0;
          for( j = 0; j < 10; j++ ) {
               rowOnes += a[ i ][ j ];  /* ones in row i */
               colOnes += a[ j ][ i ];  /* ones in col i */
          }
          if( rowOnes > max ) max = rowOnes;
          if( colOnes > max ) max = colOnes;
     }
     return max;
}
```