Name: ______________________________

1. (10 points) For the following, Check T if the statement is true, or F if the statement is false.

   (a) [✓ T] F : The expression x || y has the exact same **logical** value as the expression x | y for all values of x and y.

   The bitwise version will probably have a different numerical value, but if any bit in x or y is on, then a bit will be on in the bitwise OR of x and y, which is true, so these two expressions evaluate to the same logic answer even though their numeric values may be different.

   (b) T [✓ F] : The arguments to the main function always have the same values, no matter how the program was invoked.

   Arguments to the main function depend on what was specified on the command line when the program was invoked.

   (c) T [✓ F] : In mathematics, we learned that every integer, n has a successor, $n + 1$, that is greater than n. This is also true in the C language.

   In C, integers must have a type, which controls the number of bits used to represent that number, and since the number of bits is finite, there are limits on the size of an integer that can be represented in C.

   (d) [✓ T] F : In C, the assignment statement **x=3;** evaluates to the value **3**, but that value is typically discarded.

   (e) [✓ T] F : In C, a **while(1) { ... }** loop is an endless loop unless the loop body contains a break keyword.

   (f) T [✓ F] : A compiler is a tool that takes a binary executable file, and turns it into man-readable C code.

   A compiler turns man-readable C code into a binary executable file.

   (g) T [✓ F] : The condition in a "while" statement cannot be a single variable such as "while(x) { ...". The condition must compare x to a literal constant such as "while (x!=0) { ...".

   The condition can be any expression, including a single variable, which evaluates to true or false.

   (h) [✓ T] F : If aliens from outer space prevent all computers from typing (or cut and pasting) the character "}" (right curly brace), then no one could write any new C code.

   In C, almost all statements must be in a function, and it is impossible to create a function without using a right curly brace.

   (i) [✓ T] F : If a case block inside a switch statement does not end with the "break" keyword, then control flows to the next sequential case block.

   (j) T [✓ F] : A specific set of bits in memory will always be interpretted as the same value by the compiler, no matter what type of data is associated with that memory.

   Different types of data are interpretted differently by the compiler. For instance 11111111 is interpreted as -1 if it is a signed char, but as 255 if it is unsigned char.

Answer the following questions by checking each correct answer.

2. (10 points) Check the items that are are valid literal values in C.

[X] 21	[X] 0x3Ed6	[X] 0xdeadbeef
[ ] x	[ ] 0b0324	[X] 3 * 41.5
[X] ')'	[X] -0472	
[ ] 0960	[X] 7.54e-6F	

3. (10 points) Check the statements that are valid declaration statements in C. (You may assume **stdbool.h** has been included.)

[X] const double constant=3.5E016;	[X] int number='a';
[ ] /* int x=17; */	[X] float fraction=7;
[X] short width=-3;	[ ] bit moreData=true;
[X] int y; // int if x = 7;	[ ] short peaches&cream=17;
[ ] int while=1;	[X] float forwhile=1.275e-3;

4. (10 points) For the following, put a check in front of the statements that are syntactically correct, in other words, will compile without any errors, assuming the following declare statements:

```
int n=5; char init='W'; float g=9.8; int nums[5];
```

[X] if (n==3) init='W';	[ ] foreach(n : int nums[]) n++;
[X] n = n/init;	[ ] n = n * 3
[ ] n = n # 3;	[ ] n += n->data;
[ ] if (init >= 'M') then n==5;	[X] int y1 = n;
[X] while(init<0) init++;	[ ] if (3=n) init='X';

5. (5 points) Check the single best answer to fill in the blank in the following statements:
   (a) The ____home____ directory is the directory you start in when you open a new terminal, and which you can use to hold your own personal files.
   ☐ current ☐ backup [✓ home] ☐ blue ☐ base

   (b) When defining a C function, you must specify zero or more ____arguments____ to the function by specifying a comma separated list of types and identifiers surrounded by parenthesis to represent the data provided to the function.
   [✓ arguments] ☐ return types ☐ names ☐ bodies ☐ parrot meters

   (c) When a C program is executed in the order that the instructions appear in the C code, we call that ____sequential____ control flow. This is the default execution order in a C program.
   ☐ in-order [✓ sequential] ☐ consequential ☐ looping ☐ blocked

   (d) In C, a variable must be ____declared____ before it can be used to hold a value.
   ☐ typed ☐ instantiated [✓ declared] ☐ devoured ☐ thought up

   (e) The C compiler ignores ____white space____ other than to indicate a delimiter between tokens.
   ☐ blanks ☐ tabs ☐ new lines ☐ comments [✓ white space]

6. (5 points) What is printed when you execut the following C code:

```
int n=2;
float x = n * 2.3;
int m = x * 2;
printf("m=%d\n",m);
```

☐ m=0 ☐ m=4 ☐ m=4.6 ☐ m=8 [✓ m=9] ☐ m=9.2 ☐ None of the above

Since 2.3 is a floating point constant, n*2.3=4.6, so x=4.6. Since x is float, x*2 is calculated in floating point arithmetic, so is 9.2. However, the floating point result is assigned to an int, so is converted to in by dropping the .2 to make 9.

7. (5 points) Given the following C code:

```
---------- poundsToGrams(float pounds) {
   return 454.0* pounds;
}
```

What is the most appropriate return type for the poundsToGrams function (pick the **single best answer**.)

☐ short ☐ int ☐ long ☐ char * ☐ double [✓ float]

Cleary a floating point result is required, but double precision is misleading because it implies that the result is greater precision than the argument.

8. (5 points) Given the following code:

```
int x; int count=0;
while (x > 0) {
   count++;
   x=x - count;
}
printf("count is %d\n",count);
```

What will get printed?

- ☐ count is 0
- ☐ count is 1
- ☐ count is 12
- ☐ count is 1342
- [✓ Any of the above - results are undefined]
- ☐ None of the above

The value of **x** depends on whatever is in memory at the time the compiler assigns x to memory, which might be any value. It is acceptable to check all of the first four answers as well as "any of the above", but incorrect to select only one numeric answer.

Answer the following questions by filling in the blanks.

9. (10 points) Given the following declarations:

```
int a = 4; int b=-2; int c=5; float fx=3.1; float fy=-2.1;
```

Determine the value of the following expressions (Be sure to use a decimal point in floating point answers, and leave out decimal points in integer answers.)

b < (fx+3.0)	1	12.0	12.0
fx / (a + b)	1.55	a + b * c	-6
c * fy	-10.5	(a>c ? 1:0) * fx / fy	0.0
c / (a + b)	2	fy / (b * b / a)	- 2.1
(2==b) || !(2==b)	1	fy * b / (a * c)	0.21

10. (5 points) What does the following code print if it is invoked as stats(69,249)? (Hint: Very little arithmetic is required to answer this question!)

```
void stats(int hits, int atBats) {
   int battingAverage = (hits/atBats) * 1000;
   printf("Batting average is %d\n",battingAverage);
}
```

Batting avaerage is 0

11. (5 points) What does the following C code print when compiled and executed?

```
int a = 13; // a is 13
a = a // + 3;
/* a= 21 */* 2;
int c = /* a + /* 3 + /* 4 */ + 12 /* - 13 */ ; /*
if (a<30)
*/ { printf("a=%d, c=%d\n",a,c); }
// else { printf("a is too big\n");
```

a=26, c=12

12. (10 points) Answer the questions below based on the following C program:

```
#include <stdio.h>
int main() {
   printf("Enter a number to factor :> ");
   int n; scanf("%d",&n);
   int f=2;
   printf("%d=",n);
   while(f*f<=n) {
      if (0== n%f) { // Does n/f have a remainder of zero?
         printf("%d x ",f);
         n = n / f;
      } else f++;
   }
   printf("%d\n",n);
   return 0;
}
```

(a) When this program prints the factor, f, is that value a prime number or not, and why?

It is prime because all lesser factors have already been removed.

(b) Would this program work if the variable "f" was initialized to "1" instead of "2"? If not, why not?

No, since n%1=0 for any n. The loop in main would never end (f would never get incremented).

(c) What would this program print if compiled and invoked as "./primeFact", and the user enterred the value "60" at the prompt?

60 = 2 x 2 x 3 x 5

(d) Would the program produce the correct answer in all cases if the increment of f was performed in the then block as well as the else block?

No, the program would not work when the same factor appears twice like 4 = 2 x 2

(e) What does this program do?

Prints all the prime factors of a number in increasing order.

13. (10 points) Write a C program that reads two floating point numbers, x and y, uses a constant epsilon defined to be 1e-3, and prints to stdout based on the following rules:

- prints "Origin" if x and y are both within epsilon of zero,
- prints "X left" if y is within epsilon of zero and x is less than negative epsilon,
- prints "X right" if y is within epsilon of zero and x is greater than epsilon,
- prints "Y down" if x is within epsilon of zero and y is less than negative epsilon,
- prints "Y up" if x is within epsilon of zero and y is greater than epsilon,
- prints "Quadrant I" if x is greater than epsilon and y is greater than epsilon,
- prints "Quadrant II" if x is less than negative epsilon and y is greater than epsilon,
- prints "Quadrant III" if x is less than negative epsilon and y is less than negative epsilon
- prints "Quadrant IV" in all other cases.

You may use the library function `fabs` to compute the floating point absolute value of a number if you include `math.h`.

Solution:

```c
#include <stdio.h>
#include <math.h>

int main() {
   const float eps=1e-3;
   float x,y;
   printf("Enter two numbers :> ");
   scanf("%f %f",&x,&y);
   if (fabs(x) <= eps) {
      if (fabs(y) <= eps) printf("Origin\n");
      else if (y < -eps) printf("Y down\n");
      else printf("Y up\n");
   } else if (fabs(y) < eps) {
      if (x < -eps) printf("X left\n");
      else printf("X right\n");
   } else if (x < -eps) {
      if (y < -eps) printf("Quadrant III\n");
      else printf("Quadrant II\n");
   } else if ( y < -eps) printf("Quadrant IV\n");
   else printf("Quadrant I\n");
   return 0;
}
```

Solution: Bit Twiddler's Solution, complicated but efficient:

```
#include <stdio.h>
#include <math.h>

int main() {
   const float eps=1e-3;
   float x,y;
   printf("Enter two numbers :> ");
   scanf("%f %f",&x,&y);
   /*---flags-----------------------------------------------
      1 bit: fabs(x)<=eps
      2 bit: x>eps
      4 bit: fabs(y)<=eps
      8 bit: y>eps
    ------------------------------------------------------*/
   char flags = (fabs(x)<=eps) + 2*(x>eps) +
          4*(fabs(y)<=eps) + 8*(y>eps);
   switch(flags) {
      case 0: printf("Quadrant III\n"); break;
      case 1: printf("Y down\n"); break;
      case 2: printf("Quadrant IV\n"); break;
      case 4: printf("X left\n"); break;
      case 5: printf("Origin\n"); break;
      case 6: printf("X right\n"); break;
      case 8: printf("Quadrant II\n"); break;
      case 9: printf("Y up\n"); break;
      case 10: printf("Quadrant I\n");
      // Other cases have either 1&2 or 4&8 which are illegal
   }
   return 0;
}
```

Question:	1	2	3	4	5	6	7	8	9	10	11	12	13	Total
Points:	10	10	10	10	5	5	5	5	10	5	5	10	10	100
Bonus Points:	0	0	0	0	0	0	0	0	0	0	0	0	0	0