Name: ______________________________

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

   (a) [✓ T] F : In the ASCII alphabet, the only difference between an uppercase alphabetic character and that same character in lower case is a single bit, which can be represented as the value for a blank (0x20).

   (b) T [✓ F] : If you change two different clones of the same repository in different ways, git will be able to merge those changes automatically when you push both sets of changes into the cloud.

   Git will try to automatically merge divergent sets of development, but rarely is able to resolve differences, and almost always requires human intervention to resolve conflicts. We strongly recommend single-threaded development to avoid this conflict.

   (c) T [✓ F] : The same bits in memory are always interpreted to be the same value, no matter what type is associated with those bits.

   (d) T [✓ F] : We can use bitwise AND to turn off some bits in a value, but leave others unchanged, we can use bitwise OR to flip some bits (change 0's to 1's and 1's to 0's) but leave others unchanged, and we can use bitwise XOR to turn on some bits but leave others unchanged.

   Bitwise OR turns on some bits, bitwise XOR flips some bits.

   (e) T [✓ F] : In an assignment (or argument evaluation), C converts the value of the expression to the most general type and converts the receiver of the assignment to that type.

   C evaluates the expression using the most general type, but then converts the result to the type of the receiver.

   (f) [✓ T] F : If you continue to add 1 to a 4 byte two's complement integer, eventually, you will reach the value of -27.

   Even if you start at -26, eventually you will overflow to negative numbers and get to -27.

   (g) [✓ T] F : Even though computed results may look exactly the same, the underlying hardware on a little-endian machine is very different than the equivalent hardware on a big-endian machine.

   (h) [✓ T] F : The concept of sub-normal floating point numbers enables the representation of much smaller (closer to zero) numbers than we could represent with only normal floating point numbers, but we lose precision... as numbers get smaller and smaller, they also become less and less precise.

   This is true... every time the power of 2 gets smaller, we can use fewer FRAC bits to represent sub-normal numbers, and hence they are less precise.

   (i) [✓ T] F : C functions always pass arguments by copying the value of the argument into the parameter, but by passing a pointer, we can mimic a "call by reference" paradigm because even though the address is copied, the memory that the address is pointing to is shared between the caller and the called function.

   (j) T [✓ F] : It is possible to initialize all elements of an array to 1 using the syntax **int** vec[10]={1};.

   That will initialize vec[0] to 1, but all remaining elements of vec will be initialized to 0.

2. (5 points) If a floating point number is represented by the bits 0x8000003f, then the floating point value of that number is:

   - [ ] Less than -2 to the -127
   - [X] Greater than -2 to the -127, but less than 0
   - [ ] Zero
   - [ ] Greater than zero, but less than 2 to the -127.
   - [ ] Greater than 2 to the -127

3. (5 points) What is does the following code snippet print?

```
int x=2; int y=4;
if (x==1)
    if (y==2)
        x=6;
else
    y=9;
printf("x=%d y=%d\n",x,y);
```

[X] `x=2 y=4` [ ] `x=6 y=9` [ ] `x=6 y=4` [ ] `x=2 y=9` [ ] Compiler error.

An ambiguous else statement is associated with the closest if statement even though the indentation suggests otherwise.

4. (5 points) Check all the statements below that are correct comparisons of a float variable to a double variable.

   - [X] The float variable takes half the memory of the double variable.
   - [ ] The float variable can hold a smaller (closer to zero) value than the double variable.
   - [ ] The float variable is more precise than the double variable.
   - [X] The float variable cannot hold a non-infinite value as large as the double variable.
   - [X] There are some values which must be denormalized as a float, but can be normalized as double.

5. (5 points) What does the following snippet of code print out?

```
int x=5;
if (x=6) x++;
printf("x is %d\n",x);
```

[X] x is 7 [ ] x is 5 [ ] x is 6 [ ] Nothing (compile error) [ ] Nothing (seg. violation)

6. (5 points) What does the following snippet of code print to stdout?

```
int x[]={3, 1, 1, 4, 2};
char *y[]={"cat","dog","turkey","ant","owl"};
printf("%s\n",y[x[x[4]]]);
```

☐ cat [✓ dog] ☐ turkey ☐ ant ☐ owl

y[x[x[4]]]==y[x[2]]==y[1] which contains the address of the literal string "dog".

7. (5 points) What does the following snippet of code print to stdout?

```
int arr[] = {1, 2, 3, 4};
int i, sum=0; int *arptr=arr;
for (i = 0; i < 4; i++) { sum += *arptr; arptr++; }
printf("%d\n", sum);
```

☐ Compiler error ☐ 6 ☐ ???... reads unitialized memory [✓ 10] ☐ 24

When i==0, arptr==&arr[0]; i==1, arptr==&arr[1], and so on. So sum==1+2+3+4==10.

Answer the following questions by filling in the blanks.

8. (5 points) Given the following snippet of code, what will get printed to stdout?

```
int x[7];
x[6]=1+(x[5]=1+(x[4]=1+(x[3]=1)));x[2]=1+(x[1]=1+(x[0]=0));
printf("x[4]=%d\n",x[4]);
```

`x[4]=2`

This problem demonstrates assignment expressions. x[3]=1, x[4]=1+1=2, x[5]=1+2=3, x[6]=3+1=4, x[0]=0, x[1]=1+0=1, x[2]=1+1=2.

9. Given the following snippet of code:

```
int i;
for(i=0;i<100;i++) {
    if (0==i%2) continue;
    if (0==i%4) printf("You won $100!\n");
    if (i>15) break;
}
printf("The final value of i is %d\n",i);
```

(a) (2 points) Will the code snippet above print out `You won $100!`? ___No___

If i is divisible by 4, it is also divisible by 2, so the continue would already have happened.

(b) (3 points) The code snippet above will print out `The final value of i is` ___17___

When i=16, you will continue. The first time the break statement is true is when i=17.

10. (5 points) Given the following C code:

```
#include <stdio.h>
union {
    int num;
    unsigned char ncs[4];
} test;
int main() {
    test.num=0x000000ff;
    if (test.ncs[0]==0x??) printf("This is a little-endian machine\n");
    else printf("This is a big-endian machine\n");
    return 0;
}
```

What literal value would you specify in place of 0x?? so that the correct message gets printed?

0xff

11. (5 points) Given the following snippet of code, what will get printed to stdout?

```
float fx=2.56;
int y=fx*2;
printf("y=%d\n",y);
```

y=5

12. (5 points) What is does the following code snippet print to stdout?

```
int mat[2][3][4]; int slice; int row; int col;
for(slice=0;slice<2;slice++) {
    for(row=0;row<3;row++) {
        for(col=0; col<4; col++) {
            mat[slice][row][col] = slice*20 + row*5 + col;
        }
    }
}
printf("mat[1][2][3]=%d\n",mat[1][2][3]);
```

mat[1][2][3]=33

mat[1][2][3]==33 because it was written by mat[1][2][3]=1*20 + 2*5 + 3;

13. (5 points) Given the following snippet of code, what will get printed to stdout?

```
unsigned char x=255; // maximum unsigned char value
printf("x+1=%d ",x+1);
printf("++x=%d\n",++x);
```

x+1=256 ++x=0

x is unsigned char, but gets expanded to an int because %d expects an int, but ++x is evaluated in an unsigned char environment, so is set to zero BEFORE it's value is cast to int!

14. Consider the following program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char encode(char in,char * perm);
int main(int argc, char **argv) {
   char c;
   while(EOF !=(c=getchar())) {
      putchar(encode(c,argv[1]));
   }
   return 0;
}
char encode(char in,char * perm) {
   int i=in-'a';
   if (i<0 || i>strlen(perm)) return in;
   return perm[i];
}
```

This program is based on the fact that all ASCII lowercase letters map to consecutive numbers in alphabetical order. For instance, $a = 97, b = 98, c = 99, d = 100$, and so on.

If this code is compiled as executable file `encode`, then the command

```
echo "this is a test" | ./encode zyxwvutsrqponmlkjihgfedcba
```

... would write the following string to stdout: `gsrh rh z gvhg`

(a) (30 points) Write a C program (including the main function) which takes a permutation of the alphabet as the command line argument, and decodes the encoded result as specified by the encode function above. For instance, the command...

```
echo "this is a test" | ./encode zyxwvutsrqponmlkjihgfedcba | ./decode zyxwvutsrqponmlkjihgfedcba
```

... would write the string `this is a test` to stdout.

Solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char decode(char in,char * perm);
int main(int argc,char **argv) {
   char c;
   while(EOF !=(c=getchar())) {
      putchar(decode(c,argv[1]));
   }
   return 0;
}

char decode(char in,char * perm) {
   int i;
   for(i=0;i<strlen(perm);i++) {
      if (perm[i]==in) return 'a'+i;
   }
   return in;
}
```

Note... it turns out that the encode function will also decode itself! You could also copy exactly what was given above.

(b) (10 points (bonus)) If your code from the previous question works correctly, and is compiled as `decode`, what will get written to stdout as a result of the following command:

```
echo "vzhb zh z y x!" | ./decode zyxwvutsrqponmlkjihgfedcba
```

Solution: easy as a b c!

Question:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Total
Points:	10	5	5	5	5	5	5	5	5	5	5	5	5	30	100
Bonus Points:	0	0	0	0	0	0	0	0	0	0	0	0	0	10	10