Name: ______________________________

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

(a) ☐ T [✓ F] : A single variable in C may have several addresses at any given time.
Variables may have only a single address at any given time.

(b) [✓ T] ☐ F : C allows us to define five dimensional arrays, although it is so difficult to visualize five dimensional arrays, they are hardly ever used.

(c) [✓ T] ☐ F : In gdb the parameter of a print statement can be just about any valid C expression, including arithmetic operations or invocations of functions in our code. When gdb executes the print command, it will evaluate the expression and print the result.

(d) ☐ T [✓ F] : In gdb, the "previous" command will back up to the previous instruction executed, and reset all variables to the values they had at that previous instruction.
There is no good way to back up in gdb.

(e) ☐ T [✓ F] : The following code: `int nums[6]; for(i=0; i<=6; i++) nums[i]=i;` will result in an array bounds violation if the code is commpiled and executed.
C does not perform run time array bounds checking. The code will run without error, but write into memory it does not own, and potentially cause a later error.

(f) ☐ T [✓ F] : An empty string in C takes no memory.
An emptry string takes at least one byte of memory for the null terminator.

(g) [✓ T] ☐ F : A C program may use a pointer as an alternative to a variable name to read or write a value in memory.

(h) ☐ T [✓ F] : It is typical to start gdb, type in the "run" command, and then, when the (gdb) prompt appears again, set breakpoints at the lines where you want gdb to stop.
In gdb, set breakpoints BEFORE running.

(i) ☐ T [✓ F] : If you run the gcc compiler without the "-g" flag then your code will run much slower because it will not be optimized.
The -g flag tells the compiler to include debug information, and actually REDUCES the optimization. You code runs very slightly slower with the -g option.

(j) ☐ T [✓ F] : Every C function uses exactly one parameter.
C functions can have as many parameters as you wish, defined by how many comma separated fields appear inside the parenthesis in a function definition.

2. (10 points) Given the following C code:

```
#include <stdio.h>
int argGen(int aga) {
   static int nc=0;
   nc++;
   printf("%d - argGen(%d)\n",nc,aga);
   return aga*nc;
}

int sum(int s1, int s2, int s3, int s4) {
   return s1+s2+s3+s4;
}

int main() {
   int s=sum(argGen(1),argGen(2),argGen(3),argGen(4));
   printf("Sum is %d\n",s);
   return 0;
}
```

When this code is run on a specific machine, it produced the following somewhat counter-intuitive results:

```
>./argEval
1 - argGen(4)
2 - argGen(3)
3 - argGen(2)
4 - argGen(1)
Sum is 20
```

Clearly, this is not what the developer expected to happen. Choose the **single best** response from the developer:

- [ ] Complain to the compiler writers. The arguments should be evaluated in order.
- [ ] Depend on the fact that the arguments are always evaluated in reverse order.
- [ ] Tell the user, "Sorry - it's a bug in the operating system and I can't fix it."
- [ ] Blame the problem on my co-developer... he wrote that part of the code.
- [X] Rewrite the code so that the answer does not depend on the order in which the arguments are evaluated.
- [ ] See if changing the compiler optimization level will change the behavior.

The C specification states that the order of argument evaluation is up to the compiler, and that different compilers may choose different orders of argument evaluation. The best answer is to rewrite the code so that it doesn't depend on the order of argument evaluation.

3. (10 points) While debugging a program to track the height of a ball, the following code was in an endless loop:

```
 5  ...
 6  float v0=atof(argv[1]); float delta=1/10;
 7  float t=0.1; float h;
 8  do {
 9     h=v0 * t - ( GRAVITY * t * t / 2);
10     printf("t=%f h=%f\n",t,h);
11     t+=delta;
12  } while(h>0.0);
```

In order to figure out what was happening, I generated the following gdb output

```
(gdb) b 11
Breakpoint 1 at 0x100401154: file ballPath.c, line 11.
(gdb) run 10
Starting program: /home/Thomas/cs211/src/ballPath 10
t=0.100000 h=0.950950

Thread 1 "ballPath" hit Breakpoint 1, main (argc=2, argv=0xffffcc20)
    at ballPath.c:11
11                         t+=delta;
(gdb) p t
$1 = 0.100000001
(gdb) p h
$2 = 0.950950027
(gdb) c
Continuing.
t=0.100000 h=0.950950

Thread 1 "ballPath" hit Breakpoint 1, main (argc=2, argv=0xffffcc20)
    at ballPath.c:11
11                         t+=delta;
(gdb) p t
$3 = 0.100000001
(gdb) p h
$4 = 0.950950027
```

Check the **single most likely** reason for the endless loop

- ☐ The h variable is not initialized
- ☒ The delta variable is initialized to zero because of integer division
- ☐ The t variable is overwritten by an array bounds overflow
- ☐ The t variable is never initialized
- ☐ The value of GRAVITY was not expressed with enough decimal places (precision)
- ☐ The formula for the calculation of h is incorrect

The expression to initialize delta is 1/10, which uses integer division, and results in an integer 0. When this is assigned to delta, it is converted to float 0.0. So t will remain 0.1, and the formula for h will alsways evaluate to 0.950950, so h will always be positive.

4. (10 points) Given the following C program:

```
#include <stdio.h>
void chooseGift(int nice,char *list[]) {
    static int i=0;
    static char * gifts[]={"train","elmo","sweater"};
    if (nice) list[i]=gifts[i%(sizeof(gifts)/sizeof(gifts[0]))];
    else  list[i]="coal";
    i++;
}
int main() {
    char * gl[5];
    chooseGift(1,gl);     chooseGift(0,gl);
    chooseGift(2,gl);     chooseGift(-1,gl);
    chooseGift(-2,gl);
    printf("Gifts: %s, %s, %s, %s, and %s\n",
        gl[0],gl[1],gl[2],gl[3],gl[4]);
    return 0;
}
```

If the program is compiled and executed, what will get printed?

- ☐ Gift: elmo, train, sweater, coal, coal
- ☐ Gift: train, elmo, sweater, train, elmo
- ☐ Gift: train, coal, train, train, train
- [✓ X] Gift: train, coal, sweater, train, elmo
- ☐ Gift: elmo, coal, sweater, elmo, sweater

Answer the following questions by filling in the blanks.

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

```
int evalGate(char gateType,int i1, int i2) {
    int out=-1;
    switch(gateType) {
        case '&':
            if (i1==0 || i2==0) out=0;
            else out=1;
            break;
        case '|':
            if (i1==0 && i2==0) out=0;
            else out=1;
            break;
        case 'X':
            if (i1==i2) out=0;
            else out=1;
    }
    return out;
}
```

Evaluate the following expressions:

(a) evalGate('&',1,1) 1

(b) evalGate('|',1,1) 1

(c) evalGate('&',1,0) 0

(d) evalGate('Z',1,0) -1

(e) evalGate('X',1,1) 0

(f) evalGate('&',1,7) 1

(g) evalGate('&',12,1) 1

(h) evalGate('X',0,1) 1

(i) evalGate('|',−2,0) 1

(j) evalGate('X',12,7) 1

6. (10 points) Given the following C code:

```
#include <stdio.h>

int main() {
   char str[100]=
"We hold these truths to be self evident; that all men are created equal";
   int i;
   for (i=0; str[i]!=0x00; i++) {
      ------------------------------------;
   }
   return 0;
}
```

What line of code would you put in place of the dashes above to make the code print out `e at` n where n is the index from the beginning of the string of the location of an 'e' character. For instance "e at 1" for the e in "We".

```
if (str[i] == 'e') printf("e at %d\n",i);
```

7. (15 points) Given the following C header in stack.h:

```c
void reset();
void push(char in);
char pop();
```

And the following C code in stack.c:

```c
#include "stack.h"
#include <assert.h>
#define STACKSIZE 100
char stack[STACKSIZE]={0};
int top=0;
void reset() { top=0; }
void push(char in) {
    assert(top<STACKSIZE);
    stack[top++]=in;
}
char pop() {
    if (top==0) return 0x00;
    return stack[--top];
}
```

And the following C code in useStack.c:

```c
#include <stdio.h>
#include "stack.h"
#include <assert.h>
#include <stdlib.h>

int isPalin(char *word);
int main(int argc, char **argv) {
  for(int i=1;i<argc;i++) {
    printf("%s is%s a palindrome\n",argv[i],isPalin(argv[i])? "" : " not");
  }
  return 0;
}

int isPalin(char *word) {
  reset();
  for(int i=0;word[i]!=0x00;i++) push(word[i]);
  for(int i=0;word[i]!=0x00;i++) if (word[i]!=pop()) return 0;
  return 1;
}
```

(a) Immediately after a the reset() function is invoked, and before push() is invoked, what value will be returned by the pop() function?

0x00 (zero) or null terminator.

(b) If these are compiled with **gcc -g -Wall -o useStack useStack.c stack.c** and executed as **./useStack madam gets redder**, what will get printed to standard output?

```
madam is a palindrome
gets is not a palindrome
redder is a palindrome
```

(c) Why did the programmer assert(top<STACKSIZE) in the push function in stack.c?

When top==STACKSIZE then the buffer is full, and no more data can be added without causing errors.

8. (10 points) Given the following code:

```
int isSpecial(char texta[], char textb[]) {
   char checkMatch[256]={0};
   assert(strlen(texta)<256);     assert(strlen(textb)<256);
   int i; int j;
   for(i=0;i<strlen(texta); i++) {
      char c=texta[i];
      if (c==' ') continue;
      for (j=0; j<strlen(textb); j++) {
         if (textb[j]==c && checkMatch[j] == 0) {
            checkMatch[j]=1;
            break;
         }
      }
      if (j==strlen(textb)) return 0;
   }
   for(i=0; i<strlen(textb); i++) {
      if (checkMatch[i]==0 && textb[i]!=' ') return 0;
   }
   return 1;
}
```

(a) What is the value of isSpecial(”jim morrison”,”mr mojo risin”)? 1

(b) What is the value of isSpecial(”tom marvolo riddle”,”i am lord voldemort”)? 1

(c) Before the first loop, are all the elements of the checkMatch array initialized to 0? Yes

(d) Is it possible to return 1 if strlen(texta)!=strlen(textb)? Why or why not?

Yes, because texta and textb may have a different number of blanks, but still have the same letters.

9. (15 points) Write a C function that takes a string (a pointer to the first of a null-terminated list of characters) as an argument, and returns the number of times two adjacent letters are the same in that string. For instance, in the string "llamas have eyes and teeth", there are two ll's adjacent in llama, and two e's adjacent in teeth, so you would return 2. You may assume the same letter never occurs three times in a row in the string. You may use either pointer notation or array notation in your function.

```c
int countDup(char *str) {
    char prev=0x00; int count=0;
    while(0x00 != (*str)) {
        if ((*str)==prev) { count++; }
        prev=(*str);
        str++;
    }
    return count;
}
```

Or, using arrays...

```c
int countDup(char *str) {
    int i; int count=0;
    if (str[0]==0x00) return 0;
    for(i=1;str[i]!=0x00;i++) {
        if (str[i]==str[i-1]) count++;
    }
    return count;
}
```

Question:	1	2	3	4	5	6	7	8	9	Total
Points:	10	10	10	10	10	10	15	10	15	100
Bonus Points:	0	0	0	0	0	0	0	0	0	0