Name: ____

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

   (a) [ ] T [X] F : One of the disadvantages of using the `make` command is that even if the Makefile is written correctly, make will not guarranty that all of the pre-requisite steps in a build process are completed before evaluating the requested target.

   In fact, the real advantage of make is that, if the Makefile is written correctly, i.e. all dependencies are correctly specified, make **will** perform all required pre-requisite steps before evaluating the requested target.

   (b) [ ] T [X] F : The main function should return a true value (e.g. 1) if it ends successfully, and a false value (0) if there is a problem.

   The operating system interprets any non-zero return value from main as an error, and a zero return value as successful program completion.

   (c) [X] T [ ] F : In an assignment statement, C will convert the value of the expression on the right hand side of the equals sign to the type of the left hand side, even if this causes the value of the expression to change.

   (d) [X] T [ ] F : Upside down code can be avoided by putting function declarations at the top of a C file. However doing so requires specification of the return type, function name, and argument list twice.

   (e) [ ] T [X] F : In C, the leftmost bit of an unsigned integer can be used to determine if that number is positive or negative.

   True for signed integers, but not true for unsigned integers, which are always positive no matter what the leftmost bit value is.

   (f) [X] T [ ] F : The advantage of little-endian numbers is that it is possible to cast a little-endian number to a different width without changing its starting address.

   (g) [ ] T [X] F : Floating point arithmetic is much simpler than integer arithmetic because floating point numbers are already expressed in a normalized form.

   We learned in Lab02 and HW02 just how difficult floating point addition and multiplication are, much more complicated than integer arithmetic.

   (h) [ ] T [X] F : A pointer to an int (int *) is 32 bits wide, but a pointer to a char (char *) is only 8 bits wide.

   A pointer to anything takes as many bits as it takes to represent an address in memory, usually 64 bits.

   (i) [X] 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 ”call by reference” because even though the address is copied, the memory at that address is shared between the caller and the called function.

   (j) [X] T [ ] F : The malloc function returns a void pointer because it cannot predict what type of data you will put into the malloced memory. Before you use the pointer, you need to cast it to a pointer to a non-void type to tell the compiler what type of data that memory will hold.

Answer the following by checking all correct answers.

2. (5 points) What is the output to stdout from the following code snippet?

```
int x=2; int y=4;
if ( (x = 5) || (y = 8) ) printf("x=%d y=%d\n",x,y);
else printf("y=%d x=%d\n",y,x);
```

☐ `x=2 y=4` ☐ `y=4 x=2` ☐ `x=5 y=8` [✓ `x=5 y=4`] ☐ `x=7 y=12`

x=5 is an assignment, not a comparison, whose result is 5, and therefore true. Since the first half of the logical or expression is true, the second half is not executed, so no assignment to y occurs. Partial credit (-2) for answering x=5, y=8 because you got the assignment, but missed the short circuit.

3. (10 points) What does the following code snippet print?

```
char * buf=calloc(100,1);
strcpy(buf,"Good stuff here!");
free(buf);
// Other code here which does not change buf
printf("Buf contains: %s\n",buf);
```

☐ `Buf contains:`

☐ `Buf contains: 0`

[✓ `Buf contains:` followed by arbitrary characters which may not be printable ASCII values.]

☐ `Buf contains: Good Stuff here!`

☐ Compiler error.

After memory is freed, other parts of programs may modify that memory. It is possible that no other part of the program has modified the memory that Buf is pointing at, but the most correct answer is arbitrary characters.

4. (5 points) Which order represents increasing generality of numeric types in C.

☐ double, float, unsigned int, int, char

[✓ char, unsigned char, long, unsigned long, float, double]

☐ char, char *, string, struct record, union flint

☐ unsigned char, char, unsigned long, long, float, double

☐ char, short, long, double, short, char

5. (10 points) Given the program in Listing 1 on page 7 on the tear-off page, check all of the programming techniques used in this code:

- [X] Array of Structures
- [ ] Structure of Arrays
- [X] Pointer notation used for Array access
- [X] Array notation used for Pointer access
- [X] Binary Tree Data Structure
- [ ] Linked List Data Structure
- [ ] Use of Dynamic(heap) Memory
- [X] Pointer Arithmetic
- [ ] Union to redefine memory
- [ ] Explicit Casting

Answer the following questions by filling in the blanks.

6. (10 points) What does the following snippet of code print out?

```
int matrix[2][4]={{0,1,2,3},{10,11,12,13}};
if (matrix[1][2]==matrix[0][6]) printf("Very Cool!\n");
else printf("What were you thinking?\n");
```

`Very Cool!`

C doesn't check for array bounds overflow, and matrix[1][2] is the 6th element of the matrix vector; and so is matrix[0][6]!

7. (10 points) Given the program in Listing 1 on page 7 on the tear-off page, if this program is compiled into an executable file called **stray**, what will get printed to the screen if the following command line is run: ./stray these are arguments

`./stray are arguments these`

All four command line arguments will be put into a binary tree, and the contents of that tree will be printed in alphabetical (ASCII) order.

8. (10 points) Given the program in Listing 1 on page 7 on the tear-off page,
   (a) How many command line arguments can be specified before errors may occur?

   Up to 10 arguments (including the command itself) can be specified

   After 10 arguments, the array bounds of the node array will overflow, and all sorts of bizzare behavior may result.

   (b) Would the following change to line 15 of the program...

   for(**int** i=1; (i<argc) || (i<10) ;i++) {

   ... fix this problem?

   No, need logical AND, not logical OR.

   (c) (5 points (bonus)) If line 13 of the program were changed to...

   struct tnode node[argc];

   ... would that remove the limit on the number of command line arguments?

   Yes.

   A new feature of C allows parameters to be used to allocate arrays. The compiler inserts code to malloc the correct amount of space for you when the function starts.

9. (10 points) Given the program in Listing 1 on page 7 on the tear-off page,
   (a) Is the tnode structure "self-referential"? If so, why; if not, why not?

   Officially tnode is not self-referential because it does not refer to struct tnode.

   But, the intent is to refer to other instances of the tnode structure with the left and right fields, so even though tnode is not officially self-referential, it does have an element of self-reference built in.

   (b) If the tnode structure were re-defined so that the left and right fields were pointers intstead of integers, would an instance of the structure be larger or smaller?

   Larger - pointers take 8 bytes, integers take 4 bytes

10. Consider the program in Listing 2 on page 8 on the tearoff pages. If this code is compiled as executable file `encode`, then the command

```
echo "This is a test" | ./encode 3
```

... would (possibly) write the following string to stdout...

```
Thbilps jidgs rajx tceuyste
```

(a) (20 points) Write a C program (including the main function) which takes a single number 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 3 | ./decode 3` ... would write the string `This is a test` to stdout.

Solution: Note... encode adds extra random letters to the output. If you run encode again (and call it decode), it will just add MORE random letters to the output. This is not a code which has it's own encoder as a decoder.

Note also, if you use a loop like the following:

```c
for(i=0;i<cn;i++) {
    if (i==0) putchar(c);
}
```

This loop will execute cn times, but only the first iteration of the loop body will do anything, because only when i==0, the first iteration of the loop, will anything happen. Every other iteration of the loop does nothing. Therefore, the loop itself could be replaced by the statement:

```c
putchar(c);
```

... which accomplishes the same thing as the loop. The loop is ONLY useful if we do something in other iterations of the loop.

The entire solution is not presented so we can study this more in lab.

(b) (5 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 "I taopcekdbr thhwpism ontedssxt" | ./decode 3
```

Solution: I aced this test

Listing 1: stray.c

```c
#include <stdio.h>
#include <string.h>
struct tnode {
   char * payload;
   int left;
   int right;
};
void defineNode(struct tnode *node,char * payload);
void insertNode(struct tnode * node,int root,int n);
void printTree(struct tnode *node,int root);

int main(int argc, char **argv) {
   struct tnode node[10];
   defineNode(node,argv[0]);
   for(int i=1;i<argc;i++) {
      defineNode(node+i,argv[i]);
      insertNode(node,0,i);
   }
   printTree(node,0); printf("\n");
   return 0;
}

void defineNode(struct tnode *node,char * payload) {
   node->payload=payload;
   node->left=-1; node->right=-1;
}

void insertNode(struct tnode * node,int root,int n) {
   int c=strcmp((node+n)->payload,(node+root)->payload);
   if (c<0) {
      if ((node+root)->left==-1) (node+root)->left=n;
      else insertNode(node,(node+root)->left,n);
   } else {
      if ((node+root)->right==-1) (node+root)->right=n;
      else insertNode(node,(node+root)->right,n);
   }
}

void printTree(struct tnode *node,int root) {
   if (root==-1) return;
   printTree(node,(node+root)->left);
   printf("%s ",(node+root)->payload);
   printTree(node,(node+root)->right);
}
```

Listing 2: encode.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
char randChar();
int main(int argc, char **argv) {
    time_t t; srand((unsigned) time(&t));
    int n=atoi(argv[1]);
    int cn=1; char c;
    while(EOF !=(c=getchar())) {
        int i;
        for(i=0;i<cn;i++) putchar(i==0?c:randChar());
        cn++;
        if (cn>n) cn=1;
    }
    return 0;
}
char randChar() {
    char ltrs[]="abcdefghijklmnopqrstuvwxyz";
    return ltrs[rand()%strlen(ltrs)];
}
```

Question:	1	2	3	4	5	6	7	8	9	10	Total
Points:	10	5	10	5	10	10	10	10	10	20	100
Bonus Points:	0	0	0	0	0	0	0	5	0	5	10