1

2:55

NAME: Brandon Fletcher

Mid-term Exam 1

57/80

Instructions:

WRITE YOUR NAME AT THE TOP OF EACH PAGE IN THE SPACE PROVIDED! Please write legibly, but when writing your name and when answering questions.

Answer questions in the spaces provided, or on the last page, which is blank and provided for writing longer or additional answers. DO NOT write on the backs of any sheet. If you use the scratch sheet to answer a questions, clearly indicate so in the space for the question, and clearly mark the answer as to that question on the scratch sheet.

This exam is open book, open notes, and you are allowed to use the ZyBooks and cplusplus.com Web sites as references. You may not search the Web beyond these sites. You may not collaborate or discuss any of these questions with other students during the exam.

Point Distribution:

Question 1 is in 20 parts, each worth one points, for a total of 20 points.

Question 2 is in 10 parts, each worth two points, for a total of 20 points.

Questions 3, and 4 are worth 5 points each.

Question 5 is worth 10 points, and includes a 2 point bonus component.

Questions 6 and 7 are worth 10 points each.

There are 80 points possible. These points will be scaled to fit the 5% stated in the syllabus, and thus do not reflect the point totals for other components of final grades.

Don't panic, this is only 5 percent of your grade.

1. Answer the following TRUE/FALSE statements in the space provided at the left of each. (Print "TRUE" or "FALSE" as your answer, not T or F, to avoid confusion.)

False a. C is a high level language designed to implement graphical user interfaces.

False b. C compilers use a two-pass compilation process.

False c. In C, the size of a single **char** is uniform across all common computing platforms.

True d. An **int** type variable is always composed of four bytes or 32 bits for all C compilers.

False e. There are only two types of loop statements in C: **for** and **while**.

True f. All C variables and functions must be declared prior to referencing them.

True g. The **&** symbol in C is used to obtain the contents of a memory location referenced by the name the **&** symbol is prefixed to.

True h. To declare a **long int** variable, you may either declare it explicitly as a **long int**, or simply as a **long**.

False i. Declaration of a variable as a **long long** is not a valid declaration in C.

True j. A **typedef** statement may be used to create a type name that may then be used to declare instances of the type defined by the **typedef**.

False k. All C programs must have a **main** function of type **void**.

True l. In a C program file, a variable declared before any functions are declared may be accessed by all functions in that file.

True m. In C, a variable declared inside a function may be accessed outside that function if it is declared **public**.

True n. You cannot assign a literal to a char variable, as in **myStr = "Yes";**, in C.

True o. In C, you can assign a literal to a char array when declaring the array.

False p. Loops in C require braces for the loop body even when that body is a single statement.

True q. A **do while** loop will always execute at least once when execution reaches it.

False r. A C **if** statement cannot contain math operations such as *****, **+**, **/** and **%**.

False s. The C expressions **a > b** and **!(a <= b)** are logically equivalent.

True t. The **&&** and **||** operators in C perform bit-wise logical *and* and *or* operations, respectively.

-5

2. For each of the following actions, write C instructions that perform those actions.

a. Increase the **int** variable **x** by 10.

```
x = x + 10;
```

b. Declares a variable **pix2** of type **double** and initializes it to two times 3.14159.

```
double pix2 = (3.14159) * 2;
```

c. Assigns the value of two times **a** plus **b**, the result divided by three, to variable **c**.

```
int c=(2*(a+b))/3;
```

d. Assign the remainder of **x** divided by **y**, with one added to the result, to **z**.

```
int z = (x %y) +1;
```

e. Divide **x** by **n** minus **i** and assign the result to **z**.

```
int z= (x /n) - i;
```

f. Assign the larger value of two variables **tip** and **top** to **max**.

```
if (tip > top){ max = tip} else if (top > tip) {
max = top} else { printf("equal\n"); }
```

other works if they are equal! no need to

g. Read doubles **dX** and **dY** from **stdin** in a single function call.

```
scanf("%lf %lf", &dx, &dy);
```

h. Print the **int i** and two times **i** with three blank spaces between to **stdout**.

```
printf(" %d --- %d", &i, &(i * 2));
```

-1

i. Sums the integers **i** through **j** as **sum**. (Assume **i** < **j** before execution.)

```
while (i < j) {
    sum= i +j;
    ++i;
}
```

-2

j. Calculate integer **prod** as **i** times **j** if **i** is less than **j**, else set **prod** to zero.

```
if (i<j){
    prod= i*j;
} else {
    prod=0;
}
```

-3

3. Given the following program in C, what will the program print? Explain.

```
#include <stdio.h>
#define MESSAGE "Murphy was an optimist!\n"

int main(int argc, char *argv) {
    int n = 1;

    while (n < 4)
        printf("Rule %d: %s", n, MESSAGE);
        n++;
    printf("That's all you need to know about life.\n");
    return 0;
}
```

3

Rule 1: Murphy was an optimist!
Rule 2: Murphy was an optimist!
Rule 3: Murphy was an optimist!
That's all you need to know about life.

4. Describe what is wrong with the following loop. Assume year is an int and initialized earlier in the program.

```
for (i = year; i != 40; i = i + 2) {
    printf("At %d, you aren't old yet!\n", i);
}
```

3

Never increment

* risk infinite loop * 2.

39 + 2 = 41

!= 40 ...

-6

5. Write a program that accepts an **int** as input from **stdin**, and converts that value in seconds to hours, minutes and seconds, then writes the result to **stdout** in the format <hours>:<minutes>:<seconds>, with a new-line character after the that output, then terminates. Be sure to include any header files needed. (BONUS: Five points if the minutes and seconds values will display a leading zero (0), if they would otherwise print as single digits.)

```
#include<stdio.h>

int main(void)
{
    int userInput;
    int hours;
    int minutes;
    int seconds;
    printf("Give me seconds: ");
    scanf("%d", &userInput);
    printf("\n");
    seconds = (userInput) %60;
    minutes = (seconds %60);
    hours = (minutes %60);
    printf("%d:%d:%d\n", &hours, &minutes, &seconds);

    return 0;
}
```

-4

6. Given the following **typedef** of a **struct** and declaration of the array **roster** of that type, write a function that compares two instances and returns -1 if the first comes alphabetically before the second, 0 if they are the same and 1 if the second comes alphabetically before the first. Compare the **l_name** fields first. Only if the last names are the same compare the **f_name** fields. Only if the last and first names are the same compare the **mid_init** fields. Additionally, provide a single statement that is a call to the **qsort** function that would invoke the comparison just written and sort the array **roster**. You may assume all instances include an actual middle initial, and that all of the members of the array **roster** are to be sorted. Note that `mid_init` is not stored as a string.

```
typedef struct empNameIDRecord {
    char l_name[16];
    char f_name[15];
    char mid_init;
    char date_of_birth[8];
    char employee_ID[8];
} empNameIDRecord;

empNameIDRecord roster[1000];
```

```
#include <string.h>

for (int i=0, i< 1,000; ++i) {
    if (strcmp (roster[i].l_name, roster[i+1].l_name) > 0) {
        return -1;
    } else if( strcmp(roster[i].l_name, roster[i+1].l_name) < 0) {
        return 1;
    } else {
        return 0;
        for(int j=0, j< 1,000; ++j) {
            if(strcmp(roster[j].f_name, roster[j+1].f_name) >=0) {
                strcmp(roster[i].mid_init, roster[i+1].mid_init)=0;
            }
        } //end of else
    } //end of for
qsort ( roster, 1,000, sizeof(char), compare );
```

-3

7. Given the following global declarations, write a function named **edif** that appends a **wnode** to the end (tail) of the doubly linked list with head-node pointer passed to the function as a parameter, unless the string in the node passed to it starts with an asterisk ('*'), in which case the node is inserted at the head of the list, but the last node in the list is deleted and returned to the free node list pointed to by **freeList**. Note that the head-node may be any node in the pool of nodes, and so may not be **nodePool[0]**. A template for the function is provided.

```
Typedef struct wnode {
    char aword[16];
    wnode *last;
    wnode *next;
} wnode;
wnode nodePool[512];
wnode *freeList;
void edif(wnode *Head, wnode *new);
```

```
void edif(wnode *new, wnode *Head) {
    wnode  *here = head->next;
    while ((here!=head) && (strcmp(new->aword, here->aword)>=0))
    {
        here= here->next;
    }
    new->next = here;
    new->last= here->last;
    new->last->next = new;
    here->last = new;
    return;
}
```

-2

This page is provided as scratch paper and to complete long answers. Clearly indicate the problem being answered if answering or completing an answer here.