McGill

Fall 2018
Practice Final

VERSION # Practice

Introduction to Software Systems

COMP 206 Sec 001/002

EXAMINER: Dr. David Meger

STUDENT NAME:		**McGill ID:**									

INSTRUCTIONS:

EXAM:	CLOSED BOOK ☒ OPEN BOOK ☐
	SINGLE-SIDED ☐ PRINTED ON BOTH SIDES OF THE PAGE ☒
	MULTIPLE CHOICE ☒ NOTE: The Examination Security Monitor Program detects pairs of students with unusually similar answer patterns on multiple-choice exams. Data generated by this program can be used as admissible evidence, either to initiate or corroborate an investigation or a charge of cheating under Section 16 of the Code of Student Conduct and Disciplinary Procedures. ANSWER IN BOOKLET ☐ EXTRA BOOKLETS PERMITTED: YES ☐ NO ☐ ANSWER ON EXAM ☒ (FOR THE LONG ANSWER QUESTIONS)
	SHOULD THE EXAM BE: RETURNED ☒ KEPT BY STUDENT ☐
CRIB SHEETS:	NOT PERMITTED ☐ PERMITTED ☒ Specifications: One 8 1/2X11 double-sided sheet. Can be handwritten or printed.
DICTIONARIES:	TRANSLATION ONLY ☒ REGULAR ☐ NONE ☐
CALCULATORS:	NOT PERMITTED ☒ PERMITTED (Non-Programmable) ☐
ANY SPECIAL INSTRUCTIONS: e.g. molecular models	This is a sample exam cover page. The real one will look similar in this spot we will mention the number and type of questions. Note that this exam is mainly to give you a flavor for the format. Do not assume that the coverage of material in this example is similar to the real exam. The real final covers the whole course in about the same ratio as we spent time on each concept in lecture. We have purposely given more practice with the 2nd half of the course here because you already had a midterm to practice the first half.

Section 1: Knowledge and Definitions

Answer questions 1-5 by choosing the software system component from the following list that most directly accomplishes the described functionality. Each letter MUST be used only once.

a) gcc
b) bash
c) make
d) tcp
e) sockets

1. Performs networking operations related to congestion control and reliable transmission.
2. Provides an interface between user code and network functionality.
3. Automates compilation of only the minimal required changes.
4. Processes code files to produce programs.
5. Controls the input and output of commands.

6. Which of the following solutions to the Dining Philosopher's Problem:

a) Ordering the chopsticks and picking up the lowest first.
b) Allowing communication at the dinner table.
c) Asking the waiter to give an order.
d) All of the above.
e) None of the above, this problem is unsolvable.

7. Which of the following fields are contained within a TCP (transmission control protocol) header:

a) Sequence number
b) Port
c) Checksum
d) Header length
e) All of the above

Section 2: Find the Errors

8. Find the errors in the following C program. Select the option that best describes the lines that contain errors.

```
 1  int main(){
 2
 3    int array[10];
 4    int *p = array;
 5    int *dynamic_array = malloc(array);
 6    p[0] = 0;
 7    array[10] = 10;
 8    *(p++) = array[0];
 9    array++;
10    return 0;
11  }
```

a) 4, 5, 8
b) 5, 7, 9
c) 4, 5, 7
d) 4, 8, 9
e) There are fewer than 3 errors in this program.

9. Find the errors in the following C program. Select the option that best describes the lines that contain errors.

```
 1      #include <stdio.h>
 2      public static void main(String[] args)
 3      {
 4              long int i = (long int)printf;
 5              printf( "i holds %d.\n", int );
 6
 7              float f;
 8              scanf("%f", f);
 9              printf("user entered: $%f", f );
10
11              printf("f addr %ld.", (long int)&f
);
12      }
```

a) 4, 5, 8
b) 2, 5, 8
c) 2, 4, 5
d) 2, 4, 9
e) There are fewer than 3 errors in this program

Section 3: Describe Program Outcome

10. What does this C program display on the terminal:

```
#include <stdio.h>

void main(){

     int i = 2;
     int j = 3;

     int *p = &i;
     int *q = &j;

     int **pp = &p;
     int **qq = &q;

     q = *pp;
     *p = **qq;
     i = j;

     printf( "%d %d %d %d\n", *p, *q, **pp, **qq );
}
```

a) 2 2 2 2
b) 2 3 2 3
c) 3 2 3 2
d) 3 3 3 3
e) Segmentation fault

Section 4: Larger Systems and Tools

11. You have 3 C files that all depend on a single header, **common.h**. Two files **questions.c** and **answers.c** need to be compiled together into a shared library, and the file **main_exam.c** then links with this library to produce the executable. Consider the partial Makefile:

```
libExam.so: questions.c answers.c common.h
	gcc -shared -fPIC questions.c answers.c -o libExam.so

YOU MUST COMPLETE THIS MISSING LINE
	gcc main_exam.c -lExam -L. -o main_exam
```

What is the correct way to complete the missing line:

a) main_exam: libExam.so main_exam.c common.h
b) main_exam: questions.c main_exam.c libExam.so
c) main_exam: main_exam.c common.h
d) questions.c: libExam.so main_exam.c common.h
e) main_exam.o: questions.c answers.c main_exam.c common.h

12. We are compiling several C programs that all depend on the C source file "sort_helpers.c". Which of the following approaches allow us to re-use compilation effort when building all of the programs:

a) First create an a.out file from sort_helpers.c
b) #include "source_helpers.c" in each program
c) First create a dynamic library from sort_helpers.c
d) Add the –I argument to the gcc command line of each program
e) None of the above

Section 5: Analyzing Software Systems

The following C function is designed to test for patterns in the bits of its argument ***bit_field*** in its input and will be used for the following three questions.

```
1  void findPattern( unsigned char bit_field )
2  {
3          unsigned char test_mask = ???
4          int i;
5
6          for( i=0; i<???; i++ )
7                  if( ( bit_field>>i & test_mask) > 2 ){
8                                  printf( “Pattern detected!\n” );
9                                  return;
10                 }
11         printf( “No pattern occurred.\n” );
12 }
```

13. What value of the **test_mask** variable, set on line **3**, would allow checking for “1”s in any consecutive positions? Consecutive positions means at least two “1”s in a row without any “0”s in between. Examples of inputs that should report “detected” are 11000000, 11111111 and 10101011. Examples that should report “No pattern” are 10101010, 00000001, 00010001.

a) 1
b) 3
c) 5
d) 2
e) It cannot be done correctly without changing some other part of the code.

14. What value of the **test_mask** variable, set on line **3**, would allow checking for “0”s in consecutive positions? Here 11111100 is one example that should report “detected” and 10101010 is one example that should report “no pattern”.

a) 3
b) 5
c) 254
d) 252
e) It cannot be done correctly without changing some other part of the code.

15. In order that we consider the entirety **bit_field** but nothing further, what would be the best bound for the value of the variable **i** within the for loop on line **6**?

a) 4
b) 8 * sizeof(unsigned char)
c) 1
d) sizeof(unsigned char)
e) sizeof(unsigned char*)

Use the following declaration of a C struct that will be used to represent a linked list to answer the next 3 questions:

```
typedef struct s_list_node{
        int value;
        struct s_list_node *next;
} LIST_NODE;
```

16. Which of these are correct C statements:

a) s_list_node struct s;
b) struct LIST_NODE s;
c) LIST_NODE s;
d) struct *s_list_node s;
e) None of the above are correct

17. Consider the following variable declaration:

LIST_NODE *queues[5];

Which is a reasonable explanation for how the variable **queues** might be used?

a) As a single linked list of length 5
b) As an arbitrary length array of linked lists of length 1
c) As an array of length 5 of linked lists of length 5
d) As an array of length 1 of linked lists of length 5
e) As an array of length 5 of linked lists of arbitrary length

18. Suppose we write a function **switch_lists()** with the specification:

void switch_lists(int target, LIST_NODE *a, LIST_NODE *b);

This function is meant to find a list entry in **a**. If it exists, it should be removed from **a** and added at the end of **b.** Which of the following is true of the memory operations that would be required within this function:

a) No memory operations would be required
b) A new LIST_NODE would need to be created on the stack
c) A new LIST_NODE would need to be created on the heap
d) A LIST_NODE should be free'd from the heap
e) New heap memory is needed only if the **target** value is found

Section 6: Complete the Lines

This program is meant to use function pointers to apply the correct printing routine depending on the type of data passed in as argument **input.** Think about how to complete the missing sections for questions 27-29:

```
#include <stdio.h>

void printInt( void *data )   { /* TO BE COMPLETED IN Q27 */ }
void printChar( void *data ){ /* TO BE COMPLETED IN Q27 */ }

void main (){
   int i = 5;
   char c = ‘a’;
   void *untyped_data_array[2];
   char *type_name_array[2] = {  “int”, “char” };
   void (*fn_ptr_array[2])( void* );

   untyped_data_array[0] = /* TO BE COMPLETED IN Q28 */;
   untyped_data_array[1] = /* TO BE COMPLETED IN Q28 */;
   fn_ptr_array[0] =           /* TO BE COMPLETED IN Q29 */;
   fn_ptr_array[1] =           /* TO BE COMPLETED IN Q29 */ ;

   for( int entry=0; entry<2; entry++ ){
                    if( strcmp( type_name_array[entry], “int” ) == 0 )
                            fn_ptr_array[0](untyped_data_array[entry]);
                    else
                            fn_ptr_array[1](untyped_data_array[entry]);
    }
}
```

19. Which of the following are reasonable completions for the commented portions marked Q27?
    a) printf(“%d\n”, data); AND printf(“%c\n”, data);
    b) printf(“%d\n”, *data); AND printf(“%c\n”, *data);
    c) printf(“%d\n”, (int*)data); AND printf(“%c\n”, (char*)data);
    d) printf(“%d\n”, *(int*)data); AND printf(“%c\n”, *(char*)data);
    e) printf(“%d\n”, (int)*data); AND printf(“%c\n”, (char)*data);

20. Which of the following are reasonable completions for the commented portions marked Q28?
    a) i AND c
    b) *i AND *c
    c) (void*)&i AND (void*)&c
    d) (void*)c AND (void*)c
    e) (int*)i AND (char*)c

21. Which of the following are reasonable completions for the commented portions marked Q29?
    a) printInt(i) AND printChar(c)
    b) printInt and printChar
    c) void printInt and void printChar
    d) printInt() and printChar()
    e) None of these are appropriate

Long Answer Question:

22. One way to write the main function in a C program is: int main(int argc, char** argv)
As you know, argc gives the number of arguments and argv is an array holding one argument per array element, each a properly formed C string.

For this question, imagine another language, ***C-like***, which is identical to C in every way except that only the raw command-line (exactly the text the user types) as a single C string is available as the argument to the "main-like" function. Your job is to write a function parseArgs() that would recover the typical argc and argv (think carefully about what you know about each of these variables: you must achieve all properties guaranteed by C). The following C-like code gives an example of using ***C-like*** with your function:

```
#include <stdio.h>
#define MAX_ARGS 1000
void parseArgs( char * raw_command_line, int *argc, char *argv[MAX_ARGS] );

void main_like( char *raw_command_line ){    // Pay attention to this purposely weird line
    int argc;
    char* argv[MAX_ARGS];
    parseArgs(raw_command_line, &argc, argv ); // This line is the call to your function

    printf( "raw_command_line was %s\n", raw_command_line );
    for( int i=0; i<argc; i++ ){
      printf( "Argument %d is %s\n", i, argv[i] );
    }
}
```

When you complete your parseArgs function, the above program should run as shown in the following example, assuming it is compiled with gcc-like to produce a.out:

```
$ ./a.out Hello world
raw_command_line was ./a.out Hello world
Argument 0 is ./a.out
Argument 1 is Hello
Argument 3 is World
```

MARKING NOTES AND ALLOWED ASSUMPTIONS:

1) You cannot use libraries
2) Your code should be readable, clear and follow your design in an obvious way
3) You should attempt to solve the problem in the fewest number of operations and lines of code you can manage, while maintaining clarity
4) You can assume there is always exactly one space between each argument
5) You can assume there is always at least one argument (the program name)
6) You can assume there are never more than MAX_ARGS arguments
7) You can assume the raw_command_line string is properly null-terminated

Describe your solution design, ideally as a diagram or a few short phrases (6 marks):

Then, complete the contents of the parseArgs function, here (10 marks):

```
void parseArgs( char * raw_command_line, int *argc, char *argv[MAX_ARGS] ){




























}
```