

Format of date for midterm indicates the exam (i.e., the two different exam have different date formats).

SENG 265
Summer 2019 (Sections A01/02: CRN 30735/30736)
Midterm Exam: Tuesday, 18 June 2019

Name: _____ (please print clearly!)

UVic ID number: _____

Signature: _____

Exam duration: 50 minutes

Instructor: Michael Zastre

Section	Value	Mark
A: Multiple Choice	20	
B: Code reading/writing	20	
C: Problem solving	40	
Total	80	

Students must check the number of pages in this examination paper before beginning to write, and report any discrepancy immediately.

- **All answers are to be written on this exam paper.**
- The exam is closed book. No books or notes are permitted.
- When answering questions, please do not detach any exam pages!
- **Electronic devices are not permitted.** Cellphones must be turned off.
- Partial marks are available for the questions in sections B and C.
- There are eight (8) printed pages in this document, including this cover page.
- We strongly recommend you read the entire exam through from beginning to end before starting on your answers.
- **Please have your UVic ID card available for inspection by an exam invigilator.**

Section A (20 marks): For each question in this section, place an X beside all answers that apply. Each question is worth two (2) marks. *Partial marks are not given for incomplete answers.*

Question 1: After we use *git add* on a file in our local repository:

- ☐ Every change to that file is automatically committed to the local repository.
- ☐ Every additional file added to the same directory is automatically pushed to the remote repository by *git*.
- ☐ All other developers who have cloned our remote repo must perform the *git add* command to add that file to their local copy.
- ☐ A new file will be stored in that directory which contains the concatenation of all changes made to the file.
- ☒ None of the above.

Question 2: *git clone ssh://ironman@avengerscorp.us.com/suits/extremis*

- ☐ Can only be performed once for the *extremis* project.
- ☐ Will return an error if some other user has already performed a *git push* on the *extremis* repository.
- ☒ Creates possibly many subdirectories in a new local repository.
- ☒ Includes the full commit history of the project stored at that remote repository.
- ☐ None of the above.

Question 3: By stating that *git* is a version-control system, we are saying:

- ☒ It tracks changes to files and directories over time.
- ☒ It ensures that versions of files associated with previous commits can be obtained using the proper *git* operations.
- ☒ It permits concurrent read access of remote repositories for users who have read rights to those repositories.
- ☐ It enables a programmer to “lock out” others from accessing a file when that programmer is making changes to the file.
- ☐ None of the above.

Question 4: The UNIX *cp* command:

- ☐ Can be used to delete text files.
- ☐ Can be used to delete executable files.
- ☐ Can be used to modify a text file (i.e., “copy-and-paste”).
- ☐ Can be used to enlarge an executable file (i.e., “create-and-provision”).
- ☒ None of the above.

Question 5: A pipe (“|”) that may be constructed using the *bash* shell:

- ☒ connects the *stdin* stream of one command with the *stdout* stream of the previous command in the pipe.
- ☐ connects the *stdout* stream of one command with the *stdout* stream of the next command in the pipe.
- ☒ may be used with Unix commands and with executables we ourselves have created using *gcc*.
- ☐ is needed in order to move the location of a file or directory.
- ☐ None of the above.

Question 6: If a file has the permissions *rw-rw-r--*, then this means:

- ☐ the owner of the file has more access than those users in the file’s group or even of all other users.
- ☐ anyone is able to execute the file.
- ☐ anyone is able to write the file.
- ☒ anyone is able to read the file.
- ☐ None of the above.

Question 7: Assuming we have declarations “char commands[50]; char *cp;” in our C program, then which expressions have the same value type on either side of the assignment operator?

- ☒ cp = &commands[25];
- ☒ *commands = commands[2];
- ☐ *(&commands[10]) + 1 = cp;
- ☐ *cp = commands + 10;
- ☐ *(cp + 1) = cp;

Question 8: Consider the following loop in C where “%” is the modulo operator (the remainder operator, i.e., 7 % 2 equals 1, 8 % 2 equals 0, etc.)

```
int k;
for (k = 10; k >= 1; k--) {
    printf("%d", k);
    if (k % 3 == 1) {
        printf(" , ");
    } else if (k % 2 == 1) {
        printf(" ^ ");
    } else if (k % 1 == 0) {
        printf(" # ");
    } else {
        printf(" ");
    }
}
printf("\n");
```

Assume this code is within the *main()* function of a program and the proper headers are included. The output seen at the terminal’s console when this program is compiled and executed is:

- ☐ Nothing is output as the resulting program cannot compile given the syntax errors.
- ☐ 10 9 8 ^ 7 6 5 # 4 3 2 , 1
- ☐ 10 ^ 9 # 8 , 7 ^ 6 # 5 , 4 ^ 3 # 2 , 1 ^
- ☒ 10 , 9 ^ 8 # 7 , 6 # 5 ^ 4 , 3 ^ 2 # 1 ,
- ☐ None of the above.

Question 9: The *for*-loop in C:

- ☒ Can contain other *for*-loops within it as nested loops.
- ☐ Is a bottom-tested loop just like the *do-while* statement.
- ☒ May contain statements that dereference pointers.
- ☒ May contain function calls.
- ☐ None of the above.

Question 10: The C string function *strncat(char *dest, char *src, int len)*:

- ☒ Concatenates at most *len* characters from *src* to *dest*.
- ☐ Must be used when working with *char* arrays containing the names of cats.
- ☒ Assumes memory already exists for the destination string.
- ☐ Will terminate the program if *src* string is longer than what is indicated by the *len* parameter.
- ☐ None of the above.

Section B (20 marks): One question

Question 11: Given three arrays *a*, *b* and *c* in program below, use a loop to let the array *d* become the product of *a* and *b* plus *c*. That is, each element of *d* is the sum of corresponding elements of *a* and *b*, to which is multiplied the corresponding element of *c*. After computing the values for array *d*, print the elements of array *d*.

You must use a *while* loop for the first part of this question, and a *for* loop for the second part of the question. Write your code in the boxes provided.

```
#include <stdio.h>
```

```
int main() {  
    int a[] = {2, 6, 4, 7, 3, 9};  
    int b[] = {5, 1, 3, 2, 4, 0};  
    int c[] = {1, -1, 2, -2, 3, -3};  
    int d[6];
```

```
    /* Write code below that will let array d be the product of  
     * arrays a and b and summed to array c. While-loop only.  
     */
```

```
    /* One possible solution */  
    int i = 0;  
    while (i < 6) {  
        d[i] = (a[i] + b[i]) * c[i];  
    }
```

```
    /* Write code needed to print the elements of d on one  
     * line, with commas provided between numbers. For-loop only.  
     */
```

```
    /* Another possible solution, re-using declaration of i from above */  
    char *sep = "";  
    for (i = 0; i < 6; i++) {  
        printf("%s%d", d[i]);  
        sep = ", ";  
    }  
    printf("\n");
```

```
}
```

Here is the output expected:

11, 5, 14, 12, 15, -3

Section C (40 marks): One question

Question 12

A *pangram* is a sentence that contains each letter of the English alphabet at least once.

Here are some pangram strings (some famous, some not):

- "The quick brown fox jumps over the lazy dog."
- "Two driven jocks help fax my big quiz."
- "When zombies arrive, quickly fax judge Pat."

The following are *not* pangrams:

- "Zippee-dee doo-dah, zippee-dee day."
- "I am a very stable genius."
- "Not a pangram. Not even close."

Write a C function `int is_pangram(char *sentence)` that returns 1 if the sentence is a pangram, otherwise returns 0.

Your solution:

- may assume that `sentence` is never null;
- may use the `islower()` library function that converts a single character into its lower-case equivalent;
- must use only `strlen` from all of the C string functions (i.e., may not use `strstr`, `strncpy`, `strncat`, etc.), although your solution is not required to use `strlen`.
- must work correctly for sentences other than the examples shown above.

Hint: You can exploit the way C treats characters. For example, the character 'a' is actually the integer 97, the character 'b' is 98, the character 'c' is 99, (... etc. ...), the character 'y' is 121, and the character 'z' is 122. This means that after this line is executed:

```
int val = 'm';
```

the integer variable `val` is declared, and 109 is stored in it.

Some marks will be given for the quality of your answer.

(Use the next blank page of this exam for your solution.)

I was looking for the following five elements in your solution:

- * function header (properly written)
- * examining all characters in the sentence
- * determining whether an alphabetic character appears
- * determining whether or not all alphabetic characters appear
- * returning the results

Given the confusion around `islower()` vs. `tolower()`, I didn't focus too closely on treatment of upper- vs. lower-case letters.

Some correct solutions used nested loops, yet these solutions are not as good as those using several single loops.

A solution I considered to be "A" was given 40 mark, "B" was 32, "C" was 24, and anything below 20 is an "F"; I also when up a little or down a little as I felt more or less of a mark was warranted.

One possible solution:

```
int is_pangram(char *sentence)
{
    int freqs[26] = {0,};
    int i;
    char *c, d;

    for (c = sentence; *c; c++) {
        d = tolower(*c);

        if (d < 'a' || d > 'z') {
            continue;
        }

        freqs[(int)(d - 'a')]++;
    }

    for (i = 0; i < 26; i++) {
        if (freqs[i] == 0) {
            return (0);
        }
    }

    return 1;
}
```

(end of the exam)