University of Victoria

University of Victoria
Final Examination
December 2016

Course Name & No.	SENG 265 Software Development Methods
Section(s)	A01/A02
CRN	12857 (A01) 12858 (A02)
Instructor	Michael Zastre
Duration	Three (3) hours

Name:
Student Number: V00

This exam has a total of 16 pages including this cover page.

Students must count the number of pages and report any discrepancy immediately to the Invigilator.

This exam is to be answered on the paper.

No materials or aids of any kind are permitted. This is a closed-book exam.

All cellphones must be switched off. You must obtain permission from an invigilator to temporarily leave the examination room.

The total number of marks in this exam is 180.

Section A (60 marks): 20 questions

For each question in this section, place an X beside all answers that apply. Each question is worth three (3) marks. *Partial marks are not given for incomplete answers.*

Question 1: The boolean "false" value in C:

___ is represented by the C keyword *false* in a boolean expression.

___ is returned by *malloc()* when this function succeeds in obtaining heap memory.

___ is used by C to determine the bounds of an integer array.

___ may have its C equivalent stored in an integer variable.

___ None of the above.

Question 2: A *for*-loop in C:

___ may be used to traverse the nodes of a linked-list implementation discussed in this course.

___ uses a very different syntax compared to Python's *for* statement.

___ is limited to one per function.

___ may refer to two variables in its loop-condition expression.

___ None of the above.

Question 3: When using *malloc(), calloc()* or *realloc()*:

___ we are always guaranteed to obtain from the stack the amount of memory we request.

___ we are always guaranteed to obtain from the heap the amount of memory we request.

___ we must always specify the amount of memory requested, possibly in an expression involving the *sizeof()* macro.

___ the memory allocated in one module cannot ever be accessed by code in another module.

___ None of the above.

Question 4: Every C source-code file:

___ must contain the *main()* function as otherwise the file cannot be part of a C program.

___ may *import* a header file.

___ may include function prototypes for all other functions in all other modules.

___ can have some variables visible only within the source-code file and not outside of the source-code file.

___ None of the above.

Question 5: The C string function *strtok(somestring, someseparatorchars)*:

___ must be followed at some point with *strtok(NULL, someseparatorchars)* in order to continue tokenizing the original string.

___ can only be used when we *#include <strtok.h>*.

___ automatically allocates memory for all tokens extracted from the string.

___ returns the value -1 when there are no more tokens in the original string.

___ None of the above.

Question 6: A Python 3 variable3:

___ may be assigned a type in the same way variables are declared in C.

___ can be assigned values of different types at different points within the same function.

___ may be assigned the value of a list.

___ may be assigned the value of a tuple.

___ None of the above.

Question 7: Assuming *list1 = ["abc", "d", "ef", "ghij"]*, the following list comprehension:

*[a * 2 for a in list1 if len(a) < 3]*

is equal to:

___ *["abcabc", "dd", "efef", "ghijghij"]*

___ *["d", "ef"]*

___ *["ddd", "efef"]*

___ *["dd", "efef"]*

___ None of the above.

Question 8: When defining a Python 3 class:

___ methods have *self* listed as the first parameter.

___ we are able to write constructors as needed.

___ a class variable's name may match an instance variable's name.

___ may be contained within a file having a name different than the class's name itself.

___ None of the above.

Question 9: When using Python 3 lists:

___ a negative lookup such as *somelist[-3]* refers to list values positioned relative to the end of the list.

___ a copy of list contents is made as the result of a statement such as *some_other_list = somelist*

___ we can refer to the list from the second element to the end using *somelist[1:]*

___ Python 3 ensures all values within the same list have the same type.

___ None of the above.

Question 10: A Python 3 function:

___ may be passed as a function parameter.

___ may or may not have parameters.

___ are forbidden from containing underscores (i.e., "_") as part of the function name.

___ may be returned as a result of some function's execution.

___ None of the above.

Question 11. Considering the Python 3 expression *(True or (y < 10) and bar)*:

___ it generates a syntax error as *bar* always represents a function and must be called with parentheses.

___ is equivalent to the Python 3 expression *((y < 10) and bar)*.

___ its value may be *False* or the value of *bar*.

___ its value may be passed as a parameter to some function.

___ None of the above.

Question 12: Consider the following Python 3 dictionary:

usprez = {42:'clinton', 41:'ghwbush', 43:'gwbush', 44:'obama'}

Place an *X* beside each line below that ***does not result in an error*** when evaluated by a Python 3 interpreter.

___ *print(usprez[42])*

___ *len(usprez[41])*

___ *usprez[40] = "rreagan"*

___ *output = "President " + usprez[45]*

Question 13: *git clone ssh://jtrudeau@git.seng.uvic.ca/seng265/jtrudeau*:

___ Can only be performed once for the *jtrudeau* project.

___ Will return an error if some other user has already performed a *git push* on the *jtrudeau* repository.

___ Creates possibly many subdirectories.

___ Includes the full commit history of the project stored at that remote repository.

___ None of the above.

Question 14: The git *push* command:

___ automatically commits changes to files in the current local repository.

___ is the same as the *clone* command.

___ normally results in the most recent commit (amongst others) being transferred to some remote repository.

___ reverses the effect of the most recent *pull* command.

___ None of the above.

Question 15: The UNIX "diff" command:

___ is part of the math library which provides a differentiation utility for symbolic math.

___ automatically tests files for completion.

___ reports the ways in which two files do not have the same content.

___ may only ever be used as part of a piped Unix command.

___ None of the above.

Question 16: Consider the following regular expression:

```
"^\w(p|pp)\w+$"
```

Place an X beside each string below that is found by the regex (and note that the quotation marks are only used to denote the start and end of the string). Assume the Python 3 *re* module is being used, that the *search* function is provided only two parameters (i.e., the regex plus the string to be checked against the regex), and that raw strings are always used for a regular expression.

___ `"apple"`

___ `"piano"`

___ `"a Porsche"`

___ `"apron"`

___ `"pop"`

Question 17: Consider the following string:

```
"604-555-1515"
```

Place an X beside each regular expression below for which a match is found with the string. (The same assumptions about Python 3 given in question 15 apply here.)

___ `"\d{3}-\d{3}"`

___ `"^\d{3}-\d{3}$"`

___ `"(4|5|6).(4|5|6)"`

___ `"^.*[1-5].*$"`

___ `"^.*[1-5]-[1-5]$"`

Question 18: Place an X beside each regular expression that matches the string `"in"` and the string `"into"` but not `"bin"` or `"cinch"`. (The same assumptions about Python 3 given in question 15 apply here.)

___ `"\bin\b"`

___ `"\bin"`

___ `"in\b"`

___ `"^ch\b"`

___ `"\bin$"`

Question 19: The *unittest* framework for Python 3:

___ can be used to associate test cases with class files we write in Python 3.

___ requires special scripting support from Unix to run the tests.

___ has methods for setting up and tearing down data structures needed for tests.

___ automatically executes the tests we have written.

___ none of the above.

Question 20: *Black-box testing* as described in this course is an approach to constructing tests:

___ where code is ignored.

___ where often it is only the specification document used to design test cases.

___ where code walkthroughs occur with the programmer sitting behind a black curtain.

___ where the mathematics of "black-body radiation" must be used to design the tests.

___ None of the above.

Section B: Python 3 (Total marks: 50)

Question 21 (15 marks): Consider the following Python 3 code, named *mystery.py* (line numbers provided for your reference):

```
 1 #!/usr/bin/env python3
 2
 3 import sys
 4
 5 def main():
 6     aaa = []
 7     for bbb in sys.stdin:
 8         ccc = bbb.split()
 9         for yyy in ccc:
10             aaa.append(yyy)
11
12     ddd = [len(zzz) for zzz in aaa]
13     ddd.sort()
14
15     eee = ddd[0]
16     fff = 1
17     for ggg in ddd[1:]:
18         if eee != ggg:
19             print (fff, "-->", eee)
20             eee = ggg
21             fff = 1
22         else:
23             fff = fff + 1
24
25     print (fff, "-->", eee)
26
27 if __name__ == "__main__":
28     main()
```

and the following input in a file named *seuss.txt*:

```
the sun did not shine
it was too wet to play
so we sat in the house
all that cold cold wet day.
```

Write on the lines below the output that results when the input is redirected into the Python program (i.e., *cat seuss.txt | ./mystery.py*). Assume the file permissions for *mystery.py* to execute as a command are correctly set and the Python 3 interpreter is is available given the bang path. *You may use the next page for your rough work.*

Question 22: (35 marks)

Write a Python 3 function to determine if one string is an **anagram** of another. Your function must be named *is_anagram()* and will take two strings as parameters. The function must return *True* if the strings are anagrams of each other, otherwise it must return *False*.

An ***anagram*** is a phrase that uses all the letters—and exactly those letters—of some other phrase. Here are examples of pairs of strings that are anagrams of each other:

- Zastre: *Ersatz*
- Dormitory: *Dirty Room*
- Computer Science: *CPU Secret Income*
- The check is in the mail!: *Claim "Heck, I sent it (heh)"*
- I sat there with Sally: *Aha! Lithely twisters!*

Notice that the number of spaces and use of punctuation in an anagram can differ greatly from the original phrase. *We ignore spaces, punctuation and digits when checking if two phrases are anagrams of each other.*

You may wish to make use of the following facts

- Python strings have a *lower()* method (i.e., *"ABCD".lower()* returns *"abcd"*).
- Individual characters can be compared using: <, >, <=, >=, ==. Alphabetic characters are in the range of values from 'A' to 'Z' and from 'a' to 'z'.

(The next blank page of this exam may be used for your answer.)

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

Section C: C programming (Total marks: 70)

Question 23 [20 marks]

Write a C function *int is_palindrome(char *)* that accepts a C string as input and returns 1 if the string is a palindrome, otherwise it returns 0. A *palindrome* is a string spelling the "same" forwards and backwards.

Here are some examples of palindromes:

- *radar*
- *abaaba*
- *anna*
- *hanah*
- *was it a rat i saw*
- *yo, banana boy*

True palindromes usually include all forms of punctuation and a mix of upper- and lower-case characters. However, for this function you may assume the only punctuation is either a single space between words or a comma. You may also assume all characters are in lower case.

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

(The next blank page of this exam may be used for your answer.)

Question 24: (50 marks)

Write a C function named *remove_dups(array1)* that accepts an array of positive integers *array1* as a parameter and returns a new array that contains only the unique values in *array1*. The original *array1* must not be modified. The value *-1* is used to indicate the last value in the array. **Memory for the result array must be dynamically allocated.**

The values in the new array should be ordered in the same order they originally appear in the input array. For example, if *array1* stores the elements {10, 10, 9, 4, 10, 4, 9, 17, -1}, then *remove_dups(array1)* should return a new array with elements {10, 9, 4, 17, -1}.

The following table shows some example calls to your function and their expected results:

Array	*Returned Value*
int *a1 = {5, 2, 5, 3, 2, 5, -1};	remove_dups(a1) returns {5, 2, 3, -1}
int *a2 = {8, 2, 12, 12, 2, 8, -1};	remove_dups(a2) returns {8, 2, 12, -1}
int *a3 = {4, 17, 100, 32, 3, -1};	remove_dups(a3) returns {4, 17, 100, 32, 3, -1}
int *a4 = {11, 55, 5, 5, 902, 55, 5, 43, -1};	remove_dups(a4) returns {11, 55, 5, 902, 43, -1}
int *a5 = {1, 2, 3, 4, 5, -1};	remove_dups(a5) returns {1, 2, 3, 4, 5, -1}
int *a6 = {5, 5, 5, 5, 5, 5, -1};	remove_dups(a6) returns {5, -1}
int *a7 = {-1};	remove_dups(a7) returns {-1}

Do not modify the contents of the array passed to your function as a parameter.

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

(The next blank page of this exam may be used for your answer.)

```
int *b = (int*) malloc(sizeof(int));

int *remove_dups(int* array) {
    int *new_array = (int*) malloc(sizeof(int));
    int size = 1; i,j;
    for (i=0; i<size; i++) {
        for(j=i; array[j] != -1; j++) {
            if
```

END OF EXAM

This page is for the sole use of examination evaluators.

Section A	/60
Section B	/50
Section C	/70
Total	**/180**