University
of Victoria

University of Victoria
Final Examination
December 2012

Course Name & No.	SENG 265 Software Development Methods
Section(s)	A01 and A02
CRN	10563, 12769
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.

Ensure all cellphones are 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 (40 marks): 20 questions

For each question in this section, **clearly circle all answers** that apply. All questions have equal weight. Partial marks are not given for incomplete answers.

Question 1: Dynamic memory in C:

a. Must only be used for arrays.
b. Is unavailable for use with program- and file-scope variables.
c. Involves the use of "malloc()" and "free()".
d. Involves the use of "new()" and "free()".
e. None of the above.

Question 2: In the C language, "struct"s:

a. Are aggregate data types where each field may have a different type.
b. Are aggregate data types where each field must have the same type.
c. May contain fields that are pointers.
d. Associate functions with a data type just like "class" does in an OO language (Java or C#).
e. None of the above.

Question 3: A function pointer in C:

a. Is a function that must take pointers as arguments.
b. Is a function that must return a pointer.
c. Is a function that must use pointers within the body of the function.
d. Is a function that must use dynamic memory.
e. None of the above.

Question 4: Any C source-code file:

a. Must contain the "`main()`" function as otherwise the file cannot be part of a C program.
b. Must `#include` a header file.
c. Must include function prototypes for all other functions in all other modules.
d. Can have variables that are only visible within the file and not outside of it.
e. None of the above.

Question 5: The C string function "`strncpy(foo, bar, baz)`":

a. Takes the string in "`bar`" and copies it to "`foo`".
b. Takes the string in "`baz`" and copies it to "`bar`".
c. Automatically allocates memory for the new string.
d. Is safer to use than "`strcpy`".
e. None of the above.

Question 6: The `sizeof` operator in C:

f. Accepts the name of some C type as a parameter.
g. Is often used when determining the amount of memory that must be requested from the heap.
h. Will always return the length of a string.
i. Returns a value that can be part of a larger arithmetic expression.
j. None of the above.

Question 7: A Python *list comprehension*:

f. Is the set of functions that Python permits to be called on lists.
g. May make use of a filtering expression
h. Always produces a new (possibly empty) list.
i. Converts list from a mutable form to an immutable form.
j. None of the above.

Question 8: For a Python class `Foo` to be declared a subclass of class `Bar`, then this can be expressed as:

```
a. class Foo :: Bar:
b. class Foo extends Bar:
c. class Foo interface Bar:
d. class Foo(Bar):
e. None of the above.
```

Question 9: In order create a new instance of some class `Foo` (which is a subclass of `Bar`, the following Python statement may be used:

```
a. someVar = Foo()
b. someVar = new Foo()
c. Foo somevar = Foo()
d. Foo somevar = new Foo()
e. None of the above.
```

Question 10: The Python statement "`for i in range(lower, upper): print i`"

a. Sets the value of "`upper`" to "`lower`".
b. Sets the value of "`lower`" to "`upper`".
c. Calls the function "`range`" which returns the string "`abcde`" when lower is "a" and upper is "e".
d. Will never compile because the "`for`" operator should be replaced with a "`while`" when "`range`" appears in an expression.
e. None of the above.

Question 11: The list comprehension `[elem*2 for elem in [3, 6, 2, 7]]` produces the list

a. `[3, 3, 6, 6, 2, 2, 7, 7]`
b. `["3*2", "6*2", "2*2", "7*2"]`
c. `[elem*3, elem*6, elem*2, elem*7]`
d. `[6, 12, 4, 14]`
e. None of the above.

Question 12: When defining a function in Python:

a. You must always include the `self` variable in the function signature.
b. The block of a multiline function definition must be indented.
c. You must end the function with the `return` keyword.
d. It is possible to provide default values for parameters.
e. None of the above.

Question 13: Pipes and input/output redirection are important tools available via the UNIX shell. Which of these UNIX commands properly ensures the output of `step_a.sh` is piped into `process_b.py`, and that `process_b.py`'s output is saved in `/tmp/capture.txt`? Assume both of the scripts/programs are executable. (Remember to circle all that apply.)

a. `cat ./options.txt | step_a.sh | process_b.py > /tmp/capture.txt`
b. `cat ./options.txt > step_a.sh > process_b.py > /tmp/capture.txt`
c. `cat ./options.txt | step_a.sh | process_b.py | /tmp/capture.txt <`
d. `cat ./options.txt | step_a.sh | process_b.py || /tmp/capture.txt`
e. None of the above.

Question 14: The UNIX "`ls`" command:

a. Is meant to accept input that can be redirected to its `stdin` from a command like "`cat`".
b. Can display the contents of files.
c. Can display the attributes of files.
d. Can display the size of files.
e. None of the above.

Question 15: After executing "`svn add blarg.c`", the user who issued this command:

a. Knows the file named "`blarg.c`" is now known by the repository.
b. Knows the file named "`blarg.c`" has been committed to the repository.
c. Has created a new file in their working module.
d. Has taken a step that may allow another user to access the contents of "`blarg.c`".
e. None of the above.

Question 16: The command "`svn checkout http://somesite.org/somemodule`":

a. Can only be performed once for "`somemodule`".
b. Will return an error if some other user has already checked out "`somemodule`".
c. Creates possibly many subdirectories.
d. Is the opposite of "`svn checkin http://foo/somemodule`".
e. None of the above.

Question 17: A bug is:

a. Evidence the programmer might misunderstand the meaning of their code.
b. Can always be found using "`printf`" statements in C code.
c. Best found using a methodical, scientific approach.
d. An error caused by the compiler.
e. None of the above.

Question 18: Regular expressions in Python:

a. May be used to specify sets of strings.
b. Are accessible via either `re.match` or `re.search`.
c. Can be compiled in order to reduce the time needed to perform a match.
d. Are sometimes used as part of control-flow expressions.
e. None of the above.

Question 19: In the context of this course's treatment of software-development process, a development lifecycle:

a. Has a "requirements" phase.
b. Has a "customer negotiation" phase.
c. Has a "library assessment" phase.
d. Has a "system design" phase.
e. None of the above.

Question 20: Dependencies in a makefile rule:

a. Consist of a list of commands.
b. Consist of a list of files.
c. Consist of a list of targets.
d. Consist of a list of base rules.
e. None of the above.

Section B: Python (60 marks)

Question 21 (weight 10): Consider the following Python code, named "`mystery.py`":

```
#!/usr/bin/python
import sys

def main():
    (aaa, bbb, ccc) = (1, 0, {})

    ddd = sys.stdin.readlines()

    while (ddd != []):
        eee = ddd[0]
        fff = eee.strip().split()
        for ggg in fff:
            if ggg in ccc.keys():
                ccc[ggg] = ccc[ggg] + ", " + ("%d" % aaa)
            else:
                ccc[ggg] = ("%d" % aaa)
        aaa = aaa + 1
        ddd = ddd[1:]

    hhh = ccc.keys()
    print hhh
    hhh.sort()
    print hhh

    hhh = hhh[0:5]

    for h in hhh:
        print h, ":", ccc[h]

if __name__ == "__main__":
    main()
```

and the following input, named "`in01.txt`":

```
this time is all the time
for all good
people to come for
to the aid of this their country.
```

Write on the lines below the output that results when the input is redirected into the Python program (i.e., "`cat in01.txt | ./mystery.py`").

_____

_____

_____

_____

_____

Question 22: Weight 40

Consider a playlist stored in a text file. The example below is named `265.list` and demonstrates the way such files are formatted. Note the individual tracks in the playlist correspond to the lines following the third line.

```
title#Greatest Lectures of SENG 265
genre#spoken word
length#3
Why Python is terrific!#/Volume/MP3/GLS265/lecture12.mpg#005012
Development processes: Why?#/Volume/MP3/GLS265/lecture9.mpg#004500
What I wish I learned before industry#/Volume/MP3/Guests/jason.mpg#004920
```

- Write a Python function that accepts the name of an input file containing data such as that above and returns an instance of a `Playlist` class.
- As part of your solution you must also define the `Playlist` class. Amongst other class attributes, the tracks in a `Playlist` instance must be stored in a single list attribute of that instance.
- There must be a method in `PlayList` named `totalLength()` which returns a tuple containing the number of hours, minutes and seconds needed to play all of the tracks from start to finish.
- Regular expressions must be used where appropriate.
- Some marks will be given for the quality of your solution.

Your indentation is to be clearly indicated. Some marks will be given for the quality of your answer.

Question 23: Weight 10

Python strings and tuples are referred to as *immutable*, and lists are referred to as *mutable*. What is the difference between immutable and mutable variables? Prove examples explaining your answer.

Section C: C programming (50 marks)

Question 24: Weight 15

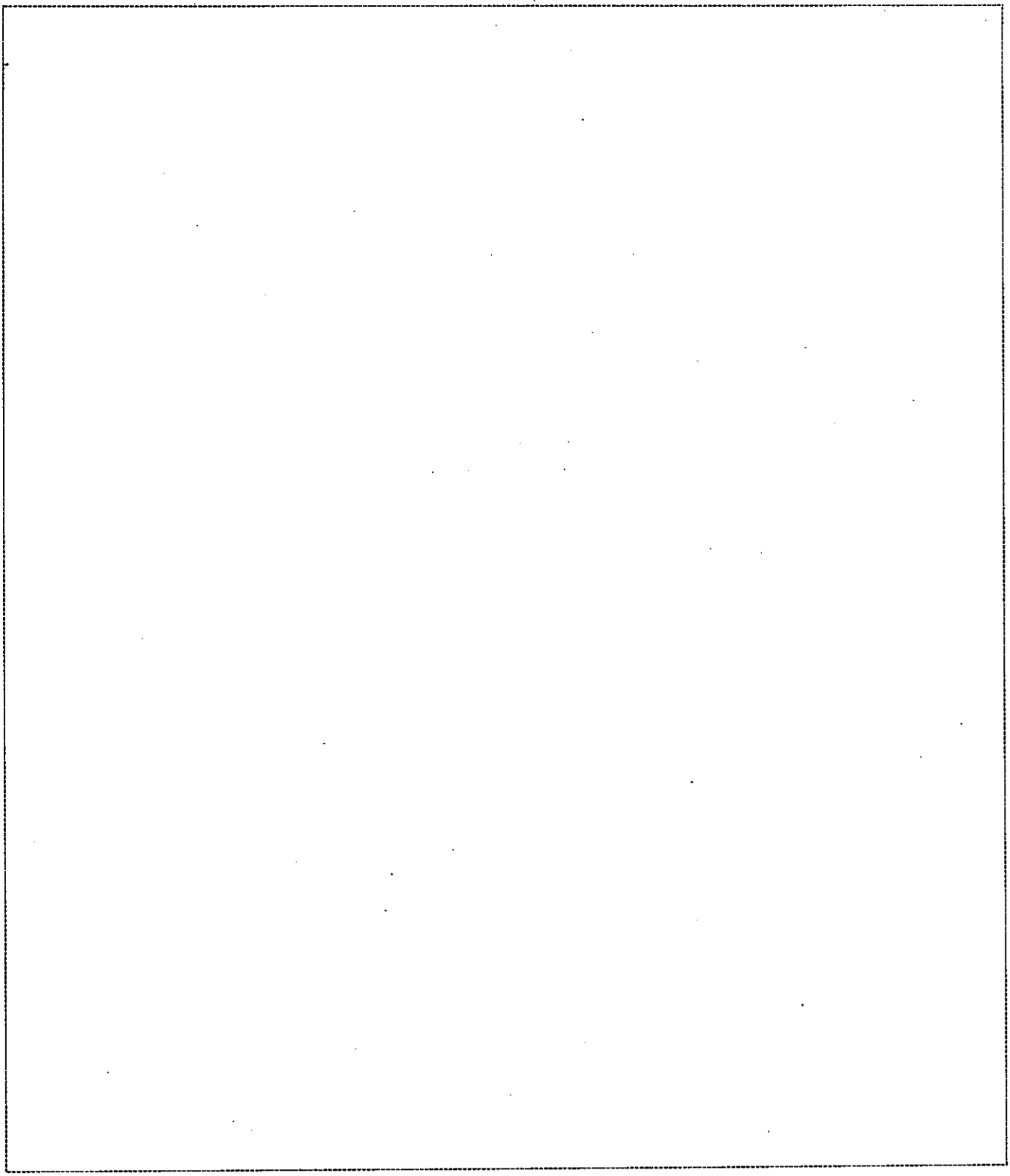
What is the output of the following program? Line numbers are provided for your own reference. Write your answer in the lines provided at the bottom of this page.

```
1    #include <stdio.h>

2    int main()
3    {
4            int a[4] = {5, 10, 15, 11};
5            int b = 1050;
6            int *p1 = &b;
7            int *p2 = a;
8            int *p3 = &a[2];

9            printf("%d %d %d %d %d %d %d\n", a[0], a[1], a[2],
                 b, *p1, *p2, *p3);

10       *p2 = 44;
11       p3 = p1;
12       *p3 = 55;
13       printf("%d %d %d %d %d %d %d\n", a[0], a[1], a[2],
           b, *p1, *p2, *p3);

14       a[2] = 99;
15       *p3 = 219;
16       printf("%d %d %d %d %d %d %d\n", a[0], a[1], a[2],
           b, *p1, *p2, *p3);

17       *(p2 + 1) = 555;
18       a[0] = 665;
19       printf("%d %d %d %d %d %d %d\n", a[0], a[1], a[2],
           b, *p1, *p2, *p3);

20       return(0);
}
```

Question 25: Weight 35

Complete the C function described below. Some marks will be given for the quality of your code. List all assumptions but please note that unreasonable assumptions will be penalized. The input parameters must not be modified by your solution.

```
/*
 * is_anagram
 *
 * input:
 * -- char *str1, *str2: two existing strings (character arrays)
 *
 * purpose: to return 1 if str1 is an anagram of str2; return 0 otherwise
 *
 * examples of anagrams:
 * -- An anagram for "Tom Cruise" is "So I'm Cuter!"
 * -- An anagram for "Waitress" is "A stew, Sir?"
 * -- An anagram for "software process" is "Cafe's worst prose"
 *
 * Note that letter case, whitespace and punctuation are ignored.
 *
 * You can use the "isalpha()" and "tolower()" functions of C to help you with
 * your work. The value of (int)'a' is 97.
 *
 */

int is_anagram(char *str1, char *str2)
```



Section D: Debugging (20 marks)

Question 26: Weight 20

(a) What are two of the programming techniques suggested for defensive programming? Explain why they are effective.

(b) Describe two approaches we can take towards debugging our code when we have good clues.

Section E: Make (10 marks)

Question 27: Weight 10

Consider the following makefile:

```
CC=gcc
FLAGS=-ansi -pendantic -Wall

parser: parse.o line.o pstack.o
	$(CC) -o parser line.o pstack.o parse.o

parse.o: parse.c token.h line.h pstack.h
	$(CC) $(FLAGS) -c parse.c

line.o: line.c line.h token.h
	$(CC) $(FLAGS) -c line.c

pstack.o: pstack.c pstack.h token.h
	$(CC) $(FLAGS) -c pstack.c

clean:
	rm -f *.o parser
```

(a) What must be recompiled if `token.h` is modified?

(b) What must be recompiled if `parse.h` is modified?

(c) Why do you think nothing follows immediately after "`clean:`"?

(d) How many rules does this makefile contain?

(e) Explain why a makefile (or equivalent tool) should be used for a non-trivial project.

END

This page is for the sole use of examination evaluators.

Section A	/40
Section B	/60
Section C	/50
Section D	/20
Section E	/10
Total	**/180**