TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS – 2017

TRIMESTER 1

NWEN241

SYSTEMS PROGRAMMING
VEC MOCK EXAM

Time Allowed: TWO HOURS

CLOSED BOOK

****Corrections to original exam****

****Solutions to the corrected exam****

Permitted materials: No calculators are allowed.

No electronic dictionaries are allowed.

Paper foreign to English language dictionaries are allowed.

Instructions: The examination contains 5 questions. You must answer ALL questions.

The exam consists of ~~120~~ 100 marks in total, with 20 marks for each of the 5 questions:

Question 1 C General Questions	**[20 marks]**
Question 2 Arrays Strings and Pointers	**[20 marks]**
Question 3 Data Structures and Memory	**[20 marks]**
Question 4 Projects, File I/O and Process Management	**[20 marks]**
Question 5 Python Fundamentals	**[20 marks]**

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 1. C General Questions **[20 marks]**

(a) **[4 marks]** Explain the four steps of compilation for C programs.

Preprocessing - Modifies the original text according to directives that begin withe the '#' character

Compilation - Compiles the text files into an assembly-language program

Assembly - Translates assembly into machine language instructions (relocatable object program)

Linking - Merges object files, resolves external links and creates an executable

(b) **[4 marks]** Explain how the Stack, Heap and Data Segment sections are used in program memory and how these sections relate to compile-time or run-time memory allocation.

Stack - (run-time) memory for data in a scope/block, for which the used portion expands and shrinks as the program runs.

Heap- (run-time) dynamically allocated memory requested by the execution

Data Segment - (compile-time) memory for global and static variables

(c) **[4 marks]** The floating point format (IEEE 754 single-precision form) consists of three sections. Name each section and how it is used to construct the decimal number.

sign (s) - determines positive/negative

exponent (e) - exponential portion (2e) and also provides representation for special values (0, inf, nan)**

fraction / mantissa / significand (m) - the fractional portion

The final float = ((s) ? -1 : 1) * m * 2e**

(d) **[4 marks]** The following macro computes the square of x. Discuss the issues with this macro:

```
#define SQ(x) x*x
```

In the current definition it has problems evaluating SQ(2+2):

SQ(2+2) => 2 + 2 * 2 + 2 => 2 + 4 + 2 => 8

Which is not the desired behaviour (16)

Similar problems with
#define SQ(x) (x)*(x) => 1.0/SQ(2) => 1.0/(2)*(2) => 0.5*2 => 1.0

and
#define SQ(x) ((x)*(x)) => x = 2; SQ(x++) => ((x++)*(x++)) => <undefined>

(e) **[4 marks]** What does the following print out:

```
#include <stdio.h>

int main(void)
{
    int i;
    float k;

    for (k=i=6; i-->2; k+=0.5)
    {
        if (i%2)
            ;
        else
            printf("k=%.2f\n", k);
    }
    return 0;
}
```

k=6.50
k=7.50

(note the extra decimal place from %.2f)

Question 2. Arrays, Strings and Pointers **[20 marks]**

(a) **[6 marks]** Consider the following code:

```
int a[3] = {1, 2, 3};
int *pa = a;

int m[4][4] = {{2, 4, 6, 8}, {22, 44, 66, 88},
               {1, 3, 5, 7}, {11, 33, 55, 77}};
int (*pm)[4] = m;
```

Give the outputs of the following `printf` statements.

`printf("%d", *a);` **[1 mark]**

1

`printf("%d", *(a+1));` **[1 mark]**

2

`printf("%d", pa[1]);` **[1 mark]**

2

`printf("%d", *(*m+2));` **[1 mark]**

6

`printf("%d", *(m[1]+2));` **[1 mark]**

66

`printf("%d", (*(pm+3))[2]);` **[1 mark]**

55

(b) **[4 marks]** Give a declaration for the variable `p` in each of the following cases.

`p` is an array of 5 elements of pointer to `char` **[1 mark]**

```
char *p[5];
```

`p` is a pointer to an array of 10 elements of `float` **[1 mark]**

```
float (*p)[10];
```

`p` is a `const` pointer to `int` **[1 mark]**

```
int *const p;
```

`p` is a function that takes a `const int` and a `float` array as arguments and returns a pointer to a `const int` **[1 mark]**

```
const int * p(const int, float[ ]);
```

(c) **[2 marks]** Consider the following code:

```
char s1[] = "Hello";
char s2[] = {'H', 'e', 'l', 'l', 'o'};
```

`sizeof(s1)` does not equal `sizeof(s2)`. Discuss why this is.

s1 = {'H', 'e', 'l', 'l', 'o', '\0'}; 6 characters

Because the array automatically allocates the right amount of memory to store the string literal, and the string literal contains the "null terminator" character, s1 is 6 chars while s2 is 5 chars meaning they are not the same size.

(d) **[3 marks]** Consider the following declarations:

```
char str1[] = "hello";
char *str2 = "hello";
const char *str3 = "hello";
```

State whether following statements would compile. If not explain why.

`str1 = "HELLO";` **[1 mark]**

No. You can not assign to an array variable after it has been declared.

ie, what happens to the original memory?
what if you assign something the wrong size?

`str2 = "HELLO";` **[1 mark]**

Yes.

`str3 = "HELLO";` **[1 mark]**

Yes.

(e) **[3 marks]** The following code code compiles but causes undefined behaviour. Why is this?

```
char *str = "hello";
str[0] = "H";
```

A string literal is stored in the Data-segment and is typically read-only. Attempting to modify it is undefined behaviour and often leads to a segfault.

error: assignment to expression with array type
str = "HELLO";
^

(f) **[3 marks]** Consider the following declarations.

```
void a(int **x);
void b(int (*x)[4]);
void c(int *x[]);
void d(int x[][4]);

int arr[2][4];
```

Which of `a`, `b`, `c`, or `d` could you use `arr` as an argument for? Briefly explain your answer.

b and d

Arrays decay into pointers when used in an expression (with few exceptions) but the type they point to doesn't change.

int arr[2][4] => int arr[ ][4] => int (*arr)[4] (this is okay)

int (*arr)[4] => int **arr (is the same as) int *p => char *p
Where you are changing the value you point to.

Question 3. Data Structures and Memory **[20 marks]**

(a) **[6 marks]** Consider the following code:

```
typedef union
{
    int *i;
    char c[8];
} Data;

char s[][] = {{'a', 'b', 'c'}};
```

Assume that this code is running on a 32-bit system. What is the result for `sizeof` for the given statements:

`sizeof(Data)` **[1 mark]**

8

`sizeof(Data*)` **[1 mark]**

4

`sizeof(&s)` **[1 mark]**

4

`sizeof(s)` **[1 mark]**

3

`sizeof(s[0])` **[1 mark]**

3

`sizeof(s[0][0])` **[1 mark]**

1

(b) **[3 marks]** Briefly explain what a **memory leak** is and describe how it may occur when using `malloc` to allocate memory.

A Memory Leak is when dynamically allocated memory can not be deallocated during program execution.

If you use 'malloc' to get a pointer to some dynamically allocated memory then lose the value of the pointer (for example, going out of scope) then you can no longer call 'free' to deallocate the memory. This leads to a memory leak.

(c) **[3 marks]** Assume the following `malloc` is successful:

```
int *ptr = malloc(10 * sizeof(int));
```

State the possible outcomes of the following statment and discuss why a temporary variable `tmp` is used:

```
int *tmp = realloc(ptr, 100 * sizeof(int));
```

In the successful case, tmp points to the new memory and original memory (ptr) is freed.

In the unsuccessful case tmp points to NULL and the original memory (ptr) is NOT freed.

If you did
int *ptr = realloc(...)
and it was unsuccessful, ptr would point to NULL and the original memory would be lost leading to a memory leak.

(d) **[2 marks]** Briefly explain the difference between `malloc` and `calloc`. Discuss when `malloc` should be used in preference to `calloc`.

Malloc takes the number of bytes you want to allocate and returns a pointer to un-initialized memory

Calloc takes a size(bytes) and a count and returns a points to size*count bytes of memory cleared to 0

If you were requesting memory for single struct or an array that you are immediately going to overwrite then malloc is more efficient because it does not need to clear the values.

(e) **[2 marks]** Explain why the following code would not compile.

```
int *p1 = malloc(128);
extern i = 0;
```

this is ment to be "extern int i"

```
int main(void)
{
    char *p2 = malloc(128);
    return 0;
}
```

You can not execute code outside of functions in C.
You can only evaluate constant expressions (such as assigning a value that can be evaluated at compile time, like a literal string).

ie: when would it get called otherwise? before main? what if it fails?

(f) **[4 marks]** What does the following print out:

```
#include <stdio.h>

static int a = 0;
int foo(int);


int main(void)
{
    static int b = 0;

    b += foo(2);
    printf("a=%d,b=%d\n", a, b);

    b += foo(3);
    printf("a=%d,b=%d\n", a, b);

    return 0;
}


int foo(int n)
{
    static int b = 0;
    int i;

    for (i = 0; i < n; ++i)
    {
        a++;
        b++;
    }

    return b;
}
```

a=2,b=2
a=5,b=7

Question 4. Projects, File I/O and Process Management **[20 marks]**

(a) **[4 marks]** Consider the following code:

```
#include <stdio.h>

char l337ify(char);


int main(int arc, char* argv[])
{
    if (arc != 3)
        return 1;

    FILE *fin, *fout;
    int c;

    if ((fin = fopen(argv[1], "r")) == NULL
    || (fout = fopen(argv[2], "w")) == NULL)
    {
        printf("Failure.\n");
        return 1;
    }

                                v '\n'
    fprintf(fout, "%d␣speak", 1337);
    while ((c = fgetc(fin)) != EOF)
    {
        fputc(l337ify(c), fout);
    }

    fclose(fin);
    fclose(fout);

    return 0;
}


char l337ify(char c)
{
    switch(c)
    {
    case 'e' : return '3';
    case 't' : return '7';
    case 'o' : return '0';
    default : return c;
    }
}
```

(Question 4 continued on next page)

(Question 4 continued)

i. **[4 marks]** Assume the program runs without errors. Given the following file as `arg[1]` what is the output to the file in `arg[2]` ?

```
Whats the good of Mercators
North Poles and Equators,
Tropics, Zones, and Meridian Lines?
```

1337 speak
Wha7s 7h3 g00d 0f M3rca70rs
N0r7h P0l3s and Equa70rs,
Tr0pics, Z0n3s, and M3ridian Lin3s?

ii. **[4 marks]** What is the difference between `"r"`, `"w"` and `"a"` as arguments for `fopen` when opening an **existing file**

r - opens the file in 'read' mode

w - opens the file and discards the original data for 'write' mode

a - opens the file ready to append to the end of existing data in 'append' mode

(b) **[3 marks]** Discuss the difference between headers files (.h) and source files (.c) and why they are used for large projects.

Modularity and abstraction.

Header files can declare an API without exposing internal mechanisms
Source files include headers and define the API.

It also reduces compilation time if you make changes to a small part of the source and not the headers.

(c) **[3 marks]** Briefly explain what a zombie process is and give an example of how it can be avoided.

A Zombie process is a process that has finished executing but is waiting to be cleaned up by the parent process.

The parent can either wait for the child.
The process can force quit with _exit() instead.

(d) **[2 marks]** `exec` is a function that only returns if an error occurs. In the successful case that `exec` is called what happens to memory of the process that calls it?

The new process's memory block entirely replaces it.

(e) **[4 marks]** Consider the following code:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main(void)
{
    int status, id1, id2;

    id1 = fork();
    if (id1)
    {
        printf("a");
        waitpid(id1, &status, 0);
    }
    else
    {
        printf("b");
        exit(0);
    }

    id2 = fork();
    if (id2)
    {
        printf("c");
        waitpid(id2, &status, 0);
    }
    else
    {
        printf("d");
        exit(0);
    }

    printf("e\n");
    return 0;
}
```

List ALL possible outputs assuming no errors occur (Hint: draw a diagram of the processes, where they fork and where they join).

abcde
abdce
bacde
badce

Question 5. Python Fundamentals **[20 marks]**

(a) **[4 marks]** Determine the **type AND value** of x in the following statements.

```
x = not(4 < 3)or (6 >= 9)
```

bool true

```
x = 'Hello'
x = x[2:5]
```

string 'llo'

```
x = 10.5//4
```

float 2.0

```
x = {2, 3, 4}
x.add('4')
```

set { 2, 3, 4, '4' }

```
x = (1, 2, 3)
x = x + x
```

tuple (1, 2, 3, 1, 2, 3)

```
x = {'a':3, 'b':2, 'c':0}
x[3] = 'd'
```

dict { 'a':3, 'b':2, 'c':0, 3:'d' }

(b) **[10 marks]** What does the following code print out:

i. **[2 marks]**

```
s = 'This takes the cake'
l = s.split(' ')
for w in l :
    print(w.upper())
```

```
THIS
TAKES
THE
CAKE
```

ii. **[2 marks]**

```
l = list('ab')
l.extend('cd')
l.append('ef')
print(l)
```

```
[ 'a', 'b', 'c', 'd', 'ef' ]
```

iii. **[2 marks]**

```
l = [0, 1, 2, 3, 4, 5]
for i in range(-3, 3) :
    print(l[i])
```

```
3
4
5
0
1
2
```

(Question 5 continued on next page)

(Question 5 continued)

iv. **[2 marks]**

```
x = [1, 3, 1, 5, 2, 1, 3]
print([e**2 for e in x[1:-1]])
```

[9, 1, 25, 4, 1]

v. **[2 marks]**

```
a = {1, 2, 3, 4}
b = {3, 4, 5, 6}

print(b - a)
print(a - b)
print(a | b)
print(a & b)
```

{ 5, 6 }
{ 1, 2 }
{ 1, 2, 3, 4, 5, 6 }
{ 3, 4 }

(c) **[2 marks]** Consider the following code:

```
import os.path
print(path.abspath(''))
```

Explain why this code will throw a runtime error?

The statment 'path.abspath('')' is missing the 'os.' prefix

Using this sort of import statement you still need to prefix the entire module before calling any functions.

(d) **[2 marks]** Explain the difference between calling a function using positional arguments and calling a function using keyword arguments.

When calling a function with positional arguments you must specify your arguments in the correct order, but you do not have to use the argument names.

When calling a function with keyword arguments you do not have to order your arguments but you will have to assign them to the names defined by the function.

(e) **[2 marks]** What does an immutable type mean in python? Give **three** examples of an immutable type.

A type whos value can not be modified.
ie x = (1, 2); x.append(3)

A variable with an immutable type may still be changed.
ie x = (1, 2); x = (1, 2, 3)

Immutable types : bool, int, float, tuple, string, frozenset

Question 6. Trivia **[361 marks]**

(a) **[361 marks]** 2016 and again in 2017 a computer algorithm developed by DeepMind beat professional players in an abstract strategy board game invented in ancient China more than 2,500 years ago. What is the Korean name for this board game?

Baduk

* * * * * * * * * * * * * * *