CPSC 213 Midterm 2019W Term 1 (Oct 21, 2019)

1. Numbers and Memory [8 marks]

a) Consider the following code running on a *little-endian* machine where the address of `i` is `0x1000`.

```
int i = 0x1234;
```

What is the hex value of the byte at address `0x1000`?

b) Consider the following code where the address of `s` is `0x1000`.

```
struct SS {
    char c;
    int i;
    char d[2];
};
struct SS s;
```

What is the value of the expression `&s.i` ?

What is the value of the expression `sizeof(s)` ?

c) Consider the following code

```
int i = 0x56789a;
char c = (char) i;
int j = c;
```

What is the hex value of `c` ?

What is the hex value of `j` ?

1 **[5 marks]** **Memory and Numbers.** Consider the execution of the following code on a *little-endian* processor. Assume that the address of `i` is `0x1000` and that the compiler allocates the three global variables contiguously, in the order they appear, wasting no memory between them other than what is required to ensure that they are properly aligned (and assuming that `char`'s are **signed**, and `int`'s and pointers are **4-bytes** long).

```
int  i;
char c;
int* j

void foo() {
    i = 0x22446688;
    j = &i;
    c = (char) i;
    *j = i | c;
}
```

1a For every byte of memory whose value you can determine from the information given above, give its value in hex on the line labeled with its address. For addresses whose value you can not determine, check the *Unknown* box.

0x1000: Unknown ○ or Value _____

0x1001: Unknown ○ or Value _____

0x1002: Unknown ○ or Value _____

0x1003: Unknown ○ or Value _____

0x1004: Unknown ○ or Value _____

0x1005: Unknown ○ or Value _____

0x1006: Unknown ○ or Value _____

0x1007: Unknown ○ or Value _____

0x1008: Unknown ○ or Value _____

0x1009: Unknown ○ or Value _____

0x100a: Unknown ○ or Value _____

0x100b: Unknown ○ or Value _____

CPSC 213, Winter 2016, Term 1

Midterm I Sample Questions

Exercise 1
What is the value of i after the following Java statements execute?

```
byte b = 0x84;

int i = b << 8;
```

Recall that int's are 4-bytes long. Give your answer as a single number in hex.

Exercise 2
Consider the following content of a portion of memory (in the form of address: value):

```
0x1000: 0x12

0x1001: 0x34

0x1002: 0x56

0x1003: 0x78
```

What is the little endian value of the 4-byte integer at address 0x1000? Give your answer as a single value in hex.

Exercise 3
Consider the following C code with three global variables, a, b, and c, that are stored at addresses 0x1000, 0x2000, 0x3000, respectively, and a procedure `foo()` that accesses them.

```
int a[1];  // at address 0x1000
int b[1];  // at address 0x2000
int* c;    // at address 0x3000
void foo() {
      a[0] = 1;
      b[0] = 2;
      c = a;
      c[0] = 3;
      c = b;
      *c = 4;
}
```

Describe what you know about the content of memory following the execution of `foo()` on a 32-bit Little Endian processor. List only memory locations whose address and value you know. List each byte of memory separately using the form "`byte_address: byte_value`". List all numbers in hex.

CPSC 213, Winter 2016, Term 2 — Final Exam

Date: April 19, 2017; Instructor: Mike Feeley and Alan Wagner

This is a closed book exam. No notes or electronic calculators are permitted. You may remove the last page.

Answer in the space provided. Show your work; use the backs of pages if needed. There are **15** questions on **16** pages, totaling **103** marks. You have **3 hours** to complete the exam.

STUDENT NUMBER: _____

NAME: _____

SIGNATURE: _____

Q1	/ 4
Q2	/ 6
Q3	/ 6
Q4	/ 8
Q5	/ 4
Q6	/ 9
Q7	/ 10
Q8	/ 9
Q9	/ 6
Q10	/ 6
Q11	/ 6
Q12	/ 8
Q13	/ 9
Q14	/ 6
Q15	/ 6

1 (4 marks) Variables and Memory. Consider the following code running on 32-bit, big-endian architecture.

```
struct S {
    char a[4];
};

int       i;
struct S* s;

char foo() {
    i = 0x12345678;
    s = (struct S*) &i;
    return s->a[1];
}
```

Does the procedure `foo()` compile and execute without error? If not, what is the error?

If so, can you determine what value it returns?

If so, what is that value?

CPSC 213, Winter 2015, Term 2 — Midterm Exam

Date: February 29, 2016; Instructor: Mike Feeley

This is a closed book exam. No notes. No electronic calculators. Note: this is a normal 1-hour-length midterm, but you have 2 hours to write it.

Answer in the space provided. Show your work; use the backs of pages if needed. There are **9** questions on **8** pages, totaling **64** marks. You have **2 hours** to complete the exam.

STUDENT NUMBER: _____

NAME: _____

SIGNATURE: _____

Q1	/ 8
Q2	/ 6
Q3	/ 6
Q4	/ 6
Q5	/ 6
Q6	/ 6
Q7	/ 8
Q8	/ 8
Q9	/ 10

1 (8 marks) **Memory and Numbers.** Consider the following C code containing global variables `a`, `b`, and `c` that is executing on a *little endian*, 32-bit processor. Assume that the address of `a[0]` is `0x1000` and that the compiler allocates global variables in the order they appear in the program without *unnecessarily* wasting space between them. With this information, you can determine the value of certain bytes of memory following the execution of `foo()`.

```
char a[2];
int  b[2];
int* c;
```

```
void foo() {
    a[1] = 0x10;
    b[1] = 0x30405060;
    c    = b;
}
```

List the address and value of every memory location whose address and value you know. Use the form "`address: value`". **List every byte on a separate line** and list all numbers in hex.

CPSC 213, Winter 2015, Term 2 — MIDTERM SAMPLE QUESTIONS

Date: Feb 18, 2016; Instructor: Mike Feeley

This sample has about twice the number of questions as the actual midterm. If you can answer this midterm completely in 100 minutes you should be able to answer the actual midterm in 50 minutes.

Answer in the space provided. Show your work; use the backs of pages if needed. There are 16 questions on 13 pages, totaling 112 marks. You have 100 minutes to complete the exam.

STUDENT NUMBER: _____

NAME: _____

SIGNATURE: _____

Q1	/ 7
Q2	/ 7
Q3	/ 6
Q4	/ 3
Q5	/ 6
Q6	/ 8
Q7	/ 7
Q8	/ 3
Q9	/ 3
Q10	/ 4
Q11	/ 4
Q12	/ 4
Q13	/ 10
Q14	/ 20
Q15	/ 10
Q16	/ 10

1 (7 marks) Variables and Memory. Consider the following C code with three global variables, `a`, `b`, and `c`, that are stored at addresses `0x1000`, `0x2000`, `0x3000`, respectively, and a procedure `foo()` that accesses them.

```
int a[1];   // at address 0x1000
int b[1];   // at address 0x2000
int* c;     // at address 0x3000

void foo() {
    a[0] = 1;
    b[0] = 2;
    c = a;
    c[0] = 3;
    c = b;
    *c = 4;
}
```

Describe what you know about the content of memory following the execution of `foo()` on a 32-bit **Little Endian** processor. List only memory locations whose address and value you know. **List each byte of memory separately** using the form "`byte_address: byte_value`". List all numbers in hex.

CPSC 213, Winter 2014, Term 1 — Midterm

Date: October 24, 2014; Instructor: Mike Feeley

This is a closed book exam. No notes. No electronic calculators.

Answer in the space provided. Show your work; use the backs of pages if needed. There are **9** questions on **8** pages, totaling **50** marks. You have **50 minutes** to complete the exam.

STUDENT NUMBER: _____

NAME: _____

SIGNATURE: _____

Q1	/ 7
Q2	/ 7
Q3	/ 6
Q4	/ 3
Q5	/ 6
Q6	/ 8
Q7	/ 7
Q8	/ 3
Q9	/ 3
Total	/ 50

1 (7 marks) **Variables and Memory.** Consider the following C code with three global variables, `a`, `b`, and `c`, that are stored at addresses `0x1000`, `0x2000`, `0x3000`, respectively, and a procedure `foo()` that accesses them.

```
int a[1];   // at address 0x1000
int b[1];   // at address 0x2000
int* c;     // at address 0x3000

void foo() {
    a[0] = 1;
    b[0] = 2;
    c = a;
    c[0] = 3;
    c = b;
    *c = 4;
}
```

Describe what you know about the content of memory following the execution of `foo()` on a 32-bit **Little Endian** processor. List only memory locations whose address and value you know. **List each byte of memory separately** using the form "`byte_address: byte_value`". List all numbers in hex.

CPSC 213, Winter 2014, Term 1 — Some More Sample Midterm Questions

Date: October 2014; Instructor: Mike Feeley

This was a closed book exam. No notes. No electronic calculators. This sample combines questions from multiple exams and so a real exam will have fewer total questions.

Answer in the space provided. Show your work; use the backs of pages if needed. There are **15** questions on **8** pages, totaling **79** marks. You have **50 minutes** to complete the exam.

STUDENT NUMBER: _____

NAME: _____

SIGNATURE: _____

Q1	/ 2
Q2	/ 4
Q3	/ 4
Q4	/ 6
Q5	/ 6
Q6	/ 3
Q7	/ 8
Q8	/ 10
Q9	/ 5
Q10	/ 3
Q11	/ 3
Q12	/ 3
Q13	/ 3
Q14	/ 9
Q15	/ 10
Total	/ 79

1 (2 marks) **Memory Alignment.** The memory address 0x2018 is aligned for certain byte size accesses but not to others. List all power-of-two sizes for which it is aligned and carefully justify your answer.

2 (4 marks) **Pointer Arithmetic.** Without using the `[]` array syntax (i.e., using only pointer arithmetic) and without introducing **any** additional variables, finish implementing this function that copies the first `n` integers of array `from` into array `to`.

```
void copy (int* from, int* to, int n) {
```

4b `&a[3]`

4c `b[3]`

5 **(6 marks)** **Instance Variables.** In the context of the following C declarations:

```
struct S {                              struct S  a;
    int i,j;                            struct S* b;
};
```

Indicate what, if anything, about the following expressions can not be computed statically. Be precise.

5a `&a.i`

5b `&b->i`

5c `(&b->j)-(&b->i)`

6 **(3 marks)** **Memory Endianness** Examine this C code.

```
char a[4];
*((int*) (&a[0])) = 1;
```

Carefully explain how this code can be used to determine the *endianness* of the machine on which it runs, recalling that on a *Big Endian* machine, the high-order (most-significant) byte of a number has the lowest address.

7 **(8 marks)** Consider the following C declarations.

```
struct S {                              int       a[10];
  int       i;                          int*      b;
  int       j[10];                      int       c;
  struct S* k;                          struct S  d;
};                                      struct S* e;
```

CPSC 213, Winter 2014, Term 1 — Sample Midterm (Winter 2013 Term 2)

Date: October 2014; Instructor: Mike Feeley

This is a closed book exam. No notes. No electronic calculators.

Answer in the space provided. Show your work; use the backs of pages if needed. There are **9** questions on **7** pages, totaling **50** marks. You have **50 minutes** to complete the exam.

STUDENT NUMBER: _____

NAME: _____

SIGNATURE: _____

Q1	/ 8
Q2	/ 4
Q3	/ 4
Q4	/ 4
Q5	/ 8
Q6	/ 4
Q7	/ 4
Q8	/ 4
Q9	/ 10
Total	/ 50

1 (8 marks) **Memory and Numbers**. Consider the following C code with global variables `a` and `b`.

```
int  a[2];
int* b;

void foo() {
    b    = a;
    a[0] = 1;
    b[1] = 2;
}

void checkGlobalVariableAddressesAndSizes() {
    if ((&a==0x2000) && (&b==0x3000) && sizeof(int)==4  && sizeof(int*)==4)
        printf ("OKAY");
}
```

When `checkGlobalVariableAddressesAndSizes()` executes it prints "`OKAY`". Recall that `sizeof(t)` returns the number of bytes in variables of type `t`.

Describe what you know about the content of memory following the execution of `foo()` on a **Little Endian** processor. List only memory locations whose address and value you know. List each byte of memory on a separate line using the form: "`byte_address: byte_value`". List all numbers in hex.

CPSC 213, Winter 2013, Term 2 — Final Exam

Date: April 14, 2014; Instructor: Mike Feeley

This is a closed book exam. No notes or electronic calculators are permitted.

Answer in the space provided. Show your work; use the backs of pages if needed. There are **12** questions on **12** pages, totaling **105** marks. You have **2.5 hours** to complete the exam.

STUDENT NUMBER: _____

NAME: _____

SIGNATURE: _____

Q1	/ 5
Q2	/ 8
Q3	/ 8
Q4	/ 10
Q5	/ 12
Q6	/ 3
Q7	/ 6
Q8	/ 12
Q9	/ 6
Q10	/ 9
Q11	/ 10
Q12	/ 16
Total	/ 105

1 (5 marks) **Variables and Memory.** Consider the following C code with three global variables, `a`, `b`, and `c`, that are stored at addresses `0x1000`, `0x2000`, `0x3000`, respectively.

```
void foo() {
    a[0] = 1;
    b[0] = 2;
    c = a;
    c[0] = 3;
    c = b;
    *c = 4;
}
```

```
int a[1];
int b[1];
int* c;
```

Describe what you know about the content of memory following the execution of `foo()` on a 32-bit **Big Endian** processor. List only memory locations whose address and value you know. List each byte of memory separately using the form "`byte_address: byte_value`". List all numbers in hex.

1 (2 marks) **Memory Alignment.** Consider the memory address 0x92. List all power-of-two sizes for which aligned memory access is possible and carefully justify your answer.

2 (8 marks) **Pointer Arithmetic.** Consider the following lines of C code. For the assignments to `i` , `j`, `k`, and `m` say (a) whether the code generates an error and why or (b) what value the variables have after the code executes. If one line generates an error but a later one does not, give the value of the later ones. Show your work.

```
int a[10] = { 9,8,7,6,5,4,3,2,1,0 };
int i = *(a+4);
int j = &a[3] - &a[1];
int k = *(a+*(a+6));
int m = *(&a[5]-a);
```

2a `i`:

2b `j`:

2c `k`:

2d `m`:

CPSC 213 Midterm 2019W Term 1 (Oct 21, 2019) – Solution

1. Numbers and Memory [8 marks]

a) Consider the following code running on a *little-endian* machine where the address of `i` is `0x1000`.

```
int i = 0x1234;
```

What is the hex value of the byte at address `0x1000`?

0x34

b) Consider the following code where the address of `s` is `0x1000`.

```
struct SS {
    char c;
    int i;
    char d[2];
};
struct SS s;
```

What is the value of the expression `&s.i` ?

0x1004

What is the value of the expression `sizeof(s)` ?

12

c) Consider the following code

```
int i = 0x56789a;
char c = (char) i;
int j = c;
```

What is the hex value of `c` ?

0x9a

What is the hex value of `j` ?

0xffffff9a

CPSC 213, Winter 2018, Term 2 — Midterm Exam **Solution**

Date: February 29, 2019; Instructor: Mike Feeley and Anthony Estey

1 [5 marks] **Memory and Numbers.** Consider the execution of the following code on a *little-endian* processor. Assume that the address of `i` is `0x1000` and that the compiler allocates the three global variables contiguously, in the order they appear, wasting no memory between them other than what is required to ensure that they are properly aligned (and assuming that `char`'s are **signed**, and `int`'s and pointers are **4-bytes** long).

```
int  i;
char c;
int* j

void foo() {
    i = 0x22446688;
    j = &i;
    c = (char) i;
    *j = i | c;
}
```

1a For every byte of memory whose value you can determine from the information given above, give its value in hex on the line labeled with its address. For addresses whose value you can not determine, check the *Unknown* box.

0x1000: 0x88 Unknown ○ or Value _____

0x1001: 0xff Unknown ○ or Value _____

0x1002: 0xff Unknown ○ or Value _____

0x1003: 0xff Unknown ○ or Value _____

0x1004: 0x88 Unknown ○ or Value _____

0x1005: Unknown Unknown ○ or Value _____

0x1006: Unknown Unknown ○ or Value _____

0x1007: Unknown Unknown ○ or Value _____

0x1008: 0x00 Unknown ○ or Value _____

0x1009: 0x10 Unknown ○ or Value _____

0x100a: 0x00 Unknown ○ or Value _____

0x100b: 0x00 Unknown ○ or Value _____

2 [10 marks] **Global Variables and Arrays.** Answer the following questions about global variables.

CPSC 213, Summer 2016, Term 2

Midterm I Sample Questions

Exercise 1
What is the value of i after the following Java statements execute?

```
byte b = 0x84;

int i = b << 8;
```

Recall that int's are 4-bytes long. Give your answer as a single number in hex.

0xffff8400

Exercise 2
Consider the following content of a portion of memory (in the form of address: value):

```
0x1000: 0x12

0x1001: 0x34

0x1002: 0x56

0x1003: 0x78
```

What is the little endian value of the 4-byte integer at address 0x1000? Give your answer as a single value in hex.

0x78563412

Exercise 3
Consider the following C code with three global variables, a, b, and c, that are stored at addresses 0x1000, 0x2000, 0x3000, respectively, and a procedure `foo()` that accesses them.

```
int a[1];  // at address 0x1000
int b[1];  // at address 0x2000
int* c;    // at address 0x3000
void foo() {
      a[0] = 1;
      b[0] = 2;
      c = a;
      c[0] = 3;
      c = b;
      *c = 4;
}
```

Describe what you know about the content of memory following the execution of `foo()` on a 32-bit Little Endian processor. List only memory locations whose address and value you know.

CPSC 213, Winter 2016, Term 2 — Final Exam **Solution**

Date: April 19, 2017; Instructor: Mike Feeley and Alan Wagner

1 **(4 marks)** **Variables and Memory.** Consider the following code running on 32-bit, big-endian architecture.

```
struct S {
    char a[4];
};

int       i;
struct S* s;
```

```
char foo() {
    i = 0x12345678;
    s = (struct S*) &i;
    return s->a[1];
}
```

Does the procedure `foo()` compile and execute without error? If not, what is the error?

Yes, it compiles

If so, can you determine what value it returns?

Yes you can.

If so, what is that value?

4 marks, In big endian it is 34 and in little endian 56

2 **(6 marks)** **Global Variables.** Consider the following C declaration of global variables. Give the SM213 assembly code for each of the following C statements. Use labels such as `a`, `b` etc. for statically know values. Comments are not required. You can treat your answers as a sequence and thus use register values loaded in one subquestion in subsequent ones to avoid duplication.

```
int  a;
int* b;
int* c[10];
```

2a `b = &a;`

```
ld   $a, r0          # r0 = &a
ld   $b, r1          # r0 = &b
st   r0,(r1)         # b = &a
```

2b `a = b[a];`

```
ld   $a, r0          # r0 = &a
ld   $b, r1          # r0 = &b
## new part
ld   0(r0), r2       # r1 = a
ld   0(r1),r3        # r1 = b
ld   (r3,r2,4),r4    # r4 = b[a]
st   r4,(r0)         # a = b[a]
```

2c `a = *c[a];`

```
ld   $a, r0          # r0 = &a
ld   0(r0), r2       # r1 = a
## new part
ld   $c,r4           # r4 = &c[0]
ld   (r4,r2,4),r5    # r5 = c[a]
ld   0(r5),r6        # r6 = *c[a]
st   r6,(r0)         # a = *c[a]
```

3 **(6 marks)** **Instance and Local Variables.** Give the SM213 assembly code for the following statements that access elements of global and local structs. Consider each statement as if it were the last line of `foo()` at the location indicated by the comment. Use labels such as `a`, `b` etc. for statically known values. Assume that `r5` stores the value of the stack pointer. Comments are not required. You can treat your answers as a sequence and thus use register values loaded in one subquestion in subsequent ones to avoid duplication.

CPSC 213, Winter 2015, Term 2 — Midterm Exam **Solution**

Date: February 29, 2016; Instructor: Mike Feeley

1 (8 marks) **Memory and Numbers.** Consider the following C code containing global variables `a`, `b`, and `c` that is executing on a *little endian*, 32-bit processor. Assume that the address of `a[0]` is `0x1000` and that the compiler allocates global variables in the order they appear in the program without *unnecessarily* wasting space between them. With this information, you can determine the value of certain bytes of memory following the execution of `foo()`.

```
char a[2];
int  b[2];
int* c;
```

```
void foo() {
    a[1] = 0x10;
    b[1] = 0x30405060;
    c    = b;
}
```

List the address and value of every memory location whose address and value you know. Use the form "`address: value`". **List every byte on a separate line** and list all numbers in hex.

```
0x1001: 0x10
0x1008: 0x60
0x1009: 0x50
0x100a: 0x40
0x100b: 0x30
0x100c: 0x04
0x100d: 0x10
0x100e: 0x00
0x100f: 0x00
```

2 (6 marks) **Static Scalars and Arrays.** Consider the following C code containing global variables `s` and `d`.

```
int  s[2];
int* d;
```

Use `r0` for the variable `i` (i.e., do not read or write memory for `i`) and use labels for static values. Give assembly code that is equivalent to each of the following C statements. Treat each question separately; i.e., do not use register values assigned in a previous answer.

2a `i = s[i];`

```
ld $s, r1             # r1 = s
ld (r1, r0, 4), r0    # i = s[i]
```

2b `i = d[i];`

```
ld $d, r1             # r1 = &d
ld (r1), r1           # r1 = d
ld (r1, r0, 4), r0    # i = d[i]
```

2c `d = &s[10];`

```
ld  $s, r1     # r1 = s
ld  $40, r2    # r2 = 10*4
add r2, r1     # r1 = &s[10]
ld  $d, r2     # r2 = &d
st  r1, (r2)   # d = &s[10]
```

3 (6 marks) **Structs and Instance Variables** Consider the following C code containing the global variable `a`.

```
struct S {
    int       x;
    int       y;
    struct S* z;
};
```

```
struct S* a;
```

CPSC 213, Winter 2015, Term 2 — MIDTERM SAMPLE QUESTIONS
Solution

Date: Feb 18, 2016; Instructor: Mike Feeley

1 (7 marks) **Variables and Memory.** Consider the following C code with three global variables, `a`, `b`, and `c`, that are stored at addresses `0x1000`, `0x2000`, `0x3000`, respectively, and a procedure `foo()` that accesses them.

```
int a[1];   // at address 0x1000
int b[1];   // at address 0x2000
int* c;     // at address 0x3000

void foo() {
    a[0] = 1;
    b[0] = 2;
    c = a;
    c[0] = 3;
    c = b;
    *c = 4;
}
```

Describe what you know about the content of memory following the execution of `foo()` on a 32-bit **Little Endian** processor. List only memory locations whose address and value you know. **List each byte of memory separately** using the form "`byte_address: byte_value`". List all numbers in hex.

```
0x1000: 0x03
0x1001: 0x00
0x1002: 0x00
0x1003: 0x00
0x2000: 0x04
0x2001: 0x00
0x2002: 0x00
0x2003: 0x00
0x3000: 0x00
0x3001: 0x20
0x3002: 0x00
0x3003: 0x00
```

2 (7 marks) **C Pointers.** Consider the following C code.

```
int  a[10] = {0,1,2,3,4,5,6,7,8,9};  // i.e., a[i] = i
int* b     = a+4;

int foo (int* x, int* y, int* z) {
    *x = *x + *y;
    *x = *x + *z;

    return *x;
}

int bar () {
    return foo (b - 2, a + (b - a) + (&a[7] - &a[6]), a + 2);
}
```

What value does `bar()` return? Justify your answer (1) by simplifying the description of the arguments to `foo()` as much as possible so that the relationship among them, if any, is clear and (2) by carefully explaining what happens when `foo()` executes.

CPSC 213, Winter 2014, Term 1 — Midterm Solution

Date: October 24, 2014; Instructor: Mike Feeley

1 (7 marks) Variables and Memory. Consider the following C code with three global variables, `a`, `b`, and `c`, that are stored at addresses `0x1000`, `0x2000`, `0x3000`, respectively, and a procedure `foo()` that accesses them.

```
int a[1];   // at address 0x1000
int b[1];   // at address 0x2000
int* c;     // at address 0x3000

void foo() {
    a[0] = 1;
    b[0] = 2;
    c = a;
    c[0] = 3;
    c = b;
    *c = 4;
}
```

Describe what you know about the content of memory following the execution of `foo()` on a 32-bit **Little Endian** processor. List only memory locations whose address and value you know. **List each byte of memory separately** using the form "`byte_address: byte_value`". List all numbers in hex.

```
0x1000: 0x03
0x1001: 0x00
0x1002: 0x00
0x1003: 0x00
0x2000: 0x04
0x2001: 0x00
0x2002: 0x00
0x2003: 0x00
0x3000: 0x00
0x3001: 0x20
0x3002: 0x00
0x3003: 0x00
```

2 (7 marks) C Pointers. Consider the following C code.

```
int  a[10] = {0,1,2,3,4,5,6,7,8,9};  // i.e., a[i] = i
int* b     = a+4;

int foo (int* x, int* y, int* z) {
    *x = *x + *y;
    *x = *x + *z;

    return *x;
}

int bar () {
    return foo (b - 2, a + (b - a) + (&a[7] - &a[6]), a + 2);
}
```

What value does `bar()` return? Justify your answer (1) by simplifying the description of the arguments to `foo()` as much as possible so that the relationship among them, if any, is clear and (2) by carefully explaining what happens when `foo()` executes.

CPSC 213, Winter 2014, Term 1 — Some More Sample Midterm Questions
Solution

Date: October 2014; Instructor: Mike Feeley

1 **(2 marks)** **Memory Alignment.** The memory address 0x2018 is aligned for certain byte size accesses but not to others. List all power-of-two sizes for which it is aligned and carefully justify your answer.

The lower 3 bits are zero, but the 4th bit is 1 and so its aligned for 2, 4, and 8 byte access, but not for anything more than that.

2 **(4 marks)** **Pointer Arithmetic.** Without using the `[]` array syntax (i.e., using only pointer arithmetic) and without introducing **any** additional variables, finish implementing this function that copies the first `n` integers of array `from` into array `to`.

```
void copy (int* from, int* to, int n) {
    while (n--)
        *to++ = *from++;
}
```

3 **(4 marks)** **Dynamic Allocation.** A *dangling pointer* exits when a program retains a reference to dynamically allocated memory after it has been freed/reclaimed. A *memory leak* exists when a program fails to free/reclaim dynamically allocated memory that it is no longer needed by the program.

3a Carefully explain the most serious symptom of a dangling-pointer bug.

The memory pointed to by the dangling-pointer may be re-allocated for some other use and then the program might use the dangling-pointer to erroneously update this newly allocated thing, thinking its updated the old (but freed) thing.

3b Carefully explain the most serious symptom of a memory-leak bug.

The program could eventually exhaust memory and slow itself, and the entire system it runs on, horribly.

3c Does Java's garbage collector completely eliminate the dangling-pointer bug? Justify your answer carefully.

Yes. It will only free memory when it is unreachable via any pointer in the program.

3d Does Java's garbage collector completely eliminate the memory-leak bug? Justify your answer carefully.

No. Sometimes program's retain references to objects that they never intend to use again. The GC can't free these things since they are still reachable.

4 **(6 marks)** **Global Arrays.** In the context of the following C declarations:

```
int a[10];
int *b;
```

Indicate what, if anything, about the following expressions can not be computed statically. Be precise.

4a `a[3]`

The value of the variable.

4b `&a[3]`

Nothing.

4c `b[3]`

The address and value of the variable.

5 **(6 marks)** **Instance Variables.** In the context of the following C declarations:

CPSC 213, Winter 2014, Term 1 — Sample Midterm (Winter 2013 Term 2)
Solution

Date: October 2014; Instructor: Mike Feeley

1 (8 marks) **Memory and Numbers**. Consider the following C code with global variables `a` and `b`.

```
int  a[2];
int* b;

void foo() {
    b    = a;
    a[0] = 1;
    b[1] = 2;
}

void checkGlobalVariableAddressesAndSizes() {
    if ((&a==0x2000) && (&b==0x3000) && sizeof(int)==4  && sizeof(int*)==4)
        printf ("OKAY");
}
```

When `checkGlobalVariableAddressesAndSizes()` executes it prints "`OKAY`". Recall that `sizeof(t)` returns the number of bytes in variables of type `t`.

Describe what you know about the content of memory following the execution of `foo()` on a **Little Endian** processor. List only memory locations whose address and value you know. List each byte of memory on a separate line using the form: "`byte_address: byte_value`". List all numbers in hex.

```
0x2000: 0x01
0x2001: 0x00
0x2002: 0x00
0x2003: 0x00
0x2004: 0x02
0x2005: 0x00
0x2006: 0x00
0x2007: 0x00
0x3000: 0x00
0x3001: 0x20
0x3002: 0x00
0x3003: 0x00
```

2 (4 marks) **Pointers in C**. Consider the following declaration of C global variables.

```
int  a[10] = {0,1,2,3,4,5,6,7,8,9};  // i.e., a[i] = i
int* b     = &a[6];
```

And the following expression that accesses them found in some procedure.

```
*(a + ((&a[9] + 5) - b))
```

When this expression is evaluated at runtime does it cause an error? If not, what value does it compute?

Briefly explain your answer as follows: if there is a runtime error, clearly explain what causes it; if there is not an error, show at least 3 lines of work with intermediate values to explain your answer, step by step.

It executes without error and computes the value 8.

```
*(a + ((&a[9] + 5) - b)) == *(a + (&a[14] - &a[6]))
                         == *(a + 8)
                         == a[8]
                         == 8
```

3 (4 marks) **Global Arrays**. Consider the following C global variable declarations.

```
int  a[10];
int* b;
```

CPSC 213, Winter 2013, Term 2 — Final Exam **Solution**

Date: April 14, 2014; Instructor: Mike Feeley (Solutions written by Maher Kader)

1 (5 marks) **Variables and Memory.** Consider the following C code with three global variables, `a`, `b`, and `c`, that are stored at addresses `0x1000`, `0x2000`, `0x3000`, respectively.

```
void foo() {
    a[0] = 1;
    b[0] = 2;
    c = a;
    c[0] = 3;
    c = b;
    *c = 4;
}
```

```
int a[1];
int b[1];
int* c;
```

Describe what you know about the content of memory following the execution of `foo()` on a 32-bit **Big Endian** processor. List only memory locations whose address and value you know. List each byte of memory separately using the form "`byte_address: byte_value`". List all numbers in hex.

0x1000: 0x00
0x1001: 0x00
0x1002: 0x00
0x1003: 0x03

0x2000: 0x00
0x2001: 0x00
0x2002: 0x00
0x2003: 0x04

0x3000: 0x00
0x3001: 0x00
0x3002: 0x20
0x3003: 0x00

You can approach this question by tracking how memory changes line by line. You can do this in your head on the exam instead of writing everything out.

Upon entering `foo()`:
0x1000: 0x00000000 `(int a[1])`
0x2000: 0x00000000 `(int b[1])`
0x3000: 0x00000000 `(int* c)`
`a[0] = 1;`
0x1000: 0x00000001
0x2000: 0x00000000
0x3000: 0x00000000
`b[0] = 2;`
0x1000: 0x00000001
0x2000: 0x00000002
0x3000: 0x00000000
`c = a;`
0x1000: 0x00000001
0x2000: 0x00000002
0x3000: 0x00001000
`c[0] = 3;`
0x1000: 0x00000003
0x2000: 0x00000002
0x3000: 0x00001000
`c = b;`
0x1000: 0x00000003
0x2000: 0x00000002
0x3000: 0x00002000
`*c = 4;`
0x1000: 0x00000003
0x2000: 0x00000004
0x3000: 0x00002000

Then translate everything to the byte-by-byte format as requested in the question. Remember it's Big Endian in this case! (what would the answer be if it was Little Endian instead?)

CPSC 213, Winter 2010, Term 1 — Midterm Exam **Solution**

Date: October 27, 2010; Instructor: Tamara Munzner

1 (2 marks) **Memory Alignment.** Consider the memory address 0x92. List all power-of-two sizes for which aligned memory access is possible and carefully justify your answer.

2. The lowest bit is zero, but the 2nd lowest bit is 1 so it is aligned only for 2-byte access. Alternately: the decimal equivalent 146 divides evenly by 2, but not by 4 or any larger power of 2.

2 marks: 1 for correct answer, 1 for correct justification. Thus mark of 1/2 if forgot to convert from hex and did computation for decimal 92 getting answer 2 and 4, or had correct logic but computational mistake.

2 (8 marks) **Pointer Arithmetic.** Consider the following lines of C code. For the assignments to `i` , `j`, `k`, and `m` say (a) whether the code generates an error and why or (b) what value the variables have after the code executes. If one line generates an error but a later one does not, give the value of the later ones. Show your work.

```
int a[10] = { 9,8,7,6,5,4,3,2,1,0 };
int i = *(a+4);
int j = &a[3] - &a[1];
int k = *(a+*(a+6));
int m = *(&a[5]-a);
```

2a `i`:

No error. Value of i is 5.

```
int i = *(a+4);
int i = *(&a[4]);
int i = a[4];
int i = 5;
```

2b `j`:

No error. Value of j is 2.

```
int j = &a[3] - &a[1];
int j = 2;
```

2c `k`:

No error. Value of i is 5.

```
int k = *(a+*(a+6);
int k = *(a+*(&a[6]));
int k = *(a+ 3);
int k = *(&a[3]);
int k = 6;
```

2d `m`:

This attempt to de-reference the address `((&a[5]-a) == (&a[5]-&a[0]) == 5`, statement will generate an error (from the compiler or at runtime). The address 5 is not unaligned. It is also a protected location on most architectures.

3 (5 marks) **Dynamic Allocation.** A *dangling pointer* exits when a program retains a reference to dynamically allocated memory after it has been freed/reclaimed. A *memory leak* exists when a program fails to free/reclaim dynamically allocated memory that it is no longer needed by the program. Answer true or false to the following questions:

- Dangling pointers can occur in C. True
- Dangling pointers can occur in Java. False
- Memory leaks can occur in C. True
- Memory leaks can occur in Java. True