COMP201 Fall 2020 Midterm Exam
Duration: **90 minutes**

Student ID: |___|___|___|___|___|___|___|___|___|

Lab Section: |___|___|

First Name(s): _____ **Last Name:** _____

---

*Do **not** turn this page until you are told to do so.*
In the meantime, please read the instructions below.

Good Luck!

This exam contains **8 multi-part questions** and you have **90 minutes** to earn 90 marks.

- When the exam begins, please write your student ID, name and lecture section on top of this page, and sign the academic integrity code given below.
- Check that the exam booklet contains a total of **16 pages**, including this one.
- This exam is an **open book and notes exam**.
- Show all work, as partial credit will be given since you will be graded not only on the correctness and efficiency of your answers, but also on your clarity that you express it. Be neat.

HC:	_____ / 4
# 1:	_____ / 9
# 2:	_____ / 13
# 3:	_____ / 8
# 4:	_____ / 12
# 5:	_____ / 13
# 6:	_____ / 14
# 7:	_____ / 8
# 8:	_____ / 9
TOTAL:	_____ / 90

I hereby declare that I have completed this exam individually, without support from anyone else.

I hereby accept that only the below listed sources are approved to be used during this open-source quiz:
(i) Coursebook,
(ii) All material that is made available to students via Blackboard for this course, and
(iii) Notes taken by me during lectures.

I have not used, accessed or taken any unpermitted information from any other source. Hence, all effort belongs to me.

Signature: _____

Question 1A. Integer Representations, Bitwise Operations

[9 POINTS] Assume that you are working on a 10-bit machine, in which

- Two's complement arithmetic is used for signed integers,
- `short` integers are encoded using 5 bits.
- Sign extension is carried out when a `short` is casted to an `int`.
- All shift operations are arithmetic.

Consider the following variable declarations Fill in the empty boxes in the table below. The first two rows have already been answered for you.

```
short sa = -7;
int b = 2*sa;
int a = -32;
short sb = (short) a;
unsigned short ua = a+28;
```

Note: You need not fill in entries marked with "–". TMax denotes the largest positive two's complement number and TMin denotes the minimum negative two's complement number.

Expression	Decimal Representation	Hexadecimal Representation
Zero	0	0x000
(short) 1	1	0x01
sa	-7	0x19
b	-14	0x3f2
sb	0	0x00
ua	28	0x1c
a >> 2	-8	0x3f8
ua >> 2	7	0x07
b << 3	-112	0x390
Tmax	511	0x1ff
Tmax - Tmin	-1	0x3ff

Question 2A. Representing Floating Point Numbers

(a) [7 POINTS] Assume that you are working on a machine that stores floating point numbers based on two different 6-bit encodings similar to the IEEE-754 format.

Format FP6A:

- The sign bit is in the most significant bit
- The next 3 bits are the exponent, with a bias of 3.
- The last 2 bits are the fraction.

Format FP6B:

- The sign bit is in the most significant bit
- The next 2 bits are the exponent, with a bias of 1.
- The last 3 bits are the fraction.

Recall that these formats will also use normalized and denormalized modes and represent infinity and not a number (NaN) in a specific way. Based on this, please fill in the boxes below.

Value	Format FP6A	Format FP6B
Negative zero	`1 000 00`	`1 00 000`
1	`0 011 00`	`0 01 000`
3	`0 100 10`	`0 10 100`
6/8	`0 010 10`	`0 00 110`
-14/8	`1 011 11`	`1 01 110`
Smallest negative normalized number	`1 001 00`	`1 01 000`
Largest positive denormalized number	`0 000 11`	`0 00 111`
Positive infinity	`0 111 00`	`0 11 000`

(b) [6 POINTS] The rise of deep learning has stimulated interest in new floating point formats since these big machine learning models do not necessarily need high precision. One such format, referred to as bloat16 (BF16) is now becoming hugely popular for deep learning research. This format has 16 bits, which is similar to the IEEE754 half-precision (FP16) format, but has the same number of exponent bits as the IEEE754 single precision (FP32) format.

Format	Total number of bits	Exponent bits	Fraction bits
FP32	32	8	23
FP16	16	5	10
BF16	16	8	7

Name one of the advantages of using BF16 over FP16. Hint: You can think of the range of possible numbers and conversion between FP32.

- Conversion between FP32 and BF16 much simpler. To convert FP32 to BF16, just cut off the last 16 bits. To convert from BF16 to FP32, just pad with zeros (except for some edge cases regarding denormalized numbers)
- The dynamic range is mostly defined based on number of exponent bits. Hence, the range is larger for BF16.

Question 3. C-Strings

In the company you are employed, your boss asked you to implement the `strtok()` function in the standard C library that you used in your second assignment. Recall that `strtok()` splits a given string into a series of tokens using a set of delimiters provided as a second input (e.g. `" \t\r\n"`). The function returns a pointer to the first token found in the string, and if there are no tokens left, it simply returns `NULL`. In doing so, `strtok()` basically identifies both the beginning and the end of the token so that the next the function is called, it continues the search from the previous ending position. `strtok()` utilizes a static variable to store the pointer to the beginning of the next token (the end of the extracted token), and let's call this pointer `pNextToken`. The implementation of `strtok()` is given below:

```
char * strtok (char *str, const char *delim) {
    /* initialize */
    if (!str)
       str = pNextToken;

    /* find start of token in str */
    str += strspn(str, delims );
    if (*str == '\0')
        return NULL;

    /* find end of token in text */
    pNextToken = str + strcspn(str, delim);

    /* insert null-terminator at end */
    if (*pNextToken != '\0')
        *pNextToken++ = '\0';

    return str;
}
```

As you may notice, the above implementation uses the two common string functions `strspn()` and `strcspn()` and you have to implement these functions by yourself, too. To make things a little bit simpler, you first implemented an helper function named `strwhere()`, declared as:

```
int strwhere(const char *str , const char ch);
```

This function can be used to locate a character in a given string.

(a) [4 POINTS] Using the above helper function, implement the `strspn()` function by writing down the missing pieces in the code below:

```
unsigned int strspn(const char * str, const char * delims) {
    unsigned int i ;
    for (i = 0; str[i] != '\0' && strwhere(delims, str[i]) > -1; i++);
    return i ;
}
```

(b) [4 POINTS] Similarly, using the helper function `strwhere()`, implement the `strcspn()` function which computes the index of the first delimiter character in our string.

```
unsigned int strcspn(const char * str, const char * delims) {
    unsigned int i ;
    for (i = 0; str[i] != '\0' && strwhere(delims, str[i]) == -1; i++);
    return i ;
}
```

Question 4A. Pointers and Working with Dynamic Memory

In this question, you will try to debug a set of programs and identify the problems associated with them. Here are some common errors and mistakes that you can refer to in your answers:

- *storage used after free,*
- *allocation freed repeatedly,*
- *insufficient space for a dynamically allocated variable,*
- *freeing unallocated storage,*
- *freeing of the stack space,*
- *memory leakage,*
- *assignment of incompatible types,*
- *returning (directly or via an argument) of a pointer to a local variable,*
- *dereference of wrong type,*
- *dereference of uninitialized or invalid pointer,*
- *incorrect use of pointer arithmetic,*
- *array index out of bounds*

(a) [6 POINTS] What is the problem with the following code? Justify your answer, otherwise your answer will not be counted. Note that there could be more than one error.

```
void myfunc(int *arr) {
    int *p_arr = (int*) malloc(2*sizeof(int));
    p_arr[0] = 42; p_arr[1] = 24;
    arr = p_arr;
}
int main(int argc, char *argv[]) {
    int *arr = NULL;
    myfunc(arr);
    printf("arr[0] = %d\n arr[1] = %d", arr[0], arr[1]);
    free(arr);
    return 0;
}
```

dereference of uninitialized or invalid pointer: arr in main is still NULL
freeing unallocated storage: free(arr)

(b) [6 POINTS] What is the problem with the following code? Justify your answer, otherwise your answer will not be counted. Note that there could be more than one error.

```
int main(int argc, char *argv[]) {
    if (argc!=3) {printf("wrong number of arguments\n"); return 1;
    char *param1 = *argv[1];
    char *param2 = *argv[2];
    char *ptr;
    ptr = (char *) malloc(strlen(param1)+strlen(param2)+1);
    while ((*ptr++ = *param1++) != 0)
         ;
    strcat(ptr+strlen(param1)+1, param2);
    printf("%s\n", ptr);
    ptr = NULL;
    return 0;
}
```

Dereference of invalid pointer: strcat could not find end of dest
memory leakage: ptr = NULL

Question 5A. Generics, Function Pointers

In this question, you are asked to implement a generic function `findmin` which can be used to find the minimum element in an array of arbitrary data. Toward this aim, assume that you have defined the following function declaration:

```
void* findmin(void* arr, int len, int elem_size_bytes, int (*cmp)(void* a, void* b))
```

where `arr` is the array to be search, `len` is the length of the array, `elem_size_bytes` is the size (in bytes) of each array element, and `cmp` is the comparison function for a particular comparable type array.

(a) [3 POINTS] Complete the function body of `findmin`.

```
void* findmin(void* arr, int length, int elem_size_bytes, int (*cmp)(void* a, void* b))
{
    void* min = arr;
    for(int i = 1; i < len; i++)
    {
        void* elem = (char*) arr + i * elem_size_bytes;
  if(cmp(elem, min) < 0) min = elem; (char*)(arr+elem_size_bytes*i)
    }
    return min;
}
```

(b) [2 POINTS] Suppose that the findmin function is to be used with an array of ints. Write the intcmp function used to compare two elements. intcmp returns 1,0,-1 based on comparison (be- havior similar to standard compare functions)

```
int double_cmp(void* x, void* y) {
    double a = *(double *) x;
    double b = *(double *) y;
    return a - b;
}
```

(c) [2 POINTS] Suppose that the function findmin is to be used with an array of strings. Write the strcmp function used to compare two string elements. intcmp returns 1,0,-1 based on comparison (behavior similar to standard compare functions)

```
int  str_cmp(void* x, void* y) {
    char *a = *(char **)x;
    char *b = *(char **)y;
    return strcmp(a, b);
}
```

(d) [4 POINTS] Complete the following `main` function using ~~`intcmp`~~ `double_cmp` and `str_cmp`.

```
int main() {
   double A[] = {3.2, -12.0, 1.7, 18, 90, 105};
   printf("The min. element in A is ~~%d~~ %ld \n",
                *(double*) findmin(A, 6, sizeof(double), double_cmp));
 // Due to the typo *(int*) findmin(A, 6, sizeof(double), intcmp)); also correct
   char* B[] = {"luke", "leia", "han"};
   printf("%s is the first in B based on alphabetical order\n",
                *(char **)findmin(B, 3, sizeof(char*), str_cmp));
   return 0;
}
```

(e) [2 POINTS] What will be the output?

```
The min. element in A is -12.0
han is the first in B based on alphabetical order
```

Question 6A. Disassembly

Please answer the questions below considering the following fragment of the assembly code generated by running the `objdump -d` command.

```
  4004ed: b9 00 00 00 00            mov    $0x0,%ecx
  4004f2: eb 16                     jmp    40050a <myfunc+0x1d>
```

(a) [2 POINTS] The `objdump` has not included the corresponding suffix to the `mov` instruction to explicitly state the size information. That being said, you can still identify the size of the destination operand. What is the size and how do you tell?

32 bits/4 bytes

(a) [2 POINTS] What is the value of `%rip` register as the CPU executes the `mov` instruction in the sequence of two instructions?

`4004f2`

(b) [10 POINTS] Let's now look at the entire function from which we took these two instructions.

```
00000000004004ed <myfunc>:
  4004ed: b9 00 00 00 00            mov    $0x0,%ecx
  4004f2: eb 16                     jmp    40050a <myfunc+0x1d>
  4004f4: 85 ff                     test   %edi,%edi
  4004f6: 74 0b                     je     400503 <myfunc+0x16>
  4004f8: ba 00 00 00 00            mov    $0x0,%edx
  4004fd: f7 f7                     div    %edi
  4004ff: 01 c1                     add    %eax,%ecx
  400501: eb 07                     jmp    40050a <myfunc+0x1d>
  400503: 8d 04 40                  lea    (%rax,%rax,2),%eax
  400506: 01 f8                     add    %edi,%eax
  400508: 01 c1                     add    %eax,%ecx
  40050a: 8b 06                     mov    (%rsi),%eax
  40050c: 89 c2                     mov    %eax,%edx
  40050e: 0f af d7                  imul   %edi,%edx
  400511: 39 ca                     cmp    %ecx,%edx
  400513: 77 df                     ja     4004f4 <myfunc+0x7>
  400515: 89 c8                     mov    %ecx,%eax
  400517: c3                        retq
```

Fill in the blanks of the corresponding C function:

```
unsigned int myfunc(unsigned int a, unsigned int *b) {
   unsigned int result = 0;
   while (result < a*(*b)) // or and equivalent expression
   if (a>0)
         result += *b/a;
   else
         result += a+3*(*b);
   return result;
}
```

Question 7A. Makefiles

[1 POINTS] In your free time you are working on a large project involving multiple source files, which have the following `#include` dependencies:

```
***************          ***************          ***************
* beans.h     *          * coffee.h    *          * milk.h      *
***************          ***************          ***************
#ifndef BEANS_H          #ifndef COFFEE_H         #ifndef MILK_H
#define BEANS_H          #define COFFEE_H         #define MILK_H
                         #include "beans.h"
...                      ...
#endif                   #endif

***************          ***************          ***************
* beans.c     *          * coffee.c    *          * main.h      *
***************          ***************          ***************
#include "beans.h"       #include "coffee.h"      #include "coffee.h"
...                      #include "milk.h"

                                                  int main(int argc, char *argv[])
                                                  {
                                                      ...
                                                  }
```

You can always compile your project using `gcc -Wall -g -std=c11 -o main *.c` to create an executable program. But as you learned in COMP 201 that this is not a good idea as it is not necessary to recompile and relink the files each time you make a change in just one of the files.

Think about the dependencies between the source files (`.c`), the compiled object (`.o`) files, and the final executable file (`main`) that is created by linking the object files, and write down the `Makefile` that you can use for the project.

```
CC = gcc
CFLAGS = -Wall -g -std=c11

main: beans.o coffee.o main.o
    $(CC) $(CFLAGS) –o main beans.o coffee.o main.o
beans.o: beans.c beans.h
    $(CC) $(CFLAGS) –c beans.c
coffee.o: coffee.c coffee.h beans.h milk.h
    $(CC) $(CFLAGS) –c coffee.c
main.o: main.c beans.h coffee.h
    $(CC) $(CFLAGS) main.c

Each target is of 2 points, and you lose 1 point if you don’t consider the dependencies
correctly.
```

Question 8. Short Answer Questions

(a) [3 POINTS] What is the output of the following C program?

```
#include<stdio.h>
int main() {
    char string[]="MAY4TH";
    char* ptr=string;
    printf("%c",*(ptr+1)+2);
    *ptr=*ptr + 3;
    ptr = ptr + 4;
    ptr--;
    printf("%c",*ptr-1);
    ptr++;
    printf("%c",*ptr);
    ptr=string;
    printf("%c",*ptr-1);
}
```

C3TO

(b) [3 POINTS] What is the output of the following C program?

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

#define SIZE 50

int main(void) {
  char str[SIZE] = "EARETEAREAREAT";
  char *ptr;
  int count = 0;

  ptr = strstr(str, "TEAR");
  printf("%s ", ptr);

  ptr = strstr(ptr, "ARE");
  printf("%s ", ptr);
  while(ptr!=NULL) {
    count++;
    ptr = strstr(ptr+1, "EAR");
  }
  printf("%d ", count);
  return 0;
}
```

TEAREAREAT AREAREAT 2

(c) [3 POINTS] What is the output of the following C program?

```
#include <stdio.h>
#define SQUARE(a) a * a
#define TRIPLE(b) 3 * b

int main() {
    int a = 2;
    int b = 3;

    int x = TRIPLE(a);
    int y = SQUARE(a+b);
    int z = TRIPLE(1+a+b);

    printf("%d %d %d\n", x, y, z); return 0;
}
```

6 11 8