COMP201 Final 2021 Final Exam
Duration: **120 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 **7 multi-part questions** and you have **120 minutes** to earn 100 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 **11 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:	_____/ 1
# 1:	_____/ 16
# 2:	_____/ 15
# 3:	_____/ 12
# 4:	_____/ 12
# 5:	_____/ 12
# 6:	_____/ 18
# 7:	_____/ 14
TOTAL:	_____/ 100

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. Calling Functions in Assembly [16 POINTS]

In the following, you are provided some assembly code generated by compiling `myprog.c` using gcc compiler.

```
                                        // myprog.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void mystery(char *str, int i, int n) {
  if (i >= n)
    return;
  str[i] += 'a'-'A';
  str++;
  mystery(str, i+1, n-1);
}

int main(int argc, char *argv[]) {
  char *str = argv[1];
  mystery(str, 1, 10);
  printf("%s", str);
  return 0;
}
```

```
mystery:
  cmpl %edx, %esi
  jge .L5
  subq $8, %rsp
  movslq %esi, %rax
  addb $32, (%rdi,%rax)
  addq $1, %rdi
  subl $1, %edx
  addl $1, %esi
  callq mystery
  addq $8, %rsp
.L5:
  rep retq
.LC0:
  .string "%s"
main:
  pushq %rbx
  movq 8(%rsi), %rbx
  movl $10, %edx
  movl $1, %esi
  movq %rbx, %rdi
  callq mystery
  movq %rbx, %rsi
  movl $.LC0, %edi
  movl $0, %eax
  callq printf
  movl $0, %eax
  popq %rbx
  retq
```

(a) [1 POINTS] What will the program output when it is called with `./myprog HAPPYNEWYEAR` from the console?

HaPpYnEwYeAR

(b) [12 POINTS] Assume that the initial values of the stack pointer `%rsp` and `%rsi` have the values `0xFFF380` and `0x400540` just before the first instruction of `main` is executed. Fill in this table to give the contents of registers immediately before the first instruction of `mystery` is executed.

	%rdi	%esi	%edx	%rsp
First call to `mystery`	0x400548	0x1 / $1 / 1	0xA / $10 / 10	0xFFF370
Second call to `mystery`	0x400549	0x2 / $2 / 2	0x9 / $9 / 9	0xFFF360
Third call to `mystery`	0x40054a	0x3 / $3 / 3	0x8 / $8 / 8	0xFFF350

(c) [1 POINTS] Does `main` perform something to save and restore any caller saved registers? Yes/No

No

(d) [1 POINTS] Does `main` perform something to save and restore any callee saved registers? Yes/No

Yes

(e) [1 POINTS] Does `main` obey the x86-64/Linux calling conventions? Yes/No

Yes

Question 1B. Calling Functions in Assembly [16 POINTS]

In the following, you are provided some assembly code generated by compiling `myprog.c` using gcc compiler.

```
                                        // myprog.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void mystery(char *str, int i, int n) {
  if (i >= n)
    return;
  str[i] += 'a'-'A';
  str++;
  mystery(str, i+1, n-1);
}

int main(int argc, char *argv[]) {
  char *str = argv[1];
  mystery(str, 2, 11);
  printf("%s", str);
  return 0;
}
```

```
mystery:
  cmpl %edx, %esi
  jge .L5
  subq $8, %rsp
  movslq %esi, %rax
  addb $32, (%rdi,%rax)
  addq $1, %rdi
  subl $1, %edx
  addl $1, %esi
  callq mystery
  addq $8, %rsp
.L5:
  rep retq
.LC0:
  .string "%s"
main:
  pushq %rbx
  movq 8(%rsi), %rbx
  movl $11, %edx
  movl $2, %esi
  movq %rbx, %rdi
  callq mystery
  movq %rbx, %rsi
  movl $.LC0, %edi
  movl $0, %eax
  callq printf
  movl $0, %eax
  popq %rbx
  retq
```

(a) [1 POINTS] What will the program output when it is called with `./myprog HAPPYNEWYEAR` from the console?

HApPyNeWyEaR

(b) [12 POINTS] Assume that the initial values of the stack pointer `%rsp` and `%rsi` have the values `0xFFF388` and `0x400548` just before the first instruction of `main` is executed. Fill in this table to give the contents of registers immediately before the first instruction of `mystery` is executed.

	%rdi	%esi	%edx	%rsp
First call to `mystery`	0x400550	0x2 / $2 /2	0xB / $11 / 11	0xFFF378
Second call to `mystery`	0x400551	0x3 / $3 /3	0xA / $10 / 10	0xFFF368
Third call to `mystery`	0x400552	0x4 / $4 / 4	0x9 / $9 /9	0xFFF358

(c) [1 POINTS] Does `mystery` perform something to save and restore any caller saved registers? Yes/No

No

(d) [1 POINTS] Does `mystery` perform something to save and restore any callee saved registers? Yes/No

No

(e) [1 POINTS] Does `main` obey the x86-64/Linux calling conventions? Yes/No

Yes

Question 2A. Data and Stack Frames [15 POINTS]

(a) [6 POINTS] Consider the following C program, in which `A`, `B` and `C` are constants expressed with `#define` directives, and the corresponding assembly code is generated by the gcc compiler. By inspecting the assembly code, try to find the values of `B` and `C`.

```
short arr1[A][B];
int   arr2[B][C];

void update_values(int x, int y, int z) {
    arr2[x][y] = 3*arr1[y][z];
}
```

```
update_values:
  movslq %edx, %rdx
  movslq %esi, %rsi
  leaq (%rsi,%rsi,2), %rax
  addq %rax, %rdx
  movswl arr1(%rdx,%rdx), %eax
  leal (%rax,%rax,2), %edx
  movslq %edi, %rdi
  leaq (%rsi,%rdi,4), %rax
  movl %edx, arr2(,%rax,4)
  ret
```

B = 3
C = 4

(a) [9 POINTS] Determine the offset of each field and the total size (in bytes) of the structure given below, considering the alignment requirements of a 64-bit machine.

Structure	a	b	c	d	e	f	g	Total

`struct mystruct {`
`  int a;`
`  short b;`
`  char c[14];`
`  double d;`
`  short e;`
`  struct mystruct *f`
`  float g;`
`}`
	0	**4**	**6**	**24**	**32**	**40**	**48**	**56**

What is the size (in bytes) of the structure if the fields in part (b) are rearranged to have minimum wasted space?
48

Question 2B. Data and Stack Frames [15 POINTS]

(b) [6 POINTS] Consider the following C program, in which `A`, `B` and `C` are constants expressed with `#define` directives, and the corresponding assembly code is generated by the gcc compiler. By inspecting the assembly code, try to find the values of `B` and `C`.

```
short arr1[A][B];
int   arr2[B][C];

void update_values(int x, int y, int z) {
    arr2[x][y] = 3*arr1[y][z];
}
```

```
update_values:
  movslq %edx, %rdx
  movslq %esi, %rsi
  leaq (%rdx,%rsi,4), %rax
  movswl arr1(%rax,%rax), %eax
  leal (%rax,%rax,2), %ecx
  movslq %edi, %rdi
  leaq (%rdi,%rdi,2), %rdx
  leaq (%rdx,%rdx), %rax
  addq %rax, %rsi
  movl %ecx, arr2(,%rsi,4)
  ret
```

B = 4
C = 6

(c) [9 POINTS] Determine the offset of each field and the total size (in bytes) of the structure given below, considering the alignment requirements of a 64-bit machine.

Structure	a	b	c	d	e	f	g	Total

`struct mystruct {`
`  short a;`
`  float b;`
`  double c;`
`  char d[9];`
`  struct mystruct *e;`
`  short f;`
`  int g;`
`}`
	0	**4**	**8**	**16**	**32**	**40**	**44**	**48**

What is the size (in bytes) of the structure if the fields in part (b) are rearranged to have minimum wasted space?
40

Question 3. Buffer Overflow [12 POINTS]

Thomas A. Anderson, also known as with the alias “Neo”, is a highly skilled programmer who works at MetaCortex software company in his daily job. But at night he is obsessed with hacking the Matrix. Finally, he located the source code of a test program (`test.c`) which he thinks it is the key to unlock accessing the Matrix code. This function first retrieves the stored passcode and then compares the input string to check whether they match with each other or not.

```
#include <stdio.h>
...
void enter_matrix() {
    char passcode[8];
    char input[8];
    retrieve_passcode(passcode);
    gets(input);
    if (strcmp(input, passcode) == 0)
        found_the_rabbit();
}
int main(void) {
    printf("Password: ");
    enter_matrix();
    return 0;
}
```

Below, you are provided some part of the assembly code generated by compiling `test.c` using gcc compiler:

```
00000000004006a1 <found_the_rabbit>:
  4006a1:       55                      push   %rbp
  ...
00000000004006b1 <enter_matrix>:
  4006b1:       55                      push   %rbp
  4006b2:       48 89 e5                mov    %rsp,%rbp
  4006b5:       48 83 ec 20             sub    $0x20,%rsp
  4006b9:       48 8d 45 f0             lea    -0x10(%rbp),%rax
  4006bd:       48 89 c7                mov    %rax,%rdi
  4006c0:       e8 b8 ff ff ff          callq  40067d <retrieve_passcode>
  4006c5:       48 8d 45 e0             lea    -0x20(%rbp),%rax
  4006c9:       48 89 c7                mov    %rax,%rdi
  4006cc:       e8 9f fe ff ff          callq  400570 <gets@plt>
  4006d1:       48 8d 55 f0             lea    -0x10(%rbp),%rdx
  4006d5:       48 8d 45 e0             lea    -0x20(%rbp),%rax
  4006d9:       48 89 d6                mov    %rdx,%rsi
  4006dc:       48 89 c7                mov    %rax,%rdi
  4006df:       e8 6c fe ff ff          callq  400550 <strcmp@plt>
  4006e4:       85 c0                   test   %eax,%eax
  4006e6:       75 0a                   jne    4006f2 <enter_matrix+0x41>
  4006e8:       b8 00 00 00 00          mov    $0x0,%eax
  4006ed:       e8 af ff ff ff          callq  4006a1 <found_the_rabbit>
  4006f2:       c9                      leaveq
  4006f3:       c3                      retq
00000000004006f4 <main>:
  4006f4:       55                      push   %rbp
  4006f5:       48 89 e5                mov    %rsp,%rbp
  4006f8:       bf d9 07 40 00          mov    $0x4007d9,%edi
  4006fd:       b8 00 00 00 00          mov    $0x0,%eax
  400702:       e8 29 fe ff ff          callq  400530 <printf@plt>
  400707:       b8 00 00 00 00          mov    $0x0,%eax
  40070c:       e8 a0 ff ff ff          callq  4006b1 <enter_matrix>
  400711:       b8 00 00 00 00          mov    $0x0,%eax
  400716:       5d                      pop    %rbp
  400717:       c3                      retq
```

(a) [4 POINTS] Which line is the root cause of buffer overflow attack here.

```
gets(input);
```

(b) [4 POINTS] When the buffer overflow occurs, which one of the following addresses gets overwritten on the stack?

1. 0x00000000004006a1
2. 0x00000000004006b1
3. 0x00000000004006d1
4. None of the above

The gets function can write to the stack locations where both the passcode and input are stored, or you can directly try to change the next instruction address at `main` (`0x0000000000400711`)

(c) [4 POINTS] Assume that you are working at the directory in which the `test` program exists and that you execute the following commands from the terminal:

```
./hex2raw < exploit | ./test
```

Write down the content of `input` (in hexadecimal digits) so that it performs the attack that will perform a function call to `found_the_rabbit` function.

Note: In your answer, if you want to repeat a specific random byte sequence, you can use the expression `(n)` to denote consecutive bytes of size n bytes to refer to that byte sequence.

`(15) 00 (15) 00` (Two identical 16 byte strings – including the NULL terminating symbol)

Also true: Directly write onto the next instruction address in `main` pushed to the stack when `call enter_matrix`

`00(40) a1 06 40 00 00 00 00 00`

Question 4. Locality [12 POINTS]

Consider the following C functions which calculate the pairwise squared Euclidean distance matrix consisting of n rows where each row corresponds to the coordinates of an n-dimensional point.

For example, for the matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix}$, the distance matrix is calculated as $D = \begin{bmatrix} 0 & 27 & 121 \\ 27 & 0 & 34 \\ 121 & 34 & 0 \end{bmatrix}$.

```
void pdist1(int N, int A[N][N], int D[N][N]) {
  int i, j, k;
  for (i = 0; i < N; i++) {
      for (j = 0; j < N; j++) {
          int temp = 0;
          for (k = 0; k < N; k++) {
              temp += (int)pow(A[i][k] - A[j][k], 2);
          }
          D[i][j] = temp;
      }
  }
}

void pdist2(int N, int A[N][N], int D[N][N]) {
  int i, j, k;
  for (j = 0; j < N; j++) {
      for (i = 0; i < N; i++) {
          int temp = 0;
          for (k = 0; k < N; k++) {
              temp += (int)pow(A[i][k] - A[j][k], 2);
          }
          D[i][j] = temp;
      }
  }
 }
```

(a) [3 POINTS] The functions `pdist1` and `pdist2` both accomplish the same task. Which one is more cache friendly?

1. `pdist1` is more cache friendly than `pdist2`
2. `pdist2` is more cache friendly than `pdist1`
3. **They perform similar in term of cache utilization.**

Here one needs to think about the worst case where the size of the matrix is very huge, hence as we discussed in the class, you only need to focus on the inner most loop.

(b) [3 POINTS] The runtime performances of `pdist1` and `pdist2` can be improved with loop unrolling [**T**/F]

(c) [3 POINTS] The memory writes to local variable `temp` has bad temporal locality [T/**F**]

(d) [3 POINTS] The memory accesses to `A[i][k]` has stride-N reference pattern and thus have good spatial locality [T/**F**] (it has stride-1 pattern)

Question 5A. Cache Memories [12 POINTS]

Suppose that you are working on a system with the following specifications:

- The device is a 24-bit machine, i.e., the physical addresses are 24 bits wide.
- The memory is byte addressable, and memory accesses are to 1-byte words.
- The device contains a 4-way set associative cache of size 4 KB with a block size of 32 bytes.

(a) [2 POINTS] How many bits are reserved for the cache block offset? **5 bits**

(b) [2 POINTS] How many bits are used for the index? **5 bits**

(c) [2 POINTS] How many bits are used for the tag? **14 bits**

Assume that the cache is initially empty when executing the following code:

```
int arr[1024][8];
for (int i=0; i<1024; i+=2) {
  arr[i][0] += arr[i][4];
}
```

(d) [2 POINTS] What is the hit rate? **2/3 (There is 2 read and 1 write operations)**

(e) [2 POINTS] What is the number of hits? **1024**

(f) [2 POINTS] The hit rate improves if one increases the associativity of the cache to 8 while keeping the other parameters the same. [T/**F**]

Question 5B. Cache Memories [12 POINTS]

Suppose that you are working on a system with the following specifications:

- The device is a 16-bit machine, i.e., the physical addresses are 16 bits wide.
- The memory is byte addressable, and memory accesses are to 1-byte words.
- The device contains a 2-way set associative cache of size 8 KB with a block size of 32 bytes.

(a) [2 POINTS] How many bits are reserved for the cache block offset? **5 bits**

(b) [2 POINTS] How many bits are used for the index? **7 bits**

(c) [2 POINTS] How many bits are used for the tag? **4 bits**

Assume that the cache is initially empty when executing the following code:

```
int arr[512][8];
for (int i=0; i<512; i+=4) {
  arr[i][0] += arr[i][4];
}
```

(d) [2 POINTS] What is the hit rate? **2/3 (There are 2 reads and 1 write, only the first is miss)**

(e) [2 POINTS] What is the number of hits? **256**

(f) [2 POINTS] The hit rate improves if one increases the associativity of the cache to 8 while keeping the other parameters the same. [T/**F**]

Question 6. Assembly and Linking [18 POINTS]

(a) [12 POINTS] Fill in the missing parts of the C functions based on the corresponding Assembly code generated by gcc. Hint: The ASCII codes of the characters `'A'`, `'C'` and `'Z'` are 65, 67 and 90, respectively.

myfile1.c

```
#include <string.h>

extern char c;

int strfun(char *str) {
   const int x = 32;
   char * ret;
   ret = strchr(str, c);
   if (ret[0]>='A' && ret[0]<='Z') {
   // *ret is also true
       ret[0] = ret[0] & ~x;
       return 1;
   }

   return 0;
}
```

myfile1.s

```
strfun:
  subq $8, %rsp
  movsbl c(%rip), %esi
  call strchr
  movzbl (%rax), %edx
  leal -65(%rdx), %ecx
  cmpb $25, %cl
  ja .L3
  andl $-33, %edx
  movb %dl, (%rax)
  movl $1, %eax
  jmp .L2
.L3:
  movl $0, %eax
.L2:
  addq $8, %rsp
  ret
```

myfile2.c

```
char c = 'C';

int strfun(char *);

typedef struct mystr {
  struct mystr *a;
  char b[10];
  char c;
} mystr;

void strfun2(mystr *s, int b) {
  s->c = 'C';
  while (s) {
    if (strfun(s->b)) {
      s->c--; // or any equivalent expr.
    }
    s = s->a;
  }
  return;
}
```

myfile2.s

```
strfun2:
  pushq %rbx
  movq %rdi, %rbx
  movb $67, 18(%rdi)
  jmp .L2
.L4:
  leaq 8(%rbx), %rdi
  call strfun
  testl %eax, %eax
  je .L3
  movzbl 18(%rbx), %eax
  subl $1, %eax
  movb %al, 18(%rbx)
.L3:
  movq (%rbx), %rbx
.L2:
  testq %rbx, %rbx
  jne .L4
  popq %rbx
  ret
```

(b) [3 POINTS] Fill in the following table to name the strong, weak and external symbols defined in the C files. If there is no such definition just write 'None'. If there are more than one, separate the names with commas.

The strong symbols defined in `myfile1.c`	The weak symbols defined in `myfile1.c`	The external symbols defined in `myfile1.c`	The strong symbols defined in `myfile2.c`	The weak symbols defined in `myfile2.c`	The external symbols defined in `myfile2.c`
`strfun`	`None`	`c`	`strfun2, c`	`None`	`strfun`

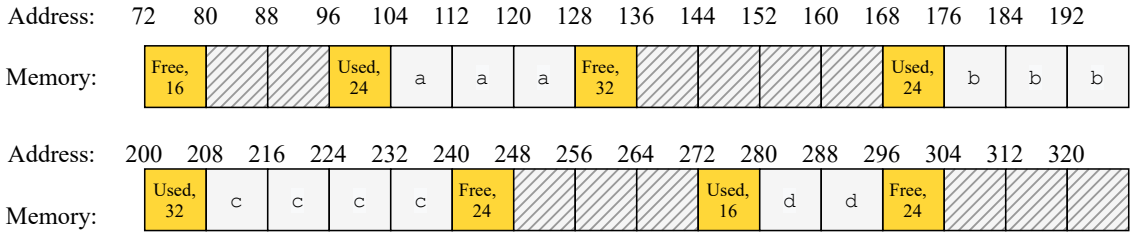
(c) [3 POINTS] Suppose that you have another file (`main.c`) which includes the main function, and two static libraries `libmyfile1.a` and `libmyfile2.a` created from `myfile1.c` and `myfile2.c`, respectively. The correct way to compile the program is `gcc -o a.out main.c -L. -lmyfile2 -lmyfile1` [T/F]

`myfile2` should come before `myfile1` to correctly resolve the symbol `strfun`.

Question 7. Heap Allocators [14 POINTS]

Consider the following state of the heap with *an explicit free-list allocator with coalescing* after performing the following memory allocation operations. The header is of 8 bytes and the allocated chunks of memory should be multiples of 16 bytes, and a best-fit policy is employed. For the sake of simplicity, the memory addresses are given in decimal numbers.

```
void *a = malloc(20);
void *b = malloc(24);
void *c = malloc(28);
void *d = malloc(16);
```



(a) [2 POINTS] What is the amount of internal fragmentation (in bytes) for the current state of the heap for the allocated parts of the heap memory? In your answer, please do count the headers.

8 // I made a typo here, I wanted to say (do not count the headers). So I also consider 40 or 72 as true here!

(b) [2 POINTS] Suppose that the next call to the heap allocator is `malloc(24)`. Specify the address that the malloc function returns?

248 or 304

(c) [2 POINTS] What if the first-fit approach were used instead for the request in part (b)?

136

(d) [2 POINTS]) Suppose that the next call to the heap allocator is `free(a)`. After this operation, can `malloc(60)` request be satisfied? Yes/No

Yes. Because of coalescing, there is enough free space.

(e) [2 POINTS] Does the heap currently have *external* fragmentation?

Yes. There are non-consecutive free blocks.

(f) [2 POINTS] Considering the layout of the heap shown at the top, what is the maximum size one can ask for the heap allocator for `realloc(c,size)`?

64. The explicit free-list allocator does not perform in-place realloc. Hence, the maximum such chunk is of 64 bytes.

(g) [2 POINTS] Suppose that you have switched to using implicit free-list allocator. Does this change increase the throughput for allocation requests or not?

No. Because it is linear in the number of free+allocated chunks, which is worse than explicit free list allocator.