CS 240: Programming in C
Final Exam
Spring 2025

Name:

Username/email: @purdue.edu

Read all instructions before beginning the exam.

- This is a closed book examination. No materials other than those provided for you are allowed.
- You need only a pencil and eraser for this examination. If you use ink, use either black or blue ink. If you use pencil, your writing must be dark and clearly visible.
- This examination contains an amount of material that a well-prepared student should be able to complete in approximately two hours.
- This examination is worth a total of 140 points. Not all questions are worth the same amount. Plan your time accordingly.
- Write legibly. You should try to adhere to the course code standard when writing your solution(s). Egregious violations may result in point deductions. Function header comments are not required.
- You may leave after you have turned in all pages of the examination booklet. You will not be able to change any answers after turning in your examination booklet.
- If you finish your exam with less than 15 minutes remaining, please remain seated until the exam is over. We will call you down by row to turn in your exam.
- Read each question *carefully* and *only do what is specifically asked for* in that problem.
- Some problems require several steps. Show all your work. Partial credit can only be rewarded to work shown.
- Unless otherwise specified, assume all code is executed on a 64-bit, little endian system.
- Do not attempt to look at other students' work. Keep your answers to yourself. Any violation will be considered academic dishonesty.
- Write your username on *EVERY* page where indicated. Any page without a username will receive a zero for the material on that page.
- Read and sign the statement below. Wait for instructions to start the examination before continuing to the next page.

"I signify that the answers provided for this examination are my own and that I have not received any assistance from other students nor given any assistance to other students."

Signature:

- Do not open the examination booklet until instructed. Submission #:

1. (2 points) Write a single command to compile and link the C source files `file1.c` and `file2.c` into an executable named `final_exam`. Include debug symbols, enable all warnings, and adhere to the C99 standard.

```
gcc -o final_exam -g -Wall -std=c99 file1.c file2.c
```

2. (2 points) For each statement below, circle whether it applies to compilation, linking, both, or neither.

Produces machine code from source code.	Compile	Link	Both	Neither
Combines object files into an executable.	Compile	Link	Both	Neither
Resolves dependencies between modules.	Compile	Link	Both	Neither
Executes the program to detect runtime errors.	Compile	Link	Both	Neither

3. (2 points) Given the following code, which of the below statements is true?

```
#include <stdio.h>

int func(float f);

int main() {
  int result = func(3.14);
  printf("%d\n", result);
  return 0;
}
```

   A. The code will result in a compiler error.

   B. The code will result in a linker error.

   C. The code will result in a runtime error.

   D. The code will compile, link, and execute without error.

4. (2 points) Briefly describe what an object file is.

An object file contains compiled machine code that has not yet been linked into an executable.

5. (5 points) Implement the function `write_data()` below. The function should open the file given by the first argument, write the string given by the second argument, and then close the file. If an error occurs, return `ERROR`, otherwise return `SUCCESS`. Include appropriate assertions and follow best practices when opening and closing the file.

```
int write_data(const char *fname, const char *data) {
  assert(fname);
  assert(data);

  FILE *fp = fopen(fname, "w");
  if (!fp) return ERROR;

  int status = SUCCESS;

  int ret = fprintf(fp, "%s", data);
  if (ret < 0 || ret < strlen(data))
    status = ERROR;

  fclose(fp);
  fp = NULL;

  return status;
}
```

6. (2 points) Provide the `fscanf()` statement to read (in this order) a float `f`, a string `s` consisting of only lowercase letters, and an integer `i`, each separated by a semicolon (`;`), from the open file pointer `fp`. Be sure to avoid buffer overflows.

```
void read_stuff(FILE *fp) {
  float f;
  char s[7236];
  int i;

  /* fscanf() call */
}
```

```
fscanf(fp, "%f;%7235[a-z];%d", &f, s, &i);
```

7. (2 points) Which of the following input lines will completely match the `scanf()` format string `"%d-%d-%d-%f"`? Circle all that apply.

   A. `3-4-5-6`

   B. `1 - 2 - 3 - 4.5`

   C. `12- 13- -8- -77.2`

   D. `-9--9-9--9.99999`

   E. None of the above

8. (2 points) True or False: `fscanf()` behaves equivalently to `scanf()` if you pass `stdin` as the first argument to `fscanf()`.

9. (2 points) Given the below code, will the value returned by `strcmp()` be positive, negative, or zero?

```
char str1[] = "retro";
char str2[] = "retroactive";
str2[5] = '\0';
int ret = strcmp(str1, str2);
```

Zero

10. (2 points) True or False: `strlen()` returns the number of characters in a string, including the `NUL` terminator.

11. (2 points) What will the following code print?

```
char str1[] = "video";
char *str2  = "audio";
printf("%d %d", sizeof(str1), sizeof(str2));
```

6 8

12. (2 points) Assume there exists a file named `"file.bin"` of size 32 bytes. What will be the value of `len` after the following code is executed? Assume a `short` is two bytes.

```
FILE *fp = fopen("file.bin", "rb");
fseek(fp, -12, SEEK_END);

short arr[3];
fread(arr, sizeof(short), 3, fp);
int len = ftell(fp);
```

26

13. (2 points) True or False: For any given text file, you can tell how many lines are in the file by seeking to the end of the file with `fseek()` and examining the return value of `ftell()`.

14. (3 points) Create a type and structure declaration named `record` that contains an array of 10 `char` named `id`, a `double` named `value`, and an array of 4 `short` named `data`.

```
typedef struct {
  char id[10];
  double value;
  short data[4];
} record;
```

15. (3 points) Using the `record` type from question 14, create a variable of that type and, in the same statement, initialize it to assign the string `"squirrel"` to `id`, the value 6.77 to `value`, and the values `127`, `255`, `365`, and `-4` to `data`.

```
record r = {
  "squirrel",
  6.77,
  { 127, 255, 365, -4 } };
```

16. (4 points) Given the following structure, indicate the location of each variable in memory by labeling the bytes in the diagram below with the letter of the variable occupying that space. Each box represents one byte. Use `P` for padding, and leave blank any bytes that are not part of the structure.

```
struct something {
  short a;
  int b;
  char c;
  short d[2];
};
```

a	a	P	P	b	b	b	b	c	P	d	d	d	d	P	P				

17. (2 points) True or False: Files written with `fwrite()` and read with `fread()` are always portable between machines with different architectures.

18. (2 points) Given the following code segment, what will be output by the `printf()` statement?

```
union stuff {
  char first[4];
  int second;
} thing = { .second = 0x44332211 };
thing.first[3] = 0x99;
printf("%x", thing.second);
```

99332211

19. (3 points) Declare an enumeration named `biome` that can take on the values: `GRASSLAND`, `FOREST`, `TUNDRA`, `DESERT`, and `OCEAN`.

```
enum biome {
  GRASSLAND,
  FOREST,
  TUNDRA,
  DESERT,
  OCEAN,
};
```

20. (2 points) Briefly describe the difference between the `|` operator and the `||` operator.

The `|` operator performs a bitwise OR on each bit of the operands, whereas the `||` operator performs a logical OR that returns `1` if either operand is non-zero, or `0` otherwise.

21. (3 points) Consider the following constants representing file permissions:

```
#define USER_READ      (1)  /* 000000001 */
#define USER_WRITE     (2)  /* 000000010 */
#define USER_EXEC      (4)  /* 000000100 */
#define GROUP_READ     (8)  /* 000001000 */
#define GROUP_WRITE   (16)  /* 000010000 */
#define GROUP_EXEC    (32)  /* 000100000 */
#define OTHER_READ    (64)  /* 001000000 */
#define OTHER_WRITE  (128)  /* 010000000 */
#define OTHER_EXEC   (256)  /* 100000000 */
```

Using only bitwise operators and the assignment operator, define an `int perms` to represent the total permissions of a given file with `USER_READ`, `USER_WRITE`, `GROUP_READ`, and `OTHER_READ` permissions. Then, in a separate statement, add the `USER_EXEC` permission, again only using bitwise and assignment operators. Finally, remove the `OTHER_READ` permission, once again using only bitwise and assignment operators.

```
int perms = USER_READ | USER_WRITE | GROUP_READ | OTHER_READ;
perms = perms | USER_EXEC;
perms = perms & ~OTHER_READ;
```

22. (2 points) Briefly describe what a pointer is.

A pointer is a variable that holds an address.

23. (2 points) Implement a function named `mult_seven()` that returns `void`. It takes one argument that is a pointer to an integer. The function should multiply by seven the value pointed to by the argument.

```
void mult_seven(int *num) {
  *num = *num * 7;
}
```

24. (2 points) Using your `mult_seven()` function from question 23, write another function named `call_mult_seven()` that takes one integer as an argument and returns an integer. The function should call the `mult_seven()` function such that the value of the argument is modified. Return the modified argument.

```
int call_mult_seven(int arg) {
  mult_seven(&arg);
  return arg;
}
```

25. (3 points) Rewrite the following code so that it does not use arrays or bracket notation (`[]`) anywhere.

```
void sma(int arr[]) {
  for (int i = 1; i < 9; i++) {
    arr[i] = (arr[i + 1] + arr[i - 1]) / 2;
  }
}
```

```
void sma(int *arr) {
  for (int i = 1; i < 9; i++) {
    *(arr + i) = (*(arr + i + 1) + *(arr + i - 1)) / 2;
  }
}
```

26. (2 points) Given the following code, what will be output by the `printf()` statement?

```
int fib[] = { 1, 1, 2, 3, 5 };
int *p = fib + 3;
printf("%d", *p + *(p + fib[*fib]));
```

```
8
```

27. (3 points) Consider the following code.

```
int global;

int *func(int *p_arg) {
  int *p_local = malloc(sizeof(int) * global);
  char str[] = "Some descriptive text";
  return p_local;
}
```

For each item below, circle whether it exists on the stack, the heap, neither, or if it's unknown.

`global`	Stack	Heap	**Neither**	Unknown
`p_arg`	**Stack**	Heap	Neither	Unknown
The data pointed to by `p_arg`	Stack	Heap	Neither	**Unknown**
`p_local`	**Stack**	Heap	Neither	Unknown
The data pointed to by `p_local`	Stack	**Heap**	Neither	Unknown
`str`	**Stack**	Heap	Neither	Unknown

28. (2 points) Briefly explain why you should never return a pointer to something on the stack.

When the function returns, the memory being pointed to is no longer valid and may be overwritten by another function's stack frame. The returned pointer will be pointing to invalid memory and dereferencing it will result in undefined behavior.

29. (2 points) True or False: Calling `malloc(sizeof(int) * 4)` is equivalent to calling `calloc(sizeof(int), 4)`.

30. (2 points) True or False: Memory allocated with either `malloc()` or `calloc()` may be freed by calling `free()`.

31. (3 points) Declare a structure named `node` that is a valid singly-linked list node containing a single `int`. Then, in a separate statement, declare a new type named `node_t` that represents the `node` structure.

```
struct node {
  int val;
  struct node *next;
};
typedef struct node node_t;
```

32. (5 points) Using the `node_t` type from question 31, write a function named `prepend_head()` that returns `void` and accepts two arguments: the *address* of the pointer to the head of a singly-linked list, and an `int`. Allocate a new `node_t` to store the value of the second argument and insert it before the current head of the list, updating the head pointer accordingly. Include appropriate assertion checks. An initially empty list is *not* an error condition.

```
void prepend_head(node_t **head, int val) {
  assert(head);

  node_t *new = malloc(sizeof(node_t));
  assert(new);

  new->val = val;
  new->next = *head;
  *head = new;
}
```

33. (2 points) Given a pointer to a node somewhere within a singly-linked list, briefly explain why you cannot swap the node with the next node in the list.

In order to swap the nodes, you would need to set the previous node's next pointer to point to the next node, but we don't have access to the previous node.

34. (2 points) How many pointer reassignments are required to insert a node before another node in the middle of an existing doubly-linked list, assuming the existing node is not the head of the list?

4

35. (3 points) Define a pointer named `fptr` that points to a function that returns a pointer to a string and accepts two arguments, a `void` pointer and an `int`. Initialize the pointer to point to a function named `callback`. Then, using the function pointer, invoke the function being pointed to, passing `NULL` for the first argument and `-9238459` for the second argument.

```
char *(*fptr)(void *, int) = callback;
fptr(NULL, -9238459);
```

36. (2 points) Which of the below statements are true about recursion? Circle all that apply.

A. A function that recurses infinitely will always cause a stack overflow.

B. Local variables are created for each invocation of a recursive function.

C. Recursive algorithms are typically slower than iterative versions.

D. Every recursive function needs a base case.

E. None of the above

37. (5 points) Assume there exists a binary search tree structure declared below:

```
struct tree {
  char *data;
  struct tree *left;
  struct tree *right;
};
```

Write a recursive function named `print_tree()` that returns `void` and accepts a single argument: a pointer to the root of a binary search tree. Print the tree in sorted order with a newline after each node.

```
void print_tree(struct tree *root) {
  if (!root) return;

  print_tree(root->left);
  printf("%s\n", root->data);
  print_tree(root->right);
}
```

38. (2 points) True or False: A pointer of one type may not be assigned to a pointer of another type unless it is cast or the assignment is to/from a `void` pointer.

39. (2 points) Define a pointer named `ptr` that points to an integer such that the integer pointed to may be modified, but the pointer itself may not be. Initialize the pointer to point to an `int val`.

```
int * const ptr = &val;
```

40. (2 points) True or False: Any `const` declaration can be disregarded by casting to a non-`const` type. Because of that, a `const` variable can never be truly constant.

41. (2 points) Give one example of why one might wish to make a variable `volatile`.

The variable might be modified from within a signal handler.

The variable might be modified by a different process or thread.

The variable might be modified by a hardware device.

To prevent the compiler from optimizing away the variable.

42. (4 points) For each statement below, circle whether it is a declaration, a definition, both, or neither.

`unsigned short val;`	Declaration	Definition	Both	Neither
`struct gizmo;`	Declaration	Definition	Both	Neither
`union u { char v; short w; } x;`	Declaration	Definition	Both	Neither
`typedef double octo[8];`	Declaration	Definition	Both	Neither
`extern char *str;`	Declaration	Definition	Both	Neither
`void *erase(void *);`	Declaration	Definition	Both	Neither
`int sub(int a, int b) { return a - b; }`	Declaration	Definition	Both	Neither
`#define INF (1.0/0.0)`	Declaration	Definition	Both	Neither

43. (2 points) True or False: `static` variables that are local to a function get initialized each time the function is invoked.

44. (2 points) Which of the following statements is true regarding the preprocessor? Circle all that apply.

    A. The preprocessor is a stage that occurs before compilation begins.

    B. The `#include` directive only pulls in the required symbols.

    C. The `#if` directive can evaluate mathematical expressions.

    D. A macro performs rudimentary type checking of its arguments.

    E. None of the above.

45. (3 points) Write a macro named `CUBE` that takes one argument and raises it to the third power.

```
#define CUBE(x) ((x) * (x) * (x))
```

46. (2 points) True or False: Using GCC's `-O3` flag can result in faster code execution at the expense of a larger executable.

47. (2 points) Write the GCC command to link the object file `farm.o` with the shared library `libcows.so`, and name the output executable `moo`.

```
gcc -o moo farm.o -lcows
```

48. (2 points) True or False: Static libraries are built into the executable, whereas shared libraries are loaded at runtime.

49. (3 points) Rewrite the following header file named `cows.h` to add include guards.

```
struct cow {
  char *name;
  void *secrets;
};

void moo(struct cow *);
```

```
#ifndef __COWS_H_
#define __COWS_H_

struct cow {
  char *name;
  void *secrets;
};

void moo(struct cow *);

#endif
```

50. (2 points) Why should you use `srandom()` prior to calling `random()`?

Using `srandom()` will initialize the seed of the random number generator. Without calling `srandom()`, the sequence of random numbers is exactly the same every time the program is run.

51. (2 points) Briefly explain what the function `SDL_PollEvent()` does.

The `SDL_PollEvent()` function checks for any pending events in the event queue. If there is one, it pops it from the queue and gives it to the user so that it can be processed.

52. (2 points) Given the following function and stack dump, which line number contains the frame pointer?

```
void hello(int value) {
  int local = 0xdddddddd;
}
```

```
1.  0x7ffd11da7d48: 00 00 00 00 ef 0f 00 00 ........
2.  0x7ffd11da7d40: d3 12 40 00 00 00 00 00 ..@.....
3.  0x7ffd11da7d38: 78 7d da 11 fd 7f 00 00 ........
4.  0x7ffd11da7d30: 50 b1 4d 5c da 00 eb 1c P.M\....
5.  0x7ffd11da7d28: 00 00 00 00 00 00 00 00 ........
6.  0x7ffd11da7d20: 86 ee 8b 4a dd dd dd dd ...J....
```

Line 3

53. (2 points) Which of the following statements are true regarding core dump files? Circle all that apply.

A. A core dump file can be used to diagnose a program that crashed.

B. Core dump files contain the entire memory image of a process at the time of termination.

C. Using a debugger with a core dump file, you can see function and variable names even when compiling without debug symbols.

D. If stack smashing occurs due to a buffer overflow, the backtrace will be incomplete.

E. None of the above

54. (2 points) True or False: in a Makefile, the target must always be the name of a file generated by the recipe.

55. (2 points) Given the following Makefile, which targets will be rebuilt if the file `cows.c` is modified before running `make`?

```
all: moo

%.o: %.c
  gcc -c -o $@ $< -Wall -Werror

libcows.a: cows.o
  ar -crvs $@ $^

moo: farm.o libcows.a
  gcc -o $@ $< -L. -lcows
```

`cows.o`, `libcows.a`, `moo`, `all`

56. (2 points) Briefly describe the difference between a system call and a function call.

A system call is executed in kernel mode with more privileges than user mode, such as the ability to interface with I/O devices.

57. (3 points) Number the below system calls from 1 to 6 in the order they must be invoked for a server to communicate over a network.

3 `listen()`

2 `bind()`

4 `accept()`

1 `socket()`

6 `close()`

5 `send()` and/or `recv()`

58. (2 points) Briefly describe the purpose of a datasheet.

A datasheet details technical characteristics of a hardware (or software) component. Typically contains everything needed to know to be able to interface with the component.