EECS-22: Advanced C Programming (Winter 2025)
Final Exam (03/19/2025)

Prepared by: Dr. Halima Bouzidi (EECS-22 Instructor)

Full Name:	
Student ID:	
Lab session:	
Signature:	

- This is a closed-book, closed-notes exam.
- All electronic devices must be turned off. No devices are permitted.
- You have 120 minutes to complete the exam.
- The exam consists of two parts:
  - MCQ Part: 35 multiple-choice questions found on pages 2 to 5 (Make sure you have all the pages!)
  - Programming Part: A programming problem on pages 6-10 (Make sure you have all the pages!)
- This page serves as the answer sheet for the MCQ Part. Mark your answers clearly by filling in the circles on this page. Answers marked on other pages will not be considered.
- Each multiple-choice question has four (04) answer choices. Mark the corresponding circle(s) on this answer sheet.
- There are two types of multiple-choice questions:
  - Questions worth one point have only one correct answer.
  - Questions worth two points may have multiple correct answers (Select all that apply).
- For multiple-choice questions, each incorrect selection negates one correct selection. If you fail to select at least half of the correct answers, you will receive no points.
- In the provided program code, line numbers are for reference only and are not part of the code itself.
- No questions are allowed during the exam. If in doubt, write a note and clearly state all your assumptions.

Q#	a	b	c	d
1	○	○	○	○
2	○	○	○	○
3	○	○	○	○
4	○	○	○	○
5	○	○	○	○
6	○	○	○	○
7	○	○	○	○
8	○	○	○	○
9	○	○	○	○
10	○	○	○	○
11	○	○	○	○
12	○	○	○	○

Q#	a	b	c	d
13	○	○	○	○
14	○	○	○	○
15	○	○	○	○
16	○	○	○	○
17	○	○	○	○
18	○	○	○	○
19	○	○	○	○
20	○	○	○	○
21	○	○	○	○
22	○	○	○	○
23	○	○	○	○
24	○	○	○	○

Q#	a	b	c	d
25	○	○	○	○
26	○	○	○	○
27	○	○	○	○
28	○	○	○	○
29	○	○	○	○
30	○	○	○	○
31	○	○	○	○
32	○	○	○	○
33	○	○	○	○
34	○	○	○	○
35	○	○	○	○

MCQ Part: Multiple Choice Questions (45 points total)

1. **Which of the following statements about the C language are true? (Select all that apply!)**
   (a) C code compiles into machine code specific to the target platform (e.g., 32-bit or 64-bit systems).
   (b) Functions in C cannot return pointers.
   (c) C supports both pointer arithmetic and arrays.
   (d) The `const` keyword can be used to make variables accessible for read-only.

---

2. **Which commands or flags are typically used in a Makefile? (Select all that apply!)**
   (a) `CC` to specify the compiler
   (b) `-Wall` to enable common warnings
   (c) `git add` to stage changes in a Github repository
   (d) `LDFLAGS` to specify linker flags

---

3. **Which operator in C can be used to get the address of a variable?**
   (a) The `++` operator
   (b) The `--` operator
   (c) The `&` operator
   (d) The `*` operator

---

4. **In C, what is the correct way to declare a pointer to an integer?**
   (a) `int p*;`
   (b) `int *p;`
   (c) `pointer int p;`
   (d) `int p->int;`

---

5. **Which library function in C would you use to compare two strings lexicographically?**
   (a) `strcmp`
   (b) `scanf`
   (c) `strcat`
   (d) `strlen`

---

6. **Which of the following statements about pointers to structures in C are true? (Select all that apply!)**
   (a) The `->` operator is used to access members via a pointer.
   (b) You can dynamically allocate structures with `malloc`.
   (c) It's recommended to cast the return of `malloc` to the structure type in C.
   (d) Accessing `ptr->member` is equivalent to `(*ptr).member`.

---

7. **Which keyword is used to prevent a variable from being modified, thereby turning it into a read-only variable in C?**
   (a) `static`
   (b) `register`
   (c) `extern`
   (d) `const`

---

8. **In C, which of the following correctly declares a function pointer named `funcPtr` that points to a function taking two `int` parameters and returning an `int`?**
   (a) `int funcPtr(int a, int b);`
   (b) `int *funcPtr(int, int);`
   (c) `int (*funcPtr)(int, int);`
   (d) `(*funcPtr)(int, int) = int;`

---

9. **What does the `sizeof` operator return in C?**
   (a) The number of elements in an array
   (b) The size of a type or object in bytes
   (c) The pointer address of the variable
   (d) The alignment requirement of a struct

---

10. **Which of the following would best describe a segmentation fault in C?**
    (a) Reading from a file that does not exist
    (b) Accessing memory outside the allocated range
    (c) An error caused by insufficient CPU speed
    (d) A syntax error detected by the compiler

---

11. **Which keyword in C is used to create a block-scope variable that persists for the duration of the program?**
    (a) `static`
    (b) `register`
    (c) `volatile`
    (d) `extern`

---

12. **In which section of memory are local (non-static) variables typically allocated at runtime in C?**
    (a) Heap
    (b) Stack
    (c) Global area
    (d) Code segment

---

13. **What is the purpose of the `fopen` in C?**
    (a) To read a line from standard input
    (b) To open a file and return a `FILE*` handle
    (c) To close an open file
    (d) To format output to the console

---

14. **Which of the following `gdb` commands are commonly used during debugging? (Select all that apply!)**
    (a) `break <location>`

(b) `print <variable>`

(c) `run [arguments]`

(d) `continue`

Program 1 (Singly Linked List):
A singly linked list that stores integers. Answer questions 15-19 by choosing the correct statements.

```c
#include <stdio.h>
#include <stdlib.h>

/* Node structure */
typedef struct Node {
    int data;
    struct Node *next;
} Node;
/* Insert a new node as the new head of list */
Node* InsertNewHead(Node *head, int value) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if(newNode == NULL) {
        printf("Allocation error\n");
        return head;
    }
    newNode->data = value;
    // Box 1
    return newNode;
}
/* Print the entire list */
void PrintList(Node *head) {
    Node *temp = head;
    // Box 2
    {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
/* Free all nodes in the list */
void FreeList(Node *head) {
    // Box 3
}
int main() {
    Node *head = NULL;
    head = InsertNewHead(head, 10);
    head = InsertNewHead(head, 20);
    head = InsertNewHead(head, 30);

    PrintList(head);
    FreeList(head);
    return 0;
}
```

15. **In `InsertNewHead`, what belongs in Box 1 to link the new node to the current head?**

    (a) `newNode->next = NULL;`

    (b) `head->next = newNode;`

    (c) `newNode->next = head;`

    (d) `head = newNode;`

---

16. **Which statements about `InsertNewHead` are true after you fix Box 1? (Select all that apply!)**

    (a) It creates a new node on the heap when it's called.

    (b) It always inserts at the beginning (head) of the list.

    (c) It can return the same head pointer if `malloc` fails.

    (d) It sets `newNode->data` before linking the node into the list.

---

17. **In `PrintList`, how should we fix Box 2 so that the loop visits all nodes until `temp == NULL`?**

    (a) `while (temp != NULL) {...}`

    (b) `while (temp->next != NULL) {...}`

    (c) `for (int i = 0; i < 5; i++) {...}`

    (d) `do {...} while (temp != NULL);`

---

18. **If `PrintList` is called after inserting 30, 20, 10, which output is expected once the code is fixed?**

    (a) `10 -> 20 -> 30 -> NULL`

    (b) `30 -> 20 -> 10 -> NULL`

    (c) `30 -> 10 -> 20 -> NULL`

    (d) `10 -> NULL`

---

19. **In `FreeList`, which is a correct approach to free all nodes?**

    (a) `free(head); head = NULL;`

    (b) `Node *tmp; while(head) {tmp = head; head = head->next; free(tmp);}`

    (c) `Node *next = head->next; free(head->next); head->next = next;`

    (d) `// No need to free anything`

---

20. **What will be the value of `x` after executing the following code snippet in C?**

```c
int x = 5;
int *p = &x;
*p = *p + 10;
```

    (a) 5

    (b) 10

    (c) 15

    (d) 0

---

21. **If `p` is a `NULL` pointer, which statement is true about dereferencing `p`?**

    (a) It safely returns 0.

    (b) It is valid only if `p` points to an integer type.

    (c) It results in undefined behavior.

    (d) It will call `free` on the pointer automatically.

---

22. **Which statement about passing structures to functions in C is correct?**

    (a) Passing a struct by value does not create a copy inside the function.

    (b) Passing a pointer to a struct copies the entire structure into the function.

    (c) Passing a struct by value creates a copy, so changes in the function do not affect the original struct.

    (d) C does not allow passing structures as function parameters.

---

23. **Which statement about singly linked lists is correct?**

(a) Each node in a singly linked list contains data and a pointer to the next node.

(b) Inserting at the head requires modifying every node in the list.

(c) Traversing the list typically starts at the tail pointer and moves backward to the head.

(d) Deleting the first node never requires updating any pointer.

Program 2 (Recursion with Errors):
A function to compute the nth Fibonacci number, but it contains errors. Answer questions 24-27 by choosing the correct statements.

```c
#include <stdio.h>

/* Returns the nth Fibonacci number */
int fibonacci(int n) {
    if (n < 0) {
    // Box 1
    }
    if (n == 0) return 0;
    if (n == 1) return 1;
    // Box 2
}
int main() {
    int val = 5;
    printf("Fibonacci of %d is %d\n", val, fibonacci
    (val));
    return 0;
}
```

24. **In Box 1, how should negative `n` be handled?**

(a) `return -1;` // Indicate an error

(b) `while (1) {}` // Infinite loop

(c) `return fibonacci(-n);` // Flip sign

(d) `/* do nothing */` // Just continue

---

25. **In Box 2, which line correctly implements the recursive Fibonacci definition for `n >1`?**

(a) `return n + 1;`

(b) `return fibonacci(n-1) * fibonacci(n-2);`

(c) `return fibonacci(n-1) + fibonacci(n-2);`

(d) `return 0;`

---

26. **Which statements about this recursive function are true once fixed? (Select all that apply!)**

(a) It will overflow the call stack for very large `n`.

(b) It returns `-1` for negative `n`, indicating an error.

(c) It uses more memory than an iterative approach.

(d) It always completes in constant time.

---

27. **If `main` calls `fibonacci(5)` after fixing errors, what is the correct output?**

(a) 8

(b) 5

(c) 120

(d) -1

Program 3 (Pointers with Gaps to Fill):
Fill in the missing pieces to correctly allocate and use a dynamic array of integer arrays. Answer questions 28-32 by choosing the correct statements.

```c
#include <stdio.h>
#include <stdlib.h>

int** createIntArray(int count, int size) {
    // Box 1
    for (int i = 0; i < count; i++) {
        // Box 2
    }
    // Box 3;
}
int main() {
    int n = 3;
    int size = 5;
    int **arr = createIntArray(n, size);
    if (!arr) {
        printf("Allocation failed!\n");
        return 1;
    }
    // Box 4
    // Box 5
    return 0;
}
```

28. **In Box 1, which statement correctly allocates an array of `count int *`?**

(a) `int **arr=malloc(count*sizeof(int ));`

(b) `int *arr=malloc(count*sizeof(int *));`

(c) `int **arr=malloc(count*sizeof(int *));`

(d) `int arr[count][size];`

---

29. **In Box 2, which is a valid way to allocate each integer array (assume size 5)?**

(a) `arr[i]=malloc(size*sizeof(int));`

(b) `arr[i]=(int)smalloc(size*sizeof(int));`

(c) `arr[i]=(int**)malloc(size*sizeof(int*));`

(d) `arr = NULL;`

---

30. **In Box 3, what should we return after successful allocation in `createIntArray`?**

(a) `return &arr;`

(b) `return *arr;`

(c) `return arr;`

(d) `return NULL;`

---

31. **In Box 4, which loop prints each integer array in `arr`?**

(a) `for (int i = 0; i < n; i++) { for (int j = 0; j < size; j++) printf("%d ", arr[i][j]);printf(" n"); }`

(b) `for (int i = 0; i < n; i++) {printf("%d n", *arr[i]);}`

(c) `while (arr) {printf("%d n", arr); arr++; }`

(d) `for (int i = 0; i < n; i++) {printf("%d n", arr[i]);}`

---

32. **How should we perform Box 5 (freeing all memory)? (Select all that apply!)**

    (a) Loop over each `arr[i]` and call `free(arr[i])`.

    (b) Then free the array of pointers itself with `free(arr)`.

    (c) Call `free(arr)`, it frees everything automatically.

    (d) Use `malloc` on `arr` to zero bytes.

---

33. **Which statements about `strcpy` in C are correct? (Select all that apply!)**

    (a) `strcpy(dest, src)` copies the string `src` (including the null terminator) into `dest`.

    (b) `strcpy` performs bounds-checking on `dest` to prevent buffer overflows.

    (c) The behavior is undefined if `dest` and `src` overlap in memory.

    (d) `strcpy` returns a pointer to `dest`.

---

34. **Which statements about function argument passing in C are correct? (Select all that apply!)**

    (a) Arguments are always passed by value.

    (b) Passing a pointer to a function means the pointer itself is passed by value.

    (c) Passing an array to a function copies the entire array onto the stack.

    (d) Changing a function parameter does not directly change the caller's variable, unless via pointers.

---

35. **Which statement about command-line arguments in C is correct?**

    (a) `argv[0]` typically contains the last argument typed on the command line.

    (b) `argc` typically contains the program name.

    (c) Accessing `argv[argc]` is always invalid.

    (d) `argv[argc-1]` is guaranteed to be `NULL`.

Programming Part: Programming Problem (55 points total)

Similar to the example discussed in the lectures (Students List), a doubly linked list is used in the following two header files. In this case, it is applied to manage a collection of books in a library, allowing efficient insertion, deletion, and filtering of book records. Note that a book record is defined by the following attributes: ISBN (International Standard Book Number), Title, and Author.

Book.h:

```
/* Book.h: header file for book records */

#ifndef BOOK_H
#define BOOK_H

#define SLEN 100

/* Book record structure */
typedef struct book
{
    int ISBN;
    char Title[SLEN + 1];
    char Author[SLEN + 1];
} BOOK;

/* Function prototypes */

/* Allocates and initializes a new book */
BOOK *NewBook(int ISBN, char *Title, char *Auth);

/* Deletes a book and frees its memory */
void DeleteBook(BOOK *b);

/* @brief Prints the details of a book */
void PrintBook(BOOK *b);

#endif /* BOOK_H */
```

BookList.h:

```
/* BookList.h: header file for book lists */

#ifndef BOOKLIST_H
#define BOOKLIST_H

#include "Book.h"

/* Structure for an entry in the book list */
typedef struct bentry
{
    struct book *Book;
    struct bentry *Next;
    struct bentry *Prev;
    struct booklist *List;
} BENTRY;

/* Structure to represent the book list */
typedef struct booklist
{
    int Length;
    BENTRY *First;
    BENTRY *Last;
} BOOKLIST;

/* Function prototypes */

/* Allocates and initializes a new book list */
BOOKLIST *NewBookList(void);

/* Allocates and initializes a new book entry */
BENTRY *NewBookEntry(BOOK *b);

/* Prints all books in the book list */
void PrintBookList(BOOKLIST *l);

/* Deletes a book entry (not the book record) */
BOOK *DeleteBookEntry(BENTRY *e);

/* Deletes an entire book list */
void DeleteBookList(BOOKLIST *l);

/* Appends a book at the end of the book list */
void AppendBook(BOOKLIST *l, BOOK *b);

/* filter a book list by specified author */
BOOKLIST *Filter(BOOKLIST *l, const char Auth);

#endif /* BOOKLIST_H */
```

Your task is to implement the functions **AppendBook**, **DeleteBookList**, and **Filter**

Assume that all other functions follow the same implementation style as covered in the lectures.

Question 1: Visualize the doubly linked list data structure (10 points)

Draw a diagram that shows the data structure (structures, members, and pointers) for an example list with several books. Your diagram should show a list header and four (04) list entries and book records (of your favorite books!) connected by arrows represening pointers.

Question 2: Implement the `AppendBook` function: (15 points)

- Start with the function signature as listed in `BookList.h` above.
- Use `assert()` statements to ensure that the values in parameters are as expected.
- Make sure the function can handle empty lists as well as lists with existing elements.
- Update all necessary data structure members.

AppendBook (15 points):

Question 3: Implement the `DeleteBookList` function: (15 points)

The `DeleteBookList` function is responsible for deleting an entire book list. It takes a book list as input and ensures that all book entries and records are properly removed from memory before deallocating the list itself. The function should handle cases where the list is already empty and ensure no memory leaks occur. After execution, the book list should no longer be accessible.

- Start with the function signature as listed in `BookList.h` above.
- Use `assert()` statements to ensure that the input list is valid.
- Traverse the list and free all allocated memory for each book entry.
- Properly deallocate the list structure itself.
- Ensure that there are no memory leaks after executing the function.
- Set the list pointer to `NULL` after deletion to prevent dangling pointers.

DeleteBookList (15 points):

Question 4: Implement the `Filter` Book function: (15 points)

The `Filter` function takes a given book list as input and filters it so that only books written by a specified author remain in the list. Books that do not match the specified author are to be removed. As a result, the `Filter` function returns the filtered list to the caller:

- Start with the function signature as listed in `BookList.h` above.
- Use `assert()` statements to ensure that the values in parameters are as expected.
- Make sure the function can handle empty lists as well as lists with existing elements.
- Update all necessary data structure members.
- Ensure that there is no memory leak after executing the function.

Filter (15 points):