Assignment — Functions and Header Files in C

Introduction to functions

A function is a named block of code that does one job. You call it when you need that job done. Functions make code shorter, clearer, and reusable.

Parts of a function:

- Return type — what it gives back (int, float, void, ...)
- Name — the function identifier.
- Parameters — inputs inside ().
- Body — code inside { ... } that does the work.

Example:

```
int add(int a, int b) {
   return a + b;
}
```

```
#include <stdio.h>
int main(void) {
   int s = add(3, 4);
   printf("sum = %d\n", s);
   return 0;
}
```

Objective 1 — Create functions

Students will implement these three functions:

1. int isArmstrong(int num); — Check if a number is an Armstrong number.

2. int isAdams(int num); — Check if a number is an Adams number (reverse of square equals square of reverse).

3. int isPrimePalindrome(int num); — Check if a number is both prime and palindrome.

Example:

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

```
// Function to reverse digits of a number
int reverseDigits(int n) {
   /* Your Code Here */

}

// Function to check if number is Armstrong
int isArmstrong(int num) {
   /* Your Code Here */

}

// Function to check if number is Adams Number
int isAdams(int num) {
   /* Your Code Here */

}

// Function to check if number is prime
int isPrime(int num) {
   /* Your Code Here */}

// Function to check if number is prime and palindrome
int isPrimePalindrome(int num) {
   /* Your Code Here */

}

// Main menu-driven program
int main() {
   int choice, num;

   do {
      printf("\n===== MENU =====\n");
      printf("1. Check Armstrong Number\n");
      printf("2. Check Adams Number\n");
      printf("3. Check Prime Palindrome Number\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
```

```
        if (choice == 4) {
            printf("Exiting program. Goodbye!\n");
            break;
        }

        printf("Enter a number: ");
        scanf("%d", &num);

        switch (choice) {
            case 1:
                if (isArmstrong(num))
                    printf("%d is an Armstrong number.\n", num);
                else
                    printf("%d is NOT an Armstrong number.\n", num);
                break;

            case 2:
                if (isAdams(num))
                    printf("%d is an Adams number.\n", num);
                else
                    printf("%d is NOT an Adams number.\n", num);
                break;

            case 3:
                if (isPrimePalindrome(num))
                    printf("%d is a Prime Palindrome number.\n", num);
                else
                    printf("%d is NOT a Prime Palindrome number.\n", num);
                break;

            default:
                printf("Invalid choice! Please select between 1–4.\n");
        }

    } while (choice != 4);

    return 0;
}
```

Page 2 — Header files: why, how, and example

Header files store function declarations, macros, and structure definitions.
They separate declaration from definition, making code modular and reusable.

For example: stdio.h is a header file which provides robust functionality of taking all types of inputs and displaying them.

Include **guard** pattern:

```
#ifndef MYLIB_H
#define MYLIB_H
/* declarations */
#endif
```

Step-by-step example:

1. Create mylib.h — declarations

```
#ifndef MYLIB_H
#define MYLIB_H

int isArmstrong(int num);
int isAdams(int num);
int isPrimePalindrome(int num);

#endif
```

2. Create mylib.c — definitions (student implements logic)

```
#include "mylib.h"

int reverseDigits(int n) {
   int r = 0;
   while (n) { r = r*10 + (n % 10); n /= 10; }
   return r;
}

int isArmstrong(int num) {
```

```
        /* Your Code Here */
}
int isAdams(int num) {
        /* Your Code Here */
}
int isPrimePalindrome(int num) {
        /* Your Code Here */
}
```

3. Create main.c — use functions

```
#include <stdio.h>
#include "mylib.h"

int main(void) {
   int n = 12;
   printf("isAdams(%d) = %s\n", n, isAdams(n) ? "Yes" : "No");
   return 0;
}
```

4. Compile & run:

```
gcc main.c mylib.c -o prog
./prog
```

Only after completing the above example, you would be able to complete the 2nd objective of this assignment.

Objective 2 — Create an array helper header for 1D arrays

Goal: make common 1-D array operations simple to call.

Suggested functions in arraylib.h:

```
#ifndef ARRAYLIB_H
#define ARRAYLIB_H
```

```
int findMaxIndex(int arr[], int size);
int findMinIndex(int arr[], int size);
float findAverage(int arr[], int size);
void displayArray(int arr[], int size);
void reverseArray(int arr[], int size);
void sortArray(int arr[], int size);
int linearSearch(int arr[], int size, int value);

#endif /* ARRAYLIB_H */
```

Sample usage in main.c:

```
#include <stdio.h>
#include "arraylib.h"



int main(void) {
   int a[] = {3, 1, 4, 1, 5};
   int n = 5;
   displayArray(a, n);
   printf("Max at index %d\n", findMaxIndex(a,n));
   reverseArray(a,n);
   displayArray(a,n);
   return 0;
}
```

Submission checklist

- mylib.h and mylib.c — Objective 1 (function logic)
- arraylib.h and arraylib.c — Objective 2 (array utilities)
- main.c — test both headers
- Short README — how to compile and run