



IOT Team 20 Assignment 1 report

Assignment background

This assignment is part of the Summer 2025 Field Training for the IoT Software & Hardware Engineering program. It aims to reinforce students’ understanding of foundational programming concepts through a series of conceptual and practical coding problems. The tasks cover core C programming topics such as arrays, pointers, memory management, and functions, along with real-world applications like financial calculations, string manipulation, and algorithmic challenges. The assignment encourages both individual and team collaboration, and emphasizes hands-on practice, logical reasoning, and structured problem-solving — essential skills in embedded and software system development.

Team Members

Team member	Id	Task
Mostafa Mohamed Morshedy (Team leader)	23011156	Problem 1: Conceptual Questions, Problem 2: Revolving Credit Account
Yousef Khaled Shawkyy	23011635	Problem 9: Search Insert Position, Problem 10: Add Binary,

		Problem 11: Word Search (Bonus), Final Report Writing
Yousef Othman Mohamed	23011645	Problem 5: Longest Substring Without Repeating Characters, Problem 6: String to Integer
Badr Ahmed Mohamed	23011235	Problem 3: Number Manipulation, Problem 4: Pizza Size Comparison, Bonus: Palindrome
Yehia Mostafa Mohamed	23011620	Problem 7: Median of Two Sorted Arrays, Problem 8: First and Last Position in Sorted Array

Problem 1

1- An array in C is a fixed-size collection of similar data items stored in contiguous memory locations

-> Main characteristics

-Collection of Same Data Type

-Contiguous Memory Allocation

-Fixed Size

-Indexing

-Multi-dimensional Array

-> Uses

-Storing and accessing data

-Searching

-Matrices

-Implementing other data structures

Ref: <https://www.geeksforgeeks.org/c/c-arrays/>,
https://www.tutorialspoint.com/cprogramming/c_arrays.htm

2-

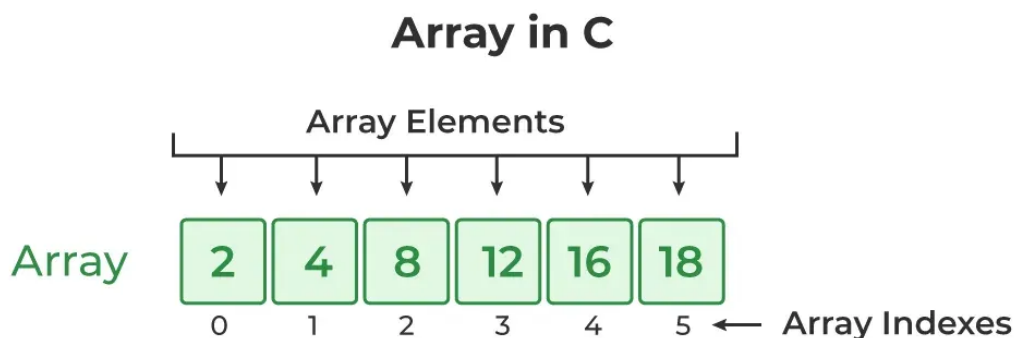
-In memory, arrays are stored in contiguous locations.

-Each element is stored in adjacent memory locations.

-The memory representation of an array is like a long tape of bytes, with each element taking up a certain number of bytes.

Ref: <https://medium.com/@pujari.thanmayee/arrays-and-memory-representation-96c07332916f>

Picture for Q1,Q2

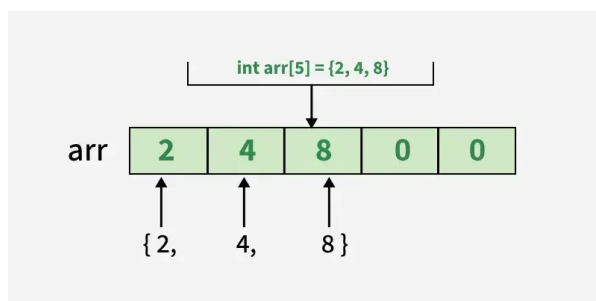
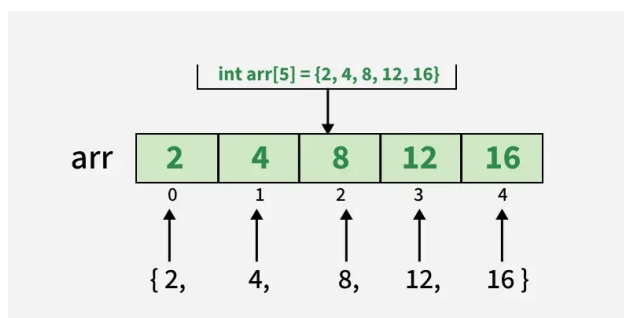


3-

Full Initialization: `int arr[5] = {2, 4, 8, 12, 16};`

Partial Initialization : `int arr[5] = {2, 4, 8}` -> The remaining elements will be assigned the value 0 (or equivalent according to the type)

Ref: <https://www.geeksforgeeks.org/c/c-arrays/>

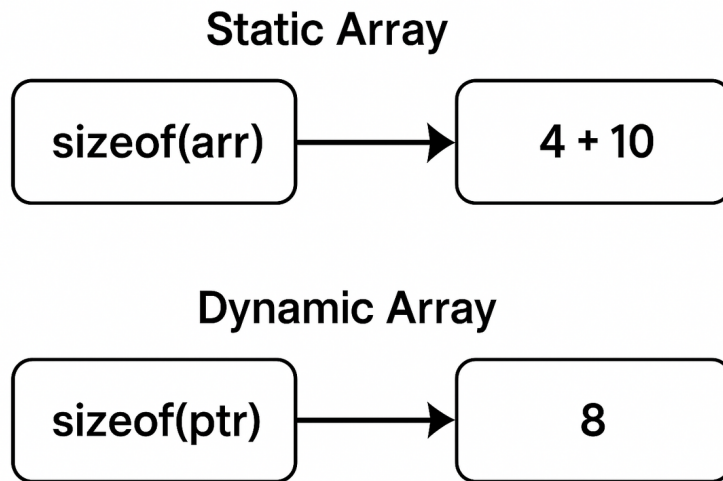


4-

-int n = sizeof(arr) / sizeof(arr[0]);

-The sizeof operator only returns the size of the pointer, not the memory block size.

Ref: <https://www.geeksforgeeks.org/c/c-arrays/>



5-

If, we use an array index which is out of bounds, then the compiler will compile and even run. But, there is no guarantee for the correct result.

-> Consequences:

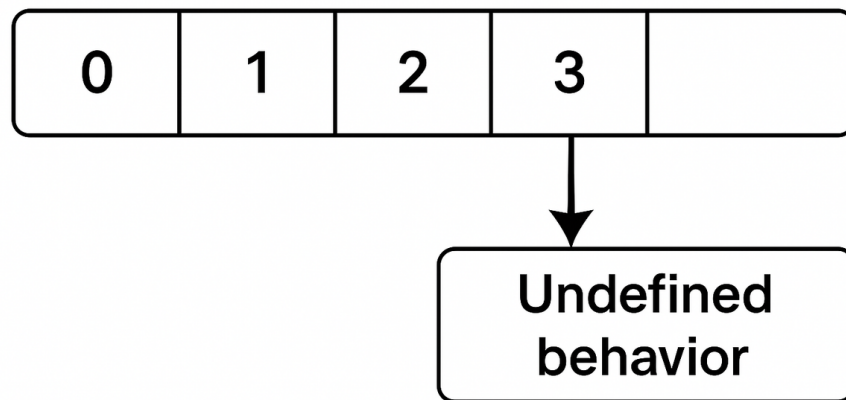
-Undefined behavior.

-May read/write garbage memory.

Why? C doesn't perform bounds checking at runtime.

Ref: <https://www.tutorialspoint.com/what-is-out-of-bounds-index-in-an-array-c-language#:~:text=Suppose%20you%20have%20an%20array,an%20index%20out%20of%20bounds.>

Accessing an Array Index Outside Its Bounds



6-

1D Array: Just a sequence of elements

```
int a[5] = {2, 4, 8, 12, 16};
```

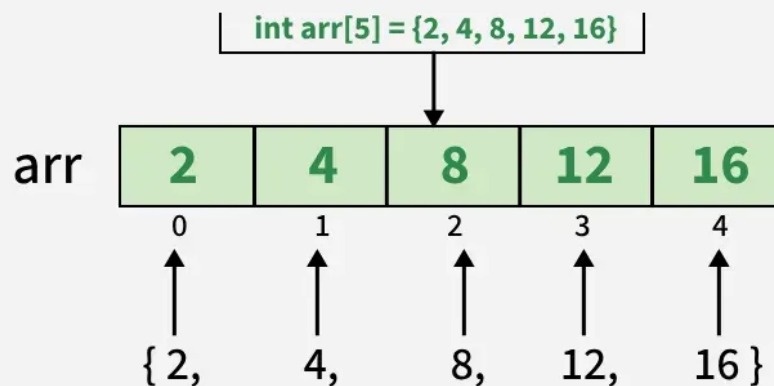
Use case: Lists, scores, names, simple sequences.

2D Array: more like table of elements with rows and columns

```
int b[2][3] = {{1,4,2},{3,6,8}};
```

Use case: Matrices, tables, grids.

Ref: https://www.w3schools.com/c/c_arrays_multi.php



	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

7-

arrays are always passed to function as pointers.

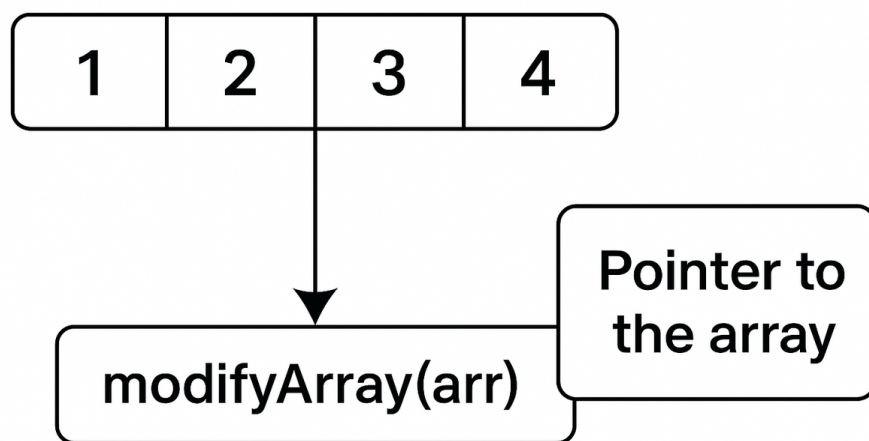
```
int arr[3] = {1,2,3};
```

```
ChangeArray(arr);
```

when we pass array like that it is actually pass the address of the array not a copy of it so any change of array in function affects the original one due to change in same location in memory;

Ref: <http://www.geeksforgeeks.org/cpp/how-arrays-are-passed-to-functions-in-c/>

Passing an Array to a Function



8-

actually name of array is treated as a pointer to first element in array with address of first element address, so here we can assign pointer to first element in array an loop over it.

```
int arr[4] = {25, 50, 75, 100};
```

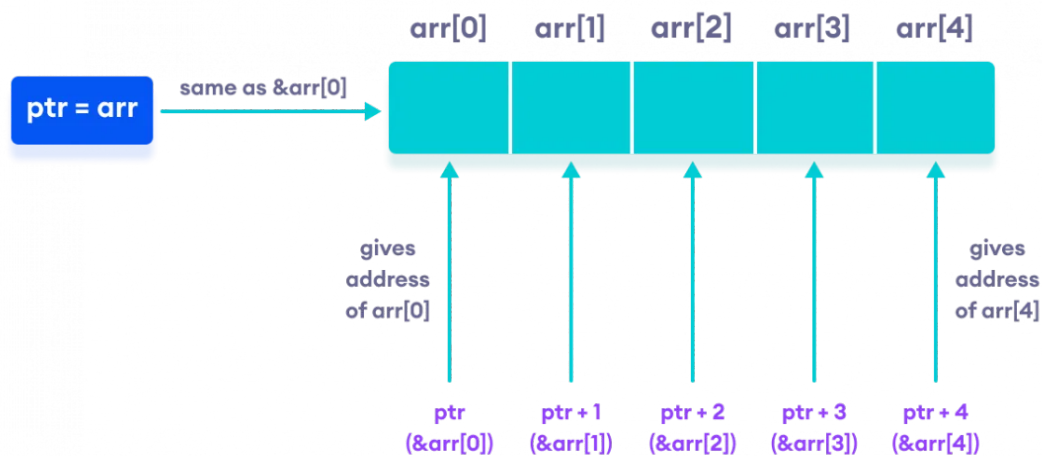
```
int *ptr = arr;
```

```
for (int i = 0; i < 4; i++) {
```

```
    printf("%d\n", *(ptr + i));
```


}

Ref: https://www.w3schools.com/c/c_pointers_arrays.php

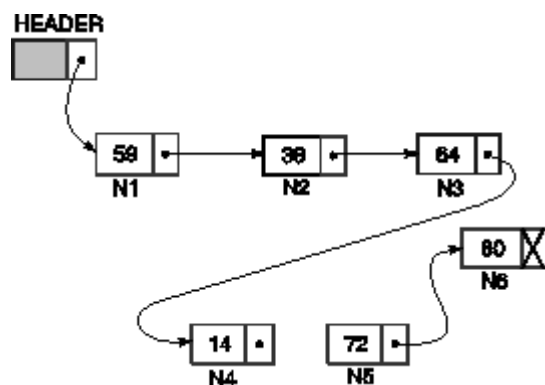


9-

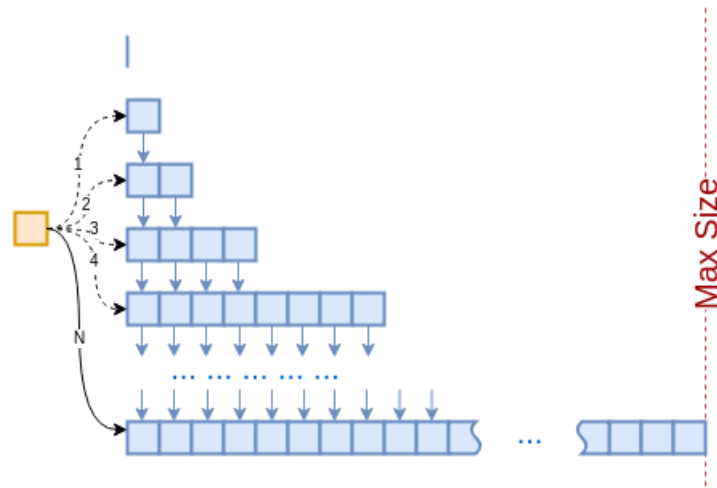
- Fixed size, we overcome this using dynamic arrays or vectors (in c++ for example)
- array is homogeneous, meaning all elements must be the same type, we can overcome this with structures.
- Array is Contiguous blocks of memory, we can overcome this with linked lists. Using pointers in linked lists make it allocate any location of memory for any elements with pointer with this this address.

Ref: <https://www.geeksforgeeks.org/c/advantages-and-disadvantages-of-array-in-c/>

Linkedlist



Vector



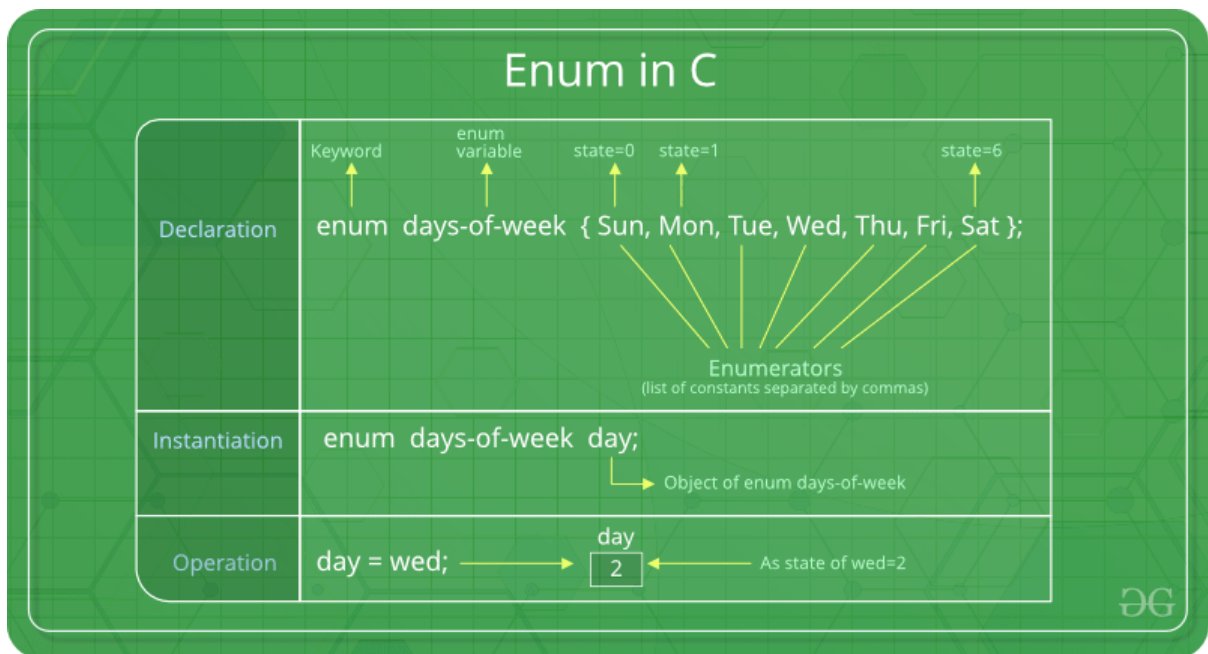
10-

In C, an enumeration (or enum) is a user defined data type that contains a set of named integer constants. It is used to assign meaningful names to integer values, which makes a program easy to read and maintain.

Example:

```
enum days-of-week { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
```

Ref: <https://www.geeksforgeeks.org/c/enumeration-enum-c/>



Problem 2

Problem Description

The task is to create a C program that calculates the **interest**, **total amount due**, and **minimum payment** for a credit account based on a given balance. The interest calculation follows a tiered rate:

- **1.5% interest** on the **first \$1000**
- **1% interest** on any amount **above \$1000**

The **minimum payment** rules are:

- If the total amount due is **\$10 or less**, the full amount is due.
- Otherwise, the minimum payment is the **greater of \$10 or 10%** of the total amount due.

The program should allow the user to repeat the process for multiple balances using a loop until the user decides to exit.

Problem 2 > p2.c

```
1  #include <stdio.h>
2  int main() {
3      double balance, interest = 0.0, totalDue, minPayment;
4      char choice;
5
6      do {
7          printf("Enter your account balance: $");
8          scanf("%lf", &balance);
9
10         if (balance <= 1000) {
11             interest = balance * 0.015;
12         } else {
13             interest = 1000 * 0.015 + (balance - 1000) * 0.01;
14         }
15
16         totalDue = balance + interest;
17
18         if (totalDue <= 10) {
19             minPayment = totalDue;
20         } else {
21             minPayment = totalDue * 0.10;
22             if (minPayment < 10)
23                 minPayment = 10;
24         }
25
26         printf("\nInterest: $%.2f\n", interest);
27         printf("Total Amount Due: $%.2f\n", totalDue);
28         printf("Minimum Payment: $%.2f\n", minPayment);
29
30         printf("\nDo you want to enter another balance? (Y/N):");
31         scanf(" %c", &choice);
32
33     } while (choice == 'Y' || choice == 'y');
34
35     printf("Thank you. Goodbye!\n");
36
37     return 0;
38 }
39
```

Code Explanation

Variables:

- `balance` : stores the user's input balance.
- `interest` : calculated interest based on the balance.
- `totalDue` : the sum of balance and interest.
- `minPayment` : the calculated minimum payment due.
- `choice` : used to control the loop based on user input.

Interest Calculation:

- If the balance is **less than or equal to \$1000**, apply **1.5%** on the whole amount.

- If the balance is **more than \$1000**, apply **1.5% on \$1000** and **1% on the remaining amount**.

Total Due:

- Calculated as `balance + interest`.

Minimum Payment Logic:

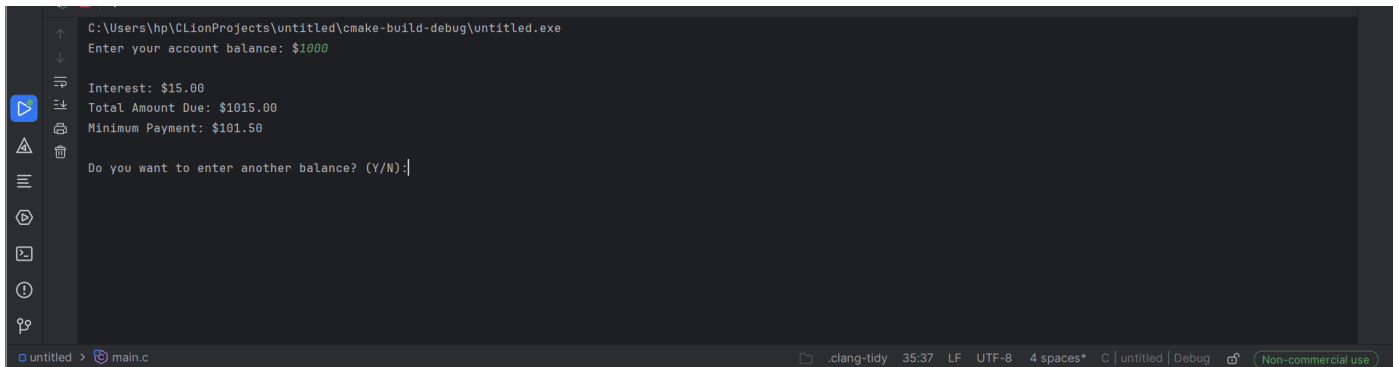
- If the total due is \leq **\$10**, the minimum payment is the full amount.
- Otherwise, calculate **10% of the total** and ensure it's at least **\$10**.

Loop:

- The program runs inside a `do-while` loop to allow the user to enter multiple balances until they type `N` or `n`.

Output:

- Displays the calculated **interest**, **total amount due**, and **minimum payment** clearly



```
C:\Users\hp\CLionProjects\untitled\cmake-build-debug\untitled.exe
Enter your account balance: $1000

Interest: $15.00
Total Amount Due: $1015.00
Minimum Payment: $101.50

Do you want to enter another balance? (Y/N):Y
```



```
Do you want to enter another balance? (Y/N):y
Enter your account balance: $5500

Interest: $60.00
Total Amount Due: $5560.00
Minimum Payment: $556.00

Do you want to enter another balance? (Y/N):n
Thank you. Goodbye!
```



```
Do you want to enter another balance? (Y/N):y
Enter your account balance: $500

Interest: $7.50
Total Amount Due: $507.50
Minimum Payment: $50.75

Do you want to enter another balance? (Y/N):
```

Problem 3

Problem Description

The task is to write a C program that takes a **positive integer** as input and performs the following three operations:

Check if it is a perfect square (e.g., 25 → Yes, 26 → No).

Reverse the digits of the number (e.g., 123 → 321).

Calculate the sum of its digits (e.g., 123 → 1+2+3 = 6).

The program should use three separate functions:

- `isPerfectSquare()` → returns `true` or `false`.
- `reverseDigits()` → returns the reversed number.
- `calculateSum()` → returns the sum of digits.

```
problem 3 > q3.c
1  #include <stdio.h>
2  #include <math.h>
3  #include <stdbool.h>
4
5  bool isPerfectSquare(int);
6  int reverseDigits(int);
7  int calculateSum(int);
8
9  int main() {
10
11     int number;
12     scanf("%d", &number);
13
14     printf("Is perfect square? %s\n", isPerfectSquare(number) ? "Yes" : "No");
15     printf("Reversed digits: %d\n", reverseDigits(number));
16     printf("Sum of digits: %d\n", calculateSum(number));
17
18     return 0;
19 }
20
21 bool isPerfectSquare(int num) {
22
23     if (num < 0) return false;
24
25     for (int i = 1; i * i <= num; i++) {
26         if (i * i == num)
27             return true;
28     }
29     return false;
30 }
31
32 int reverseDigits(int num) {
33
34     int reversed = 0;
35
36     while (num != 0) {
37         reversed = reversed * 10 + num % 10;
38         num /= 10;
39     }
40     return reversed;
41 }
42
43 int calculateSum(int num) {
44
45     int sum = 0;
46
47     while (num != 0) {
48         sum += num % 10;
49         num /= 10;
50     }
51     return sum;
52 }
53
```

```
"D:\code blocks projects\assig" x + v
12344321
Is perfect square? No
Reversed digits: 12344321
Sum of digits: 20

Process returned 0 (0x0)   execution time : 21.602 s
Press any key to continue.
```

```
1 #include <stdio.h>
2 #include <math.h>

25
Is perfect square? Yes
Reversed digits: 52
Sum of digits: 7

Process returned 0 (0x0)   execution time : 4.013 s
Press any key to continue.
```

```
"D:\code blocks projects\assig" x + v
10
Is perfect square? No
Reversed digits: 1
Sum of digits: 1

Process returned 0 (0x0)   execution time : 3.626 s
Press any key to continue.
```

Code Explanation

1727. Main Function:

- Reads a positive integer from the user.
- Calls and prints the result from the three functions:

- Whether the number is a perfect square.
- Its reverse.
- The sum of its digits.

isPerfectSquare(int num)

- Checks if `num` is a perfect square by looping `i` from 1 upwards.
- If `i*i` equals the number, return `true`. If loop ends, return `false`.

reverseDigits(int num)

- Extracts the last digit using `% 10`, builds the reversed number.
- Divides by 10 to drop the last digit in each step.

calculateSum(int num)

- Similar loop to above, but instead of reversing, it adds the digit to a `sum`.
-

Problem 4

Problem Description

The objective is to write a C program that helps a user decide which of two pizzas is the better value for money based on their **cost per square inch**.

The program:

- Accepts the **diameter** and **price** of two pizzas.
- Calculates the **area** of each pizza using the formula:

$$\text{Area} = \pi \times (\text{radius})^2 \quad \text{Area} = \pi \times (\text{radius})^2$$
- Determines the **cost per square inch** by dividing the price by the area.
- Compares the two costs and displays which pizza is the better buy.
 If the costs are nearly equal, the program indicates both are the same value.

problem 4 > q4.c

```
1  #include <stdio.h>
2  #include <math.h>
3
4  double calculateCostPerInch(double, double);
5
6  int main() {
7
8      double d1, p1, d2, p2;
9
10     printf("Enter diameter and price of pizza 1: ");
11     scanf("%lf %lf", &d1, &p1);
12
13     printf("Enter diameter and price of pizza 2: ");
14     scanf("%lf %lf", &d2, &p2);
15
16     double cost1 = calculateCostPerInch(d1, p1);
17     double cost2 = calculateCostPerInch(d2, p2);
18
19     printf("\nCost per sq.inch:\n");
20     printf("Pizza 1: %.4f\n", cost1);
21     printf("Pizza 2: %.4f\n", cost2);
22
23     if (fabs(cost1 - cost2) < 1e-6) {
24         printf("Both pizzas are the same value.\n");
25     } else if (cost1 < cost2) {
26         printf("Pizza 1 is the better buy.\n");
27     } else {
28         printf("Pizza 2 is the better buy.\n");
29     }
30
31     return 0;
32 }
33
34 double calculateCostPerInch(double diameter, double price) {
35
36     double PI = 22.0/7.0;
37
38     double radius = diameter / 2.0;
39     double area = PI * radius * radius;
40     return price / area;
41 }
42
```

```
"D:\code blocks projects\assig" x + v
Enter diameter and price of pizza 1: 30 200
Enter diameter and price of pizza 2: 30 300

Cost per sq.inch:
Pizza 1: 0.2829
Pizza 2: 0.4244
Pizza 1 is the better buy.

Process returned 0 (0x0)   execution time : 30.685 s
Press any key to continue.
```

```
"D:\code blocks projects\assig" x + v
Enter diameter and price of pizza 1: 40 500
Enter diameter and price of pizza 2: 50 400

Cost per sq.inch:
Pizza 1: 0.3979
Pizza 2: 0.2037
Pizza 2 is the better buy.

Process returned 0 (0x0)   execution time : 37.400 s
Press any key to continue.
```

```
"D:\code blocks projects\assig" x + v
Enter diameter and price of pizza 1: 40.1 200
Enter diameter and price of pizza 2: 40 200

Cost per sq.inch:
Pizza 1: 0.1584
Pizza 2: 0.1592
Pizza 1 is the better buy.

Process returned 0 (0x0)   execution time : 15.742 s
Press any key to continue.
```

```
"D:\code blocks projects\assig" x + v
Enter diameter and price of pizza 1: 30 200
Enter diameter and price of pizza 2: 30 200

Cost per sq.inch:
Pizza 1: 0.2829
Pizza 2: 0.2829
Both pizzas are the same value.

Process returned 0 (0x0)   execution time : 10.083 s
Press any key to continue.
```

Code Explanation

Input:

The user enters the **diameter** and **price** for each of the two pizzas.

`calculateCostPerInch()` Function:

- Converts diameter to radius.
- Computes area of the pizza.
- Returns the cost per square inch by dividing price by area.

Comparison Logic:

- Compares the two costs using `fabs()` to handle floating-point precision.
 - Prints which pizza is the better buy, or whether they are equal in value.
-

Problem 5

Problem Description

The goal is to write a C program that reads a string from the user and finds the **length of the longest substring** that contains **no repeating characters**.

A **substring** is a continuous sequence of characters in the string. For example:

- Input: `"abcabcbb"` → Output: `3` (Longest: `"abc"`)
- Input: `"bbbbbb"` → Output: `1` (Longest: `"b"`)
- Input: `"pwwkew"` → Output: `3` (Longest: `"wke"`)

The program should scan the string and evaluate all possible substrings to find the maximum length without repeated characters.

problem 5.7 problem 5.c

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5      char str[1000];
6      printf("Enter a string: ");
7      scanf("%s", str);
8
9      int maxLength = 0;
10     int start = 0;
11
12     for (int i = 0; str[i] != '\0'; i++) {
13         int used[256] = {0};
14         int length = 0;
15         for (int j = i; str[j] != '\0'; j++) {
16             if (used[(unsigned char)str[j]] == 1) {
17                 break;
18             } else {
19                 used[(unsigned char)str[j]] = 1;
20                 length++;
21             }
22         }
23         if (length > maxLength) {
24             maxLength = length;
25         }
26     }
27     printf("Longest substring without repeating characters = %d\n", maxLength);
28     return 0;
29 }
```

Output

```
Enter a string: abcabcbb
Longest substring without repeating characters = 3
```

=== Code Execution Successful ===

Output

```
Enter a string: bbbb
Longest substring without repeating characters = 1
```

=== Code Execution Successful ===

Output

```
Enter a string: pwwkey
Longest substring without repeating characters = 4

=== Code Execution Successful ===
```

Code Explanation

Input Handling:

- The program reads a string from the user using `scanf`.

Main Logic:

- For every possible **starting index** `i` in the string:
 - It uses a `used[256]` array to keep track of which characters have been seen.
 - It loops from `i` forward, adding characters until a duplicate is found.
 - Keeps track of the length of this substring.
 - Updates `maxLength` if the current length is the longest found.

Output:

- Prints the length of the **longest substring** without repeating characters.

Efficiency:


- This is a simple and intuitive $O(n^2)$ approach — good for learning and small inputs.
-

Problem 6

Problem Description

This task is to implement a function in C that simulates the behavior of the standard `atoi()` function — which converts a string into an integer. The program should handle the following:

- **Ignore leading whitespaces**
- **Handle optional '+' or '-' sign**
- **Convert digits into an integer until a non-digit is found**
- **Clamp the result within the range of a 32-bit signed integer**
(from `-231` to `231 - 1`)

m 6 >  problem 6.c

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

int myAtoi(char *s) {
    int i = 0;
    int sign = 1;
    long result = 0;
    while (s[i] == ' ') {
        i++;
    }
    if (s[i] == '+' || s[i] == '-') {
        if (s[i] == '-') {
            sign = -1;
        }
        i++;
    }

    while (isdigit(s[i])) {
        result = result * 10 + (s[i] - '0');

        if (result * sign >= INT_MAX) return INT_MAX;
        if (result * sign <= INT_MIN) return INT_MIN;

        i++;
    }
    return (int)(result * sign);
}

int main() {
    char *tests[] = {
        "42",
        "-042",
        "1337cod3",
        "0-1",
        "words and 987"
    };
    for (int i = 0; i < 5; i++) {
        printf("Input: \"%s\" → Output: %d\n", tests[i], myAtoi(tests[i]));
    }

    return 0;
}

/* This program solves the problem of converting a string into an integer. The
```



```
Output
Input: "42" → Output: 42
Input: " -042" → Output: -42
Input: "1337cod3" → Output: 1337
Input: "0-1" → Output: 0
Input: "words and 987" → Output: 0

=== Code Execution Successful ===
```

Code Explanation

Skip Leading Spaces:

- The program starts by ignoring all spaces at the beginning of the input string.

Check for Sign:

- If the next character is '+' or '-', it records the sign accordingly.

Digit Extraction:

- It reads each digit and forms the integer using `result = result * 10 + digit`.

Clamp if Out of Range:

- If the result goes beyond `INT_MAX` or `INT_MIN`, it returns the clamped limit.
-

Problem 7

Problem Description

You are asked to write a C program that finds the **median** of two sorted arrays. The median is:

- The **middle value** if the total number of elements is **odd**.
- The **average of the two middle values** if the total is **even**.

The arrays are sorted in ascending order and are entered by the user.

```

7 >  problem7.c
#include <stdio.h>

double findMedianSortedArrays(int* nums1, int nums1Size, int* nums2, int nums2Size) {
    double s;
    int b;
    int t[nums1Size+nums2Size];
    int i=0; int j=0; int k=0;
    // Merging two sorted arrays
    while((i<nums1Size)&&(j<nums2Size))
    {
        if(nums1[i]<nums2[j])
        {
            t[k]=nums1[i];
            i++;
            k++;
        }
        else
        {
            t[k]=nums2[j];
            j++;
            k++;
        }
    }
    // If there are remaining elements in nums1
    if(i== nums1Size)
    {
        while(j<nums2Size)
        {
            t[k]=nums2[j];
            j++;
            k++;
        }
    }
    // If there are remaining elements in nums2
    if(j== nums2Size)
    {
        while(i<nums1Size)
        {
            t[k]=nums1[i];
            i++;
            k++;
        }
    }
    // Calculate the median
    if((nums1Size+nums2Size)%2!=0)
    {
        // Odd length, return the middle element
        b=t[(nums1Size+nums2Size)/2];
        s=(double)b;
        return(s);
    }
    else
    {
        // Even length, return the average of the two middle elements
        s=((double)((t[(nums1Size+nums2Size)/2])+(t[(nums1Size+nums2Size)/2-1])))/2;
        return(s);
    }
}

```

```

52 }
53 int main() {
54     int size1, size2;
55
56     // Get first array from user
57     printf("Enter size of first sorted array: ");
58     scanf("%d", &size1);
59     int nums1[size1];
60     printf("Enter %d elements for first sorted array (in ascending order):\n", size1);
61     for (int i = 0; i < size1; i++) {
62         scanf("%d", &nums1[i]);
63     }
64
65     // Get second array from user
66     printf("Enter size of second sorted array: ");
67     scanf("%d", &size2);
68     int nums2[size2];
69     printf("Enter %d elements for second sorted array (in ascending order):\n", size2);
70     for (int i = 0; i < size2; i++) {
71         scanf("%d", &nums2[i]);
72     }
73
74     // Calculate and display median
75     double result = findMedianSortedArrays(nums1, size1, nums2, size2);
76     printf("Median: %.1f\n", result);
77
78     return 0;
79 }
80

```

```

PS C:\Users\yahya\Yahya VS> & 'c:\Users\yahya\.vscode\extensions\ms-vscode.cpptools-1.26.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-ejcgpt2m.5xe' '--stdout=Microsoft-MIEngine-Out-2iq41gsd.i3u' '--stderr=Microsoft-MIEngine-Error-mosy132s.qzr' '--pid=Microsoft-MIEngine-Pid-2paopnj.ess' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Enter size of first sorted array: 3
Enter 3 elements for first sorted array (in ascending order):
2
4
8
Enter size of second sorted array: 2
Enter 2 elements for second sorted array (in ascending order):
1
3
Median: 3.0
PS C:\Users\yahya\Yahya VS>

```

```

PS C:\Users\yahya\Yahya VS> & 'c:\Users\yahya\.vscode\extensions\ms-vscode.cpptools-1.26.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-x141fygg.fro' '--stdout=Microsoft-MIEngine-Out-00yyysa1.iu1' '--stderr=Microsoft-MIEngine-Error-hcvylrmh.sb1' '--pid=Microsoft-MIEngine-Pid-0qjxitzy.dgv' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Enter size of first sorted array: 2
Enter 2 elements for first sorted array (in ascending order):
1
3
Enter size of second sorted array: 1
Enter 1 elements for second sorted array (in ascending order):
2
Median: 2.0
PS C:\Users\yahya\Yahya VS>

```

```

PS C:\Users\yahya\Yahya VS> & 'c:\Users\yahya\.vscode\extensions\ms-vscode.cpptools-1.26.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-x141fygg.fro' '--stdout=Microsoft-MIEngine-Out-rggagfi2.f4g' '--stderr=Microsoft-MIEngine-Error-52bghmew.lro' '--pid=Microsoft-MIEngine-Pid-zxrcc1o.x0e' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Enter size of first sorted array: 2
Enter 2 elements for first sorted array (in ascending order):
1
2
Enter size of second sorted array: 2
Enter 2 elements for second sorted array (in ascending order):
3
4
Median: 2.5
PS C:\Users\yahya\Yahya VS>

```

Code Explanation

Input Stage:

- The user enters two arrays (already sorted in ascending order).

Merging:

- A new array `t[]` is used to merge the two arrays into one sorted array using the two-pointer technique.

Finding the Median:

- If the total number of elements is **odd**, the middle element is returned.
- If the total number is **even**, the average of the two middle elements is calculated and returned.

Output:

- The program prints the median rounded to **one decimal place**.
-

Problem 8

Problem Description

The goal is to write a C program that finds the **first** and **last position** of a given target value in a **sorted array** using an efficient approach.

If the target is **not found**, the program should return `[-1, -1]`.

- Time complexity must be **$O(\log n)$** .
- This is ideal for **binary search**, which repeatedly divides the search space in half.

Problem 8 >  problem 8.c

```
1  #include <stdio.h>
2
3  // Function to find first and last position of a target using binary search
4  void searchRange(int nums[], int numsSize, int target, int result[2]) {
5      int left = 0, right = numsSize - 1;
6      result[0] = -1;
7
8      // Find first occurrence
9      while (left <= right) {
10         int mid = left + (right - left) / 2;
11         if (nums[mid] == target) {
12             result[0] = mid;
13             right = mid - 1;
14         } else if (nums[mid] < target) {
15             left = mid + 1;
16         } else {
17             right = mid - 1;
18         }
19     }
20
21     left = 0;
22     right = numsSize - 1;
23     result[1] = -1;
24
25     // Find Last occurrence
26     while (left <= right) {
27         int mid = left + (right - left) / 2;
28         if (nums[mid] == target) {
29             result[1] = mid;
30             left = mid + 1;
31         } else if (nums[mid] < target) {
32             left = mid + 1;
33         } else {
34             right = mid - 1;
35         }
36     }
37 }
38
39 int main() {
40     int result[2];
41
42     int nums1[] = {5, 7, 7, 8, 8, 10};
43     searchRange(nums1, sizeof(nums1) / sizeof(nums1[0]), 8, result);
44     printf("Input: nums = [5,7,7,8,8,10], target = 8 Output: [%d, %d]\n", result[0], result[1]);
45
46     searchRange(nums1, sizeof(nums1) / sizeof(nums1[0]), 6, result);
47     printf("Input: nums = [5,7,7,8,8,10], target = 6 Output: [%d, %d]\n", result[0], result[1]);
48
49     int nums2[] = {}; // empty array
50     searchRange(nums2, 0, 0, result);
51     printf("Input: nums = [], target = 0 Output: [%d, %d]\n", result[0], result[1]);
52
53     return 0;
54 }
```

```

59
60  There are many algorithms that achieve logarithmic performance, but here I chose Binary Search because it's simple, efficient, and
61
62  Binary Search is a powerful algorithm used to find a specific element in a sorted array or List.
63  Instead of scanning elements one by one (as in linear search), it repeatedly divides the search range in half, significantly reducing
64
65  This makes it ideal for large datasets where performance matters.
66  */
67

```

PROBLEMS 5 OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```

PS C:\Users\youse\OneDrive\Desktop\Ai for business> & 'c:\Users\youse\.vscode\extensions\ms-vscode.cpptools-1.26.3-win32-x64\debugAdapters\bin\
\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-fibnwukb.3xa' '--stdout=Microsoft-MIEngine-Out-xymgkpfb.uo0' '--stderr=Microsoft-MIEngine-Error-0zmtgcn.3lt' '--pid=Microsoft-MIEngine-Pid-ofwgywp2.cca' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Input: nums = [5,7,7,8,8,10], target = 8 → Output: [3, 4]
Input: nums = [5,7,7,8,8,10], target = 6 → Output: [-1, -1]
Input: nums = [], target = 0 → Output: [-1, -1]
PS C:\Users\youse\OneDrive\Desktop\Ai for business>

```

Code Explanation

Binary Search Logic (Twice):

- First, to find the **leftmost (first) position** of the target.
- Second, to find the **rightmost (last) position** of the target.

Why Binary Search?

- Binary search divides the array in half each time, achieving **$O(\log n)$** time complexity.
- It's highly efficient, especially for large sorted arrays.

Handling Edge Cases:

- If the array is empty or the target doesn't exist → returns `[-1, -1]`.

Problem 9


Problem Description

Given a **sorted array** of distinct integers and a **target value**, write a C program that returns the **index** of the target if it's found.

If the target is **not present**, return the **index where it would be inserted** in order to maintain the sorted order.

🚩 Required time complexity: **$O(\log n)$**

🔍 Best suited for a **binary search** approach.

n 9 >  problem 9.c

```
#include <stdio.h>

int searchInsert(int nums[], int numSize, int target) {
    int left = 0, right = numSize - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (nums[mid] == target)
            return mid;
        else if (nums[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return left;
}

int main() {
    int nums1[] = {1, 3, 5, 6};

    int index;

    // Test case 1: target = 5 → Output: 2
    index = searchInsert(nums1, 4, 5);
    printf("Input: nums = [1,3,5,6], target = 5 → Output: %d\n", index);

    // Test case 2: target = 2 → Output: 1
    index = searchInsert(nums1, 4, 2);
    printf("Input: nums = [1,3,5,6], target = 2 → Output: %d\n", index);

    // Test case 3: target = 7 → Output: 4
    index = searchInsert(nums1, 4, 7);
    printf("Input: nums = [1,3,5,6], target = 7 → Output: %d\n", index);

    return 0;
}
```

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\youse\OneDrive\Desktop\Ai for business> & 'c:\u
\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-ov
gine-Error-of13j5uk.c1b' '--pid=Microsoft-MIEngine-Pid-pfuts
Input: nums = [1,3,5,6], target = 5 → Output: 2
Input: nums = [1,3,5,6], target = 2 → Output: 1
Input: nums = [1,3,5,6], target = 7 → Output: 4
PS C:\Users\youse\OneDrive\Desktop\Ai for business>
```

Code Explanation

What It Does:

- Performs a binary search on the sorted array to locate the `target`.
- If found → returns its **index**.
- If not → returns the index where the target **should be inserted**.

Binary Search Logic:

- `left` and `right` define the current search range.
- If the loop ends without finding the target, `left` will be at the correct insert position.

Time Complexity:

- The binary search ensures **$O(\log n)$** performance, making it efficient for large arrays.
-

Problem 10

Problem Description

You are given two binary strings `a` and `b`. The task is to write a C program that **adds these two binary numbers** and returns the result as a binary string.

Example inputs and outputs:

- Input: `a = "11"`, `b = "1"` → Output: `"100"`
- Input: `a = "1010"`, `b = "1011"` → Output: `"10101"`

```
1  #include <stdio.h>
2  #include <string.h>
3  char* addBinary(char* a, char* b) {
4      static char result[10005];
5      int i = strlen(a) - 1;
6      int j = strlen(b) - 1;
7      int carry = 0;
8      int k = 0;
9      while (i >= 0 || j >= 0 || carry) {
10         int bitA = 0;
11         int bitB = 0;
12
13         if (i >= 0) {
14             bitA = a[i] - '0';
15             i--;
16         }
17
18         if (j >= 0) {
19             bitB = b[j] - '0';
20             j--;
21         }
22
23         int sum = bitA + bitB + carry;
24
25         if (sum == 0) {
26             result[k] = '0';
27             carry = 0;
28         }
29         else if (sum == 1) {
30             result[k] = '1';
31             carry = 0;
32         }
33         else if (sum == 2) {
34             result[k] = '0';
35             carry = 1;
36         }
37         else if (sum == 3) {
38             result[k] = '1';
39             carry = 1;
40         }
41
42         k++;
43     }
44     for (int x = 0; x < k / 2; x++) {
```

```

44     for (int x = 0; x < k / 2; x++) {
45         char temp = result[x];
46         result[x] = result[k - 1 - x];
47         result[k - 1 - x] = temp;
48     }
49
50     result[k] = '\0';
51     return result;
52 }
53
54 int main() {
55     printf("Sum of 11 and 1 = %s\n", addBinary("11", "1"));
56     printf("Sum of 1010 and 1011 = %s\n", addBinary("1010", "1011"));
57     return 0;
58 }
59

```

```

15     i--;
16 }
17
18 if (j >= 0) {
19     bitB = b[j] - '0';
20     i--;

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```

PS C:\Users\youse\OneDrive\Desktop\Ai for business> & 'c:\Users\youse\.vscode\extensions\ms-vscode.cpptools-1.26.3-win32-x64\debugAdapters\bin\
\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-bxghbkyg.n24' '--stdout=Microsoft-MIEngine-Out-gprykob1.x33' '--stderr=Microsoft-MIEn
gine-Error-ra3n2fhd.ov1' '--pid=Microsoft-MIEngine-Pid-fvtzhjhx.eqq' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Sum of 11 and 1 = 100
Sum of 1010 and 1011 = 10101
PS C:\Users\youse\OneDrive\Desktop\Ai for business>

```

Code Explanation

Input:

- Two binary strings `a` and `b` (e.g., `"1010"` and `"1011"`).

Logic:

- Start from the **end of both strings** (least significant bit).
- Use a **carry** variable to keep track of binary carry-over.
- Add bit by bit:
 - `0+0 = 0`, `1+0 = 1`, `1+1 = 10`, etc.
- Store result in reverse order, then reverse it back at the end.

Edge Handling:

- The loop continues until all bits are processed and no carry remains.
- The result is reversed before returning since we built it from LSB to MSB.

Problem 11 (Bouns)

The objective is to determine whether a given **word** can be found in a **2D grid of characters**.

The word must be formed by **adjacent letters** in the grid — where adjacent means **horizontally or vertically neighboring**.

Each cell can only be used **once per word construction**.

Examples:

- Input: "ABCCED" → Output: true
- Input: "SEE" → Output: true
- Input: "ABCB" → Output: false

m 11 > C problem 11.c

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

char board[3][4] = {
    {'A', 'B', 'C', 'E'},
    {'S', 'F', 'C', 'S'},
    {'A', 'D', 'E', 'E'}
};

bool visited[3][4];

bool dfs(int i, int j, char* word, int index) {
    if (word[index] == '\0') return true;
    if (i < 0 || i >= 3 || j < 0 || j >= 4) return false;
    if (visited[i][j]) return false;
    if (board[i][j] != word[index]) return false;

    visited[i][j] = true;

    bool found = dfs(i+1, j, word, index+1) ||
                 dfs(i-1, j, word, index+1) ||
                 dfs(i, j+1, word, index+1) ||
                 dfs(i, j-1, word, index+1);

    visited[i][j] = false;
    return found;
}

bool exist(char* word) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            if (dfs(i, j, word, 0)) return true;
        }
    }
    return false;
}

int main() {
    printf("ABCCED → %s\n", exist("ABCCED") ? "true" : "false");
    printf("SEE → %s\n", exist("SEE") ? "true" : "false");
    printf("ABCB → %s\n", exist("ABCB") ? "true" : "false");
    return 0;
}
```

```
PS C:\Users\youse\OneDrive\Desktop\Ai for business>
her.exe' '--stdin=Microsoft-MIEngine-In-mdazfeii.u1y
rosoft-MIEngine-Pid-b2itvxttd.fpx' '--dbgExe=C:\msys64
ABCCED → true
SEE → true
ABCB → false
```


Code Explanation

Main Logic:

- The function `exist()` tries to start the word from every cell on the board.
- For each starting point, it uses **DFS (depth-first search)** to explore all valid directions (up/down/left/right).

DFS Function:

- Matches one character at a time.
 - Uses a `visited[][]` array to avoid reusing the same cell.
 - Uses recursion to explore all 4 directions.
 - Backtracks if a path fails by unmarking the visited cell.
-

Palindrome problem (Bouns)

```

ouns (palindrome) > C main.c
1  #include <stdio.h>
2  #include <math.h>
3  #include <stdbool.h>
4
5  bool isPalindrome(char str[]);
6
7  int main()
8  {
9
10     char str[] = "";
11
12     scanf("%s", str);
13
14     printf("%s", isPalindrome(str)? "Palindrome" : "not Plindrome");
15
16     return 0;
17 }
18
19 bool isPalindrome(char str[]) {
20
21     int i = 0;
22     int j = strlen(str) - 1;
23
24     while (i < j) {
25         if (str[i] != str[j]) {
26             return false;
27         }
28         i++;
29         j--;
30     }
31     return true;
32 }
33

```

```

D:\code blocks projects\assig
+ v
abaaba
Palindrome
Process returned 0 (0x0)   execution time : 42.656 s
Press any key to continue.

```

```
file
ner
rce
'D:\code blocks projects\assig
mom
Palindrome
Process returned 0 (0x0)   execution time : 5.853 s
Press any key to continue.
```

```
t1
badr
not Plindrome
Process returned 0 (0x0)   execution time : 2.510 s
Press any key to continue.
```