Table of Contents

## 0.1 FunctionDef findMax(arr[], size)

# 1 Function: findMax(int arr[], int size)

## 1.1 Overview

The findMax function iterates through an array of integers to find and return the largest value.

## 1.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| arr | int[] | An array of integers from which the maximum value will be found. |
| size | int | The total number of elements in the arr array. |

## 1.3 Description

This function provides a straightforward way to determine the maximum value within an integer array.

The logic begins by initializing a local integer variable, max, with the value of the first element of the input array, arr[0]. This value serves as the initial benchmark for comparison.

Next, the function enters a for loop that iterates through the array, starting from the second element (at index 1) up to the element just before the specified size. Inside the loop, each element arr[i] is compared to the current value stored in max. If the current element arr[i] is greater than max, the max variable is updated to hold this new, larger value.

After the loop completes, having checked all elements from the second to the last, the function returns the final value of max. This returned value is the largest integer found in the entire array.

## 1.4 Usage Notes

* The input array arr must not be empty (i.e., size must be at least 1). The function assumes arr[0] is a valid element to initialize the max value. Passing an empty array will result in undefined behavior.
* The size parameter must accurately reflect the number of elements in the array to prevent out-of-bounds memory access and ensure correct results.

**Output Example**: The function returns a single integer representing the maximum value found in the array. For an input array {10, 5, 45, 12, 8}, the output would be 45.

## 1.5 Example

#include <stdio.h>  
  
// The findMax function definition  
int findMax(int arr[], int size) {  
 if (size <= 0) {  
 // Handle empty or invalid array size  
 return -1; // Or some other error indicator  
 }  
 int max = arr[0];  
 for (int i = 1; i < size; i++) {  
 if (arr[i] > max)  
 max = arr[i];  
 }  
 return max;  
}  
  
// Example usage in a main function  
int main() {  
 int numbers[] = {10, 5, 45, 12, 8};  
 int n = sizeof(numbers) / sizeof(numbers[0]);  
   
 int maxValue = findMax(numbers, n);  
   
 printf("The maximum value is: %d\n", maxValue);  
   
 return 0;  
}

**Output:**

The maximum value is: 45

## 1.6 FunctionDef findMin(arr[], size)

# 2 Function: findMin(int arr[], int size)

## 2.1 Overview

The findMin function iterates through an array of integers to find and return the smallest value.

## 2.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| arr | int[] | The array of integers to be searched. |
| size | int | The total number of elements in the arr. |

## 2.3 Description

This function provides a straightforward way to find the minimum value within an integer array.

The logic begins by initializing a variable min with the value of the first element of the array, arr[0]. This variable acts as a temporary placeholder for the smallest value found so far.

Next, the function enters a for loop that starts from the second element of the array (at index 1) and continues until it has iterated through all remaining elements up to the index size - 1. Inside the loop, it compares the current element, arr[i], with the value stored in min.

If the current element arr[i] is less than min, the value of min is updated to arr[i]. This process ensures that min always holds the smallest value encountered in the portion of the array that has been scanned.

After the loop completes, the function returns the final value of min, which is the absolute minimum value in the entire array.

## 2.4 Usage Notes

* The array passed to the function must not be empty. The function directly accesses arr[0] without checking if size is greater than 0. Passing an empty array will result in undefined behavior.
* The size parameter must accurately represent the number of elements in the array. Providing a size larger than the actual array length will lead to out-of-bounds memory access.
* The function is designed to work with arrays of standard integers, including both positive and negative values.

**Output Example**: The function returns a single integer representing the minimum value found.

12

## 2.5 Example

#include <stdio.h>  
  
// Definition of the findMin function  
int findMin(int arr[], int size) {  
 int min = arr[0];  
 for (int i = 1; i < size; i++) {  
 if (arr[i] < min)  
 min = arr[i];  
 }  
 return min;  
}  
  
int main() {  
 int numbers[] = {45, 21, 89, 12, 5};  
 int n = sizeof(numbers) / sizeof(numbers[0]);  
  
 // Example usage  
 int minValue = findMin(numbers, n);  
  
 printf("The minimum value is: %d\n", minValue);  
  
 return 0;  
}

**Output:**

The minimum value is: 5

## 2.6 FunctionDef calculateAverage(arr[], size)

# 3 Function: calculateAverage(int arr[], int size)

## 3.1 Overview

The calculateAverage function computes the arithmetic mean of the elements in a given integer array.

## 3.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| arr | int[] | An array of integers whose average value is to be calculated. |
| size | int | The number of elements in the arr array. |

## 3.3 Description

This function provides a straightforward way to calculate the average of a set of integers.

The logic begins by initializing an integer variable, sum, to 0. This variable serves as an accumulator for the total value of all elements in the array.

The function then iterates through the input array arr using a for loop. The loop runs from index 0 up to, but not including, the size parameter. In each iteration, the value of the current element arr[i] is added to the sum.

After the loop has processed all elements, the function calculates the final average. It performs a type cast on the sum variable, converting it to a double before dividing it by size. This step is crucial to ensure that the division is a floating-point operation, which preserves any fractional part of the result, rather than an integer division, which would truncate it. The resulting double value is then returned.

## 3.4 Usage Notes

* The size parameter must be a positive integer. Providing a size of 0 will result in division by zero, which causes undefined behavior.
* Ensure that the size parameter accurately reflects the number of elements in the arr array to avoid reading from out-of-bounds memory.
* If the sum of the array elements is expected to be very large, the sum variable might exceed the maximum value for an int. In such cases, consider modifying the function to use a larger data type like long long for the accumulator.

**Output Example**: A floating-point number representing the average, such as 6.4.

## 3.5 Example

#include <stdio.h>  
  
// The function being documented  
double calculateAverage(int arr[], int size) {  
 int sum = 0;  
 for (int i = 0; i < size; i++) {  
 sum += arr[i];  
 }  
 return (double)sum / size;  
}  
  
// Example usage in a main function  
int main() {  
 int numbers[] = {10, 2, 8, 5, 7};  
 int count = 5;  
 double average = calculateAverage(numbers, count);  
   
 printf("The average is: %f\n", average);  
   
 return 0;  
}

**Output:**

The average is: 6.400000

## 3.6 FunctionDef stringLength(str[])

# 4 Function: stringLength

## 4.1 Overview

The stringLength function calculates the length of a null-terminated string, excluding the final null character.

## 4.2 parameters

* **str** char[]: The input character array (string) that must be null-terminated.

## 4.3 Description

The stringLength function provides a manual implementation for determining the length of a C-style string. It operates by iterating through the characters of the input array until it finds the null terminator character ('\0').

The function begins by initializing an integer variable length to 0. This variable serves a dual purpose: it acts as an index to access characters in the str array and as a counter for the string’s length.

A while loop forms the core of the function. The loop’s condition checks if the character at the current index, str[length], is not the null terminator '\0'. - If the character is not '\0', the loop body executes, incrementing length by one. - This process continues, moving to the next character in the array in each iteration.

When the loop finally encounters the null terminator '\0', the condition str[length] != '\0' becomes false, and the loop terminates. At this point, the value of length is equal to the number of characters that were read before the null terminator.

Finally, the function returns the integer value of length.

// The loop iterates until the null character is found.  
while (str[length] != '\0') {  
 // The length counter is incremented for each character.  
 length++;  
}  
// The final count is returned.  
return length;

## 4.4 Usage Notes

* The input array str **must** be a valid null-terminated string. Failure to provide a null terminator will cause the function to read beyond the buffer’s boundary, leading to undefined behavior.
* The returned length does not include the null terminator character '\0'. For example, the string "hi" has a length of 2, but it occupies 3 bytes in memory ('h', 'i', '\0').
* An empty string "" will correctly return a length of 0.

**Output Example**: An integer representing the number of characters in the string. For the input "example", the output would be 7.

## 4.5 Example

The following C code demonstrates how to use the stringLength function.

#include <stdio.h>  
  
// Assuming stringLength function is defined in the same file or included  
int stringLength(char str[]) {  
 int length = 0;  
 while (str[length] != '\0') {  
 length++;  
 }  
 return length;  
}  
  
int main() {  
 char myString[] = "Hello, World!";  
 int len = stringLength(myString);  
 printf("The length of the string is: %d\n", len);  
  
 char emptyString[] = "";  
 int emptyLen = stringLength(emptyString);  
 printf("The length of the empty string is: %d\n", emptyLen);  
  
 return 0;  
}

**Output:**

The length of the string is: 13  
The length of the empty string is: 0

## 4.6 FunctionDef getRandom(min, max)

# 5 Function: getRandom(int min, int max)

## 5.1 Overview

The getRandom function generates a pseudo-random integer within a specified inclusive range.

## 5.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| min | int | The minimum possible value for the random number (inclusive). |
| max | int | The maximum possible value for the random number (inclusive). |

## 5.3 Description

This function provides a reliable way to obtain a random integer between two boundary values, min and max.

The function first ensures that the range is valid. It checks if min is greater than max. If this condition is true, it swaps the values of min and max internally. This makes the function robust, as it will work correctly even if the arguments are passed in the wrong order.

The core logic for generating the random number is in the return statement: return min + rand() % (max - min + 1);. - rand(): This standard C library function returns a pseudo-random integer. - (max - min + 1): This expression calculates the size of the desired range. For example, for a range from 5 to 10, this evaluates to 10 - 5 + 1 = 6. - rand() % (max - min + 1): The modulo operator (%) constrains the result of rand() to a number between 0 and max - min. - min + ...: This adds the min value to the result, effectively shifting the range from [0, max - min] to [min, max].

## 5.4 Usage Notes

* The range is inclusive, meaning both the min and max values are potential return values.
* The function automatically handles cases where min > max by swapping the values.
* For rand() to produce different sequences of numbers each time the program runs, it must be seeded. This is typically done once at the start of the program (e.g., in main) by calling srand(time(NULL)). You must include the <stdlib.h> and <time.h> headers for this.

**Output Example**: A single integer value within the specified range. For getRandom(1, 100), a possible output is 42.

## 5.5 Example

The following C code demonstrates how to use the getRandom function. It includes the necessary seeding for the random number generator.

#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
// Definition of the getRandom function  
int getRandom(int min, int max) {  
 if (min > max) {  
 int temp = min;  
 min = max;  
 max = temp;  
 }  
 return min + rand() % (max - min + 1);  
}  
  
int main() {  
 // Seed the random number generator once at the beginning  
 srand(time(NULL));  
  
 int min\_val = 1;  
 int max\_val = 10;  
  
 // Generate and print a random number between 1 and 10  
 int random\_number = getRandom(min\_val, max\_val);  
 printf("A random number between %d and %d is: %d\n", min\_val, max\_val, random\_number);  
  
 return 0;  
}

**Output:**

(The actual output will vary on each execution)

A random number between 1 and 10 is: 7

## 5.6 FunctionDef isPrime(n)

# 6 Function: isPrime(int n)

## 6.1 Overview

The isPrime function determines whether a given integer is a prime number.

## 6.2 Parameters

* n: The integer to be checked for primality.

## 6.3 Description

This function provides an efficient method to check if an integer n is a prime number.

The logic begins by handling the base cases. According to the definition of prime numbers, any integer less than or equal to 1 is not prime. The function first checks if n <= 1 and, if this condition is met, it immediately returns false.

If n is greater than 1, the function proceeds to check for factors. It iterates through integers starting from i = 2 up to the square root of n. The loop condition i \* i <= n is an optimization that avoids checking divisors beyond the square root of n, as any factor larger than the square root would have a corresponding factor smaller than it.

Inside the loop, it uses the modulo operator (%) to check if n is perfectly divisible by the current integer i. If n % i == 0 is true, it means a factor other than 1 and n has been found, so n is not a prime number. In this case, the function returns false and terminates.

If the loop completes without finding any factors, it confirms that n is only divisible by 1 and itself, thus it is a prime number, and the function returns true.

## 6.4 Usage Notes

* Numbers less than or equal to 1 will always return false.
* The function returns a boolean value: true if the number is prime, and false otherwise.
* This implementation is optimized to check for divisors only up to the square root of the input number, making it efficient.

**Output Example**: The function returns a boolean value, which in C might be represented as 1 (for true) or 0 (for false) when printed or used in integer contexts.

## 6.5 Example

#include <stdio.h>  
#include <stdbool.h>  
  
// Assuming the isPrime function is defined here or included  
bool isPrime(int n) {  
 if (n <= 1)  
 return false;  
 for (int i = 2; i \* i <= n; i++) {  
 if (n % i == 0)  
 return false;  
 }  
 return true;  
}  
  
int main() {  
 int num1 = 29;  
 int num2 = 10;  
  
 bool result1 = isPrime(num1);  
 bool result2 = isPrime(num2);  
  
 printf("Is %d a prime number? %s\n", num1, result1 ? "true" : "false");  
 printf("Is %d a prime number? %s\n", num2, result2 ? "true" : "false");  
  
 return 0;  
}

**Output:**

Is 29 a prime number? true  
Is 10 a prime number? false

## 6.6 FunctionDef main

# 7 Function: main

## 7.1 Overview

The main function serves as the entry point for the program, demonstrating the usage of several utility functions to find a maximum value in an array, calculate string length, and determine if a number is prime.

## 7.2 parameters

This function does not take any parameters.

## 7.3 Description

The main function executes a sequence of demonstrations for other utility functions.

1. **Find Maximum in Array**:
   * An integer array nums is initialized with the values {5, 12, 3, 19, 7}.
   * The size of the array is calculated by dividing the total size of the array in bytes (sizeof(nums)) by the size of a single integer element (sizeof(nums[0])).
   * The findMax function is called with nums and size as arguments. The returned value, which is the largest number in the array, is then printed to the console.
2. **Calculate String Length**:
   * A character array name is initialized with the string "Prateek".
   * The stringLength function is called with the name array. The function returns the number of characters in the string, which is then printed.
3. **Prime Number Check**:
   * An integer variable number is initialized with the value 17.
   * The isPrime function is called with number. The function returns a boolean-like value (true or false).
   * An if-else block checks the return value and prints a message indicating whether 17 is a prime number or not.

Finally, the function returns 0 to the operating system, which conventionally signifies that the program executed successfully without errors.

## 7.4 Usage Notes

* This main function is a driver function, designed to showcase the functionality of findMax, stringLength, and isPrime.
* For this code to compile and run, the implementations for findMax, stringLength, and isPrime must be available and linked.
* The program requires the <stdio.h> header for the printf function.

**Output Example**: A possible console output from running the program.

Maximum number: 19  
Length of 'Prateek': 7  
17 is a prime number.

## 7.5 Example

The main function is the entry point of a C program and is not called by other functions within the code. It is executed when the compiled program is run. To use it, you would compile the source file and run the resulting executable.

// Assuming the code is in a file named 'program.c'  
// and the helper functions are also defined.  
  
// 1. Compile the code using a C compiler like GCC  
// > gcc program.c -o program  
  
// 2. Run the executable  
// > ./program

**Output:**

Maximum number: 19  
Length of 'Prateek': 7  
17 is a prime number.