# WEEKLY INTERNSHIP ACTIVITY 4

**Students will implement functions using variable argument lists (stdarg.h) to understand flexible function design similar to printf-like behaviour.**

Source Code

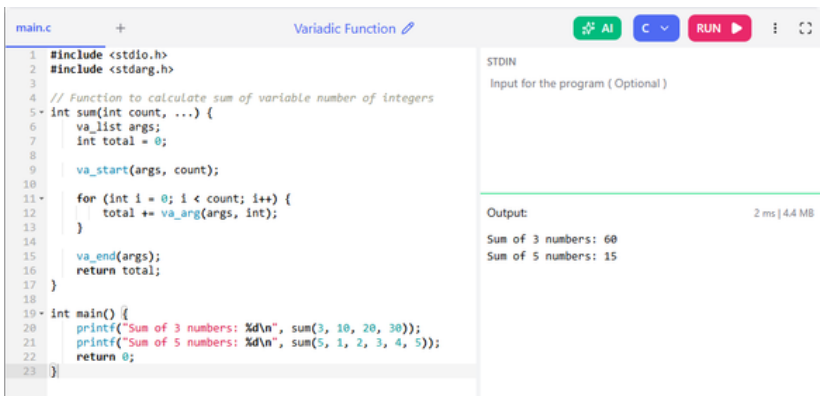
```c
#include <stdio.h>
#include <stdarg.h>

// Function to calculate sum of variable number of integers
int sum(int count, ...) {
    va_list args;
    int total = 0;

    va_start(args, count);

    for (int i = 0; i < count; i++) {
        total += va_arg(args, int);
    }

    va_end(args);
    return total;
}

int main() {
    printf("Sum of 3 numbers: %d\n", sum(3, 10, 20, 30));
    printf("Sum of 5 numbers: %d\n", sum(5, 1, 2, 3, 4, 5));
    return 0;
}
```

Execution Output

```
Sum of 3 numbers: 60
Sum of 5 numbers: 15
```

Try Online (Click to open)

- Onecompiler > c



## Explanation:

In C programming, functions normally accept a fixed number of arguments defined at compile time. However, there are situations where the number of inputs is not known beforehand. To handle such cases, C provides variable argument functions, also known as variadic functions, through the **stdarg.h** header file.

A variadic function allows a function to receive a variable number of arguments. A classic real-world example is the **printf()** function, which can take different numbers and types of arguments depending on the format string. This flexibility is essential for designing reusable and adaptable functions.

To implement a variadic function, four macros from stdarg.h are used:

- **va_list:** Declares a variable that will hold the argument list.
- **va_start:** Initializes the argument list and points it to the first variable argument.
- **va_arg:** Retrieves the next argument from the list.
- **va_end:** Cleans up the argument list.

In this program, the function **sum()** is designed to calculate the sum of a variable number of integers. The first parameter **count** specifies how many arguments follow. This is important because C does not automatically track the number of arguments passed to a variadic function.

Inside the function, a **va_list** named **args** is declared. The **va_start(args, count)** macro initializes the argument list. A loop then runs count times, retrieving each integer using **va_arg(args, int)** and adding it to the total. After processing all arguments, **va_end(args)** is called to release resources.

The **main()** function demonstrates multiple uses of the same function with different numbers of arguments. This shows how variadic functions enhance flexibility while reducing code duplication.

Variadic functions are commonly used in logging systems, debugging utilities, and formatted input/output operations. However, they must be used carefully because type checking is not enforced by the compiler. The programmer must ensure correct argument types and counts to avoid runtime errors.