1. Calculate product of two integers

Write a C program that accepts two integers from the user and calculates the product of the two integers. Test Data :

```
> Input the first integer: 25 > Input the second integer: 15 Expected Output:
> Product of the above two integers = 375
```

1.1 Solution

```c
#include <stdio.h>

#define HEIGHT 7
#define WIDTH 5
#define RADIUS 6
#define M_PI 3.14159265358979323846

// global scope variable declaration
int num1, num2, result;

```

Listing 1: 1.1 Code Solution

when you declaring a number using a `#define`, It has no type. It is a simple text substitution. although PHI is already defined by C itself, but i found on a stack overflow thread, they were defining PHi with `#define`. this kind of problem can be tackled easyly by creating a separate *function* outside of `main()` function, and declaring those variable using *a global scope variable* for efficiency and avoiding conflict within other function.

1.2 Pros And Cons

by storing a number in `#define`, it's not the best practice in the modern C. because as i said, **It has no data types, hard to debug, there's no scope, and etc.** i'd reccomend using a `const` variable instead. for example: `const double PI = 3.1415926535;`. a `#define` is more suitable for storing a macro rather than a number for mathematical operation unless you know what are you doing.

2. Calculate average weight for purchases

Write a C program that accepts two item's weight and number of purchases (floating point values) and calculates their average value. for example:

```
Weight - Item1: 15, No. of item1: 5, Weight - Item2: 25, No. of item2: 4,
Expected Output: Average Value = 19.444444
```

2.1 Solution

by using math shit for this problem, i've figured out using average value for calculating the result, we can implement this solution to our code:

$$\text{Average Value} = \frac{(w_1 \times n_1) + (w_2 \times n_2)}{n_1 + n_2}$$

By creating a separate function and declaring the required variable first using `double` data types, finally we can implement the math formula above to our code solution.

```c
void CalculatePurchases(){
  double wi1, ci1, wi2, ci2, resultDouble;

  printf("Enter Weight for item 1: ");
  scanf("%lf", &wi1);

  printf("Enter Weight for ci1: ");
  scanf("%lf", &ci1);

  printf("Enter Weight for weight item 2: ");
  scanf("%lf", &wi2);

  printf("Enter Weight for ci2: ");
  scanf("%lf", &ci2);

  resultDouble = ((wi1 * ci1) + (wi2 * ci2)) / (ci1 + ci2);
  printf("The result is %lf\n", resultDouble);
}

```

Listing 2: 1.2 Code Solution

Why does* `scanf()` *require to use & ? well, It needs to change the variable. Since all arguments in C are passed by value you need to pass a pointer if you want a function to be able to change a parameter. Here's a super-simple example showing it:

```c
void nochange(int var) {
// Here, var is a copy of the original number. &var != &value
var = 1337;
}
void change(int *var) {
    // Here, var is a pointer to the original number. var == &value
    // Writing to `*var` modifies the variable the pointer points to
    *var = 1337;
}
int main() {
    int value = 42;
    nochange(value);
    change(&value);
    return 0;
}

```

Listing 3: 1.3 scanf example

C function parameters are always "pass-by-value", which means that the function scanf only sees a copy of the current value of whatever you specify as the argument expression. In this case &i is a pointer value that refers to the variable i. scanf can use this to modify i. If you passed i, then it would only see an uninitialized value, which (a) is UB, (b) is not sufficient information for scanf to know how to modify i.