why we use int instate of char in case 'c':
printf("%c", va_arg(argu, int));
break;

In variable argument list functions (varargs) like the one using `va_arg`, `char` and `short` integer types are automatically promoted to `int` when passed to the function. This is part of the default argument promotions in C's calling conventions for variadic functions.

These promotions occur because the C standard specifies that when you have a function with variable arguments (those using ellipsis `...`), the default argument promotions are applied before the function call is made. For variadic functions, there's no function prototype that specifies types for the ellipsis, so the compiler doesn't have any information about what types the function will receive beyond those that are explicitly specified. To make it more consistent and efficient for the machine's calling conventions, smaller types like `char` are promoted to `int`.

Therefore, when you use `va_arg` to retrieve the next argument of a `char` type, you should use `int` and then cast it back to `char` if necessary. The same principle applies to `float`, which is promoted to `double`, so you would retrieve it with `va_arg(argu, double)` even if the original argument was a `float`.

Here's the relevant part of your code, with an inline comment explaining it:

```
case 'c':
    // char is promoted to int, so retrieve it as int
    printf("%c", va_arg(argu, int));
    break;
```

This is why even though you are expecting a `char`, you use `va_arg(argu, int)` and pass it to `printf` with `%c`. The `printf` function then correctly interprets it as a `char` because of the `%c` format specifier.

From ChatGPT, designed with FancyGPT