

Named argument view for IDE and more

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2004-04/2828.html

From: Booted Cat (coolspeech_at_hotmail.com)

Date: 04/23/04

Date: 23 Apr 2004 13:37:16 -0700

I've seen lots of discussions on the proposed inclusion of "function call with named arguments" to C/C++ on these newsgroups. My proposal is slightly different in that:

- * No ANSI approval is needed
- * No conflicts with existing language features such as function overloading
- * No need to modify the language spec or the compiler
- * No need to modify your source code

As you may have guessed, I think it possible to make this a feature of the IDE.

A first thought was to keep the argument order as specified by the function declaration, but allow the editing and viewing of function arguments in a named manner. That is to say, the argument list is displayed as a closeable table immediately after every function call. The table include argument names and fields where you can view or fill the value or expression for each argument. But the tables are just visual effects of the code editor window, and the source code is stored in the original format in memory and on disk.

If you also want customized argument order, the order information can be saved by the IDE as special comments in the source file, or as a separate non-source file included by your project file.

Another idea of mine is also about programming language design, although a bit theoretical. It's about a freer syntax. The idea comes from my research on controlled language design and translation (the BabelCode methodology); controlled natural languages and programming languages have something in common that is they're both formal languages. We know modern programming language design is evolving toward a goal of "programming like speaking a natural language"; OOP has unleashed the most of the semantic potential of programming languages, but the syntactic freedom still has room to improve.

A typical C function call (enhanced by the "named argument" feature) is like this:

```
myfunc(arg1 value1, arg2 value2, ...);
```

Compared to English syntax, myfunc is like the main verb of a sentence, and argvalues are the subject, the objects (direct and indirect, argument name omitted unless in passive voice), prepositional phrases (argument names are the preposition, values are the prepositional objects), adverbials (presence of argument name is a flag delivering a boolean value) and nonfinite verb phrases (passing function pointers). This is like the freeform case-based natural languages such as Russian and German. In OOP programming languages like C++, the "subject" or "object" argument is isolated and put before the "main verb", the other arguments being remained in the argument list:

```
subject.myfunc(arg2 value2, arg3 value3, ...);
```

This way it's more like how we form a natural language sentence. But note that this is only in favor of English-like syntax. Many natural languages even including English prefer to place some prepositional phrases before the main verb. The Chinese language prefer:

```
subject.(arg2 value2, arg3, value3, ..., myfunc objvalue, ...);
```

Russian and German prefer the "good old C function call" style:

```
.(arg2 value2, subjvalue, arg3 value3, myfunc, ..., OBJ objvalue, ...);
```

What's worse, many Chinese monolingual speakers can't distinguish the v.t. part-of-speech from the prep part-of-speech. In their eyes, these are both "relational words with an object and belonging to a predicate" so they're likely to take a PP (viewed from English) such as "with" for the main verb. This makes me to try break the POS difference between prep and vt and treat them as equal entities. So Chinese speakers may prefer this syntax:

```
subject.argx(valuex, arg1 value1, arg2 value2, ..., myfunc objvalue, ...);
```

Some programming languages are promoting the idea of "everything is an object", and here "argument names" also become objects, at least with equal syntactic status to that of the function name.

The above discussion shows that:

1. The SUBJECT argument may not necessarily come first;
2. Virtually any argument can play the role of "main verb" (function name);
3. There may not necessarily be a distinct place for "main verb", in languages that sometimes don't distinguish verbs from preps.
4. The order of arguments should be able to vary.

Handling such function call syntax is technically possible as long as the real "main verb" identifier can be found.

Although this syntactic freedom is more significant to my controlled language authoring and translation project, it does have some use in programming languages. Take an example from the C library function `memcpy`:

```
void *memcpy(  
    void *dest,  
    const void *src,  
    size_t count  
);
```

We would be able to call it in these ways:

```
memcpy(to p2; from p1; amount count); // the original order  
memcpy(amount count; from p1; to p2); // the english order  
(from p1; memcpy amount count; to p2); // the chinese order  
...
```

Yao Ziyuan