Subject: Re: New packages announcement
Posted by piotr5 on Mon, 15 Jun 2009 11:58:00 GMT
View Forum Message <> Reply to Message

I really don't want to go off-topic. a discussion of which language is better or which philosophy has its merits is beyond this thread and beyond my own knowledge. so I probably should repeat what I just wrote above:

c is designed in a way that it is impossible to implement type-checking without breaking working code. if you want that, then you need to extend c++ or similar (like the java8 that was suggested). even python is a better starting-point than c! 0-terminated strings are theoretically faster than anything else
since on processor-level testing for zero takes much less ressources than comparison or even a seperate counter. therefore I can understand the design-decision of c. if you want working code, don't write in c, it's easy. and as far as I learned in school, it is absolutely impossible to create a code-checker which would be able to predict all crashes. I think people writing programs in old c has nothing to do with desire for working or non-working code, I rather think it's a personal decision on how much influence one wants to have on the compilation-process vs. having all the gory details hidden. seemingly programmers don't want to get the magic hidden from them. my choice would be a more low-level language than c, but apart from assembler there doesn't exist anything. you seem to seek something which is higher-level than c, and here you have a very large range of programming-languages to test out. that's why I fail to understand why you wish to create yet another one. with so many theoretical concepts on how a programming language should look like, all around the world, you can never claim that you know them all -- even less that what you plan to create wouldn't already exist somewhere. my favourite programming-language was Sather (because of its innovative use of iterators and its c-alike philosophy). then I noticed that all the stuff I liked about it is already present in c++ (one can write std::accumulate(it1,it2,0) and similar things to get what sather has managed with its iterator-concept). I liked Sather because it's low-level and because it requires shorter sourcecode (than assembler). in upp I can see this is also a viable goal for c++. I did know of c++ before knowing of Sather, I just didn't know it well enough to judge. but naturally, I really see no advantage of c++ over c except for the shorter source-code, so if you want to extend it in any direction it can only be an improvement. but I see no use in extending c into a higher level language. again my question: why did you choose c as a basis? the stuff about bison and yacc I have only written because I suspected this might be your plan, but seemingly you prefer your own parsers instead of reusing other people's work. what is it that you want? why do you think a c-parser could help? as I implied, I have strong feelings against the c-command "for" and all other loop-commands, for all the reasons you mentioned and much more. do you feel the same? is that the reason? in my experience one can hide most loops in c++, and if there are no more explicit loops in the main-program, then also code-checkers will reliably work...

btw, a code-checker would be also a good application of tcc. this is my major reason why I write all this stuff on programming-languages in this thread. when I manage to hide all loops in the used libraries and put my code through some specialized code-checker (which I would like to write), then there is absolutely no possibility to shoot into the own foot -- instead the libraries will take over that duty...