Subject: Re: New packages announcement
Posted by cbpporter on Mon, 15 Jun 2009 06:42:19 GMT
View Forum Message <> Reply to Message

piotr5 wrote on Sun, 14 June 2009 22:53I think the c-extension is a bad idea. there are a lot of things possible in c, from objects and abstract interfaces, right down to iterators and whatever programming-paradigm. what we really need is a replacement for bison and yacc. what do you plan to put into your c-language, cbpporter? what do you feel is missing in c? as I am saying, bison or yacc are supposed to create c-files, and a slow incremental non-breaking development would fit those tools better than any c-slang. whenever some file or script needs to be parsed, there is a major problem with redundancy within the sources of a parsing-program. solve that! c is really not missing anything (except maybe for type-checking, but his would break code and is already covered by c++). please show me that I am wrong. from the point of view of assembler there are several things one could add to c. for example an inlined 2-log realized in assembler would be nice -- unfortunately not every machine-language has the fitting op-code. and what I really miss about c is a way to tell the compiler how it should optimize my code. for example if the compiler doesn't know of the low-level load-zero optimization (i.e. whenever a register has to contain a zero-value then it's faster to xor the register with itself), if it really does load an explicit "0" from memory to the registers whenever it is asked to, then there is no possibility in c to tell the compiler to apply that optimization! in such a case one is forced to switch to a different compiler as the c-standard has no possibility of achieving an influence on pre-processing and post-processing of the program. I'm just not sure how such a intrusion into the compiler's competences could be implemented...

as for tcc, it's really a good idea. a good application would be to create a graphical front-end to a c-interpreter. preferably in the style of upp with the possibility to browse the various libraries and headers and maybe occasional sources of them, and to look up things in various documentations. if only I would understand upp-sources better, I could transform it into such a beast. instead of source-packages there would be library-packages and program-interfaces (for already running programs, in the style of a debugger). still, I'm not ready yet, and I don't have enough time. I really can't help now...
So first of all I have no illusions that I'm going to change C. This is not the plan. Nor is it to make even a single person except for me of course while testing use this language. The plan for now is to do a semi-formal design paper, and maybe some day implement it. The implementation would quite short and easy, seeing that the design is layered on top of C and C would be doing all the heavy lifting like optimization.

But C is missing a lot. What is wrong with it? Everything! I don't want to insult nobody using C, but it is not fit for ANY sizable practical purpose. We use C, a language which makes it "easy to shoot yourself in the foot", and we say: "no problem, I'm a professional, I have learned the intricacies of C and I wont shoot myself or anyone else in the foot". And this approach works for the most part. We have almost all software written in C. But what I'm designing is a C which makes it "almost impossible to shoot yourself in the foot" and we'll say: "no problem, we're professionals, we have learned the intricacies of this language and if it is our desire we can shoot people's feet right off". The idea is that all small operations are safe, but feel free to cast them around, do freaky stuff with unions, etc. etc. if you please. When something breaks, you'll know that it's somewhere where you have bent the rules.

I don't want to spent a lot of time here talking about C or how I plan to make it safe. Maybe in a coffee corner thread. Just to enumerate some items: proper module support, real symbolic constants, arrays have sizes but at compile and run time, overall reduction of pointer use, things like memset removed, etc.

Just to give an example:
puts(s);
Even this is unsafe. As long as your string was the result of proper calls to strcat and friends you are probably safe, as long as no buffer overflow accured without you noticing until you got to puts. Or maybe someone filled s with code which does stuff manually and forgot to put the null terminator. But if arrays are strongly typed and have a length, there is no possible way you could have gotten either a buffer overflow or an invalid s. The only way to make this not work is to cast s to something it is not, and write random stuff to it. This is clearly intentional and can be caught easily. No use for expensive and bulky C code checkers which we use BTW. Also null terminated strings have proven time and time again that they have abyssal performance, but this is very easy to fix: every single piece of code that I have seen passes a pointer and a size when doing non trivial stuff. There are companies who even insist on this practice and have a new string and memory "standard" library.

As for your second concern, what is exactly wrong with Bison and Yacc? They are somewhat slow and generated code is ugly but readable. They work.

But I have used a lot of such tools and abandoned them all because once you get to an ambiguity, you generally have to jump though loops. I'm sorry, I don't want to normalize anything. When I write my hand parsers they never suffer from ambiguities. At one project I spent more time trying to make Antrl do it's job than the actual grammar.
