Subject: New packages announcement
Posted by koldo on Sat, 13 Jun 2009 22:49:45 GMT
View Forum Message <> Reply to Message

Hello all

This is just to inform you that I am working in three new packages for Bazaar, all three running in Linux and Windows (MinGW and MSC), using libraries with licenses compatible with BSD (Upp) and to be released this year:

- Tcc
Tcc (http://bellard.org/tcc/) is a C compiler so small and fast to compile that can be used as an scripting language.
The package is ready in an 80% and complies with LGPL license also for MSC.

- Matrix algebra and numerical methods
It uses BLAS and LAPACK matrix library through ATLAS, so it is LGPL compatible and just as fast as Matlab (tenths of times faster than pure C or Fortran!!). For you to know BLAS and LAPACK are for matrix algebra and linear equations solving like OpenGL or DirectX for graphics.

- Image analysis
Package focused to machine analysis and recognition more than to picture improvement (but also including functions for that).

If anybody is working on some of them and wants to join efforts or simply wants some capability just post it.

Best regards
Koldo

Subject: Re: New packages announcement
Posted by ptDev on Sun, 14 Jun 2009 12:49:53 GMT
View Forum Message <> Reply to Message

Thank you very much for all this excellent work.

Are you also planning to make a C interpreter engine/class based on TCC?

Subject: Re: New packages announcement
Posted by koldo on Sun, 14 Jun 2009 13:55:28 GMT
View Forum Message <> Reply to Message

Hello ptDev

These are the main methods in Tcc class:

```
void Compile(const char *my_program);
void AddSymbol(const char *funName, void *fun);
void *GetSymbol(const char *funName);
void Relocate();
```

And this is the detail of a demo:

```
tcc.Compile(my_program);
tcc.AddSymbol("myFunction", (void *)&myFunction);
tcc.Relocate();
double (*myMain)(char *str) = (double (*)(char *str))tcc.GetSymbol("myMain");
printf("\nThe result for Test is %.1f", myMain("Hello"));
```

where:
- my_program is a text with the source code
- myFunction is a function in the main program that can be called from the tcc program
- myMain is a function in the tcc program that can be called from the main program

It is like a dinamic linking to a program that can be done on the fly, and also this program can call to caller program functions.

Tcc also lets to do standalone programs but I have done the class to only compile and run a program in memory.

Best regards
Koldo

Subject: Re: New packages announcement
Posted by cbpporter on Sun, 14 Jun 2009 15:18:21 GMT
View Forum Message <> Reply to Message

Very interesting! I'm very close to starting a small research project in which I would like to "fix" C. The idea was to parse my "safe & enriched" C and output normal C, C being something like an universal "assembly" language. The idea was to add incremental non-breaking additions, so old C code would still be valid for several versions. I don't know when I'll have time to actually code, but by efforts with TCC could prove useful for me. With TCC maybe I don't have to parse C myself, just modify existing parser. I hope I'll have time to get this off the drawing board.

Subject: Re: New packages announcement
Posted by piotr5 on Sun, 14 Jun 2009 19:53:09 GMT
View Forum Message <> Reply to Message

I 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...

Subject: Re: New packages announcement
Posted by tojocky on Sun, 14 Jun 2009 20:11:43 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...

May be would be nice to appreciate V8 javascript elaborasted by Google? May be is sense to contribute and use embeded compile with c++?

Subject: Re: New packages announcement
Posted by Mindtraveller on Sun, 14 Jun 2009 20:45:24 GMT
View Forum Message <> Reply to Message

IMO the most important parts of choosing embedded scripting are:
1) Compile speed/memory usage
2) Binding simplicity. It must be easy to, i.e. open main program' dialog (1-2 lines of code).

May be we should choose with these criteria, not by "the only one True Language".

Subject: Re: New packages announcement
Posted by tojocky on Sun, 14 Jun 2009 21:05:13 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sun, 14 June 2009 23:45IMO the most important parts of choosing embedded scripting are:
1) Compile speed/memory usage
2) Binding simplicity. It must be easy to, i.e. open main program' dialog (1-2 lines of code).

May be we should choose with these criteria, not by "the only one True Language".
About compile speed I can't compare, but in comparison of JavaScript from Firefox, IExplorer, the V8 is more faster. Memory do not use so much Firefox, IE or Opera.
Binding is a little problems because is not so simple to bind with U++, but is more flexible!

As the 3-th point I should add the run speed!

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.

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...

Subject: Re: New packages announcement
Posted by cbpporter on Mon, 15 Jun 2009 12:29:43 GMT
View Forum Message <> Reply to Message

If I extend C I will break working code. If I extend C++ I will break working code again. This is not an issue, it is a design choice. As I said C is not a good practical language and making it backward compatible would "doom" it from the start. And I don't really want to extend it, rather than replace all unsafe features with safe ones, basically creating a new systems programming language which is as safe and clear Java and has the performance of C, but with slightly more memory consumption (arrays have sizes). The memory consumption of 8 bytes per array is not

the length along the pointer so you can optimize by structure size. Also, our build machines have

And believe me, I have tried every single programming language that has a fair user share and about 20 GUI toolkits before I have stumbled upon U++. U++ is the perfect GUI toolkit for me, and C++ is the perfect language (but with extremely archaic technology behind it). When I started using U++ I was doing desktop software, but I'm not doing that anymore.

But one cannot choose his programming language. In the domain I work (industrial printers) there are only two choices: C and C++. I'm researching my language proposal to see if one can merge the ease of use of C++ RAII semantics for easy resource management with a classic non OOP

based way of working like in C, but in the meantime making C extremely safe.

And NULL terminated strings are not fast. You need to check the whole string to determine the length. I understand that a lot of operations on strings are serial by nature and this disadvantage disappears, but there are also a lot of operations which are not serial. Take for example strings which have a known length which is padded with zeros up to the platforms integer size (32/64 bits). You can write an optimized strcpy which will use 32 bit moves and will never have to compare against zero vs. 4 times as many 8bit moves and comparing to zero. Which do you think is faster? Or 64 bit moves vs. 8 times as many 8 bit moves with null checking? Actually U++ does this for short strings. Or string concatenation, reverse find, binary search, etc. They all are more efficient when the length of the string is known, and since a string is an array, these algorithms will work on all arrays.

Subject: Re: New packages announcement
Posted by forlano on Wed, 17 Jun 2009 12:14:51 GMT
View Forum Message <> Reply to Message

koldo wrote on Sun, 14 June 2009 00:49
- Image analysis
Package focused to machine analysis and recognition more than to picture improvement (but also including functions for that).

Hello Koldo,

does it mean that with this package you can, for example, recognise a number in a bitmap image?

Luigi

Subject: Re: New packages announcement
Posted by piotr5 on Wed, 17 Jun 2009 15:57:01 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Mon, 15 June 2009 14:29But one cannot choose his programming language. In the domain I work (industrial printers) there are only two choices: C and C++. I'm researching my language proposal to see if one can merge the ease of use of C++ RAII semantics for easy resource management with a classic non OOP based way of working like in C, but in the meantime making C extremely safe.

Ah, now I understand: you want to backport c++ features into c! you sound like the gnu-hurd project: they too have a problem with insufficient info about what a program wants to do. is that what you tried to say? do you want to give c-programmers a possibility for revealing more info on what they want to do with the ressources they got from the operating-system? especially your string-handling sounds like that: how large is it? is it constant size or will it grow? do you plan to

read it, or do you only wish to make a copy? all this strong typing could be an improvement in c-programming -- even when it's merely implemented as a lib. in that case I would really like to help with the implementation of a compiler. this is also a feature I always missed. as you said, for string-handling the string-size is an important thing: if stringsize is large, then 0-terminating is a speed-up (as long as it's 4-8 bytes for 32bit respectively 8-16 bytes for 64bit filled with zeroes), with small strings it's a big hassle...

and don't get me wrong, I really do not believe you would be naive or something. I really just want to learn more about your project! and by the way, in my experience it is nonsense to present any design-paper without even trying to implement it first. during programming many issues will show up which could never have been noticed with a pure text description. you probably also know that, otherwise you wouldn't have mentioned that project.

Subject: Re: New packages announcement
Posted by koldo on Wed, 17 Jun 2009 20:14:58 GMT
View Forum Message <> Reply to Message

forlano wrote on Wed, 17 June 2009 14:14koldo wrote on Sun, 14 June 2009 00:49
- Image analysis
Package focused to machine analysis and recognition more than to picture improvement (but also including functions for that).

Hello Koldo,

does it mean that with this package you can, for example, recognise a number in a bitmap image?

Luigi

Hello Luigi

For now it only can analize an image and by adjustable edge detection and image segmentation it gets a list of elements and information about them like centroid position, surface, bounding box, orientation and basic shape.

From this a higher level should have to be included probably to thin objects and apply them a pattern matching against different letter shapes.

Best regards
Koldo

Subject: Re: New packages announcement
Posted by avpavp on Fri, 16 Oct 2009 02:12:34 GMT
View Forum Message <> Reply to Message

Where do I find these packages? Specifically the Matrix algebra and numerical methods?

thanks

Subject: Re: New packages announcement
Posted by koldo on Fri, 16 Oct 2009 07:19:16 GMT
View Forum Message <> Reply to Message

avpavp wrote on Fri, 16 October 2009 04:12Where do I find these packages? Specifically the Matrix algebra and numerical methods?

thanks

Hello avpavp

Matrix and Image are not ready yet. As I told in first post I have time until end of year .

If you cannot wait, the package will be just the Eigen library (http://eigen.tuxfamily.org/index.php?title=Main_Page) with some samples and perhaps some additions.

Best regards
Koldo
