Subject: "(national) C compiler"
Posted by kbyte on Tue, 31 Mar 2009 16:10:43 GMT
View Forum Message <> Reply to Message

Hi all,

I would like to implement or to port an existing extremely small C compiler. I just want to give support for the basic standard C instructions (about 30, right?) and no libs at all. Just a function to read from keyboard and another one to write to screen.

After that, my main objective is to translate every
C keyword into my native country keywords, the same C number of keywords but all translated to my country language. I already tryed to find such a small c compiler but no hits till now.
What is the best practice? Port or implement?
Please could you give me on this thought?

Target is x86 architecture

Many thanks

Kim

Subject: Re: "(national) C compiler"
Posted by Mindtraveller on Tue, 31 Mar 2009 18:07:08 GMT
View Forum Message <> Reply to Message

I`ve heard about a pair of such experiments here in Russia. But due to native complexity of Russian language (which is really very close to any east- or central- European language) the resulting code is horrible.
We do even have something like national standard for documentation processing app called 1C - which has embedded programming language based on Russian. At least in a number of versions. And 1C programmers consider it a complete nightmare.

But. I think this is good research direction.
The ability to express algorithm in native language and ability to program it are definitely tightly interconnected.

Subject: Re: "(national) C compiler"
Posted by kbyte on Tue, 31 Mar 2009 18:26:00 GMT
View Forum Message <> Reply to Message

I dont know russian lang. But mine language as a 1:1 relation to the english keywords and thus to the C keywords. Do you think that this could be a nighmare too?

Kim

Subject: Re: "(national) C compiler"
Posted by cbpporter on Tue, 31 Mar 2009 18:38:45 GMT
View Forum Message <> Reply to Message

I would recommend porting something. C is an extremely complex language and unfortunately you do have to implement most features to have something functional. I would recommend implementing at first two basic types: char and int. That should get you started. Combine with simple pointers, and you have something capable of most tasks that don't require structs. Programing languages are my hobby and I implemented once a "new C++", but I gave up the moment I discovered D.

Code generation is another hard part. Witting x86 assembly is relatively difficult and very difficult if you want to write an optimizing compiler. But if you leave out code generation phase and pass it on to a different tool, you could get a functional mini compiler in about 3000-5000 lines of code. With code generation and optimization you will need hundreds of thousands of lines of code, a full team and at least 10 years of development .

I would recommend checking out Clang. It is based on LLVM. It's not quite ready yet, but the code is relatively simple to understand. Or maybe D sources. They are not that long, but more difficult to understand. And there was a "simple retargetable C compiler" I played around about 5 years ago called lcc.

I hope you find something to get you started in those projects. With Clang you should be able to change the keywords in hours. But still, please don't waste too much time on C .

Subject: Re: "(national) C compiler"
Posted by kbyte on Tue, 31 Mar 2009 19:18:52 GMT
View Forum Message <> Reply to Message

Thanks a lot for these tips and advices.

Why do you say to not pend too much time with C?

Do you mean that I should point my target to object language?

Kim

Subject: Re: "(national) C compiler"
Posted by mr_ped on Tue, 31 Mar 2009 20:21:41 GMT
View Forum Message <> Reply to Message

I'm not sure what you want..
You want to take original C source and translate it into native language text file?
Then maybe U++ CParser connected with search&replace would work in many cases.

Or do you want to compile code with native words like it was C?
What about #include "native_definition.h" at the start of the file with
#define NATIVE_WORD_IF if
etc.?
Can mess up for more complex sources, maybe then a translation as functions would be better.

Anyway, I don't see the point of such effort. If you would succeed, you would get yet another C or C++, and why would anyone sane learn some obscure native language C with very limited use instead of proper universal world known C?
And every programmer should know English, otherwise he will cut himself from many resources.

Subject: Re: "(national) C compiler"
Posted by kbyte on Tue, 31 Mar 2009 21:03:53 GMT
View Forum Message <> Reply to Message

As far as i know C language as about 30 keywords, right?

What I want is to translate each C standard keyword for a country language keyword and then compile and generate the exe. It is just a rename of the keywords their implementation (translation for assembly are the same)

Supose this, instead of while(1) I want to use Enquanto(1), where enquanto is the translation of the while to my language.

Fisrt I thought to use macros to wrapp all C keywords and compilation and linking etc remain untoutchable. But if there is an error the error will refere the C keywork and not the related macro.

So, what do you to advice me?

Kim

Subject: Re: "(national) C compiler"
Posted by mirek on Wed, 01 Apr 2009 03:33:43 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Tue, 31 March 2009 14:38
C is an extremely complex language and unfortunately you do have to implement most features to have something functional.

Actually, C is pretty simple. I believe I could do basic C compiler in a week (badly optimizing, compiling to e.g. x86 assembly).

On the other side, C++ is complex - I would say year is not enough to implement it (just to showcase relative difficulity).

Mirek

Subject: Re: "(national) C compiler"
Posted by cbpporter on Wed, 01 Apr 2009 07:28:13 GMT
View Forum Message <> Reply to Message

kbyte wrote on Wed, 01 April 2009 00:03
So, what do you to advice me?

Well, I wouldn't invest effort into this except with macros. You could parse the compiler errors and with basic regex get it to talk about your keywords.

But if you want a "true" solution, I still suggest taking a C compiler and just changing what the keywords are. If you're lucky, you wont have to edit some lexer tool generated tables. LCC and clang are good candidates. If you're feeling brave, you could try to tackle GCC. GCC may be complex and not very coder friendly (it's sources), but changing the lexer should be accessible.

Quote:Actually, C is pretty simple. I believe I could do basic C compiler in a week (badly optimizing, compiling to e.g. x86 assembly).

On the other side, C++ is complex - I would say year is not enough to implement it (just to showcase relative difficulity).
Sure, one week is doable. As I said, 3000-5000 lines as a rough estimate and with a back end to handle platform specific code generation you could even have something relatively cross platform. Clang is significantly larger than this because it is modular and tries do do a lot of fancy stuff, like keeping all precompiler information so it can generate meaningful compiler errors based on source code before precompilation. Also tries to emulate GNU extensions.

But C is IMO very complex, not from a pure language point of view, but from an implementation point of view. You have to get a lot of features together and working before your compiler can do anything. Just to write a compiler that can genuinely handle the requirements of hello world, you need to implement the preprocessor, pointers, strings with at least one escape character and functions with variable argument list. A lot of small features, but these do ramp of the complexity in the initial stages. With other languages, you could get simple cases working with considerably less effort. Also, operators in C are not that easy to implement with optimizations because C accepts a wide range of combinations and expressions which are hardly readable by human being

sometimes make their way into production code.

Subject: Re: "(national) C compiler"
Posted by kbyte on Wed, 01 Apr 2009 09:20:50 GMT
View Forum Message <> Reply to Message

Ok, three ideas:

1- "Actually, C is pretty simple. I believe I could do basic C compiler in a week (badly optimizing, compiling to e.g. x86 assembly)."

This is the most interesting but even not wanting an optimized compiler I think it requires a great knowledge on assembly, which is not my case.

2-"Well, I wouldn't invest effort into this except with macros. You could parse the compiler errors and with basic regex get it to talk about your keywords."

It seems very interesting for me . Just define the macros and parse existing compiler output...

3- But if you want a "true" solution, I still suggest taking a C compiler and just changing what the keywords are. If you're lucky, you wont have to edit some lexer tool generated tables. LCC and clang are good candidates. If you're feeling brave, you could try to tackle GCC. GCC may be complex and not very coder friendly (it's sources), but changing the lexer should be accessible."

Seems interesting too for me.

I will try third option first and if too complex or takes too long i will try the second option.

Thanks once again
Kim

(Our team: Kim, Alex and Jose)

Subject: Re: "(national) C compiler"
Posted by mr_ped on Wed, 01 Apr 2009 09:58:26 GMT
View Forum Message <> Reply to Message

kbyte wrote on Wed, 01 April 2009 11:20
1- "Actually, C is pretty simple. I believe I could do basic C compiler in a week (badly optimizing,

compiling to e.g. x86 assembly)."

This is the most interesting but even not wanting an optimized compiler I think it requires a great knowledge on assembly, which is not my case.

Actually in your special case as you don't care about the performance too much as much you care about language, you don't need to compile into assembly, you can compile to C (simple search & replace compilation) and then call some C compiler (GCC for example).

This is same to macro approach, but instead of #include with macros you can handle this with your own source preprocessor and then postprocess compiler output.

This returns me back to CParser and search&replace concept, but I'm not sure how much CParser can be used in case it does not understand some more complex class or macros.
Thinking about it more, this is quite a special case. You basically don't need macros, you need to translate the source into C before preprocessing will apply (but you have to translate all native language includes too, maybe doable separately without parsing their includes... basic clib includes I would left as they are)

Maybe you can have .hn files for includes with native language, and parse those separately into .h files, then in the final C source include .h instead of .hn.
All you need is special preprocessor which will take turn ahead of C/C++ preprocessor? Which will replace all native keywords into C keywords, but will recognize C/C++ strings/comments and replace just code. With ordinary code this should be very simple with CParser class, with macros you may run into some problems, maybe Mirek will get the idea and answer you with more details which cases it would handle well.
And the postprocess parser for compiler output, that's pure text search&replace I think.

edit:
I would start probably with the 3rd option too ... with some friendly compiler which is easy to build and allow an easy change of keywords, you are done withing minutes without actually coding anything.

Subject: Re: "(national) C compiler"
Posted by kbyte on Wed, 01 Apr 2009 10:50:59 GMT
View Forum Message <> Reply to Message

Yes, the third is the easier i think.
When i thought in macros approach will be replace all C and preprocessor using macros and it is not easy when some expression have spaces in the middle. I think that will always a parsing stage for macros too. Example:

abc.h
```
#define main_include "#include <stdio.h>"
#define main_func "int main(int argc, char**argv)"
#define bloc_b {
```

#define bloc_e }

Final program:
#include abc.h

main_include
main_func
bloc_b
bloc_e

And not compiles...

So, maybe the gcc lexical translation for C only language can make it.

(Sorry about my thoughs. I am not a C or C++ pro)

Kim

Subject: Re: "(national) C compiler"
Posted by cbpporter on Wed, 01 Apr 2009 12:46:38 GMT
View Forum Message <> Reply to Message

You should drop the quotation marks. They are not needed and you get compilation errors because you are actually inserting strings all over the place. And using defines that expand to #include is problematic. But this is not replacing the keywords. Why do you want to replace "{". And main function?

Subject: Re: "(national) C compiler"
Posted by kbyte on Wed, 01 Apr 2009 13:03:55 GMT
View Forum Message <> Reply to Message

The idea id to give to the new C native programming the shape and the lexical of native language.

So, may be at the end i decided to let stay the { } but for now i would like to customize everything... Is this a bad idea?

Kim

Subject: Re: "(national) C compiler"
Posted by mr_ped on Wed, 01 Apr 2009 13:56:13 GMT
View Forum Message <> Reply to Message

depends what you hit.
{ and } are IMHO ideal for macro (although maybe I overlook something, but it looks very safe to me right now) except when used inside macros themselves.
#include is probably impossible to change (outside of compiler), because it's directive of preprocessor, thus it is handled before the translation in preprocessor begins. (basically any preprocessor directive will be difficult to change)
In this case it would better to create your own native language -> C compiler.

I'm still missing the point of this effort.
Such verbose language exists, it's Pascal, and the only reason why I did code simple application faster in Pascal then in C was that Borland Pascal did compile executable usually under 3 seconds while Borland C++ took often over minute. Otherwise having "begin end" make productivity suffer IMHO. You have to write it longer, and in the end I find the {} even easier to read and to match (in well intended code), so I think it's better to spend couple of hours to train somebody to recognize { } without second thought, then having "begin" "end" in programming language.
Then there are C keywords. There's only handful of them, and knowing the C original allows you to use C compilers (and they are many and almost everywhere). Any knowledge of native translation of them will leave you in very impractical position of knowing something what is not used anywhere except your special case.
And sources of application usually consist lot more of calls to library functions, then C keywords. Unless you plan to create wrapper for every common library to have their native language names, most of your application will look far from native language, only the keyword part of them.

I'm not saying it's not good to see your algorithm in native language first, but in my 15+ years experience I find paper+pencil much more stronger then any programming language. After I sort this one out, I simply translate it into C/C++ or other programming language in my head, but usually (for me) it's far easier to work with paper then computer during the design of algorithm phase.
But I think it's wasting someone's time, to make him learn something as impractical as native language version of C.

Subject: Re: "(national) C compiler"
Posted by kbyte on Wed, 01 Apr 2009 14:07:46 GMT
View Forum Message <> Reply to Message

I just plain to attract people to program their programs... helder people, young people, people that do not understand english...
May be I am following a wrong strategy...

Thank you very much

Kim

Subject: Re: "(national) C compiler"
Posted by cbpporter on Wed, 01 Apr 2009 14:43:43 GMT
View Forum Message <> Reply to Message

kbyte wrote on Wed, 01 April 2009 17:07I just plain to attract people to program their programs... helder people, young people, people that do not understand english...
May be I am following a wrong strategy...

Thank you very much

Kim
Well if this is your objective you may be following the wrong strategy.

First of all, there is no good reason for non professionals to learn C. I can be even damaging. While the core of the language is as fresh as ever and will remain like that forever (I mean writing algorithms, reasoning and principles, and even syntax), other features are very dated and counter productive. C is not something to which you want to attract people. C is what you do when you need to work on embedded systems, need extreme performance code (but here also C++ does the job as fast), need very fast compile speed by careful structuring of you code, need dynamic libraries, need to create a new compiler by having it generate C and thus opening up all existing platforms and so on. If you want to attract people to programming try some scripting language, ObjectPascal or Java. ObjectPascal is based on Pascal, which has didactic roots and while being a competent tool it makes it very hard to shoot yourself in the proverbial foot. But don't try Turbo Pascal, because people generally hate it .

As for using native keywords to make learning easier and more attractive, I don't think it's going to count for C. C is not a verbose language. You use about 5 keywords very often in an imperative way: if, else, while, return and class. And even if you add the rest, their number is quite low. These could benefit from native language form, but still people will grasp quite easily "if" even if they don't understand the word. As for the rarer ones like volatile and register, I doubt that by renaming them you will make the concept easier to grasp. On the other hand, "char *(*(**foo [][8])())[];" will look just as but no matter what you put in the place of char. And here unfortunately preprocessor wont help. And you can use typedef to replace all the keywords that define types.

Subject: Re: "(national) C compiler"
Posted by kbyte on Wed, 01 Apr 2009 21:50:25 GMT
View Forum Message <> Reply to Message

Yes, you are right...
I need to reformulate the formula to attract people...

Thanks again!

Kim

Subject: Re: "(national) C compiler"
Posted by mingodad on Fri, 01 May 2009 18:13:40 GMT
View Forum Message <> Reply to Message

Have you heard about tinycc ?
http://www.tiny.cc/

Subject: Re: "(national) C compiler"
Posted by wilf123 on Thu, 28 May 2009 17:19:10 GMT
View Forum Message <> Reply to Message

Do you really think that translating the keywords will make it much easier for beginners? I think not.
-) Technically, a preprocessor that replaces all the keywords in one pass is a simple possibility.
-) You could play around and test your ideas with a C interpreter. There should be plenty around.
-) There are many programs for teaching programming:
http://scratch.mit.edu/
http://wiki.laptop.org/go/Etoys
...