The original English text of an interview with the Russian news organization CNEWS:

C++ future

Soon we will have a new standard for C++. What is going to change in this programming language? What new features will it have?

- ⇨ We have a "Final Committee Draft" for C++0x. I expect that to be corrected based on official comments and approved by final vote on May 12, 2011. After that it becomes an official ISO Standard after a few month of bureaucratic process. No significant changes are expected from now on.

  First: The changes will be almost completely compatible. The committee tries very hard not to break existing valid ISO C++ code. What breakage there will be will be along the lines of a new keyword breaking old code that used that keyword as an identifier. If you have identifiers called **static_assert**, **constexpr**, or **nullptr**, we have to apologize and you'll probably have to change your code. Similarly, it would probably be a good idea for you to start migrating away from the more prominent names from the new standard library, such as **thread**, **unordered_map**, and **regex**.

  C++0x was designed to make C++ a better language for systems programming and library building. It also aims to make C++ easier to teach and learn. I think it meets these high-level goals through hundreds of minor changes rather than one or two massive extensions. One measure is that the language sections of the standard text grew by 27% and the library sections by 100%, but that probably overstates the degree of change – maybe half of that added text is clarification mostly aimed at implementers.

  What features? Long lists of feature won't help you to really understand C++0x, but you can find them in my C++0x FAQ: www.research.att.com/~bs/C++FAQ.html . One thing that springs to mind is standard type-safe support for traditional concurrent programming using threads, locks, etc. Regular expressions, hash tables, and random number generators will appeal to many. In the language itself the support for generic programming has improved with type deduced from initializers (**auto**), lambda expressions, and variadic templates. There is general and uniform initialization based on **{}**-lists, improved constant expression evaluation, and a new simplified **for**-loop. In other words, C++0x has a lot to offer, but it will take time for programmers to figure out where the various new prices fit into effective programming styles.

How the new versions of the language standard are created? How can Russian programmers contribute to this process?

- ⇨ Different languages use different processes and have different meanings for the word "standard." C++ uses the formal, slow, and democratic ISO process that is generally considered the gold standard for standards. The ISO process differs dramatically from a what is done for a language owned by a corporation; there, "the standard" is defined by the latest release of a product.

The ISO C++ standard committee meets three times a year for one week each. National standards committees also have meetings and send representatives. A major part of the standards work consists of a constant stream of email discussing issues and proposals and a series of formal papers presented to the committee. You can find the papers by looking for “WG21” on the web – all the documents are freely available. I have links to the ones I co-authored on my publications page.

Getting anything done in the committee takes consensus among several dozen people and about two dozen nations. This gives a bias against radical change. The process does move slower than some of us would like, but also ensures that the end product is very widely accepted and respected. Technical votes are by organization (one vote per organization) and final votes are by nation (one vote per nation).

The way you take part in this process is in principle through your national standards body. In the past Russia has sent representatives, but I don’t know how active they are now. I suspect that it might be an idea for Russian programmers to get together, revive the Russian national C++ committee, have meetings (in person or through the phone and by email) and make their voice heard. The most effective way of having influence involves attending meetings (and voting).

It is too late to make significant changes to the upcoming C++0x standard (for most people’s notion of “significant”), so maybe the aim should be to take part of the work for C++1x.

What, do you think, will happen with C++ in 10 years? In 20 years?

⇨ At the current rate of progress, there should be two or three revised standards in that period of time. I hope and expect such revisions to reflect a large and vibrant user community and support new and better programming techniques. Please note that 10 and 20 years are rather long periods of time in computing. C++ will have its 25 anniversary as a commercially available language while I’m in Moscow. The first commercial release (by AT&T) and the publication of the first edition of my “The C++ Programming Language” happened on October 14, 1985.

What new developments might we see? I have no really firm ideas. I hope that higher level concurrency models will be supported and that the various programming styles that we sometimes refer to as “multi-paradigm” will be properly integrated into a well articulated system of design and programming (and given a good descriptive name). I hope a significant increase of type safety of then modern code. The ideal is as ever perfect type safety.

Tell us whether and how your view on programming has changed over the recent decades? What would you do differently if you had to design C+ now from scratch?

⇨ You can never do major things from scratch. Major systems – such as mainstream programming languages – are grown in stages and have to be viable at each stage. I would have liked for C++ to provide much better type safety, but C++ grew from C and C grew up on tiny computers where serious flexibility and performance were incompatible with enforced static type safety. I would have liked a type safe base for C++ and a less messy syntax, but I don’t see how that could have happened in real life. As it is C++0x provides some help in the form on a new uniform initialization syntax that doesn't allow for narrowing conversions and an array template class that match built-in arrays for performance and notation without the implicit conversion to pointers. Examples:

```
int i1 = 7.5;                   // traditional; narrows 7.5 to 7
int u2 = {7.5};                 // C++0x: error: implicit narrowing not allowed
vector<int> v = { 1,2,3,4};     // more flexible initialization
```

Another dream that would require the availability of a time machine would have been to provide good high-level type checking of template arguments without loss of flexibility or performance. For example:

```
template<Forward_iterator Iter) Iter find(Iter,Iter,Iter::value_type);
vector<string> v ;
int a[10];
int  I,j;
auto r1 = find(v.begin(),v.end(),"Brian");          // ok
auto r2 = find(a,a+10,42);                          // ok
auto r3 = find (I,j,42);          // error: an int is not a Forward_iterator
```

Note the use of auto. That's a C++0x feature allowing the programmer to specify that the type of a variable is to be that of the variable's initializer. C++0x has quite a few such little useful extensions.

Is there a "perfect programmer"? If she existed, how would she look like?

⇨ "Programming" is really a cluster of related skills and activities. Nobody can be best at all aspects of programming. I'd like my programmers to have a solid grounding in the computing field (algorithms, data structures, machine architecture, etc.) and a specialization (e.g. graphics or applications to history research). A good programmer should also know how to communicate (verbally and in writing) and how to be an effective team member. I think a master's degree is the proper level of education for a developer of non-trivial systems. I wrote a short opinion piece on that: What should we teach software developers? Why? CACM. January 2010.

10-20 years ago new programming languages were created very often. Now it is different – we have fewer newcomers. Is there still a need for new programming languages? What they should be about?

⇨ Were there really more languages created 10 to 20 years ago? I remember that when I started on C++ the conventional wisdom was that the time for new languages had past. The older languages were too established and Ada would steamroller anything new. Over the last 50 or so years the rate of new languages has been about 2,000 a decade and I see no signs of that slowing down – but then I have no reliable sources of statistics.

I do not think that the current languages are anywhere near perfect. Some are primitive, some are bloated, some as toys, and some depends on huge infrastructures (often proprietary) so I hope that there will be newer and better languages coming.

I hope for perfect type checking, better abstraction mechanisms ,better integration of static and run-time checking, better integration with tools, better support for high-level concurrency models – all in relatively simpler languages.

General

You will visit Moscow in mid-October and will do a keynote and a workshop at the CEE-SECR conference. What are you going to talk about?

- ⇨ About C++0x, what it offers, and in particular the general design rules and the direction we want to take the language. I'll try to convince people that C++0x will be the best language for building infrastructure and for applications with serious resource constraints. I'll have to show how C++0x differs from the C++ they know already – or at least think they know. C++0x features and libraries are appearing in mainstream C++ implementations, so this is not a science fiction talk.

  I'll also give a few tutorials focusing on ways to use the current C++ well. In both cases, I'll emphasize the importance of flexible and efficient abstraction. The tutorials will go into some detail on the use of exceptions, discuss generic programming, and more.

Are you working on the new books? What will these books be about?

- ⇨ Last year, my first programming textbook "Programming: Principles and Practice using C++" came out in the US. As its title indicates it is primarily aimed at teaching how to program. However, it is also the best book for beginners and for many programmers with little C++ experience. This book was written for a first-year university course for Electrical Engineers and Computer Engineers, but is now also used for Computer Science students. I refined that book through three years of teaching hundreds of students. The book is now available in English, Polish, German, Chinese, and French. As of mid-October, it will also be available in Russian!

  Currently, I'm working on a 4th edition of "The C++ Programming Language." That's a massive project that I can't finish quickly. I need to explain what language features and standard library components C++0x offers, but I also have to present the fundamental techniques for using them. C++0x feels like a new language, so this task is not just non-trivial, it is also exciting.

Did you have a chance to work with Russian programmers? What do you think about them?

- ⇨ Not really. I have known Russian programmers in the US, but I suspect that the ones I have worked with are exceptional. For example, Alex Stepanov – the father of the STL and a pioneer of generic programming – is hardly typical of anything. I have worked with him on and off for 20 years or more. I guess my impression of Russian programmers is warped by knowing a few stars.

High-tech innovations are a hot topic now in Russia – the government wants to boost innovations in our country, so that the economy will depend less on oil and gas export. What would be your advice to Russian government – what should the Kremlin do to develop a solid high-tech eco-system in Russia?

- ⇨ Hmmm, this is a difficult topic and I have the impression that governments don't like unsolicited advice. However, since you ask, I'll give my opinion, which is personal and not based on high-level government experience. My impression is that when governments and high-level management tries to pick winners among technologies they fail and they fail in very expensive ways.

You need good universities and probably some research institutes. They must have stable funding for the longer term (decades) and the government must try very hard not to micromanage. Major breakthroughs have a nasty habit of happening where higher management doesn't expect them. Look at success stories such as MIT, Stanford, and Bell Labs. That's where you get your trained scientific and technical manpower. Don't discourage students, professors, and researchers from becoming entrepreneurs or working with entrepreneurs. Encourage a mix of fundamental and applied research. Encourage a mix of small projects and larger ones. Encourage a mix of short, medium, and long-term projects. Don't try to structure the technology transfer. Provide a funding stream for entrepreneurs simply by a steady stream of purchases from smaller, less established companies. Given a chance, they'll prosper and grow.

Do you want to pass a message to Russian programmers?

⇨ Real progress requires a curious mix of idealism and pragmatism. Don't just go for the quick buck; try to do something to improve the world for the longer term. However, don't be so idealistic that all seem hopeless because the perfect solution obviously isn't possible. Gradual, evolutionary change is possible! When I started essentially nobody had heard of object-oriented programming and almost all of those who had "knew" that it was infeasible (too complicated, too hard to learn, too slow, too specialized, etc.) and yet today we all use variants of the idea every day. We are making similar progress with generic programming – just imagine where we may be in ten year's time. Progress is possible – even when it seems so frustratingly slow.