Document number: N2510=08-0020

Alisdair Meredith
on behalf of BSI Panel 2008-02-04

BSI Position on Lambda Functions

Background

One of the major topics of discussion at the last BSI C++ Panel Meeting was the shape of the Lambda Function feature as proposed for C++0x. The last referenced version of this proposal was N2413 which has since been succeeded by N2487 in the mid-term mailing.

Background

There is strong support for the broad implementation of such a feature. In addition to simplifying work with the standard algorithms, we anticipate this greatly enhancing the thread library and other concurrency features that are seen as having critical strategic importance to the long term success of the C++ platform. We are encouraged that the Evolution Working Group remained open and active long enough to provide a viable specification.

Concerns

Despite its clear value, we share a number of concerns that might yet cause us to vote against adopting this feature in C++0x. *Most importantly we strongly advocate a conservative approach that can be liberalised later if experience justifies it. We do not want to see the Standard effectively locked in to effectively irrevocable choices. This is particularly the case with the three Lambda-Introducers.*

Proposals

1. All three Lambda-Introducers should be keywords, rejecting whitespace. They should appear in the Keywords table, and not preprocessor-op-or-punc.

Whilst we do not currently have any keywords that are spelt using symbols other than letters there is nothing that prevents us from doing so. We suspect that many expect a keyword to be a 'word' in the natural language sense.

The Lambda-Introducers are not punctuation, they have nothing to do with either the pre-processor or operators. They are conceptually keywords and so should be listed as such. The fact that they are spelt using non-alphabetic symbols is a bonus because it means that there can be no conflict with user selected 'names'.

Note that having them as keywords makes it rather more important that the default version is NOT spelt '<>'. That spelling is already used in the language as part of the syntax for template specialisation.

2. For consistency all three Lambda-Introducers should feature a symbol between the angle-brackets, and should be chosen from Clark's list (in N2487) to minimize parsing concerns.

Recommend:

- `<.>` *No default convention*
- `<&>` *Capture-by-reference*
- `<+>` *Capture-by-copy*

3. It should not be possible to override the capture convention of the `<&>` and `<+>` forms of lambda.

The three forms of lambda provide distinct semantics that should be preserved, so the semantics can be relied on. The <&> form creates a 'pure' lambda that is efficient and preserves dynamic behaviours. The <+> form creates a classic 'closure' capturing a copy of the enclosing stack-frame that can safely be used beyond that frame's lifetime, and the 'hybrid' form <.> advertises it requires explicit overrides to introduce any element from the enclosing frame with a convention.. If all 3 forms become equivalent (beyond a default) then code becomes more demanding to read and reason about. In particular, allowing overriding of 'pass all by copy' would make the use of the pure form as a closure very dangerous. When the user writes <+> they should know that the implementation will enforce closure safe semantics.

4. The documentation should refer to `<&>` as a Lambda, `<+>` as a Closure, <.> as an 'Explicit' Lambda and 'Anonymous Function' to refer to all three.

Distinct naming will clarify usage throughout the standard, and beyond.

5. The new function declaration syntax should overload the new Lambda-Introducer keyword for 'no default convention' rather than `auto`.

One difference between a lambda function and other functions is that the former is nameless. It would seem to be logical to use the same or similar syntax for 'nameless functions' as we do for named ones. As we are introducing a new syntax for named functions in the same revision of C++ that we are introducing lambda functions we should keep the syntax as harmonious as possible. That it reduces the amount of overloading of the `auto` *keyword is an advantage.*