C++0x

New String Literals and User-defined Literals

Pascal Andreas d’Hermilly
Anders Boesen Lindbo Larsen

Datalogisk Institut
Københavns Universitet

April 25, 2008
A literal is a notation for representing a value within source code, e.g.:

- "hello" (string)
- ’c’ (character)
- 14 (integer)
- 3.1 (double)
- 3.1f (float)

We call the f in 3.1f the suffix modifier. The type of a literal is thus determined from its syntactic form.

In the following we shall see how the C++ handling of literals is changed with C++0x.
String Literals

- Lots of demands for cross-platform functionality.
- Current standing for C++: not good, compared to e.g. java or python
- The language doesn’t use a standard character-encoding set for non-ascii, but only provides containers for the bits.
String literals is an array consisting of either:

- char
- wchar_t

Examples of current string literals:

- "A string literal in double quotes" being a const char
- L"Example" being a const wchar_t

In C++0x, better support for Unicode will be introduced. Encodings that will be supported: UTF-8, UTF-16, and UTF-32
New String Literals

New character types:
- char16_t
- char32_t

Examples of new string literals:
- u8"UTF-8 string"
- u"UTF-16 string"
- U"UTF-32 string"

using const char[], const char16[], and const char32[] respectively.

Also inline insertion of unicode-codepoint will be available:
- u8"Unicode character: \u2018 "

The code that follows \u is hex for a 16 bit Unicode codepoint. 32 bit, use \U
Raw Strings

Raw strings allow you to create strings without having to escape special characters.

Examples:

- R"[ls /home/pascal/ | grep ".pdf" ]"
- R"delim[ls /home/pascal/ | grep ".pdf" ]delim"
The current problem

- C++ recognizes literals only for the built-in data types, e.g.: integers, strings and booleans.
- These types are not sufficient to maintain C compatibility! (C99 has introduced new data types, that requires recognition of new literals)

The goal of C++0x

- Make it possible to define new kinds of literal modifiers that will construct user-defined objects.
- C compatibility should be maintained as far as possible
Examples of User-defined Literals

Some of the possibilities with user-defined literals:

- "hello"s (std::string)
- 3.5i (imaginary type for complex numbers)
- 10011011b (binary literals)
- 1234567890123456789xxl (arbitrary range/precision)

Notice the use of suffixes to identify the literal type.
Every user-defined literal is interpreted as a call to a new kind of operator (called *literal operator*):

```
X operator "suffix" (parameters);
```

where `X` is the return type of the newly defined literal with the suffix `suffix`. *parameters* decides in which format the literal operator receives the value of the literal.

For example, the literal `3.5i` will perform the following call:

```
operator "i" (parameters);
```
Examples of Literal Operators

A *cooked-form* literal operator can have the following form:

X operator "suffix" (unsigned int);

For example:

```c
cm_t operator "cm" (unsigned int);
```

```c
cm_t n = 45cm;
```

In this case, 45 is given as parameter to the literal operator.

A *raw-form* literal operator can have the following form:

X operator "suffix" (char const*);

Example:

```c
long_number operator "xxl" (const char*);
```

```c
long_number n = 123456789012345xxl;
```

In this case, the null-terminated C string "123456789012345" is given as parameter to the literal operator.
Sources

The C++ Standards Committee, n2378

The C++ Standards Committee, n2442

Wikipedia, C++0x
http://en.wikipedia.org/wiki/C++0x