Subject: is assist++ ready for c++11?
Posted by piotr5 on Wed, 24 Apr 2013 08:23:11 GMT
View Forum Message <> Reply to Message

clone https://github.com/kennytm/utils.git and add the sources to an empty project. it will hang parsing variant.hpp, at least for me it does write this would be the file it is parsing.

Subject: Re: is assist++ ready for c++11?
Posted by mirek on Thu, 21 Nov 2013 07:17:17 GMT
View Forum Message <> Reply to Message

Should be now fixed.

Subject: Re: is assist++ ready for c++11?
Posted by piotr5 on Sun, 01 Dec 2013 10:57:44 GMT
View Forum Message <> Reply to Message

thanks. now it would be nice to actually support c++11.
syntax highliting is already added (although "__int128" is missing in *cpp[], even though c++11 now supports long long). so I did take a look at CppBase package but it's quite messy. this code really needs some cleanup. as it is, it's unclear if one should use Key() or Lex::operator[] or whatever. the keywords are hardcoded into the parsing sources even if it means if-statements of several lines length, and generally there is lots of long procedures without description of where which state is handled. anyway, I did think up the following changes, apart from
addingCPPID(__int128)
CPPID(constexpr)
CPPID(nothrow)
 to keyword.i:

```
in Parser.cpp--- uppsrc/CppBase/Parser.cpp   (revision 6623)
+++ uppsrc/CppBase/Parser.cpp   (working copy)
@@ -23,6 +23,7 @@
 static String s_virtual("virtual");
 static String s_inline("inline");
 static String s_static("static");
+static String s_constexpr("constexpr");

 static inline bool sSpaces(String& res, const char *& s)
 {
@@ -74,7 +75,7 @@
                    const char *b = s++;
                    while(iscid(*s)) s++;
                    String q(b, s);
-                   if(q != s_virtual && q != s_inline && q != s_static && !InScList(q, pname)) {
+                    if(q != s_virtual && q != s_inline && q != s_static && q != s_constexpr &&
```

```
!InScList(q, pname)) {
                        if(iscid(*res.Last()))
                                res.Cat(' ');
                        res.Cat(q);
@@ -123,7 +124,7 @@
                    const char *b = s++;
                    while(iscid(*s)) s++;
                    String q(b, s);
-                   if(q != s_virtual && q != s_inline && q != s_static)
+                    if(q != s_virtual && q != s_inline && q != s_constexpr && q != s_static)
                        res.Cat(q);
                    else
                        while((byte)*s <= ' ' && *s) s++;
@@ -143,7 +144,7 @@
                    const char *b = s++;
                    while(iscid(*s)) s++;
                    String q(b, s);
-                   if(q != s_virtual && q != s_inline)
+                    if(q != s_virtual && q != s_constexpr && q != s_inline)
                        res.Cat(q);
                    else
                        while((byte)*s <= ' ' && *s) s++;
@@ -292,7 +293,7 @@
                    gp = false;
                }
                else
-               if(Key('>')) {
+                if(Key('>')&&gp) {
                    level--;
                    if(level <= 0) {
                        ScAdd(param, id);
@@ -308,7 +309,7 @@
                    }
                }
                else
-               if(Key('<'))
+                if(Key('<')&&gp)
                    level++;
                else
                    ++lex;
@@ -445,7 +446,7 @@
                return Null;
        }
        if(Key(tk_int) || Key(tk_char) ||
-          Key(tk___int8) || Key(tk___int16) || Key(tk___int32) || Key(tk___int64)) return Null;
+           Key(tk___int8) || Key(tk___int16) || Key(tk___int32) || Key(tk___int64) || Key(tk___int128))
return Null;
        if(sgn) return Null;
```

```
        const char *p = lex.Pos();
        bool cs = false;
@@ -477,7 +478,7 @@
 {
        Key(tk_const);
        Key(tk_volatile);
-       if(Key(tk_throw)) {
+       if(Key(tk_throw) || Key(tk_nothrow)) {
                while(lex != t_eof && !Key(')'))
                        ++lex;
        }
@@ -540,8 +541,17 @@
 void Parser::EatInitializers()
 {
        if(Key(':'))
-               while(lex != '{' && lex != t_eof)
+               while(lex != '{' && lex != t_eof) {
                        ++lex;
+                       while(lex!='(' && lex != '{' && lex != t_eof) ++lex;
+                       int lev=1;
+                       while(lev>0 && lex != t_eof) {
+                               ++lex;
+                               if(lex=='('||lex=='{') ++lev;
+                               else if(lex==')'||lex=='}') --lev;
+                       }
+                       while(lex <= ' ' && lex != t_eof) ++lex;
+       }
 }

 void Parser::Declarator(Decl& d, const char *p)
@@ -553,7 +563,7 @@
                d.isptr = true;
                return;
        }
-       if(Key('&')) {
+       if(Key('&') || Key(t_and)) {
                Declarator(d, p);
                return;
        }
@@ -677,7 +687,7 @@
                if(Key(tk_virtual))
                        d.s_virtual = true;
                else
-                       if(!(Key(tk_inline) || Key(tk_force_inline)))
+                       if(!(Key(tk_inline) || Key(tk_force_inline) || Key(tk_constexpr)))
                                break;
        }
        Qualifier();
```

```
@@ -827,8 +837,8 @@
        int t = lex[q];
        if(t == tk_int || t == tk_bool || t == tk_float || t == tk_double || t == tk_void ||
           t == tk_long || t == tk_signed || t == tk_unsigned || t == tk_short ||
-          t == tk_char || t == tk___int8 || t == tk___int16 || t == tk___int32 || t == tk___int64) {
-              while(lex[q] == '*' || lex[q] == '&')
+           t == tk_char || t == tk___int8 || t == tk___int16 || t == tk___int32 || t == tk___int64 || t ==
tk___int128) {
+               while(lex[q] == '*' || lex[q] == '&' || lex[q] == t_and)
                      q++;
               if(!lex.IsId(q))
                      return false;
@@ -854,7 +864,7 @@
                      return false;
               type << Tparam(q);
        }
-       while(lex[q] == '*' || lex[q] == '&')
+       while(lex[q] == '*' || lex[q] == '&' || lex[q] == t_and)
               q++;
        if(!lex.IsId(q))
               return false;
```

what I wasn't able to fix is that assist sees an error in using the template keyword multiple times, like for example when the class is templated and also its memberfunction has template parameters of its own. also I'd like to add basic preprocessor parsing by reading what gets defined in a #define and if it's a keyword then the macro should be treated as if it were of that type. also some basic #if parsing would be nice by storing the current state and restoring it upon an #else. also "#if 0" really should be treated as an alternative comment ending with "#else" or "#endif"...

compilers like gcc have a nice todo-list of what still needs to be done for full c++11 support, shouldn't we have something similar for Assist++ and such? what I implemented above is just support for r_value, support for constexpr and nothrow, and I also tried to fix template-parameter parsing by reading the "gp" flag that already was present (but I've noticed there's another place where the parser could get confused by inequalities inside of template parameters). also the initializers that use {} instead of () for their initial value I hope will now correctly get parsed as such and not as the function-body. I still have to test it though...

actual support and application for [[attributes]] would be nice, also lambda functions and defining the return-value after function-declaration are important.
edit: sorry, it hangs and doesn't work since my patch prevents parsing template-parameters with defaults -- not sure how to fix. however, after an edit, now at least no more hang...

Subject: Re: is assist++ ready for c++11?
Posted by Klugier on Sun, 01 Dec 2013 19:29:03 GMT
View Forum Message <> Reply to Message

Hello piotr5,

I think "__int128" isn't official c++11 keyword. This extension family ("__int*") is only for M$ Compiler and shouldn't be use in prototable applications.

If you want to use prototable declaration try to include <cstdint> header. Following header is available on most standardized c++ compilers such as g++, clang++ and MSC.

For me, we should remove all "__int*" highlights from ultimate++ repository, because it contributes to the unnecessary mess.

External resources:
cstdint reference

Sincerely,
Klugier
