obcp-parse.y:

```
| delete objc_openbracket.expr objc_closebracket cast_expr %prec UNARY
        { $$ = delete_sanity ($4, $2, 2, $1);
          if (yychar == YYEMPTY)
            yychar = YYLEX; }
```

The **$1** in delete_sanity wasnot in 2.5.8, so it may be wrong.

Files modified to fix problems with Objective-C++ (cc-723):

typeck.c:

convert_for_assignment():

Bug: ttr was uninitialized.
Result: compiler would crash during type checking.
Test case: NSConnection.m. Unable to reproduce in a small test case.
Fix: Move the clause down a couple statements.

convert_for_assignment():

Annoyance: stupid warning about implicit casting from "void *"
Fix: Shutup if compiling ObjC code.

decl.c:

warn_extern_redeclared_static():

Bug: extern/static errors during ObjC code generation.
Result: compiler would refuse to generate code.
Test case:

```
// considered illegal by C++
extern int a;
static int a = 5;
```

Fix: Ignore "artificial" (i.e. internally generated) decls.

grokdeclarator():

Bug: C++ constructors were totally broken.
Result: the basics didn't work.

```
class foo
{
public:
        foo(int);
};

foo bar(5);
```

Fix: Put the code back where it belongs. John removed this while experimenting with another bug (I believe).

grokparms():

Bug: Very complex declarators causes grokparms to fail.
Result: The compiler would consciously about.

```
@interface Bug
- sortedArrayUsingFunction: compare ;
@end

@implementation Bug
- sortedArrayUsingFunction:(int (*)(void *))compare   { }
@end
```

Fix: Return! As the comment in the code indicates, I don't feel great about this fix. Nevertheless, I believe it is valid (see code for more details). The bottom line is "grokparms" is subtly different between the two front ends (and I don't have the time to dig any deeper). If a new wave a "grokparms" bugs appears, this should be revisited.

lex.c:

init_lex():

Annoyance: Remove C++ keywords when compiling straight ObjC (for testing purposes)

do_identifier():

Bug: Scoping mechanism for sending ObjC messages is broken.
Result: The compiler wouldn't "see" the argument (resulting in an error).

```
struct exception {};

@interface Foo
- (void)reply:(int)exception;
@end

static void func(int exception)
{
```

```
        id obj;
        [obj reply:exception];
    }
```

Fix: Last argument to lookup_name() needs to be 0, not 1.

real_yylex():

Bug: ObjC string constants didn't work.
Result: Couldn't use a string constant (at all).

```
    NSString *foo = @"xx";
```

Fix: Missing a "break" statement. Wrong token was being returned.

lex.h:

Bug: Objective-C type qualifiers (in, out, etc) are broken.
Result: Parse error.

```
@interface NSString
-(oneway void)addConversation:(int *)conversation;
@end
```

Fix: RID_LAST_MODIFIER needs to be our last modifier, not C++'s.

objc-act.c:

xref_tag():

Bug: Static typing/scoping mechanism for looking up types (from ObjC) is broken.
Result: ObjC types and internally synthesized types were being added to the wrong scope. This resulted in bogus type errors in both user code and internally generated code (for example, this also caused "super" code generation to fail).

```
struct _NSStringBuffer {
      NSString *string;
};
```

```
void func(struct _NSStringBuffer *b, NSString *s)
{
	b->string = s;
}
```

Fix: The last argument to xref_tag needs to be 1, not 0.

Bug: Static typing/scoping mechanism for looking up types (from ObjC) is broken.
Result: ObjC types and internally synthesized types were being added to the wrong scope. This resulted in bogus type errors in both user code and internally generated code (for example, this also caused "super" code generation to fail).

create_builtin_decl():

Bug: Errors during Objective-C code generation for categories.
Result: Module wouldn't compile because of "redefinition" errors.

```
@interface Object (cat)
```

```
@end

@implementation Object (cat)
@end
```

synth_module_prologue ():

Bug: @protocol() didn't work.
Result: type errors.
Fix: install correct type.

build_module_descriptor():

Bug: ObjC constructor generation fails.
Result:

```
NSConnection.m: At top level:
variable or field `_GLOBAL_$I$NSConnectionReplyMode' declared void
parse errors have confused me too much
```

build_objc_string_decl():

Bug: Extern/static warning (also changed decl.c, see above).

start_method_def():

Bug: ObjC arguments very broken. Only primitive/simple types worked.
Result:

```
@interface NSString
-(void)simple:(int *)conversation;
@end

@implementation NSString
-(void)simple:(int *)conversation { conversation; }
@end

a.cpp: In function `void -[NSString simple:](struct NSString *, struct
```

objc_selector *)':

a.cpp:7: warning: conflicting types for `-(void)simple:(int *)conversation'
a.cpp: In function `':
a.cpp:3: warning: previous declaration of `-(void)simple:(int *)conversation'
a.cpp: In function `void -[NSString simple:](struct NSString *, struct objc_selector *)':
a.cpp:7: `conversation' undeclared (first use this function)
a.cpp:7: (Each undeclared identifier is reported only once
a.cpp:7: for each function it appears in.)

Fix: The C++ parse tree for arguments is shallow (remove a "level" of tree node).

really_start_method():

Bug: Unused warnings for "self" and "_cmd".
Fix: Set the "used" bit.

parse.y:

initlist:

Bug: "labeled" inititializers (a GNU extension) cause ambiquity with message expressions.

```
void func() {
    struct foo aFoo =   { [obj bar] };
}
```

Fix: Remove the GNU feature. This is what we did in the previous release of the compiler. This also removes quite a few shift/reduce errors in the yacc grammar.

Bug: "labeled" inititializers (a GNU extension) cause ambiquity with message expressions.

opt.component_decl_list:

Bug: C++ is not allowing a semi-colon after @defs.

```
@interface Object { int a; } @end

struct foo { @defs(Object); };
```

Fix: Add an explicit rule in the grammar. To localize the effect, I just made a duplicate clause...there might be a more elegant solution, however I didn't want to risk screwing up the mainline C++ grammar.