INITIALIZATION OF CALL-BY-VALUE OBJECTS

James W. Welch
Watcom International Corp.
jww@watcom.on.ca

Doc No. WG21/N0780, X3J16/95-0180

Abstract

Existing vendors implement call-by-value of destructable objects in at least two ways, because of differing interpretations of the Draft Working Paper. This paper proposes changes to the draft to eliminate the two interpretations. The proposal is to define call-by-value as an initialization of the parameter of the called function that conceptually occurs in the context of the calling function. The lifetime of the parameter is the time during which the called function is activated.

(1) Existing Practice

Consider the following fragment of C++ code:

```cpp
struct Obj {
    Obj( int );
    Obj( Obj const & );
    ~Obj();
};

extern void PlaceHolder();

void Called( Obj object )
{
    PlaceHolder();
}

void Caller( void )
{
    Called( Obj( 1995 ) ), Called( Obj( 1996 ) );
}
```

Inspection of the example reveals that the function Caller twice passes by value an object of type Obj to the called function Called.

It has been reported that there are at least two ways in which vendors have implemented the compilation of the indicated fragment. The first method, used by WATCOM and EDG (Edison Design Group), is to create temporaries in the calling function, to merely use these temporaries as required in the called function, and then to destruct the temporaries at end
of the expression containing the call(s) to the called function. This results in the following actions:
- Obj(1995) is constructed
- Called is invoked which calls PlaceHolder and returns
- Obj(1996) is constructed
- Called is invoked which calls PlaceHolder and returns
- Obj(1996) is destructed
- Obj(1995) is destructed

The second method, reportedly used by Sun and Microsoft compilers, is to construct the temporaries in the caller and to destruct the call-by-value object during the return from the called function. This results in the following actions:
- Obj(1995) is constructed
- Called is invoked, PlaceHolder is called, Obj(1995) is destructed, return to Caller
- Obj(1996) is constructed
- Called is invoked, PlaceHolder is called, Obj(1996) is destructed, return to Caller

These two methods have different temporary lifetimes and orders of destruction, which is not desireable. It is proposed to add language of the working paper which is a compromise between the two methods. Essentially, it is proposed that the calling function is responsible for construction and destruction of the parameter and that the parameter has a lifetime matching the activation of the called function (not the expression containing the call). An optimizing compiler may do a direct initialization of the parameter by existing language in the working paper.
As well, the following program fragment was compiled under several compilers.

```cpp
class CALLER;

class CALLEE {
    CALLEE( CALLEE const & );
    ~CALLEE();
    friend void callee( CALLEE, CALLER );
public:
    CALLEE();
    void dummy();
};

class CALLER {
    CALLER( CALLER const & );
    ~CALLER();
    friend void caller1( CALLEE &, CALLER & );
    friend void caller2( void );
public:
    CALLER();
    void dummy();
};

void callee( CALLEE x, CALLER y ) {
    x.dummy();
    y.dummy();
}

void caller1( CALLEE &rx, CALLER &ry ) {
    callee( rx, ry );
}

void caller2( void ) {
    callee( CALLEE(), CALLER() );
}
```

Access errors were detected by the indicated compilers at the following spots.

<table>
<thead>
<tr>
<th>CALLER</th>
<th>CALLEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>copy</td>
</tr>
<tr>
<td>ctor</td>
<td>ctor</td>
</tr>
<tr>
<td>dtor</td>
<td>dtor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>start of callee</th>
<th>MBHG</th>
</tr>
</thead>
<tbody>
<tr>
<td>end of callee</td>
<td>B</td>
</tr>
<tr>
<td>call-point in caller1</td>
<td>PBHWEGS</td>
</tr>
<tr>
<td>call-point in caller2</td>
<td>PBE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W -- Watcom 10.5</th>
<th>E -- Edison Design Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>M -- Microsoft 2.0</td>
<td>G -- g++ 2.6.3</td>
</tr>
<tr>
<td>B -- Borland 4.02</td>
<td>S -- Sun 4.0.1</td>
</tr>
<tr>
<td>H -- Metaware 3.1</td>
<td>P -- proposal</td>
</tr>
</tbody>
</table>

It is concluded that there is little consistency among vendors for detecting access errors.
(2) Model of call-by-value

It is proposed that call-by-value be mandated to be an initialization of the argument in the called function with the value specified in the caller. The lifetime of that argument is the time during which the called function is active. The initialization and destruction of the parameter in the called function occurs in the calling function.

By existing rules, an optimizing compiler may eliminate temporaries (see 12.2 [class.temporary]) and directly initialize a parameter.

An optimizing compiler could compile the called function in such a way that the actual call for destruction could be done from code for the called function (and not from the caller). Of course, accessibility checks are done in the context of the caller, not the context of the called function. When there are multiple call-by-value parameters to a function, the compiler would need to ensure that the order of destruction of the parameters is in reverse of the order of construction.

(3) Formal Proposal

It is proposed that the Draft Working Paper be amended as follows:

5.2.2 Function call [expr.call] paragraph 3: add the following after the second sentence.

The lifetime of each parameter is until there is a return from that activation of the function. The initialization and destruction of each parameter occurs within the context of the calling function. An implementation may eliminate construction of extra temporaries by combining the construction and/or conversion with the initialization of the associated parameter (see 12.2 [class.temporary]).

(4) Conclusion

The intent of this paper is to eliminate one more difference between implementations. As usual, I am not invested in the specific wording; those more capable in standardese can likely improve my amendment.

Acknowledgements

Anthony Scian and John Spicer have made several useful criticisms of the paper. I have adopted some of their suggestions. Any errors or omissions are, of course, my responsibility.