Document Number: X3J16/92-0126
WG21/N0203
Date: November 5, 1992
Project: Programming Language C++
Reply to: Tom Pennello
tom@metaware.com

Some issues in temporaries destroyed at end of block

Assume that some temporaries are destroyed at the end of a block; we can assume for convenience that no temporary persists through a conditional expression, so there is no need for flags.

```
{
p = s+t;
printf("%s",p);
...
}       temp.~someclass() called here.
```

This is the common example where string concatenation + produces a temporary referenced from p. Let us assume that the temporary is destroyed at the end of the block }. Now what happens when we insert a goto:

```
  {
  if (dont_evaluate_p) goto L;    //1
  p = s+t;
  printf("%s",p);
L: ;
  ...
  }       //2 temp.~someclass() called here.
```

Now we have the problem that at //2 the temporary is destroyed when it wasn't created. All right, to solve this, then, we elevate temporaries to the same status as user-declared variables; i.e., skipping the construction of a temporary is invalid, just as skipping the construction of a variable is invalid. So the compiler issues a diagnostic at //1.

This diagnostic would tend to reject lots of programs. Another example:

```
switch(expn) {
    case 1:
        p = s+t;
        printf("%s",p);
        break;
    case 2:
        ...
        break;
    }
```

The control flow to case 2 produces a diagnostic. Don't believe it? Many compilers don't get this right, but consider the user-declared variable case:

```
switch(expn) {
    case 1:
        int i = 1;
        break;
```

```
        case 2:
            printf("%d",i);
            break;
        }
```

We MUST produce an error here; the usage of i in case 2 is illegal, because i will hold trash. Basically, the switch(expn) is a FORTRAN computed goto to each of the cases, and the goto to case 2 enters a scope in which "i" is still active (but was not constructed = initialized).

Another problem with giving temporaries the rank of user variables is that one compiler may use a temporary, while another optimizes it away, so that a transfer of control would be an error for the first compiler, but not for the other; this would make it difficult to write a strictly- conforming program. Alternatively all compilers would have to behave consistently and issue diagnostics about transfer of control around "potential" temporaries, even though the temporary might not be materialized.

To reduce getting errors for temps around which control is transferred, we have some possibilities.

1 Introduce flags to say whether a temp has been constructed or not
2 Have the compiler destroy temps at labels.
3 have the compiler optionally destroy a temp earlier if it knows there are no later uses of the temp.

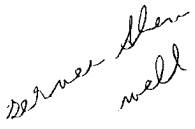
Consider 3 first.

```
{
if (dont_evaluate_p) goto L;    //1
p = s+t;
printf("%s",p);
// No more uses of the temp, so destroy it here.
L: ;
...
}
```

If the temp is destroyed early enough -- say before L -- the goto is OK. The problem encountered now is: should this work on all implementations? Some compilers may have flow analysis for the use of temps, and others might not. It would result in not being able to write strictly-conforming programs that used temps and transfers of control (remember the switch example).

Consider next 2. This means that programs will start to break if you insert labels. At first glance this sounds OK -- discourage labels, right? -- but some of the labels won't affect the temps.

```
p = s+t;
printf("1: %s",p);      //1
if (something) goto L; ... L: ;
printf("2: %s",p);      //2
```



So here the goto doesn't affect the construction of the temp, but the label would (unnecessarily) destroy the temp.

Finally consider 1. Flags. This would work, but many people are against the idea of flags.

In summary, destruction at end-of-block seems to require one of two things:

- an error if transfer of control skips the construction of a temporary -- i.e., temporaries are given the same rank as user-defined variables -- and that unless all compilers treat temporaries identically, a program will generate errors on one compiler and no errors on another;

or

- flags

This is true even if temporaries are destroyed in conditional contexts to avoid flags.

On 11/5/92 at the Boston meeting, there were some straw votes taken indicating a slight preference for end-of-statement (EOS). Interestingly, at the previous (Toronto) meeting, during the core discussion on temps, a straw vote at the beginning of the session was 2/1 in favor of EOB; after considerable discussion of the problems mentioned in this paper and other problems (see the Toronto minutes), the core group had a slight majority in favor of EOS.

Absent a strong proposal to consistenly handle temporaries at EOB without flags, it seems likely that we should go for EOS (and destruction at conditionals), so that there need be no flags.