Doc No: SC22/WG21/N1470

J16/03-0053

Date: 04/02/2003

Project: JTC1.22.32

Reply to: Daniel F. Gutson
danielgutson@hotmail.com

Enum Type checking for SWITCH statements

1. The Problem

When performing SWITCH statements on enum variables, labels may have any integral literal, without checking whether the value belongs or not to the enum elements (therefore allowing -maybe- unintended unreachable code).
Moreover, this represents a weakness in the type system, as far as the type information is lost in the SWITCH statement as it performs a cast to integral type.

Why is the problem important?
Given a SWITCH receiving an enum variable, three situations on the case labels can be found:
a) (trivial): a label is an enum element
b) a label is a literal or constant with the same value of an enum element
c) a label is a literal or constant which value does not match with any enum element.

In case b), if the enum is modified, such places (situation b) have to be updated, requiring additional maintenance effort (performing a search of every SWITCH using the enum, which is hard to find as far as SWITCH may receive a variable whose declaration can be in other scopes).
Additionally, if any of such places is not updated, a situation c) is reached, representing error prone code.
Finally, the reader is forced to check the value of the constant for finding it in the enum definition, representing additional effort (and therefore less readability due to the redundance).

A situation c) may come from an error occurred in a violation of the process mentioned in b). There is no standard way of detecting these situations (unreachable code due to unupdated cas labels on enums), despite many compilers provide warning mechanisms.

Enums are importants both in the sense of self-documentation, and in type-checking (safety) . Both aspects are impacted in the lack of checking of the case labels.

Whom does it affect?
Large/legacy code, maintained by many people.

What are the consequences of not addressing it?
Current lack of checking allows:
-code redundance
-maintenance overhead (therefore decrease of maintainability)
-error prone code
-coding errors due to confusion (cases could have the value of another enum)

How are people addressing, or working around, the problem today?

There is no way of ensuring (force) that the case labels are of the type of the enum.

Which of the categories that we're interested in addressing does this fit into?
* improve support for systems programming: type system is enhanced (or at least coherent with the type-spirit of enums), augmenting safety.

* remove embarrassments: redundant values are detected and candidates to be eliminated; unreachable code is detected and (forced to be) either corrected or eliminated.

2. The Proposal

When the SWITCH receives an enum type, require case labels to be enum elements of such enum.

2.1 Basic Cases

```
enum Greatings
{
   Hello,
   Goodbye,
   SeeYou
};

const int X = SeeYou;

void f(Greatings g)
{
      switch(g)
      {
      case Hello: //ok
            break;
      case X:           //error
            break;
      case 3:           //error
            break;
      case Goodbye:     //ok
            break;
      }
}
```

2.2 Advanced Cases

Enum checking is disabled when casted to int. Example:

```
      switch((int)g)
      {
      case Hello: //ok
            break;
      case X:           //ok
            break;
      ...
      }
```

3. Interactions and Implementability

3.1 Interactions

This proposal may break -intentionally- existing code.
Errors could be classified according to situations b) and c) mentioned above.

In situation b), the change is straight forward: replace the literal or constant by the enum element.
In situation c), the unreachable code should be analyzed, and this type of errors will help to discover possible bugs, being an opportunity to inspect these situations.
As mentioned above, the feature can be easily (quickly) disabled by explicit casting to ´int´ the SWITCH parameter.
By this, the intention to have ´unreachable´ code is explicited in the code (as the cast is explicited), serving as a documentation factor (and a llowing warning disablement according to the compiler implementation).

3.2 Implementability

The compiler should perform type-checking for each case label.