Document number: NXXXX
Date: 2014-08-19
Project: Programming Language C++,
Language Evolution Working Group
Reply to: Marcel Wid
<marcel.wid@ods-solutions.de>

Explicitly defaulted relational and equality operators

Contents

I. Introduction

Sorting and searching is a common task in everyday programming. In order to be able to search or sort objects of a type `T`, the type `T` must provide relational and equality operators. For convenience of the users of type `T` it is necessary to implement all variants of these operators, which results in a lot of boilerplate code. This is needlessly verbose, error prone and contrary to modern C++. Generally it is enough to provide only one relational operator, namely `operator<=()`, and all other relational and equality operators can be defined in terms of `operator<=()`. We propose an extension of the core language by allowing the relational and equality operators to be explicitly defaulted.

II. Motivation and Scope

Consider the following snipplet:

```
#include <string>
#include <vector>

struct Author
{
    std::string lastname;
    std::string firstname;
};

struct Book
```

```
{
    std::string title;
    Author author;
    std::string publisher;
    unsigned short year;
};

std::vector<Book> books;
```

How do we want to sort the vector `books`? One reasonable possibility is to use the lexicographical order of the `struct Book`, i.e. we first sort by `title` then by `author` then by `publisher` and finally by `year`. Therefor, we first have to implement all operators for the `struct Author`:

```
struct Author
{
    // ...
    bool operator<=(Author const& other)
    {
        if(lastname != other.lastname)
            return lastname < other.lastname;
        return firstname <= other.firstname;
    }
};
```

Here we used the operators `!=`, `<` and `<=` of `std::string`. The other comparison operators are now straightforward to implement:

```
struct Author
{
    // ...
    bool operator==(Author const& other)
    {
        return ((*this <= other) && (other <= *this));
    }

    bool operator!=(Author const& other)
    {
        return !(*this == other);
    }

    bool operator>=(Author const& other)
    {
        return other <= *this;
    }

    bool operator<(Author const& other)
    {
        return ((*this <= other) && !(*this == other));
    }

    bool operator>(Author const& other)
    {
        return other < *this;
    }
};
```

Of course, we could implement all other operators solely in terms of `operator<=()`.

III. Design Decisions

IV. Relation to N4126

Smolsky (2014c)Smolsky (2014a)Smolsky (2014b)

```
struct A
{
  short x;
  int   y;
  char  z;
};

struct B
{
  char  z;
  short x;
  int   y;
};

struct C
{
  int   y;
  short x;
  char  z;
};

int main()
{
  std::cout << sizeof(A) << ”\n” << sizeof(B) << ”\n” << sizeof(C);
}
```

V. Technical Specifications

VI. Acknowledgments

References

Smolsky, O. (2014a). N3950: Defaulted comparison operators. Technical report, C++ standards committee paper. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n3950.html.

Smolsky, O. (2014b). N4114: Defaulted comparison operators. Technical report, C++ standards committee paper. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4114.htm.

Smolsky, O. (2014c). N4126: Explicitly defaulted comparison operators. Technical report, C++ standards committee paper. http://isocpp.org/files/papers/n4126.htm.