| | |
|---|---|
| **Document number:** | *N3986* |
| **Date:** | *2014-04-25* |
| **Project:** | Programming Language C++, Evolution Working Group |
| **Reply-to:** | *"*Sebastian Davalle <sebastian.davalle@tallertechnologies.com>,<br>Daniel Gutson <daniel.gutson@tallertechnologies.com>,<br>Angel Bustamante <angel.bustamante@tallertechnologies.com>*"* |

# Adding Standard support to avoid padding within structures

## I. Motivation

### Target audience

This can be useful for the C++ Networking TS (Technical Specification). The proposal described here arises mostly when working in Telecommunications, Real-time and Embedded Systems.

### Overview

Sometimes programmers might need to avoid the bit-fields' alignment to get a compact type. Although this behavior can be obtained in most compilers by specifying the "packed" attribute, a more standard and simpler way to force it should exist (similar to the :0 syntax).

Consider the following example, from the communications protocol of a Fiber Optic device:

```
struct FogFrameHeader
{
     uint8_t         : 4;
     uint8_t valid   : 1;
     uint8_t         : 1;
     uint8_t id      : 2;
};
```

On some platforms, bit fields don't straddle bytes, on others they do. So this struct will unlikely take 1 byte.

```
struct FogFrame
{
     FogFrameHeader header;
     uint8_t data0;
     int32_t data1 :22;
     int32_t      :10;
};
```

In the example above, there is no guarantee that sizeof(FogFrame) will be 1 + 1 + 4 (implementation-defined).

# II. Proposal

Enable this feature by specifying a bit field of width -1 to tell the compiler that it should not insert spaces in such position; i.e, that the previous and next members should be adjacent in memory. With this syntax, the FogFrame could be written as:

```
struct FogFrame
{
     FogFrameHeader header;
     uint8_t: -1;                // No spaces here
     uint8_t  data0;
     uint8_t: -1;                // Ditto
     int32_t data1 :22;
     int32_t      :10;
};
```

Now consider the following situation:

```
struct S
{
   uint8_t c;
   char: -1;
   int32_t t;
};
```

This may lead to a memory access violation.

However, this should still be valid:

```
struct S
{
  uint8_t c;
  char: -1;
  int32_t t : 22;
};
```

When the structure member after the -1 bit field size is not a bit field, the rule of thumb for the validation should be:

**offsetof(S, t) % sizeof(t) == 0**

On the other hand, when the structure member after the -1 bit field size is a bit field, the compiler should determine whether to access the bit field bits as the underlying type, or with smaller units. For instance:

```
struct S
{
  uint8_t c;
  char: -1;
  int32_t t : 30;
};
```

The layout of this struct should be a 5 bytes memory region (sizeof(S) == 5, wasting 2 bits of the integer), and could be accessed as 5 adjacent bytes, or in a more optimized fashion (in terms of memory accesses) as one byte plus one 32-bit integer.

# II. Impact On the Standard

This proposal is an extension of the core language. It does not require changes to any standard classes, functions or headers.
As the standard specifies in 9.6/2, an unnamed bit field of width 0 forces alignment of the next bit field to the next type boundary, where type is the type of the member.

Add new bullet to **9.6 Bit-fields** **[class.bit]**

- A declaration for a bit-field of width -1 shall not align the next bit-field at an allocation unit boundary; in addition, the previous and next members shall be adjacent in memory. As a

special case, when the structure member after the -1 bit field size is not a bit field, the validation rule shall be: **offsetof(S, t) % sizeof(t) == 0** .
If the structure member after the -1 bit field size is a bit field, the compiler should determine whether to access the bit field bits as a the underlying type, or with smaller units. [ *Example:*

```
struct S
{
  uint8_t c;
  char: -1;
  int32_t t : 30;
};
```

The layout of this struct should be a 5 bytes memory region (sizeof(S) == 5 wasting 2 bits of the integer), and could be accessed as 5 adjacent bytes, or in a more optimized fashion (in terms of memory accesses) as one 32-bit integer plus one byte.
— *end example* ]

(9.6/5)

# III. Technical Specifications

None identified.

# IV. Acknowledgements


- Thanks to Pablo Miguel Oliva for additional comments on the proposal and the reviews.


# V. References


- ISOCPP Standard C++ - ISO/IEC JTC1 SC22 WG21 N 3690
- Sample code: http://compgroups.net/comp.std.c++/proposal-new-bitfield-value-to-ensure-structure/2126142