

# GCC Support

## General GCC Support Notes

C++test supports the following distributions of the GCC compiler:

- Windows
  - GCC 4.0.x
  - GCC 4.1.x
  - GCC 4.2.x
  - GCC 4.3.x
  - GCC 4.4.x
  - GCC 4.5.x
  - GCC 4.6.x
  - GCC 4.7.x
  - GCC 4.8.x (x86 and x86-64)
  - GCC 4.9.x (x86 and x86-64)
  - GCC 5.x (x86 and x86-64)
  - GCC 6.x (x86 and x86-64)
  - GCC 7.x (x86 and x86-64)
- Linux
  - GCC 4.0.x (x86 and x86-64)
  - GCC 4.1.x (x86 and x86-64)
  - GCC 4.2.x (x86 and x86-64)
  - GCC 4.3.x (x86 and x86-64)
  - GCC 4.4.x (x86 and x86-64)
  - GCC 4.5.x (x86 and x86-64)
  - GCC 4.6.x (x86 and x86-64)
  - GCC 4.7.x (x86 and x86-64)
  - GCC 4.8.x (x86 and x86-64)
  - GCC 4.9.x (x86 and x86-64)
  - GCC 5.x (x86 and x86-64)
  - GCC 6.x (x86 and x86-64)
  - GCC 7.x (x86 and x86-64)
- GCC-based cross-compilers listed in [Compilers](#).
- Other GCC-based cross-compilers and custom GCC compiler builds based on GCC compiler versions listed above. Heavily modified GCC-based compilers, as well as their non-standard extensions, may not be supported.

To use any of these supported distributions, the directory containing the GCC executable must be included in the \$PATH environment variable.

## Unsupported Compiler Extensions for GCC Compilers

The following sections detail the GCC compiler extensions that C++test *does not* currently support. <http://gcc.gnu.org/onlinedocs> was used as the source of information on GNU compiler extensions.

These limitations also apply to appropriate versions of GCC-based cross-compilers listed in [Compilers](#), other GCC-based cross-compilers, and custom GCC compiler builds.

### GCC 4.0.x

#### Unsupported Features

- **Arrays of Variable Length as Arguments to Functions**

```
void tester (int len, char data[len][len])
{
}
```

- **Complex Numbers**
- **Nested Functions**
- **Virtual Function With Overridden Return Type**

```

class A
{
    public:
        virtual void* a();
};
class B:
    public A
{
    public:
        virtual B* a(); // Return type changed from void* to (compatible) B*.
                        // It is ok in GCC but EDG will complain
};

```

- **Java Extensions (like extern "Java", java attributes)**
- **Offsetof extension**
- **Restricting Pointer Aliasing for Member functions**

```

class T
{
    public:
        void fn();
};
void T::fn () __restrict__ // EDG won't compile this
{
}

```

## Incompatibilities

- **Function Attributes**

```

void fatal () __attribute__ ((noreturn));
void fatal () { }
typedef void voidfn ();
volatile voidfn fatal; // EDG: declaration is incompatible with
// "void fatal()"

```

- **Friend Declarations**

```

class A;
namespace N {
    class B {
        friend class A; // In GCC 4.0+ it refer to N::A (which has not been
                        // declared yet) But EDG and older GCCs refer to ::A
        int _private;
    };
    class A {
        void foo()
        {
            B b;
            b._private = 0; // EDG inaccessible field.
        }
    };
}

```

## GCC 4.1.x

Same as for GCC 4.0.x.

## GCC 4.2.x

Same as for GCC 4.0.x.

## **GCC 4.3.x**

Same as for GCC 4.0.x

## **GCC 4.4.x**

Same as for GCC 4.0.x

## **GCC 4.5.x**

Same as for GCC 4.0.x

## **GCC 4.6.x**

Same as for GCC 4.0.x

## **GCC 4.7.x**

Same as for GCC 4.0.x

## **GCC 4.8.x**

Same as for GCC 4.0.x

## **GCC 4.9.x**

Same as for GCC 4.0.x

## **GCC 5.x**

Same as for GCC 4.0.x.

## **GCC 6.x**

Same as for GCC 4.0.x

Missing C++17 features:

- N4295 Folding expressions
- P0036R0 Unary Folds and Empty Parameter Packs

## **GCC 7.x**

Same as for GCC 4.0.x

Missing C++17 features:

- N4295 Folding expressions
- P0036R0 Unary Folds and Empty Parameter Packs
- P0017R1 Extension to aggregate initialization
- P0091R4 Template argument deduction for class templates
- P0127R2 Declaring non-type template parameters with auto
- P0135R1 Guaranteed copy elision
- P0136R1 Rewording inheriting constructors (core issue 1941 et al)
- P0145R3 Refining Expression Evaluation Order for Idiomatic C++
- P0195R2 Pack expansions in using-declarations
- P0522R0 DR 150, Matching of template template arguments