Runtime Type Information 5

This chapter explains the use of Runtime Type Information (RTTI).

In C++, pointers to classes have a *static* type, which is the type written in the pointer declaration, and a *dynamic* type, which is determined by the actual type referenced. The dynamic type of the object could be any class type derived from the static type.

```
class A {};

class B: public A {};

extern B bv;

extern A* ap = &bv;
```

Here, `ap` has the static type A* and a dynamic type B*.

RTTI allows the programmer to determine the dynamic type of the pointer.

RTTI Options

RTTI support requires significant resources to implement. To enable RTTI implementation and to enable recognition of the associated typeid keyword, use the option `-features=rtti`. To disable RTTI implementation and to disable recognition of the associated typeid keyword, use the option `-features=no%rtti` (default).

`typeid` Operator

The typeid operator produces a reference to an object of class `type_info`, which describes the most-derived type of the object. In order to make use of the `typeid()` function, source code must `#include` the `<typeinfo.h>` header file. The primary value of this operator/class combination is in comparisons. In such comparisons, the top-level const-volatile qualifiers are ignored, as in the following example.

```
#include <typeinfo.h>

#include <assert.h>

void use_of_typeinfo( )

{

      A a1;

      const A a2;

      assert( typeid(a1) == typeid(a2) );

      assert( typeid(A)  == typeid(const A) );

      assert( typeid(A)  == typeid(a2) );

      assert( typeid(A)  == typeid(const A&) );
```

```
    B b1;

    assert( typeid(a1) != typeid(b1) );

    assert( typeid(A)  != typeid(B) );
}
```

The typeid operator will raise a bad_typeid exception when given a null pointer.

`type_info` Class

The class `type_info` describes type information generated by the `typeid` operator. The primary functions provided by `type_info` are equality, inequality, `before` and `name`. From `<typeinfo.h>`, the definition is:

```
class type_info {

    public:

        virtual ~type_info( );

        bool operator==( const type_info &rhs ) const;

        bool operator!=( const type_info &rhs ) const;

        bool before( const type_info &rhs ) const;

        const char *name( ) const;

    private:

        type_info( const type_info &rhs );

        type_info &operator=( const type_info &rhs );
```

```
    };
```

The before function compares two types relative to their implementation-dependent collation order. The `name` function returns an implementation-defined, null-terminated, multi-byte string, suitable for conversion and display.

The constructor is a private member function, so there is no way for a programmer to create a variable of type "`type_info`". The only source of "`type_info`" objects is in the "`typeid`" operator.
