Der **typeid** - Operator erlaubt es, den Typ eines Objektes zur Laufzeit zu bestimmen:

```
typeid( type-id )
typeid( expression )
```

Ergebnis von **typeid** ist eine Referenz auf ein Objekt der Klasse type_info: **const type_info &**.

Die Referenz auf ein **type_info**-Objekt repräsentiert entweder die **`type-id`** oder den Typ des **`expression`**, abhängig von einer der beiden Formen von **`typeid`**.

Die **class type_info** beschreibt Type-Information, die **class type_info** wird intern vom Compiler generiert.

Objekte dieser Klasse speichern einen Zeiger auf einen Namen des Typs. Die **class type_info** speichert einen codierten Wert, der zwei Typen vergleichbar oder anordenbar macht.

```
class type_info {
public:
    _CRTIMP virtual ~type_info();
    _CRTIMP bool operator==(const type_info& rhs) const;
    _CRTIMP bool operator!=(const type_info& rhs) const;
    _CRTIMP int before(const type_info& rhs) const;
    _CRTIMP const char* name() const;
    _CRTIMP const char* raw_name() const;
};
```

Der **typeid**-Operator macht einen **run-time check**, wenn er auf einen **l-value** einer **polymorphen Klasse** angewendet wird, wobei der wahre Typ des Objekts nicht durch die statischen Informationen zur Compiletime geliefert werden. Eine **polymorphe Klasse** ist eine Klasse mit mindestens einer **virtuellen Methode**.

Wenn ein Ausdruck eine Zeigervariable auf einen Basisklassentyp ist, die Zeigervariable jedoch auf ein abgeleitetes Objekt zeigt, dann ist eine **type_info**-Referenz für die abgeleitete Klasse das Ergebnis.

Der Ausdruck muß wiederum auf einen polymorphen Typ (Klasse mit virtuellen Methoden) verweisen. Wird nicht auf einen polymorphen Typ verwiesen, dann ist das Ergebnis die Referenz auf ein type_info-Objekt , welches vom Typ des Ausdrucks zur Compilezeit abhängt.

Zeiger müssen dereferenziert werden, um das genutzte Objekt zu identifizieren. Ohne Dereferenzierung des Zeigers ist das Ergebnis eine Referenz auf ein type_info-Objekt des Zeigers zur Compile-Time, nicht jedoch auf das aktuell referenzierte Objekt.

```
#include <iostream>          // rtti3.cpp,   compile with: /GR /EHsc
#include <typeinfo.h>
using namespace std;

class Base {
    public: virtual void vvfunc() {}
};

class Derived : public Base {};

void main()
{
   Derived* pd = new Derived;
   Base* pb = pd;
   cout << typeid( pb ).name() << endl;    //prints "class Base *"
   cout << typeid( *pb ).name() << endl;   //prints "class Derived"
   cout << typeid( pd ).name() << endl;    //prints "class Derived *"
   cout << typeid( *pd ).name() << endl;   //prints "class Derived"
   delete pd; pd = 0;
}
```

```
// Projekteigenschaften | C/C++ | Sprache |
// Laufzeit-Typinformationen aktivieren | Ja (/GR)
#include <iostream>                                    //rtti2.cpp
#include <typeinfo.h>
using namespace std;

class Basis {
    public:
    // Die folgende Methode ist notwendig fuer die korrekte Ausfuehrung der Anweisung
    // cout<<typeid(*p).name()<<endl; (siehe unten), ohne eine virtual-Methode wird
    // bei der genannten Anweisung nur "class Basis" ausgegeben, obwohl "class
    // Abgeleitet" korrekt ist !
       virtual void f(){}
};

class Abgeleitet: public Basis{};

void main(){
   Abgeleitet *d = new Abgeleitet;
   Basis *p = d, *pNull=0;
   Abgeleitet Objekt1, Objekt2;

   cout<<typeid(p).name()<<endl;  // class Basis *
   cout<<typeid(*p).name()<<endl; // class Abgeleitet
   cout<<typeid(d).name()<<endl;  // class Abgeleitet *
   cout<<typeid(*d).name()<<endl; // class Abgeleitet

   if(typeid(*d)==typeid(*p)) cout<<"true\n";  // true
   else cout<<"false\n";

   if(typeid(Objekt2)==typeid(Objekt1)) cout<<"true\n"; // true
   else cout<<"false\n";

   cout<<typeid(pNull).name()<<endl;   // class Basis

   if(typeid(&Objekt2)==typeid(pNull)) cout<<"true\n";
   else cout<<"false\n";               // false


   try {
         cout<<typeid(*pNull).name()<<endl; // error
   }
   catch(...){
         cout<<"Error: typeid(*pNull).name()\n";
   }

   try {
         if(typeid(Objekt2)==typeid(*pNull)) cout<<"true\n"; //error
         else cout<<"false\n";
   }
   catch(...){
               cout<<"Error: typeid(*pNull)\n";
   }
   cin.get();
}
```

Ausgabe:

```
class Basis *
class Abgeleitet
class Abgeleitet *
class Abgeleitet
true
true
class Basis *
false
Error: typeid(*pNull).name()
Error: typeid(*pNull)
```

```
#include <iostream>                //rtti1.cpp
#include <typeinfo.h>
using namespace std;

// Beispiel fuer die Bestimmung der Typinformation zur Laufzeit
//
// Eigenschaften | C,C++ | Sprache | Laufzeit- u. Typinformationen
// aktivieren:   Ja (/GR)

class X {
    protected: int wert;
    public:
       X(int w):wert(w){ cout<<"Konstruktor X"<<endl; }
       virtual ~X(void){ cout<<"Destruktor X, wert = "<<wert<<endl; }
       virtual void out(){ cout<<"type = "<<typeid(*this).name()<<endl; }
};

class Y : public X {
    public:
       Y(int a) : X(a) { cout<<"Konstruktor Y"<<endl; }
       virtual ~Y(void){ cout<<"Destruktor Y, wert = "<<wert<<endl; }
       virtual void out(){ /*cout<<"type = "<<typeid(*this).name()<<endl;*/
                           X::out();
       }
};

void main(){
            X *py = new Y(5);
            cout<<typeid(*py).name()<<endl;
            X *px = new X(3);
            cout<<typeid(*px).name()<<endl;
            px->out();
            py->out();
            delete px; px = 0; delete py; py = 0;
            cin.get();
}
```

Ausgabe:

```
Konstruktor X
Konstruktor Y
class Y
Konstruktor X
class X
type = class X
type = class Y

Destruktor X, wert = 3
Destruktor Y, wert = 5
Destruktor X, wert = 5
```