Run-Time Type Information

CP3 Presentation

What does RTTI mean ?

Run-Time Type Identification

Run-Time Type Information

Purpose of RTTI

- To provide information about the type of an object during the run-time.

- As a consequence – to support polymorphism.

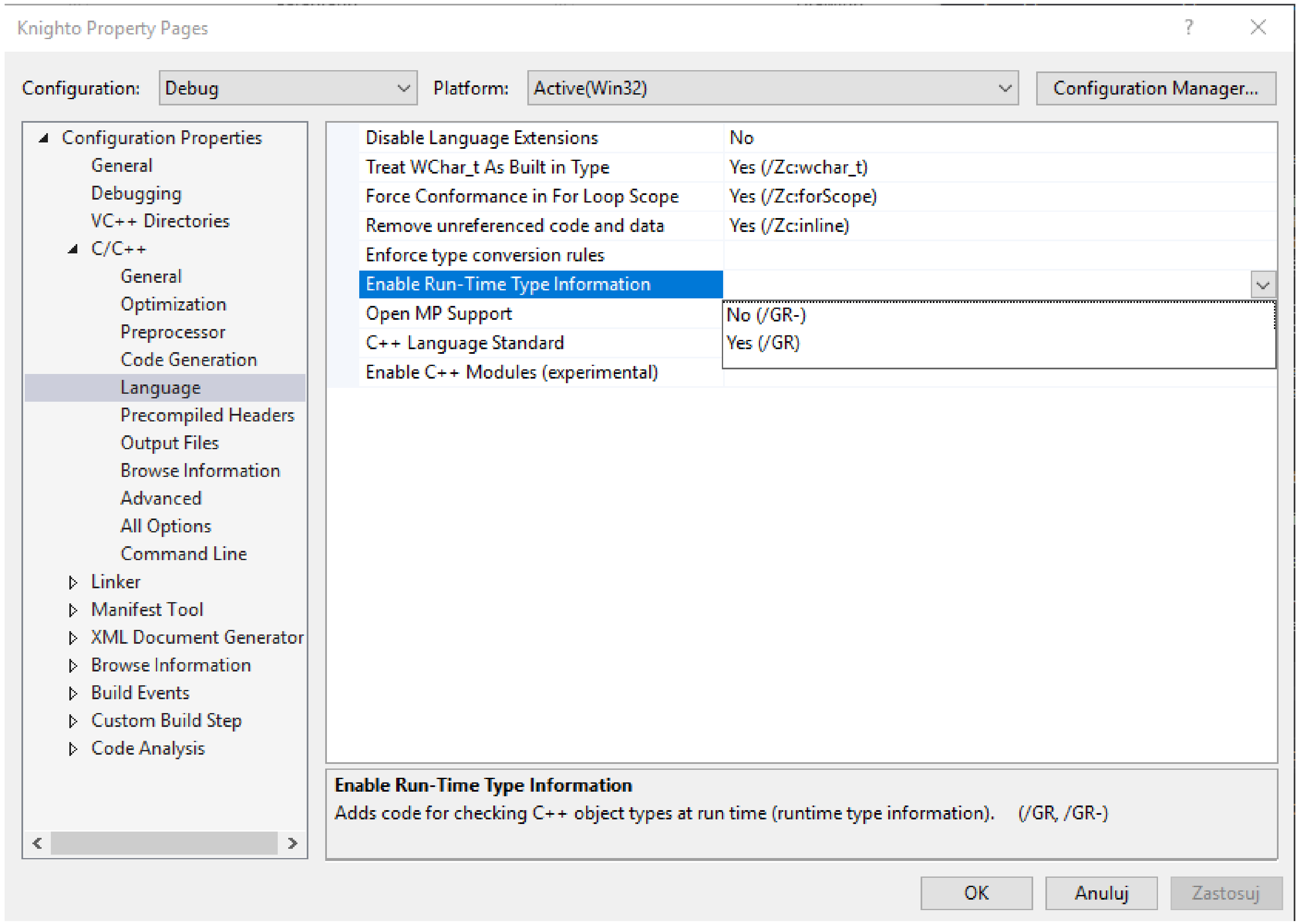
How RTTI works ?

- C++ RTTI can be used to do safe typecasts, using the dynamic_cast<> operator, and to manipulate type information at run time, using the typeid() operator and std::type_info class.

- **RTTI is available only for classes which are polymorphic, which means they have at least one virtual method**. In practice, this is not a limitation because base classes must have a virtual destructor to allow objects of derived classes to perform proper cleanup if they are deleted from a base pointer.

- RTTI is optional with some compilers; the programmer can choose at compile time whether to include the functionality. There may be a resource cost to making RTTI available even if a program does not use it.

Enabling RTTI in Visual Studio 2017



Disable Language Extensions	No
Treat WChar_t As Built in Type	Yes (/Zc:wchar_t)
Force Conformance in For Loop Scope	Yes (/Zc:forScope)
Remove unreferenced code and data	Yes (/Zc:inline)
Enforce type conversion rules	
Enable Run-Time Type Information	
Open MP Support	No (/GR-)
C++ Language Standard	Yes (/GR)
Enable C++ Modules (experimental)	

Enable Run-Time Type Information
Adds code for checking C++ object types at run time (runtime type information). (/GR, /GR-)

dynamic_cast operator

Most often used element of RTTI. Operator dynamic_cast creates pointer of derived class from the pointer of base class.

```
TYPE& dynamic_cast<TYPE&> (object);
TYPE* dynamic_cast<TYPE*> (object);
```

For pointers it returns NULL if TYPE is neither type of object nor its parent

For references it throws exception std::bad_cast if TYPE is neither type of expression nor its parent

Example

```
#include <typeinfo>

class A{
public:
	virtual ~A() {}
};
class B : public A {};
class C       {};

int main()
{
	A objectA;
	B objectB;

	A* pointerA = &objectB;
	B* b1 = dynamic_cast<B*> (&objectA);     // NULL, because 'a' is not a 'B'
	B* b2 = dynamic_cast<B*> (pointerA);     // 'b'
	C* c = dynamic_cast<C*> (pointerA);      // NULL.

	A& ar = dynamic_cast<A&> (*pointerA);    //Ok.
	B& br = dynamic_cast<B&> (*pointerA);    //Ok.
	C& cr = dynamic_cast<C&> (*pointerA);    //std::bad_cast
	system("pause");
	return 0;
}
```

typeid operator

- The typeid() operator returns the reference to the object of type type_info, which is a class defined in header file <typeinfo>.

Syntax:

```
typeid(type name)
typeid(expression)
```

Type_info class:

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

Obtaining a class (type) name

Syntax:

typeid(expression).name()

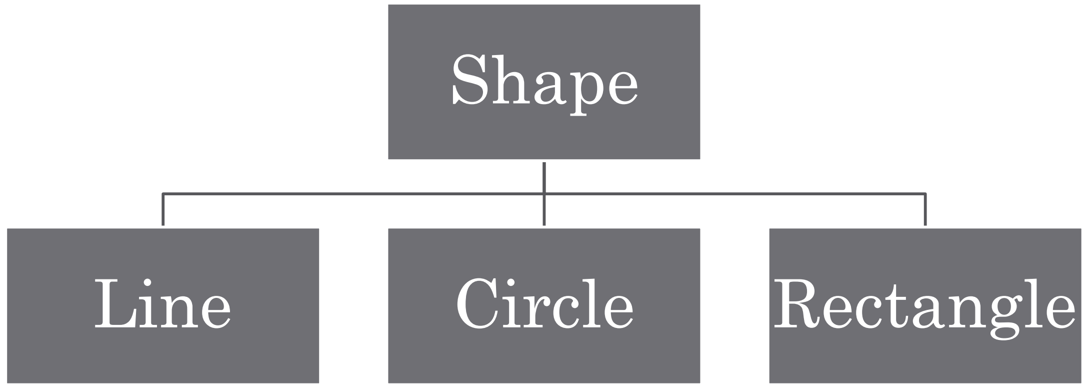
Purpose:

Used to print the human-readable name of the type.

Example of typeid() use

```
class A{
public:
	virtual ~A() {}
};
class B : public A {};

int main()
{
	typeid(A).name();   // returns "A"

	A *ptr;
	typeid(ptr).name();        // returns "A*"
	typeid(*ptr).name();       // returns "A"

	typeid(A).before(typeid(B)); // returns true

}
```

Example 2



```
Shape *sp = new Circle;

typeid(shape) == typeid(*sp)          //returns false
typeid(shape).before(typeid(*sp))     //returns true
typeid(sp).name()                     //returns „circle*”
typeid(*sp).name()                    //returns „circle”
```

RTTI – laboratory assignment

Sources of information:

- https://en.wikibooks.org
- C++ Primer Plus, Sixth Edition – Stephen Prata