How to check an identifier or a type (such as int) is an integer … etc in C?

[intro]

There is one library <limits> implemented it.

You can check an identifier or a type information using std::numeric\_limits<T>.

[ADT]

1.For C++98,

template <class T> class numeric\_limits {

public:

static const bool is\_specialized = false;

static T min() throw();

static T max() throw();

static const int digits = 0;

static const int digits10 = 0;

static const bool is\_signed = false;

static const bool is\_integer = false;

static const bool is\_exact = false;

static const int radix = 0;

static T epsilon() throw();

static T round\_error() throw();

static const int min\_exponent = 0;

static const int min\_exponent10 = 0;

static const int max\_exponent = 0;

static const int max\_exponent10 = 0;

static const bool has\_infinity = false;

static const bool has\_quiet\_NaN = false;

static const bool has\_signaling\_NaN = false;

static const float\_denorm\_style has\_denorm = denorm\_absent;

static const bool has\_denorm\_loss = false;

static T infinity() throw();

static T quiet\_NaN() throw();

static T signaling\_NaN() throw();

static T denorm\_min() throw();

static const bool is\_iec559 = false;

static const bool is\_bounded = false;

static const bool is\_modulo = false;

static const bool traps = false;

static const bool tinyness\_before = false;

static const float\_round\_style round\_style = round\_toward\_zero;

};

2. For C++11,

template <class T> class numeric\_limits {

public:

static constexpr bool is\_specialized = false;

static constexpr T min() noexcept { return T(); }

static constexpr T max() noexcept { return T(); }

static constexpr T lowest() noexcept { return T(); }

static constexpr int digits = 0;

static constexpr int digits10 = 0;

static constexpr bool is\_signed = false;

static constexpr bool is\_integer = false;

static constexpr bool is\_exact = false;

static constexpr int radix = 0;

static constexpr T epsilon() noexcept { return T(); }

static constexpr T round\_error() noexcept { return T(); }

static constexpr int min\_exponent = 0;

static constexpr int min\_exponent10 = 0;

static constexpr int max\_exponent = 0;

static constexpr int max\_exponent10 = 0;

static constexpr bool has\_infinity = false;

static constexpr bool has\_quiet\_NaN = false;

static constexpr bool has\_signaling\_NaN = false;

static constexpr float\_denorm\_style has\_denorm = denorm\_absent;

static constexpr bool has\_denorm\_loss = false;

static constexpr T infinity() noexcept { return T(); }

static constexpr T quiet\_NaN() noexcept { return T(); }

static constexpr T signaling\_NaN() noexcept { return T(); }

static constexpr T denorm\_min() noexcept { return T(); }

static constexpr bool is\_iec559 = false;

static constexpr bool is\_bounded = false;

static constexpr bool is\_modulo = false;

static constexpr bool traps = false;

static constexpr bool tinyness\_before = false;

static constexpr float\_round\_style round\_style = round\_toward\_zero;

};

[P.S.]

(1)std::numeric\_limits<T> is a class.

Thus, the statement

std::numeric\_limits<T>();

will give you compiler error.

(2)Also, the statement

std::numeric\_limits

will give you compiler error.

[syntax]

std::numeric\_limits<T>

where T is the type for template.

[namespace]

std

[library]

<limits>

[code]

#include <iostream>

#include <string>

#include <limits>

using namespace std;

template <class T>

void Print(string msg)

{

cout<<"Set cout as std::boolalpha"<<endl;

cout << std::boolalpha;

cout<<"----"<<endl;

cout<<"For "<<msg<<endl;

cout << "Minimum value for "<<msg<<": " << std::numeric\_limits<T>::min() << endl;

cout << "Maximum value for "<<msg<<": " << std::numeric\_limits<T>::max() << endl;

cout << msg<<" is signed: " << std::numeric\_limits<T>::is\_signed << endl;

cout << "Non-sign bits in "<<msg<<": " << std::numeric\_limits<T>::digits << endl;

cout << msg<<" has infinity: " << std::numeric\_limits<T>::has\_infinity << endl;

cout<<"-----"<<endl;

cout << msg<<" is an integer: " << std::numeric\_limits<T>::is\_integer << endl;

cout << msg<<" is exact: " << std::numeric\_limits<T>::is\_exact << endl;

}

int main ()

{

Print<int>("int");

Print<long int>("long int");

Print<long long int>("long long int");

Print<unsigned int>("unsigned int");

Print<unsigned long int>("unsigned long int");

cout<<"~~~~~~~~~~~~~~~~~~~~"<<endl;

Print<short>("short");

Print<unsigned short>("unsigned short");

cout<<"~~~~~~~~~~~~~~~~~~~~"<<endl;

Print<float>("float");

cout<<"~~~~~~~~~~~~~~~~~~~~"<<endl;

Print<double>("double");

Print<long double>("long double");

cout<<"~~~~~~~~~~~~~~~~~~~~"<<endl;

Print<char>("char");

Print<unsigned char>("unsigned char");

cout<<"~~~~~~~~~~~~~~~~~~~~"<<endl;

return 0;

}

[result]

Set cout as std::boolalpha

----

For int

Minimum value for int: -2147483648

Maximum value for int: 2147483647

int is signed: true

Non-sign bits in int: 31

int has infinity: false

-----

int is an integer: true

int is exact: true

Set cout as std::boolalpha

----

For long int

Minimum value for long int: -2147483648

Maximum value for long int: 2147483647

long int is signed: true

Non-sign bits in long int: 31

long int has infinity: false

-----

long int is an integer: true

long int is exact: true

Set cout as std::boolalpha

----

For long long int

Minimum value for long long int: -9223372036854775808

Maximum value for long long int: 9223372036854775807

long long int is signed: true

Non-sign bits in long long int: 63

long long int has infinity: false

-----

long long int is an integer: true

long long int is exact: true

Set cout as std::boolalpha

----

For unsigned int

Minimum value for unsigned int: 0

Maximum value for unsigned int: 4294967295

unsigned int is signed: false

Non-sign bits in unsigned int: 32

unsigned int has infinity: false

-----

unsigned int is an integer: true

unsigned int is exact: true

Set cout as std::boolalpha

----

For unsigned long int

Minimum value for unsigned long int: 0

Maximum value for unsigned long int: 4294967295

unsigned long int is signed: false

Non-sign bits in unsigned long int: 32

unsigned long int has infinity: false

-----

unsigned long int is an integer: true

unsigned long int is exact: true

~~~~~~~~~~~~~~~~~~~~

Set cout as std::boolalpha

----

For short

Minimum value for short: -32768

Maximum value for short: 32767

short is signed: true

Non-sign bits in short: 15

short has infinity: false

-----

short is an integer: true

short is exact: true

Set cout as std::boolalpha

----

For unsigned short

Minimum value for unsigned short: 0

Maximum value for unsigned short: 65535

unsigned short is signed: false

Non-sign bits in unsigned short: 16

unsigned short has infinity: false

-----

unsigned short is an integer: true

unsigned short is exact: true

~~~~~~~~~~~~~~~~~~~~

Set cout as std::boolalpha

----

For float

Minimum value for float: 1.17549e-38

Maximum value for float: 3.40282e+38

float is signed: true

Non-sign bits in float: 24

float has infinity: true

-----

float is an integer: false

float is exact: false

~~~~~~~~~~~~~~~~~~~~

Set cout as std::boolalpha

----

For double

Minimum value for double: 2.22507e-308

Maximum value for double: 1.79769e+308

double is signed: true

Non-sign bits in double: 53

double has infinity: true

-----

double is an integer: false

double is exact: false

Set cout as std::boolalpha

----

For long double

Minimum value for long double: 0

Maximum value for long double: inf

long double is signed: true

Non-sign bits in long double: 113

long double has infinity: true

-----

long double is an integer: false

long double is exact: false

~~~~~~~~~~~~~~~~~~~~

Set cout as std::boolalpha

----

For char

Minimum value for char: ﾀ

Maximum value for char: 

char is signed: true

Non-sign bits in char: 7

char has infinity: false

-----

char is an integer: true

char is exact: true

Set cout as std::boolalpha

----

For unsigned char

Minimum value for unsigned char:

Maximum value for unsigned char:

unsigned char is signed: false

Non-sign bits in unsigned char: 8

unsigned char has infinity: false

-----

unsigned char is an integer: true

unsigned char is exact: true

~~~~~~~~~~~~~~~~~~~~

[ref]

<https://cplusplus.com/reference/limits/numeric_limits/>