### 一、VS2022支持c++20配置

1. 创建一个C++空项目；

2. 添加一个hello.cpp源文件

import <iostream>;

using namespace std;

int main()

{

cout << "Hello, World!" << endl;

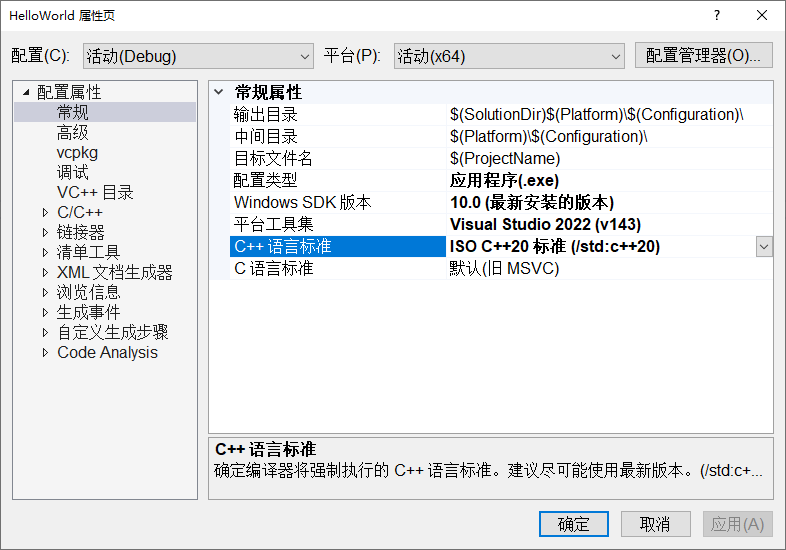
return 0;

}

1. 点击“属性”



1. 选择“C++语言标准”为“ISO C++20标准”



1. 新建一个标头文件HeaderUnits.h

#pragma once

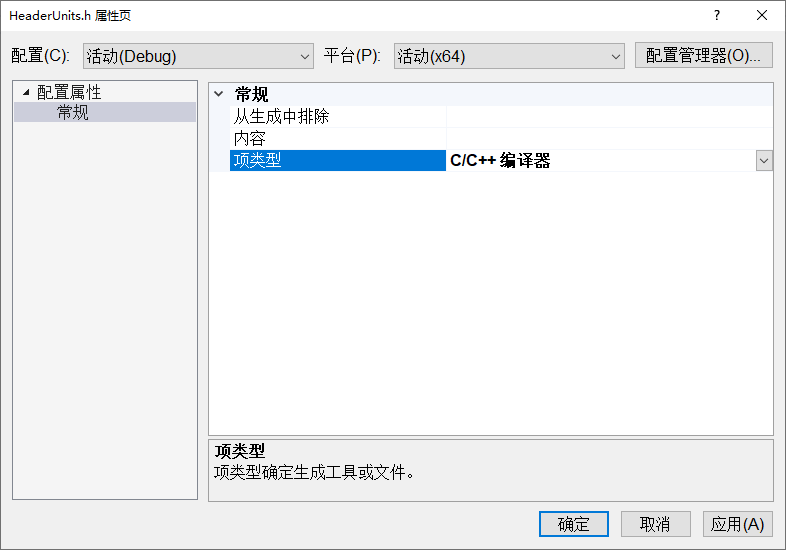
import <iostream>;

//..

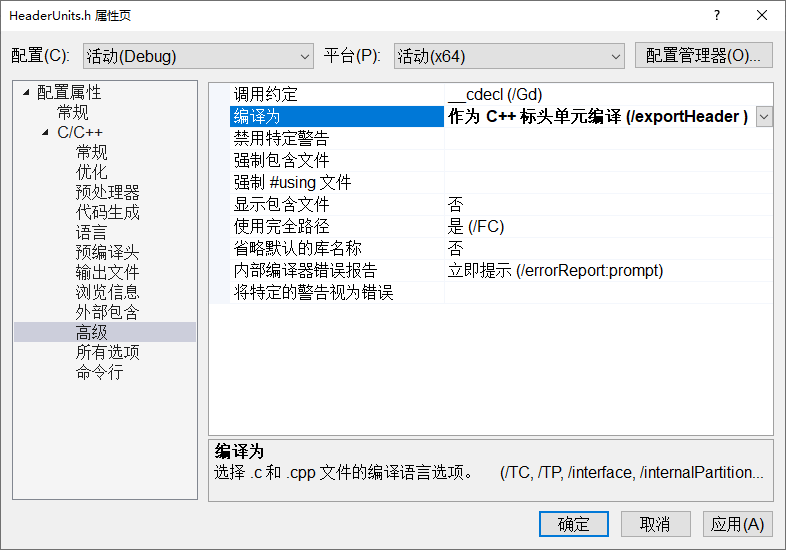
1. 右击HeaderUnists.h文件，选择属性



1. 在项类型中选择C/C++编译器，选择完了不要忘记点“应用”



1. 选择“C/C++高级-编译为”“作为C++标头单元编译export/Header”，点击“应用”



1. 点击“确定”，然后就可以编译运行了。

### Chapter 1 A Crash Course in C++ and the Standard Library

// helloworld.cpp

import <iostream>;

using namespace std;

int main()

{

cout << "Hello, World!" << endl;

return 0;

}

//usingnamespaces.cpp

import <iostream>;

namespace mycode

{

void foo()

{

std::cout << "foo() called in the mycode namespace\n";

}

}

using namespace mycode;

int main()

{

mycode::foo(); // Calls the "foo" function in the "mycode" namespace

foo(); // implies mycode::foo();

}

//01\_variables.cpp

// NOTE: Most compilers will issue a warning or an error

// when code is using uninitialized variables. Some compilers

// will generate code that will report an error at run time.

import <iostream>;

//需在标头文件中也加入这一段代码，以屏蔽错误提示

#if \_\_INTELLISENSE\_\_

#include <format>

#else

import <format>;

#endif

using namespace std;

int main()

{

int uninitializedInt;

int initializedInt{ 7 };

cout << format("{} is a random value\n", uninitializedInt);

cout << format("{} was assigned an initial value\n", initializedInt);

}

//02\_numeric\_limits.cpp

import <iostream>;

import <limits>;

#if \_\_INTELLISENSE\_\_

#include <format>

#else

import <format>;

#endif

using namespace std;

int main()

{

cout << "int:\n";

cout << format("Max int value: {}\n", numeric\_limits<int>::max());

cout << format("Min int value:{}\n", numeric\_limits<int>::min());

cout << format("Lowest int value: {}\n", numeric\_limits<int>::lowest());

cout << "\ndouble:\n";

cout << format("Max double value: {}\n", numeric\_limits<double>::max());

cout << format("Min double value: {}\n", numeric\_limits<double>::min());

cout << format("Lowest double value: {}\n", numeric\_limits<double>::lowest());

}

//03\_casting.cpp

int main()

{

float myFloat{ 3.14f };

int i1{ (int)myFloat };// method 1

int i2{ int(myFloat) };// method 2

int i3{ static\_cast<int>(myFloat) };// method 3

short someShort{ 16 };

long someLong{ someShort };// no explicit cast needed

}

//operators.cpp

import <iostream>;

using namespace std;

int main()

{

int someInteger{ 256 };

short someShort;

long someLong;

float someFloat;

double someDouble;

someInteger++;

someInteger \*= 2;

someShort = static\_cast<short>(someInteger);

someLong = someShort \* 10000;

someFloat = someLong + 0.785f;

someDouble = static\_cast<double>(someFloat) / 100000;

cout << someDouble << endl;

}

//StronglyTypedEnums.cpp

enum class PieceType

{

King = 1,

Queen,

Rook = 10,

Pawn

};

int main()

{

{

PieceType piece{ PieceType::King };

if (piece == PieceType::King)

{

/\* ... \*/

}

}

{

using enum PieceType;

PieceType piece{ King };

}

{

using PieceType::King;

PieceType piece{ King };

piece = PieceType::Queen;

}

}

//structtest.cpp

import <iostream>;

#if \_\_INTELLISENSE\_\_

#include <format>

#else

import <format>;

#endif

import employee;

using namespace std;

int main()

{

// create and populate an employee

Employee anEmployee;

anEmployee.firstInitial = 'J';

anEmployee.lastInitial = 'D';

anEmployee.employeeNumber = 42;

anEmployee.salary = 80000;

// output the values of an employee

cout << format("Employee: {}{}\n", anEmployee.firstInitial, anEmployee.lastInitial);

cout << format("Number: {}\n", anEmployee.employeeNumber);

cout << format("Salary: ${}\n", anEmployee.salary);

}

//employee.cppm

export module employee;

export struct Employee {

char firstInitial;

char lastInitial;

int employeeNumber;

int salary;

};

//01\_if\_else.cpp

int main()

{

int i{ 3 };

if (i > 4) {

// Do something.

}

else if (i > 2) {

// Do something else.

}

else {

// Do something else.

}

}

//02\_fallthrough.cpp

int main()

{

enum class Mode { Default, Custom, Standard };

int value{ 42 };

Mode mode{ Mode::Custom };

switch (mode) {

using enum Mode;

case Custom:

value = 84;

[[fallthrough]];

case Standard:

case Default:

// Do something with value ...

break;

}

}

//03\_ConditionalOperator.cpp

import <iostream>;

using namespace std;

int main()

{

int i{ 3 };

cout << ((i > 2) ? "yes" : "no")<<endl;

cout << (i > 2 ? "yes" : "no");

}

//01\_SpaceshipOperator.cpp

import <compare>;

import <iostream>;

using namespace std;

int main()

{

int i{ 11 };

strong\_ordering result{ i <=> 0 };//三向比较运算

if (result == strong\_ordering::less) { cout << "less" << endl; }

if (result == strong\_ordering::greater) { cout << "greater" << endl; }

if (result == strong\_ordering::equal) { cout << "equal" << endl; }

}

//02\_NamedComparisonFunction.cpp

import <compare>;

import <iostream>;

using namespace std;

int main()

{

int i{ 11 };

strong\_ordering result{ i <=> 0 };

if (is\_lt(result)) { cout << "less" << endl; }

if (is\_gt(result)) { cout << "greater" << endl; }

if (is\_eq(result)) { cout << "equal" << endl; }

}

//functions.cpp

import <iostream>;

#if \_\_INTELLISENSE\_\_

#include <format>

#else

import <format>;

#endif

using namespace std;

void myFunction(int i, char c);

int addNumbers(int number1, int number2);

double addNumbers(double a, double b);

int main()

{

int someInt{ 6 };

char someChar{ 'c' };

myFunction(8, 'a');

myFunction(someInt, 'b');

myFunction(5, someChar);

int sum{ addNumbers(5, 3) };

cout << format("{}\n", addNumbers(1, 2)); // Calls the integer version

cout << format("{}\n", addNumbers(1.11, 2.22)); // Calls the double version

}

void myFunction(int i, char c)

{

cout << format("the value of i is {}\n", i);

cout << format("the value of c is {}\n", c);

}

int addNumbers(int number1, int number2)

{

cout << format("Entering function {}\n", \_\_func\_\_);//函数名称

return number1 + number2;

}

double addNumbers(double a, double b)

{

return a + b;

}

//01\_nodiscard.cpp

[[nodiscard]] int func()

{

return 42;

}

int main()

{

func();//返回值被忽略

int x{ func() };

}

//02\_maybe\_unsued.cpp

int func(int param1, [[maybe\_unused]] int param2)

{

return 42;

}

int main()

{

int result{ func(1, 2) };

}

//03\_noreturn.cpp

#include <cstdlib>

import <iostream>;

[[noreturn]] void forceProgramTermination();

bool isDongleAvailable();

bool isFeatureLicensed(int /\*featureId\*/);

int main()

{

bool isLicensed{ isFeatureLicensed(42) };

std::cout << isLicensed << std::endl;

}

[[noreturn]] void forceProgramTermination()

{

std::exit(1);// Defined in <cstdlib>

}

bool isDongleAvailable()

{

bool isAvailable{ false };

// Check whether a licensing dongle is available...

return isAvailable;

}

bool isFeatureLicensed(int /\*featureId\*/)

{

if (!isDongleAvailable()) {

// No licensing dongle found, abort program execution!

forceProgramTermination();

}

else {

bool isLicensed{ false };

// Dongle available, perform license check of the given feature...

return isLicensed;

}

}

//04\_deprecated.cpp

[[deprecated("Unsafe method, please use xyz")]] void func() {}

int main()

{

func();

}

//05\_likelihood.cpp

//These attributes are rarely required

int main()

{

int value{ 4 };

if (value > 11) [[unlikely]] { /\* Do something ... \*/ }

else { /\* Do something else... \*/ }

switch (value)

{

[[likely]] case 1:

// Do something ...

break;

case 2:

// Do something...

break;

[[unlikely]] case 12:

// Do something...

break;

}

}

//01\_c\_array.cpp

import <array>;

int main()

{

{

int myArray[3];

myArray[0] = 0;

myArray[1] = 0;

myArray[2] = 0;

}

{

int myArray[3] = { 0 };

}

{

int myArray[3] = {};

}

{

int myArray[3]{};

}

{

int myArray[]{ 1, 2, 3, 4 }; // The compiler creates an array of 4 elements.

}

{

int myArray[3]{ 2 };

size\_t arraySize{ std::size(myArray) };

}

}

//02\_std\_array.cpp

import <iostream>;

#if \_\_INTELLISENSE\_\_

#include <format>

#include <array>

#else

import <format>;

import <array>;

#endif

using namespace std;

int main()

{

array<int, 3> arr{ 9, 8, 7 };

// array arr{ 9, 8, 7 };  // Using CTAD类模板参数推导

cout << format("Array size = {}\n", arr.size());

cout << format("2nd element = {}\n", arr[1]);

}

//vector.cpp

import <vector>;

import <iostream>;

#if \_\_INTELLISENSE\_\_

#include <format>

#else

import <format>;

#endif

using namespace std;

int main()

{

// Create a vector of integers

vector<int> myVector{ 11, 22 };

// vector myVector { 11, 22 };  // Using CTAD

// Add some more integers to the vector using push\_back()

myVector.push\_back(33);

myVector.push\_back(44);

// Access elements

cout << format("1st element: {}\n", myVector[0]);

}

//pair.cpp

import <utility>;

import <iostream>;

#if \_\_INTELLISENSE\_\_

#include <format>

#else

import <format>;

#endif

using namespace std;

int main()

{

pair<double, int> myPair{ 1.23, 5 };

// pair myPair { 1.23, 5 };  // Using CTAD

cout << format("{} {}\n", myPair.first, myPair.second);

}

//optional.cpp

import <optional>;

import <iostream>;

using namespace std;

optional<int> getData(bool giveIt)

{

if (giveIt) {

return 42;

}

return nullopt;// or simply return {};

}

int main()

{

optional<int> data1{ getData(true) };

optional<int> data2{ getData(false) };

cout << "data1.has\_value = " << data1.has\_value() << endl;

if (data2) {

cout << "data2 has a value." << endl;

}

cout << "data1.value = " << data1.value() << endl;

cout << "data1.value = " << \*data1 << endl;

try {

cout << "data2.value = " << data2.value() << endl;

}

catch (const bad\_optional\_access& ex) {

cout << "Exception: " << ex.what() << endl;

}

cout << "data2.value = " << data2.value\_or(0) << endl;

//value\_or() can be used to return either the value of an optional

//or another value when the optional is empty

}

//StructuredBindings.cpp

import <utility>;

import <iostream>;

#if \_\_INTELLISENSE\_\_

#include <format>

#include <array>

#else

import <format>;

import <array>;

#endif

using namespace std;

int main()

{

{

// Structured bindings with std::array.

array values{ 11, 22, 33 };

auto [x, y, z] { values };

}

{

// Structured bindings with struct.

struct Point { double m\_x, m\_y, m\_z; };

Point point;

point.m\_x = 1.0; point.m\_y = 2.0; point.m\_z = 3.0;

auto [x, y, z] { point };

}

{

// Structured bindings with std::pair.

pair myPair{ "hello", 5 };

auto [theString, theInt] { myPair };// Decompose using structured bindings

cout << format("theString: {}\n", theString);

cout << format("theInt: {}\n", theInt);

}

}

//loops.cpp

import <iostream>;

#if \_\_INTELLISENSE\_\_

#include <format>

#include <array>

#else

import <format>;

import <array>;

#endif

using namespace std;

int main()

{

{

// The while loop

int i{ 0 };

while (i < 5) {

cout << "This is silly.\n";

++i;

}

}

cout << "\n";

{

// The do/while loop

int i{ 100 };

do {

cout << "This is silly.\n";

++i;

} while (i < 5);

}

cout << "\n";

{

// The for loop

for (int i{ 0 }; i < 5; ++i) {

cout << "This is silly.\n";

}

}

cout << "\n";

{

// The range-based for loop

array arr{ 1, 2, 3, 4 };

for (int i : arr) {

cout << format("{}\n", i);

}

}

{

// The range-based for loop with initializer (C++20)

for (array arr{ 1, 2, 3, 4 }; int i : arr)

{

cout << format("{}\n", i);

}

}

}

//InitializerLists.cpp

import <iostream>;

import <initializer\_list>;//初始化列表用于可变数量参数

using namespace std;

int makeSum(initializer\_list<int> values)

{

int total{ 0 };

for (int value : values) {

total += value;

}

return total;

}

int main()

{

int a{ makeSum({ 1, 2, 3 }) };

int b{ makeSum({ 10, 20, 30, 40, 50, 60 }) };

cout << a << endl;

cout << b << endl;

}

//string.cpp

import <iostream>;

#if \_\_INTELLISENSE\_\_

#include <format>

#include <string>

#else

import <format>;

import <string>;

#endif

using namespace std;

int main()

{

string myString{ "Hello, World" };

cout << format("The value of myString is {}\n", myString);

cout << format("The second letter is {}\n", myString[1]);

}

//AirlineTicket.cppm

export module airline\_ticket;

#if \_\_INTELLISENSE\_\_

#include <string>

#else

import <string>;

#endif

export class AirlineTicket

{

public:

double calculatePriceInDollars();

std::string getPassengerName();

void setPassengerName(std::string name);

int getNumberOfMiles();

void setNumberOfMiles(int miles);

bool getHasEliteSuperRewardsStatus();

void setHasEliteSuperRewardsStatus(bool status);

private:

std::string m\_passengerName{ "Unknown Passenger" };

int m\_numberOfMiles{ 0 };

bool m\_hasEliteSuperRewardsStatus{ false };

};

//AirlineTicket.cpp

module airline\_ticket;

using namespace std;

double AirlineTicket::calculatePriceInDollars()

{

if (getHasEliteSuperRewardsStatus()) {

// Elite Super Rewards customers fly for free!

return 0;

}

// The cost of the ticket is the number of miles times 0.1.

// Real airlines probably have a more complicated formula!

return getNumberOfMiles() \* 0.1;

}

string AirlineTicket::getPassengerName()

{

return m\_passengerName;

}

void AirlineTicket::setPassengerName(string name)

{

m\_passengerName = name;

}

int AirlineTicket::getNumberOfMiles()

{

return m\_numberOfMiles;

}

void AirlineTicket::setNumberOfMiles(int miles)

{

m\_numberOfMiles = miles;

}

bool AirlineTicket::getHasEliteSuperRewardsStatus()

{

return m\_hasEliteSuperRewardsStatus;

}

void AirlineTicket::setHasEliteSuperRewardsStatus(bool status)

{

m\_hasEliteSuperRewardsStatus = status;

}

//AirlineTicketTest.cpp

import <iostream>;

#if \_\_INTELLISENSE\_\_

#include <format>

#else

import <format>;

#endif

import airline\_ticket;

using namespace std;

int main()

{

AirlineTicket myTicket;// Stack-based AirlineTicket

myTicket.setPassengerName("Sherman T. Socketwrench");

myTicket.setNumberOfMiles(700);

double cost{ myTicket.calculatePriceInDollars() };

cout << format("This ticket will cost ${}\n", cost);

}

//scope.cpp

import <iostream>;

import <memory>;

using namespace std;

class Demo

{

public:

int get() { return 5; }

};

int get() { return 10; }

namespace NS

{

int get() { return 20; }

}

int main()

{

Demo d;

cout << d.get() << endl; // prints 5

cout << NS::get() << endl; // prints 20

cout << ::get() << endl; // prints 10

cout << get() << endl; // prints 10

}

//01\_UniformInitialization.cpp

import <vector>;

#include <string>

using namespace std;

struct CircleStruct//定义圆结构

{

int x, y;

double radius;

};

class CircleClass//定义圆类

{

public:

CircleClass(int x, int y, double radius)

: m\_x{ x }, m\_y{ y }, m\_radius{ radius } {}

private:

int m\_x, m\_y;

double m\_radius;

};

void func(int i) { /\* ... \*/ }

class MyClass

{

public:

MyClass() : m\_array{0, 1, 2, 3} {}

private:

int m\_array[4];

};

struct Employee {

char firstInitial;

char lastInitial;

int employeeNumber;

int salary;

};

int main()

{

// Old pre-C++11 way

CircleStruct myCircle1 = {10, 10, 2.5};

CircleClass myCircle2(10, 10, 2.5);

// C++11 uniform initialization

CircleStruct myCircle3 = {10, 10, 2.5};

CircleClass myCircle4 = {10, 10, 2.5};

CircleStruct myCircle5{10, 10, 2.5};

CircleClass myCircle6{10, 10, 2.5};

// Initializing a struct

Employee anEmployee;

anEmployee.firstInitial = 'J';

anEmployee.lastInitial = 'D';

anEmployee.employeeNumber = 42;

anEmployee.salary = 80'000;

// Initializing a struct with uniform initialization

Employee anEmployee2{ 'J', 'D', 42, 80'000 };

int a = 3;

int b(3);

int c = { 3 }; // Uniform initialization

int d{ 3 }; // Uniform initialization

int e{}; // Uniform initialization, e will be 0

// Narrowing

int x = 3.14;

func(3.14);

// Preventing narrowing

//int x{3.14}; // Error because narrowing

//func({3.14}); // Error because narrowing

// Uniform initialization can also be used with dynamically allocated arrays

int\* myArray = new int[]{0, 1, 2, 3};

// int\* myArray = new int[4]{0, 1, 2, 3}; // Pre-C++20.

delete[] myArray;

myArray = nullptr;

// Create a MyClass object

MyClass myClass;

// Uniform initialization also works with vectors

vector<string> myVec{"String 1", "String 2", "String 3"};

}

//02\_DesignatedInitializers.cpp

struct Employee {

char firstInitial;

char lastInitial;

int employeeNumber;

int salary{ 75'000 };

};

int main()

{

// Initialize all data members

Employee anEmployee1{

.firstInitial = 'J',

.lastInitial = 'D',

.employeeNumber = 42,

.salary = 80'000

};

// Initialize all data members, except employeeNumber

Employee anEmployee2{

.firstInitial = 'J',

.lastInitial = 'D',

.salary = 80'000

};

// Initialize all data members, except employeeNumber, and salary

Employee anEmployee3{

.firstInitial = 'J',

.lastInitial = 'D'

};

}

//01\_pointers.cpp

import <iostream>;

using namespace std;

struct Employee {

char firstInitial;

char lastInitial;

int employeeNumber;

int salary{ 75'000 };

};

Employee\* getEmployee()

{

return new Employee{ 'J', 'D', 42, 80'000 };

}

int main()

{

{

int\* myIntegerPointer{ nullptr };

if (!myIntegerPointer) { /\* myIntegerPointer is a null pointer \*/ }

myIntegerPointer = new int;

\*myIntegerPointer = 8;

delete myIntegerPointer;

myIntegerPointer = nullptr;

}

{

int i{ 8 };

int\* myIntegerPointer{ &i }; // Points to the variable with the value 8

}

{

Employee\* anEmployee{ getEmployee() };

cout << (\*anEmployee).salary << endl;

cout << anEmployee->salary << endl;

bool isValidSalary1{ (anEmployee && anEmployee->salary > 0) };

bool isValidSalary2{ (anEmployee != nullptr && anEmployee->salary > 0) };

delete anEmployee;

}

}

//02\_arrays.cpp

int main()

{

int arraySize{ 8 };

int\* myVariableSizedArray{ new int[arraySize] };

myVariableSizedArray[3] = 2;

delete[] myVariableSizedArray;

myVariableSizedArray = nullptr;

}