### 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;

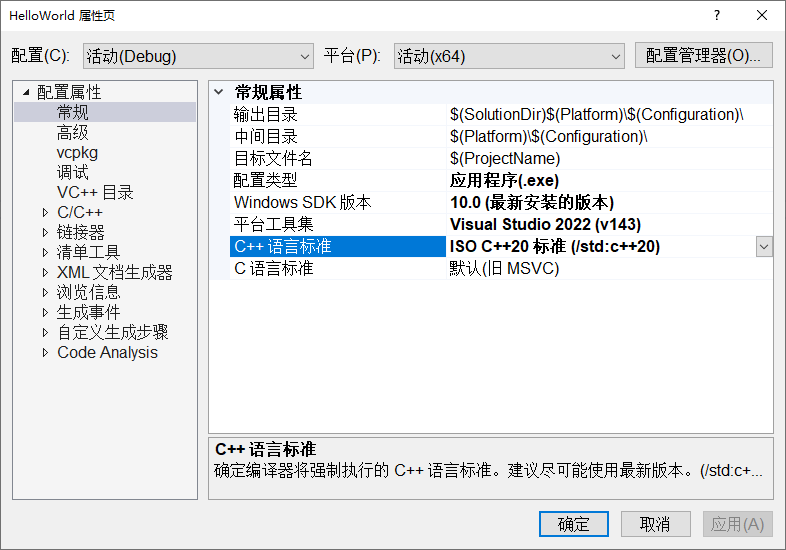
return 0;

}

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

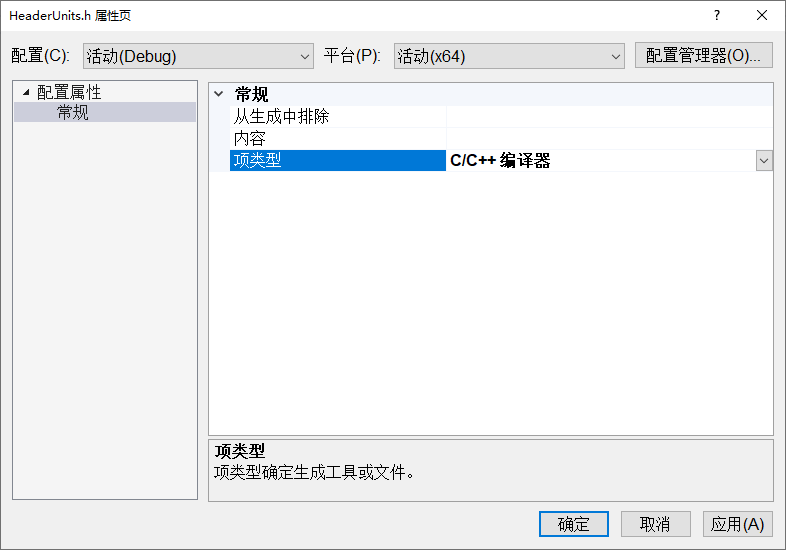
import <iostream>;

//..

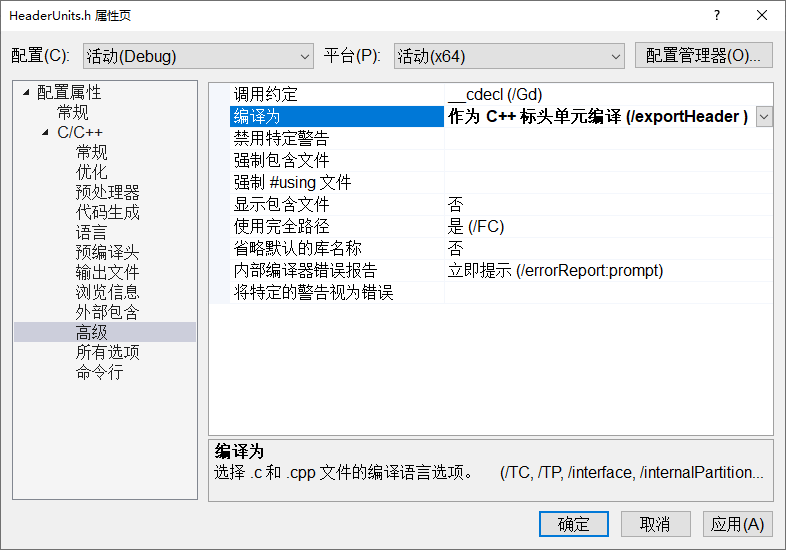
1. 右击HeaderUnists.h文件，选择属性



1. 在项类型中选择C/C++编译器，选择完了不要忘记点“应用”



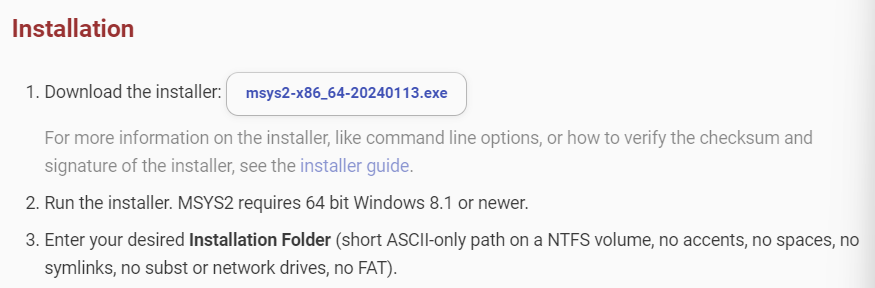
1. 选择“C/C++高级-编译为”“作为C++标头单元编译export/Header”，点击“应用”



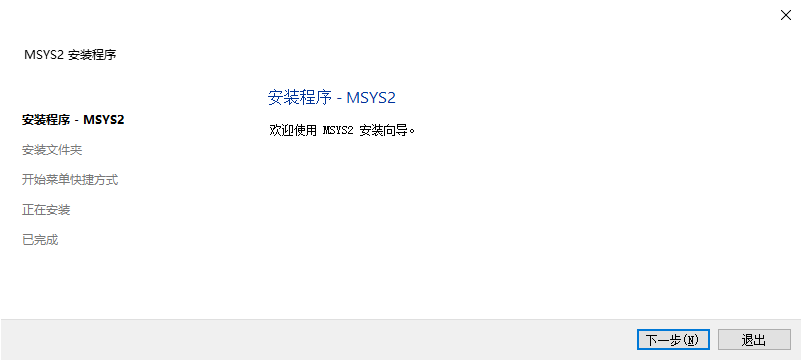
1. 点击“确定”，然后就可以编译运行了。

### 二、VSCode支持c++20配置

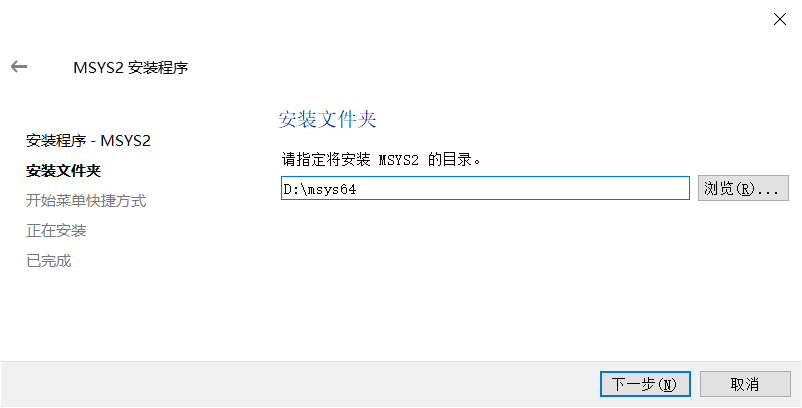
1. 去https://www.msys2.org/网站下载mingw-w64



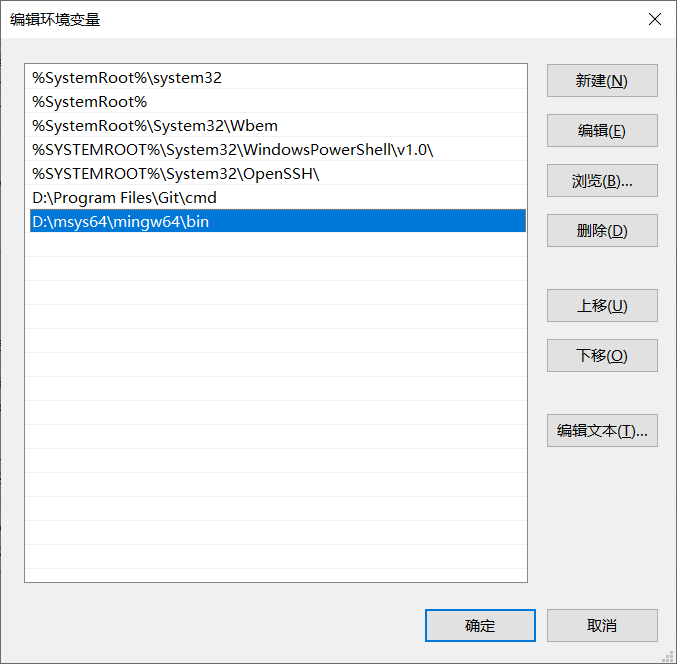
1. 下载msys2-x86\_64-20240113.exe
2. 执行msys2-x86\_64-20240113.exe开始安装



1. 按照提示进行安装即可



1. 配置系统环境变量



1. 点击Windows的开始，执行msys2 mingw64(蓝色)



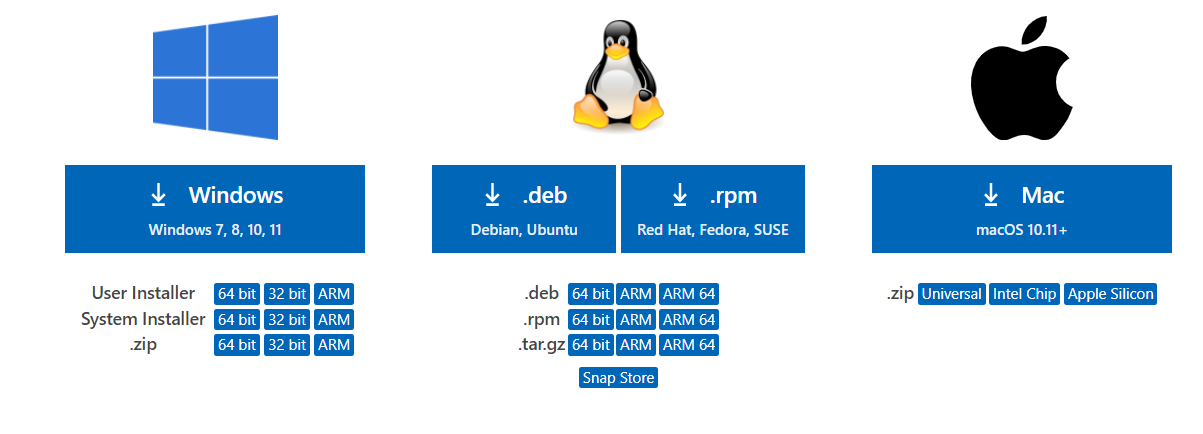
1. 在终端模式下执行如下三条命令：

pacman -S mingw-w64-x86\_64-gcc --disable-download-timeout

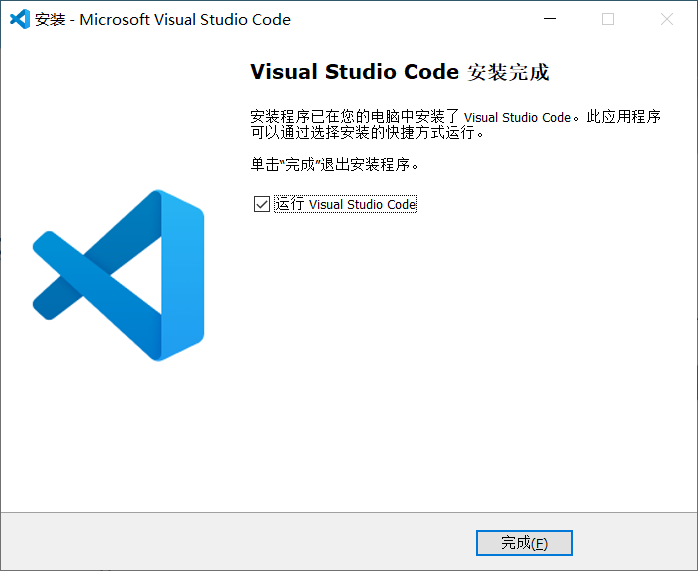
pacman -S mingw-w64-x86\_64-make --disable-download-timeout

pacman -S mingw-w64-x86\_64-gdb --disable-download-timeout

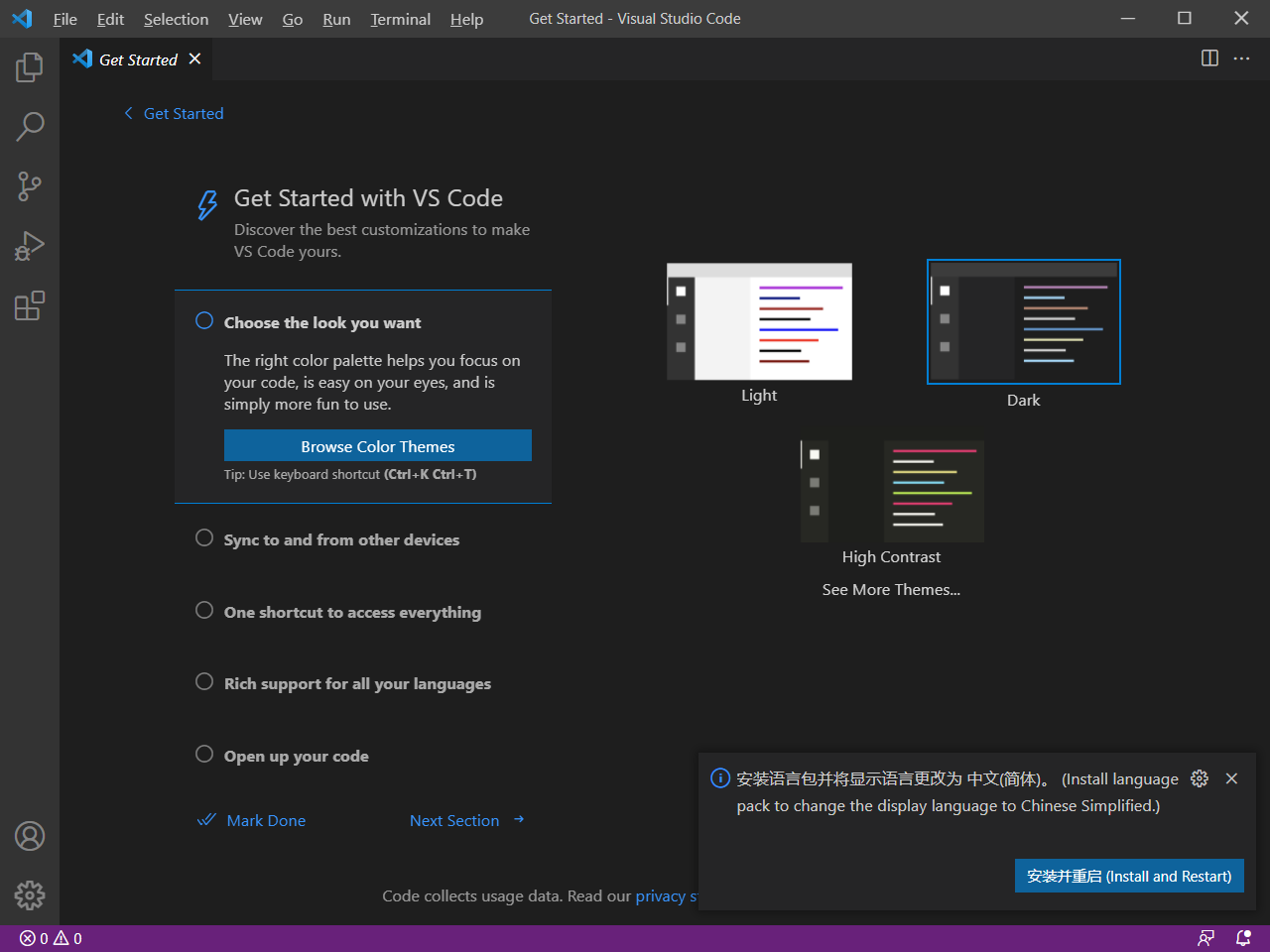
1. mingw-w64安装完毕。可以支持C++20。可以打开D:\msys64\mingw64\bin查看其中的内容，如果有，表示安装成功
2. 安装Visual Studio Code，简称VSCode，打开VSCode主页https://code.visualstudio.com/
3. 点击右上角“Download”下载系统版VSCode



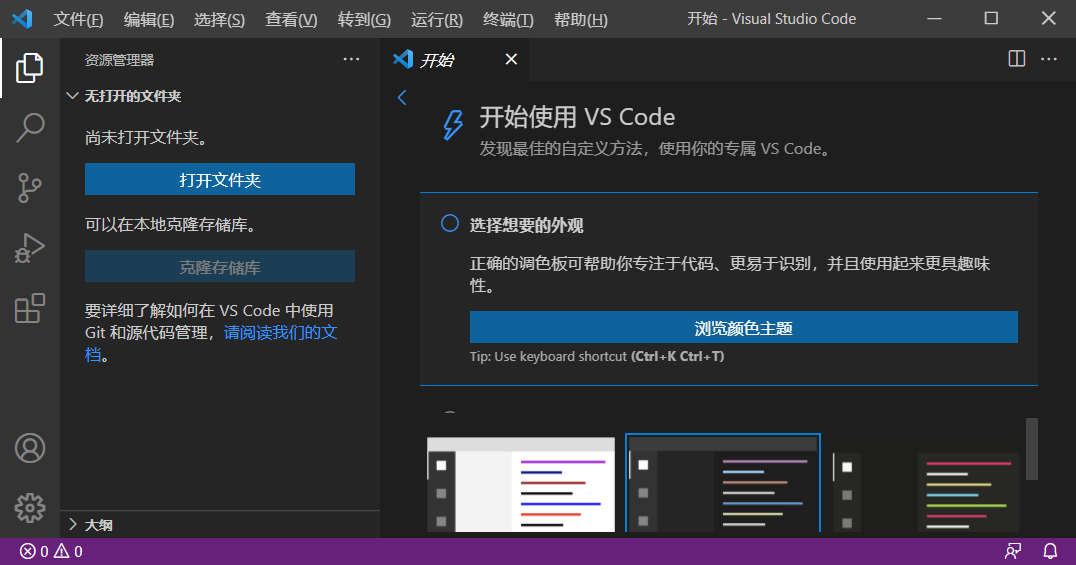
1. 选择“System Installer” - “64bit”，下载开始，完成后可以在“下载”目录中看到
2. VSCodeSetup-x64-1.82.1.exe，双击执行该文件
3. 安装完成后，点击“完成”



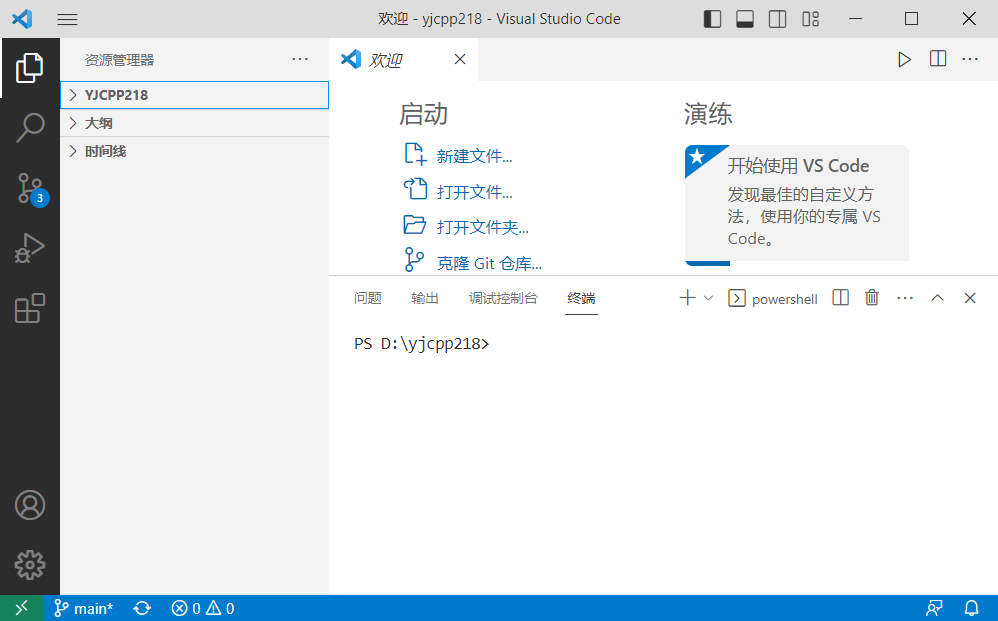
1. 启动VSCode



1. 在中文Windows环境下，通常会出现自动提示配置中文环境，点击右下角“安装并重启”会自动配置中文环境



1. 打开之前创建的文件夹“d:\vscpp”，作为自己常用的编程工作目录
2. 点击左下角的“管理” - “颜色主题”，可以将颜色主题修改为浅色



1. 在打开的工作文件夹vscpp的旁边，点击“新建文件”按钮，在文件名称vscpp旁边，创建一个helloWorld.cpp文件
2. 此时右下角会出现安装c/c++扩展提示，点击安装即可



1. 输入代码

// helloWorld.cpp

#include<iostream>

using namespace std;

int main()

{

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

return 0;

}

注意：一定要点击保存，最好选择“自动保存”

1. 系统会创建一个hello.cpp文本文件，右下角出现一个提示，提示你是否安装c/c++扩展，该扩展插件支持c/c++编辑器的操作



1. 点击“安装”



1. 回到程序编辑器，配置c/c++编译器
2. 按快捷键Ctrl+Shift+P调出命令面板，输入C/C++，选择“Edit Configurations(UI)”进入配置



1. 在配置界面中选择合适的配置选项:

D:\msys64\mingw64\bin\g++.exe

编译器参数

-std=c++23

-Mmodules

windows-gcc-x64

包含路径

${workspaceFolder}/\*\*

${vcpkgRoot}/\*\*

c17

c++23

1. 此时会在vscpp目录中生成一个.vscode文件夹，里面有一个c\_cpp\_properties.json文件，这个文件主要用来指示智能感知系统，进行搜索，避免提示错误。需要说明的是，这里vcpkg并不是必须的，可以去掉，如果已经安装了vcpkg，这行则有效。以下配置均涉及可选的vcpkg
2. 创建一个tasks.json文件来告诉VSCode如何构建（编译）程序。该任务将调用g++编译器基于源代码创建可执行文件。
3. 按快捷键Ctrl+Shift+P调出命令面板，输入tasks，选择“Tasks:Configure Default Build Task”



1. 选择“C/C++: g++.exe生成活动文件”



1. 然后会生成一个tasks.json文件

在指定的位置添加四行：

"-std=c++23",

"-Mmodules",

"-I",

"${vcpkgRoot}//\*\*"



1. 按Ctrl+F5，执行调试，选择C++(GDB/LLDB)





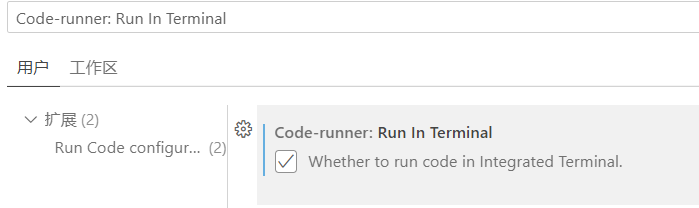
1. 程序运行后在集成终端显示：

PS D:\vscpp> & 'c:\Users\yinji\.vscode\extensions\ms-vscode.cpptools-1.14.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-2vgnblpj.ou0' '--stdout=Microsoft-MIEngine-Out-1yrt3zky.a2h' '--stderr=Microsoft-MIEngine-Error-0wnwjq3d.wby' '--pid=Microsoft-MIEngine-Pid-0sgly0gu.bwh' '--dbgExe=D:\msys64\mingw64\bin\gdb.exe' '--interpreter=mi'

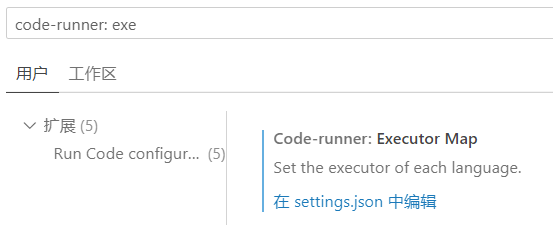
Hello, World!

PS D:\vscpp>

1. Ctrl+F5是调试运行程序，以后按F5可直接运行程序
2. Code Runner是一个很好的工具，方便调试运行。
3. 在左边管理框中点击扩展按钮，在搜索框中输入Code Runner
4. 安装Code Runner插件
5. 在左边管理框中点击设置，在搜索框中输入Code-runner: Run In Terminal
6. 勾选Whether to run code in Integrated Terminal
7. 此后便可在集成环境的终端环境下调试运行程序



1. 打开设置，查找Code-runner: Executor Map



1. 在settings.json中编辑

找到有cpp那一行，修改需如下：

"cpp": "cd $dir && g++ $fileName -std=c++23 -Mmodules -I \"D:\\vcpkg\\installed\\x64-windows\\include\" -o $fileNameWithoutExt && $dir$fileNameWithoutExt",

1. 输入代码，测试

// test.cpp

#include<iostream>

#include<format>

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);

}

### 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;

}

#include <cstdlib>

import <string>;

const int versionNumberMajor{ 2 };

const int versionNumberMinor{ 1 };

const std::string productName{ "Super Hyper Net Modulator" };

const double PI{ 3.141592653589793238462 };

void func(const int /\*param\*/)

{

// Not allowed to change param...

}

void constIntOne()

{

const int\* ip;//不能修改指针所指的单元

ip = new int[10];

// ip[4] = 5; // DOES NOT COMPILE!

}

void constIntTwo()

{

int const\* ip;//不能修改指针所指的单元

ip = new int[10];

// ip[4] = 5; // DOES NOT COMPILE!

}

void constPtrOne()

{

int\* const ip{ nullptr };//不能修改指针

// ip = new int[10]; // DOES NOT COMPILE!

ip[4] = 5; // Error: dereferencing a null pointer

}

void constPtrTwo()

{

int\* const ip{ new int[10] };//不能修改指针

ip[4] = 5;

}

void constIntPtrOne()

{

int const\* const ip{ nullptr };//既不能修改指针，也不能修改指针所指的单元

}

void constIntPtrTwo()

{

const int\* const ip{ nullptr };//既不能修改指针，也不能修改指针所指的单元

}

void manyLevelConst()

{

const int\* const\* const\* const ip{ nullptr };//三级指针，既不能修改指针，也不能修改指针所指的单元

}

int main()

{

int\* ip;

ip = new int[10];

ip[4] = 5;

}

//ArilineTicket.cppm

export module airline\_ticket;

import <string>;

export class AirlineTicket

{

public:

double calculatePriceInDollars() const;

std::string getPassengerName() const;

void setPassengerName(std::string name);

int getNumberOfMiles() const;

void setNumberOfMiles(int miles);

bool getHasEliteSuperRewardsStatus() const;

void setHasEliteSuperRewardsStatus(bool status);

private:

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

int m\_numberOfMiles{ 0 };

bool m\_hasEliteSuperRewardsStatus{ false };

};

//ArilineTicket.cpp

module airline\_ticket;

using namespace std;

double AirlineTicket::calculatePriceInDollars() const

{

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() const

{

return m\_passengerName;

}

void AirlineTicket::setPassengerName(string name)

{

m\_passengerName = name;

}

int AirlineTicket::getNumberOfMiles() const

{

return m\_numberOfMiles;

}

void AirlineTicket::setNumberOfMiles(int miles)

{

m\_numberOfMiles = miles;

}

bool AirlineTicket::getHasEliteSuperRewardsStatus() const

{

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 ${}", cost) << endl;

}

//01\_constexpr.cpp

constexpr int getArraySize()

{

return 32;

}

int main()

{

int myArray[getArraySize()]; // OK

myArray[0] = 1;

}

//02\_constexprClasses.cpp

#include <cstddef>

import <iostream>;

import <array>;

using namespace std;

class Rect

{

public:

constexpr Rect(size\_t width, size\_t height)

: m\_width{ width }, m\_height{ height }

{

}

constexpr size\_t getArea() const//既可以是编译期函数，也可以是运行期函数

{

return m\_width \* m\_height;

}

private:

size\_t m\_width { 0 }, m\_height { 0 };

};

int main()

{

constexpr Rect r{ 8, 2 };//编译期常量

int myArray[r.getArea()];

cout << std::size(myArray) << endl;

}

//consteval.cpp

consteval double inchToMm(double inch)//编译期常量

{

return inch \* 25.4;

}

int main()

{

constexpr double const\_inch{ 6.0 };

constexpr double mm1{ inchToMm(const\_inch) }; // at compile time

double dynamic\_inch{ 8.0 };

//double mm2{ inchToMm(dynamic\_inch) }; // compile-time error

}

//01\_ReferenceVariables.cpp

import <string>;

import <utility>;

using namespace std;

string getString()

{

return "Hello world!";

}

int main()

{

int x{ 3 }, y{ 4 }, z{ 5 };

int& xRef{ x };

xRef = 10;

// int& emptyRef; // DOES NOT COMPILE!

xRef = y; // changes value of x to 4. Doesn't make xRef refer to y.

int& zRef{ z };

zRef = xRef; // Assigns values, not references

const int& zRef2{ z };

// zRef2 = 4; // DOES NOT COMPILE

// int& unnamedRef1{ 5 }; // DOES NOT COMPILE

const int& unnamedRef2{ 5 }; // Works as expected

//不能创建一个指向常量的引用，但常引用可以

// string& string1{ getString() }; // DOES NOT COMPILE

const string& string2{ getString() }; // Works as expected

int\* intP{ nullptr };

int\*& ptrRef{ intP };

ptrRef = new int;

\*ptrRef = 5;

int\* xPtr{ &xRef }; // address of a reference is pointer to value

\*xPtr = 100;

pair myPair{ "hello", 5 };

const auto& [theString, theInt] { myPair }; // Decompose into references-to-const

}

//02\_ReferenceDataMembrs.cpp

class MyClass

{

public:

MyClass(int& ref) : m\_ref{ ref } { /\* Body of constructor \*/ }

private:

int& m\_ref;

};

int main()

{

int i{ 123 };

MyClass m{ i };

}

//03\_ReferenceParameters.cpp

void addOneA(int i)

{

i++; // Has no real effect because this is a copy of the original

}

void addOneB(int& i)

{

i++; // Actually changes the original variable

}

void swap(int& first, int& second)

{

int temp{ first };

first = second;

second = temp;

}

int main()

{

int myInt{ 7 };

addOneA(myInt);

addOneB(myInt);

int x{ 5 }, y{ 6 };

swap(x, y);

// swap(3, 4); // DOES NOT COMPILE

int\* xp{ &x }, \* yp{ &y };

swap(\*xp, \*yp);

}

//04\_ConstReferenceParameters.cpp

import <string>;

import <iostream>;

using namespace std;

void printString(const string& myString)

{

cout << myString << endl;

}

int main()

{

string someString{ "Hello World" };

printString(someString);

printString("Hello World"); // Passing literals works

}

//05\_OddsEvensPtrs.cpp

#include <cstddef>

import <array>;

import <iostream>;

using namespace std;

void printIntArr(const int arr[], size\_t size)

{

for (size\_t i{ 0 }; i < size; ++i)

{

cout << arr[i] << " ";

}

cout << endl;

}

void separateOddsAndEvens(const int arr[], size\_t size, int\*\* odds,

size\_t\* numOdds, int\*\* evens, size\_t\* numEvens)

{

// Count the number of odds and evens

\*numOdds = \*numEvens = 0;

for (size\_t i = 0; i < size; ++i) {

if (arr[i] % 2 == 1) {

++(\*numOdds);

} else {

++(\*numEvens);

}

}

// Allocate two new arrays of the appropriate size.

\*odds = new int[\*numOdds];

\*evens = new int[\*numEvens];

// Copy the odds and evens to the new arrays

size\_t oddsPos = 0, evensPos = 0;

for (size\_t i = 0 ; i < size; ++i) {

if (arr[i] % 2 == 1) {

(\*odds)[oddsPos++] = arr[i];

} else {

(\*evens)[evensPos++] = arr[i];

}

}

}

int main()

{

int unSplit[]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int\* oddNums{ nullptr };

int\* evenNums{ nullptr };

size\_t numOdds{ 0 }, numEvens{ 0 };

separateOddsAndEvens(unSplit, std::size(unSplit), &oddNums, &numOdds, &evenNums, &numEvens);

printIntArr(oddNums, numOdds);

printIntArr(evenNums, numEvens);

delete[] oddNums; oddNums = nullptr;

delete[] evenNums; evenNums = nullptr;

}

#include <cstddef>

import <array>;

import <iostream>;

using namespace std;

void printIntArr(const int arr[], size\_t size)

{

for (size\_t i{ 0 }; i < size; ++i) {

cout << arr[i] << " ";

}

cout << endl;

}

void separateOddsAndEvens(const int arr[], size\_t size, int\*& odds,

size\_t& numOdds, int\*& evens, size\_t& numEvens)

{

numOdds = numEvens = 0;

for (size\_t i{ 0 }; i < size; ++i) {

if (arr[i] % 2 == 1) {

++numOdds;

} else {

++numEvens;

}

}

odds = new int[numOdds];

evens = new int[numEvens];

size\_t oddsPos{ 0 }, evensPos{ 0 };

for (size\_t i{ 0 }; i < size; ++i) {

if (arr[i] % 2 == 1) {

odds[oddsPos++] = arr[i];

} else {

evens[evensPos++] = arr[i];

}

}

}

int main()

{

int unSplit[]{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

int\* oddNums{ nullptr };

int\* evenNums{ nullptr };

size\_t numOdds{ 0 }, numEvens{ 0 };

separateOddsAndEvens(unSplit, std::size(unSplit), oddNums, numOdds, evenNums, numEvens);

printIntArr(oddNums, numOdds);

printIntArr(evenNums, numEvens);

delete[] oddNums; oddNums = nullptr;

delete[] evenNums; evenNums = nullptr;

}

import <iostream>;

import <vector>;

using namespace std;

void printVec(const vector<int>& arr)

{

for (int i : arr) {

cout << i << " ";

}

cout << endl;

}

void separateOddsAndEvens(const vector<int>& arr,

vector<int>& odds, vector<int>& evens)

{

for (int i : arr) {

if (i % 2 == 1) {

odds.push\_back(i);

} else {

evens.push\_back(i);

}

}

}

int main()

{

vector<int> vecUnSplit{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

vector<int> odds, evens;

separateOddsAndEvens(vecUnSplit, odds, evens);

printVec(odds);

printVec(evens);

}

import <iostream>;

import <vector>;

using namespace std;

void printVec(const vector<int>& arr)

{

for (int i : arr) {

cout << i << " ";

}

cout << endl;

}

struct OddsAndEvens { vector<int> odds, evens; };

OddsAndEvens separateOddsAndEvens(const vector<int>& arr)

{

vector<int> odds, evens;

for (int i : arr) {

if (i % 2 == 1) {

odds.push\_back(i);

} else {

evens.push\_back(i);

}

}

return OddsAndEvens{ .odds = odds, .evens = evens };

}

int main()

{

vector<int> vecUnSplit{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

auto oddsAndEvens{ separateOddsAndEvens(vecUnSplit) };

printVec(oddsAndEvens.odds);

printVec(oddsAndEvens.evens);

}

import <iostream>;

import <stdexcept>;

import <format>;

using namespace std;

double divideNumbers(double numerator, double denominator)

{

if (denominator == 0) {

throw invalid\_argument{ "Denominator cannot be 0." };

}

return numerator / denominator;

}

int main()

{

try {

cout << divideNumbers(2.5, 0.5) << endl;

cout << divideNumbers(2.3, 0) << endl;

cout << divideNumbers(4.5, 2.5) << endl;

} catch (const invalid\_argument& exception) {

cout << format("Exception caught: {}", exception.what()) << endl;

}

}

import <string>;

import <iostream>;

using namespace std;

const string message{ "Test" };

const string& foo() { return message; }

int main()

{

auto x{ 123 }; // x is of type int

auto f1{ foo() }; // f1 is of type string

const auto& f2{ foo() }; // f2 is of type const string&

string str{ "C++" };

auto result{ as\_const(str) }; // result is of type string!

int i{ 123 };

//auto p{ &i }; // p is of type int\*

auto\* p{ &i }; // p is of type int\*

const auto p1{ &i }; // p1 is of type int\* const

auto const p2{ &i }; // p2 is of type int\* const

const auto\* p3{ &i }; // p3 is of type const int\*

auto\* const p4{ &i }; // p4 is of type int\* const

const auto\* const p5{ &i }; // p5 is of type const int\* const

}

import <initializer\_list>;

int main()

{

// Copy list initialization

auto a = { 11 }; // initializer\_list<int>

auto b = { 11, 22 }; // initializer\_list<int>

// Direct list initialization

auto c{ 11 }; // int

//auto d{ 11, 22 }; // Error, too many elements.

}

//Employee.cppm

export module employee;

import <string>;

using namespace std;

namespace Records {

const int DefaultStartingSalary{ 30'000 };

export const int DefaultRaiseAndDemeritAmount{ 1'000 };

export class Employee

{

public:

Employee(const string& firstName, const string& lastName);

void promote(int raiseAmount = DefaultRaiseAndDemeritAmount);

void demote(int demeritAmount = DefaultRaiseAndDemeritAmount);

void hire(); // Hires or rehires the employee

void fire(); // Dismisses the employee

void display() const;// Outputs employee info to console

// Getters and setters

void setFirstName(const string& firstName);

const string& getFirstName() const;

void setLastName(const string& lastName);

const string& getLastName() const;

void setEmployeeNumber(int employeeNumber);

int getEmployeeNumber() const;

void setSalary(int newSalary);

int getSalary() const;

bool isHired() const;

private:

string m\_firstName;

string m\_lastName;

int m\_employeeNumber{ -1 };

int m\_salary{ DefaultStartingSalary };

bool m\_hired{ false };

};

}

//Employee.cpp

module employee;

import <iostream>;

import <format>;

using namespace std;

namespace Records {

Employee::Employee(const string& firstName, const string& lastName)

: m\_firstName{ firstName }

, m\_lastName{ lastName }

{

}

void Employee::promote(int raiseAmount)

{

setSalary(getSalary() + raiseAmount);

}

void Employee::demote(int demeritAmount)

{

setSalary(getSalary() - demeritAmount);

}

void Employee::hire()

{

m\_hired = true;

}

void Employee::fire()

{

m\_hired = false;

}

void Employee::display() const

{

cout << format("Employee: {}, {}", getLastName(), getFirstName()) << endl;

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

cout << (isHired() ? "Current Employee" : "Former Employee") << endl;

cout << format("Employee Number: {}", getEmployeeNumber()) << endl;

cout << format("Salary: ${}", getSalary()) << endl;

cout << endl;

}

// Getters and setters

void Employee::setFirstName(const string& firstName)

{

m\_firstName = firstName;

}

const string& Employee::getFirstName() const

{

return m\_firstName;

}

void Employee::setLastName(const string& lastName)

{

m\_lastName = lastName;

}

const string& Employee::getLastName() const

{

return m\_lastName;

}

void Employee::setEmployeeNumber(int employeeNumber)

{

m\_employeeNumber = employeeNumber;

}

int Employee::getEmployeeNumber() const

{

return m\_employeeNumber;

}

void Employee::setSalary(int salary)

{

m\_salary = salary;

}

int Employee::getSalary() const

{

return m\_salary;

}

bool Employee::isHired() const

{

return m\_hired;

}

}

export module database;

import <string>;

//Database.cppm

import <vector>;

import employee;

using namespace std;

namespace Records {

const int FirstEmployeeNumber{ 1'000 };

export class Database

{

public:

Employee& addEmployee(const string& firstName,

const string& lastName);

Employee& getEmployee(int employeeNumber);

Employee& getEmployee(const string& firstName,

const string& lastName);

void displayAll() const;

void displayCurrent() const;

void displayFormer() const;

private:

vector<Employee> m\_employees;

int m\_nextEmployeeNumber{ FirstEmployeeNumber };

};

}

//Database.cpp

module database;

import <stdexcept>;

using namespace std;

namespace Records {

Employee& Database::addEmployee(const string& firstName,

const string& lastName)

{

Employee theEmployee{ firstName, lastName };

theEmployee.setEmployeeNumber(m\_nextEmployeeNumber++);

theEmployee.hire();

m\_employees.push\_back(theEmployee);

return m\_employees.back();

}

Employee& Database::getEmployee(int employeeNumber)

{

for (auto& employee : m\_employees) {

if (employee.getEmployeeNumber() == employeeNumber) {

return employee;

}

}

throw logic\_error{ "No employee found." };

}

Employee& Database::getEmployee(const string& firstName, const string& lastName)

{

for (auto& employee : m\_employees) {

if (employee.getFirstName() == firstName &&

employee.getLastName() == lastName) {

return employee;

}

}

throw logic\_error{ "No employee found." };

}

void Database::displayAll() const

{

for (const auto& employee : m\_employees) {

employee.display();

}

}

void Database::displayCurrent() const

{

for (const auto& employee : m\_employees) {

if (employee.isHired()) {

employee.display();

}

}

}

void Database::displayFormer() const

{

for (const auto& employee : m\_employees) {

if (!employee.isHired()) {

employee.display();

}

}

}

}

//DatabaseTest.cpp

import <iostream>;

import database;

using namespace std;

using namespace Records;

int main()

{

Database myDB;

Employee& emp1{ myDB.addEmployee("Greg", "Wallis") };

emp1.fire();

Employee& emp2{ myDB.addEmployee("Marc", "Gregoire") };

emp2.setSalary(100'000);

Employee& emp3{ myDB.addEmployee("John", "Doe") };

emp3.setSalary(10'000);

emp3.promote();

cout << "all employees: " << endl << endl;

myDB.displayAll();

cout << endl << "current employees: " << endl << endl;

myDB.displayCurrent();

cout << endl << "former employees: " << endl << endl;

myDB.displayFormer();

}

//UserInterface.cpp

import <iostream>;

import <stdexcept>;

import <exception>;

import <format>;

import <string>;

import database;

import employee;

using namespace std;

using namespace Records;

int displayMenu();

void doHire(Database& db);

void doFire(Database& db);

void doPromote(Database& db);

int main()

{

Database employeeDB;

bool done{ false };

while (!done) {

int selection{ displayMenu() };

switch (selection) {

case 0:

done = true;

break;

case 1:

doHire(employeeDB);

break;

case 2:

doFire(employeeDB);

break;

case 3:

doPromote(employeeDB);

break;

case 4:

employeeDB.displayAll();

break;

case 5:

employeeDB.displayCurrent();

break;

case 6:

employeeDB.displayFormer();

break;

default:

cerr << "Unknown command." << endl;

break;

}

}

}

int displayMenu()

{

// Note:

// One important note is that this code assumes that the user will

// "play nice" and type a number when a number is requested.

// When you read about I/O in Chapter 13, you will learn how to

// protect against bad input.

int selection;

cout << endl;

cout << "Employee Database" << endl;

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

cout << "1) Hire a new employee" << endl;

cout << "2) Fire an employee" << endl;

cout << "3) Promote an employee" << endl;

cout << "4) List all employees" << endl;

cout << "5) List all current employees" << endl;

cout << "6) List all former employees" << endl;

cout << "0) Quit" << endl;

cout << endl;

cout << "---> ";

cin >> selection;

return selection;

}

void doHire(Database& db)

{

string firstName;

string lastName;

cout << "First name? ";

cin >> firstName;

cout << "Last name? ";

cin >> lastName;

auto& employee{ db.addEmployee(firstName, lastName) };

cout << format("Hired employee {} {} with employee number {}.",

firstName, lastName, employee.getEmployeeNumber()) << endl;

}

void doFire(Database& db)

{

int employeeNumber;

cout << "Employee number? ";

cin >> employeeNumber;

try {

auto& emp{ db.getEmployee(employeeNumber) };

emp.fire();

cout << format("Employee {} terminated.", employeeNumber) << endl;

} catch (const std::logic\_error& exception) {

cerr << format("Unable to terminate employee: {}", exception.what()) << endl;

}

}

void doPromote(Database& db)

{

int employeeNumber;

cout << "Employee number? ";

cin >> employeeNumber;

int raiseAmount;

cout << "How much of a raise? ";

cin >> raiseAmount;

try {

auto& emp{ db.getEmployee(employeeNumber) };

emp.promote(raiseAmount);

} catch (const std::logic\_error& exception) {

cerr << format("Unable to promote employee: {}", exception.what()) << endl;

}

}

### Chapter 2 Working with Strings and String Views

**//CppStrings.cpp**

import <string>;

import <iostream>;

import <compare>;

using namespace std;

int main()

{

{

string a{ "12" };

string b{ "34" };

string c;

c = a + b; // c is "1234"

cout << c << endl;

a += b; // a is "1234"

cout << a << endl;

}

{

string a{ "12" };

string b{ "34" };

auto result{ a.compare(b) };

if (result < 0) { cout << "less" << endl; }

if (result > 0) { cout << "greater" << endl; }

if (result == 0) { cout << "equal" << endl; }

}

{

// C++20 three-way comparison operator

string a{ "12" };

string b{ "34" };

auto result{ a <=> b };

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

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

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

}

{

string myString{ "hello" };

myString += ", there";

string myOtherString{ myString };

if (myString == myOtherString) {

myOtherString[0] = 'H';

}

cout << myString << endl;

cout << myOtherString << endl;

}

{

string strHello{ "Hello!!" };

string strWorld{ "The World..." };

auto position{ strHello.find("!!") };

if (position != string::npos) {

// Found the "!!" substring, now replace it.

strHello.replace(position, 2, strWorld.substr(3, 6));

}

cout << strHello << endl;

}

{

auto string1{ "Hello World" }; // string1 is a const char\*.

auto string2{ "Hello World"s }; // string2 is an std::string.

}

}

**//stoi.cpp**

import <iostream>;

import <string>;

import <format>;

using namespace std;

int main()

{

const string toParse{ " 123USD" };

size\_t index{ 0 };

int value{ stoi(toParse, &index) };

cout << format("Parsed value: {}", value) << endl;

cout << format("First non-parsed character: '{}'", toParse[index]) << endl;

}

**//ToFormChars.cpp**

#include <cstddef>

#include <charconv>

#include <iostream>

using namespace std;

int main()

{

    const size\_t BufferSize{ 50 };

    {

        // to\_chars() without structured binding.

        string out(BufferSize, ' '); // A string of BufferSize space characters.

        auto result{ to\_chars(out.data(), out.data() + out.size(), 12345) };

        if (result.ec == errc{})

        {

            cout << out << endl; /\* Conversion successful. \*/

        }

    }

    {

        // to\_chars() with structured binding.

        string out(BufferSize, ' '); // A string of BufferSize space characters.

        auto [ptr, error] { to\_chars(out.data(), out.data() + out.size(), 12345) };

        if (error == errc{})

        {

            cout << out << endl; /\* Conversion successful. \*/

        }

    }

    {

        // Demonstrating perfect round-tripping.

        double value1{ 0.314 };

        string out(BufferSize, ' '); // A string of BufferSize space characters.

        auto [ptr1, error1] { to\_chars(out.data(), out.data() + out.size(), value1) };

        if (error1 == errc{})

        {

            cout << out << endl; /\* Conversion successful. \*/

        }

        double value2;

        auto [ptr2, error2] { from\_chars(out.data(), out.data() + out.size(), value2) };

        if (error2 == errc{})

        {

            if (value1 == value2)

            {

                cout << "Perfect roundtrip" << endl;

            }

            else

            {

                cout << "No perfect roundtrip?!?" << endl;

            }

        }

    }

}

**//StringViews.cpp**

#include <cstddef>

#include <iostream>

#include <string>

#include <string\_view>

#include <format>

using namespace std;

string\_view extractExtension(string\_view filename)

{

    return filename.substr(filename.rfind('.'));

}

void handleExtension(const string& extension) { /\* ... \*/ }

int main()

{

    // C++ std::string.

    string filename{ R"(c:\temp\my file.ext)" };

    cout << format("C++ string: {}", extractExtension(filename)) << endl;

    // C-style string.

    const char\* cString{ R"(c:\temp\my file.ext)" };

    cout << format("C string: {}", extractExtension(cString)) << endl;

    // String literal.

    cout << format("Literal: {}", extractExtension(R"(c:\temp\my file.ext)")) << endl;

    // Raw string buffer with given length.

    const char\* raw{ "test.ext" };

    size\_t length{ 8 };

    cout << format("Raw: {}", extractExtension({ raw, length })) << endl;

    cout << format("Raw: {}", extractExtension(string\_view{ raw, length })) << endl;

    // Assigning the result to a std::string.

    string extension{ extractExtension(filename).data() };

    handleExtension(extractExtension("my file.ext").data());    // data() method

    handleExtension(string{ extractExtension("my file.ext") }); // explicit ctor

    // Concatenation with string\_view.

    string str{ "Hello" };

    string\_view sv{ " world" };

    //auto result{ str + sv };  // Error, does not compile.

    auto result1{ str + sv.data() };

    string result2{ str };

    result2.append(sv.data(), sv.size());

}

**//01\_basic.cpp**

#include <iostream>

#include <string>

#include <format>

using namespace std;

int main()

{

    int n{ 42 };

    auto s1{ format("Read {} bytes from {}", n, "file1.txt") };

    auto s2{ format("Read {0} bytes from {1}", n, "file1.txt") };

    auto s3{ format(L"从{1}中读取{0}个字节。", n, L"file1.txt") };

}

**//02\_examples.cpp**

#include <iostream>

#include <string>

#include <format>

using namespace std;

int main()

{

    {

        // width

        int i{ 42 };

        cout << format("|{:5}|", i) << endl;      // |   42|

        cout << format("|{:{}}|", i, 7) << endl;  // |     42|

    }

    cout << endl;

    {

        // [fill]align

        int i{ 42 };

        cout << format("|{:7}|", i) << endl;    // |     42|

        cout << format("|{:<7}|", i) << endl;   // |42     |

        cout << format("|{:\_>7}|", i) << endl;  // |\_\_\_\_\_42|

        cout << format("|{:\_^7}|", i) << endl;  // |\_\_42\_\_\_|

    }

    cout << endl;

    {

        // sign

        int i{ 42 };

        cout << format("|{:<5}|", i) << endl;   // |42   |

        cout << format("|{:<+5}|", i) << endl;  // |+42  |

        cout << format("|{:< 5}|", i) << endl;  // | 42  |

        cout << format("|{:< 5}|", -i) << endl; // |-42  |

    }

    cout << endl;

    {

        // Integral types

        int i{ 42 };

        cout << format("|{:10d}|", i) << endl;   // |        42|

        cout << format("|{:10b}|", i) << endl;   // |    101010|

        cout << format("|{:#10b}|", i) << endl;  // |  0b101010|

        cout << format("|{:10X}|", i) << endl;   // |        2A|

        cout << format("|{:#10X}|", i) << endl;  // |      0X2A|

    }

    cout << endl;

    {

        // String types

        string s{ "ProCpp" };

        cout << format("|{:\_^10}|", s) << endl; // |\_\_ProCpp\_\_|

    }

    cout << endl;

    {

        // Floating-point types

        double d{ 3.1415 / 2.3 };

        cout << format("|{:12g}|", d) << endl;                         // |    1.365870|

        cout << format("|{:12.2}|", d) << endl;                        // |        1.37|

        cout << format("|{:12e}|", d) << endl;                         // |1.365870e+00|

        int width{ 12 };

        int precision{ 3 };

        cout << format("|{2:{0}.{1}f}|", width, precision, d) << endl; // |       1.366|

    }

    cout << endl;

    {

        // 0

        int i{ 42 };

        cout << format("|{:06d}|", i) << endl;   // |000042|

        cout << format("|{:+06d}|", i) << endl;  // |+00042|

        cout << format("|{:06X}|", i) << endl;   // |00002A|

        cout << format("|{:#06X}|", i) << endl;  // |0X002A|

    }

}

**//03\_exceptions.cpp**

#include <iostream>

#include <format>

using namespace std;

int main()

{

    try

    {

        cout << format("An integer: {:.}", 5);

    }

    catch (const format\_error& caught\_exception)

    {

        cout << caught\_exception.what(); // "missing precision specifier"

    }

}

//04\_custom\_types.cpp

#include <iostream>

#include <string>

#include <string\_view>

#include <format>

using namespace std;

class KeyValue

{

public:

    KeyValue(string\_view key, int value) : m\_key{ key }, m\_value{ value } {}

    const string& getKey() const { return m\_key; }

    int getValue() const { return m\_value; }

private:

    string m\_key;

    int m\_value;

};

template<>

class formatter<KeyValue>

{

public:

    constexpr auto parse(auto& context)

    {

        auto iter{ context.begin() };

        const auto end{ context.end() };

        if (iter == end || \*iter == '}') {  // {} format specifier

            m\_outputType = OutputType::KeyAndValue;

            return iter;

        }

        switch (\*iter) {

            case 'a':  // {:a} format specifier

                m\_outputType = OutputType::KeyOnly;

                break;

            case 'b':  // {:b} format specifier

                m\_outputType = OutputType::ValueOnly;

                break;

            case 'c':  // {:c} format specifier

                m\_outputType = OutputType::KeyAndValue;

                break;

            default:

                throw format\_error{ "Invalid KeyValue format specifier." };

        }

        ++iter;

        if (iter != end && \*iter != '}') {

            throw format\_error{ "Invalid KeyValue format specifier." };

        }

        return iter;

    }

    auto format(const KeyValue& kv, auto& context)

    {

        switch (m\_outputType) {

            using enum OutputType;

            case KeyOnly:

                return format\_to(context.out(), "{}", kv.getKey());

            case ValueOnly:

                return format\_to(context.out(), "{}", kv.getValue());

            default:

                return format\_to(context.out(), "{} - {}", kv.getKey(), kv.getValue());

        }

    }

private:

    enum class OutputType

    {

        KeyOnly,

        ValueOnly,

        KeyAndValue

    };

    OutputType m\_outputType{ OutputType::KeyAndValue };

};

int main()

{

    KeyValue keyValue{ "Key1", 11 };

    cout << format("{}", keyValue) << endl;

    cout << format("{:a}", keyValue) << endl;

    cout << format("{:b}", keyValue) << endl;

    cout << format("{:c}", keyValue) << endl;

    try {

        cout << format("{:cd}", keyValue) << endl;

    } catch (const format\_error& caught\_exception) {

        cout << caught\_exception.what() << endl;

    }

}