**基本类的定义**

类可以使用struct或class标识

struct：默认成员访问权限为public

class：默认成员访问权限为private

类一般使用头文件和源文件分离分离的方式，如下示例类的一般定义

SalesData.h文件

#include <string>

#include <vector>

#ifndef SalesData\_H

#define SalesData\_H

// SalesData声明

struct SalesData{

// 友元函数声明，不属于类的成员，但可以访问类实例的私有成员

friend SalesData add(const SalesData &, const SalesData &);

public:

SalesData() = default;

// 有参构造函数，\_money(money)为为属性\_money赋值

SalesData(const int money) : \_money(money){ };

int GetMoney() { return \_money; }

SalesData combine(const SalesData &);

// 常量成员函数

int Sum(std::vector<SalesData>) const;

private:

int \_money;

};

// 函数声明

SalesData add(const SalesData &, const SalesData &);

#endif

SalesData.cpp文件

#include "SalesData.h"

#include <vector>

SalesData SalesData::combine(const SalesData & data){

return SalesData(this->\_money + data.\_money);

}

int SalesData::Sum(std::vector<SalesData> datas) const{

int sum = 0;

for(auto data : datas){

sum = sum + data.\_money;

}

return sum;

}

SalesData add(const SalesData & left, const SalesData & right)

{

return SalesData(left.\_money + right.\_money);

}

**友元类与友元类成员函数**

class Screen

{

// 声明友元类，WindowMgr可以访问Screen的私有成员

friend class WindowMgr;

// 声明友元函数，WindowMgr::Clear可以访问Screen的私有成员

friend void WindowMgr::Clear();

public:

private:

};

**类内定义数据类型别名**

class Screen

{

public:

// 为 size\_type 定义别名，该别名为 public 访问权限，外部可以使用该别名

typedef std::string::size\_type pos;

pos GetCursor();

private:

};

**构造函数**

class Screen

{

public:

// 默认构造函数

Screen() = default;

// 委托构造函数，这个构造函数会调用下面的构造函数

Screen(pos height, pos width): Screen(height, width, ' '){ };

// index提供默认值0，:后面为初始化属性列表

Screen(pos height, pos width, char c, int index = 0) : Height(height),

Width(width),

Contents(height \* width, c),

index(index), // 常量数据成员必须赋予默认值或在初始化列表中初始化

index1(index) // 引用数据成员必须赋予默认值或在初始化列表中初始化

{

};

// 常量数据成员

const int index = 1;

// 引用数据成员

int &index1;

private:

};

**隐式内联函数，const函数**

// 隐式内联函数（在类声明中定义函数）

// const成员函数（设计类的时候，一个原则就是对于不改变数据成员的成员函数都要在后面加 const）

char Get() const { return Contents[Cursor]; }

**\*this返回当前对象**

// 返回当前对象（this存放的时当前对象指针）

Screen &Set(pos line, pos col, char c)

{

Contents[line \* Width + col] = c;

return \*this;

}

**const函数与this**

const函数使用的this是const \*类型

// 由于 GetConstThis 是常量，所以 GetConstThis 使用的this是const \*类型

Screen const &GetConstThis() const { return \*this; }

**类不能包含自身类**

// 错误：类不能包含自身，对象的空间是根据类型分配的，如果类包含自身则无法计算类的空间

// Screen ChildScreen;

// 正确：ChildScreen的类型是指针，空间大小固定

Screen \*ChildScreen;

**静态成员**

// 静态成员函数，外部使用 Screen::GetIntanceNum() 访问

static int GetIntanceNum();

// 静态成员变量，静态成员变量必须在类外定义并赋值

static int abc;

// 类外部定义定义静态成员变量

int Screen::abc = 0;

**常量类对象**

const SalesData colorPan(30); // 声明colorPan为常量（colorPan在生成之后不会改变）

// colorPan.GetMoney(); // 错误，这里GetMoney并非常量函数，常量对象只能调用常量函数（如果可以调用非常量函数，这意味着GetMoney可以改变colorPan的数据成员）

**默认构造函数生成对象**

// 下面的定义是错误的

// 下面的语句被解读成，声明colorPan1，colorPan1是函数，函数返回SalesData

// SalesData colorPan1();

// colorPan1.GetMoney();

// 下面的定义是正确的，使用默认构造函数

SalesData colorPan1;

**类类型隐式转换和显示转换**

// 类类型隐式转换

// combine接受的是一个SalesData类型，但SalesData类型有一个接受int的构造函数

// 下面的语句等价于 book.combine(SalesData(10));

// 通过在构造函数前加 explicit 可以阻止这种行为 如： explicit SalesData(const int money) : \_money(money){ };

book.combine(10);

// 类的显示转换

book.combine(static\_cast<SalesData>(10));

**constexpr对象**

// constexpr对象，这里使用默认构造函数生成对象，这样constexprClass的默认构造函数必须是constexpr

constexpr ConstexprClass constexprClass;

// constexprClass.field = 1; // 错误constexpr对象不能进行修改

class ConstexprClass

{

private:

/\* data \*/

public:

// constexpr构造函数，必须初始化所有数据成员

constexpr ConstexprClass(/\* args \*/): field(1){};

int field;

};