CPlusPlus编程语言基础

1.2.2. 算数类型

1.2.2.1. 整型字面值

1.2.2.1.1. 第一位是1~9的整数是十进制

1.2.2.1.2. 第一位是0第二位是1~7的整数是八进制

1.2.2.1.3. 前两位是0x或0X的整数是十六进制

1.2.2.1.4. 进制

1.2.2.1.5. 以l或L结尾的整数是long型

1.2.2.1.6. 以ll或LL结尾的整数是long long型

1.2.2.1.7. 以u或U结尾的整数是unsigned型

1.2.2.1.8. 类型

1.2.2.2. 字符型字面值

1.2.2.2.1. char字面值

1.2.2.2.2. 字符编码

1.2.2.2.3. 转义字符

1.2.2.3. bool型字面值：true、false

1.2.2.4. 浮点型字面值

1.2.2.4.1. 小数点表示法

1.2.2.4.2. 科学计数法

1.2.2.4.3. 表示方法

1.2.2.4.4. 以f或F结尾的浮点数是float型

1.2.2.4.5. 以L结尾的浮点数是long double型

1.2.2.4.6. 类型

1.2.3. 枚举

1.2.3.1. 关键字enum

1.2.3.2. 作用域内枚举

1.2.3.3. 枚举量

1.2.3.3.1. 默认值

1.2.3.3.2. 显式赋值

1.2.4. 符号常量

2. 数据类型

2.1. 类型

2.1.1. 类型 （程序开发语言）

2.1.1.1. 指定基本类型完成了三项工作

2.1.1.2. 不同数据类型占用的字节数根系统有关

2.2. 分类

2.2.1. 内置类型

2.2.1.1. 基本类型

2.2.1.1.1. 整型

2.2.1.1.1.1. 5种整型关键字：char、int、long、long long

函数库

2.2.1.1.1.2. 无符号类型关键字：unsigned

2.2.1.1.1.3. 字符类型扩展：char -> wchar_t、char16_t、char32_t

2.2.1.1.2. 浮点型

2.2.1.1.2.1. float、double、long double

2.2.1.1.3. 单位类型

2.2.1.1.3.1. size_t

sizeof( )

2.2.1.1.3.2. size_type

2.2.1.2. 复合类型

2.2.1.2.1. 数组

2.2.1.2.1.1. 数组声明

数组名与数组的地址

2.2.1.2.1.2. 运算符[ ]

2.2.1.2.1.3. 数组初始化规则

2.2.1.2.1.4. 二维数组

理解 arr[M][N]

char二维数组、字符串指针数组、string对象数组

2.2.1.2.1.5. 函数指针数组

2.2.1.2.1.6. 数组的替代品

静态数组：模板类array

动态数组：模板类vector

面相数值计算的数组：模板类valarray

对比

2.2.1.2.2. 字符串

2.2.1.2.2.1. 分类

- string类
  - 操作
    - 拼接+
    - 复制=
    - 字符数size()、length()
- C风格字符串
  - 初始化方法
    - 双引号法
    - 数组法
  - 字符串字面值
  - 空字符：\0
  - C库函数
    - 拼接strcat()
    - 复制strcpy()、strncpy()
    - 大小strlen()
    - 比较strcmp()
- 原始字符串
  - R"字符串"
- 其它形式
  - wchar_t、char16_t、char32_t

u16string、u32string

2.2.1.2.3. 结构

2.2.1.2.3.1. 关键字struct

2.2.1.2.3.2. 成员运算符.

2.2.1.2.3.3. 结构数组

2.2.1.2.3.4. 结构中的位字段

2.2.1.2.4. 共用体

2.2.1.2.4.1. 关键字union

2.2.1.2.5. 指针

2.2.1.2.5.1. 解除引用*

2.2.1.2.5.2. 分配内存

运算符 new

堆

运算符delete

内存泄漏

数据对象

管理数据内存的方式

动态存储

静态存储

自动存储

根据用于分配内存的方法

局部变量、栈

static、静态区

new、堆

2.2.1.2.5.3. 指针运算

递增（减）运算符和指针： *++pt、++*pt

2.2.1.2.5.4. 指针的应用

指针与数组

创建与删除

使用动态数组

遍历数组等价式

arr[i] == *(arr + i)

&arr[i] == arr + i

指针与字符串

指针与结构

箭头成员运算符->

指针与类

2.2.1.2.5.5. 指针和关键字const

指向const对象的指针

const指针

指向const对象的const指针

2.2.1.2.5.6. 分类

函数指针

函数指针声明

函数名不等于函数指针

作用：调用函数和做函数的参数

C++实现多态性的虚函数表是通过函数指针实现

指向函数指针数组的指针

空指针

关键字nullptr

智能指针：帮助管理动态内存分配

悬挂指针（野指针）

广义指针

迭代器

2.2.1.2.6. 引用

2.2.1.2.6.1. 声明引用

2.2.1.2.6.2. 引用的特点

2.2.1.2.6.3. 主要适应对象

结构和类

2.2.1.2.6.4. 主要作用

作为函数参数

尽量使用const

从函数中返回左值

函数返回引用

2.2.1.2.6.5. 左值引用与右值引用

分类

左值引用&

拷贝语义

右值引用&&

移动语义和右值引用

强制移动

std::move( )

左值、左值引用、右值、右值引用的比较

2.2.2. 自定义类型

2.2.2.1. 类型 （技术名词）

2.2.2.2. 类

2.2.3. void类型

2.2.3.1. 关键字void

2.3. 类型转换

2.3.1. 按数据类型分类

2.3.1.1. 标准转换

2.3.1.1.1. 基本数据类型之间的转换

2.3.1.1.2. 指针、引用、指向成员的指针派生类型的转换

2.3.1.2. 基本类型->自义定类型（类对象）

2.3.1.2.1. 方法1.构造函数（只能是单参数值）

2.3.1.2.2. 方法2.重载赋值运算符

2.3.1.2.3. 对比

2.3.1.3. 自义定类型（类对象）->基本类型

2.3.1.3.1. 转换函数（特殊的运算符函数）

2.3.1.3.2. 转换函数、友元函数、重载函数

2.3.2. 按是否强制分类

2.3.2.1. 自动类型转换

2.3.2.1.1. 存在问题

2.3.2.2. 强制类型转换

2.3.2.2.1. 显式强制类型转换

2.3.2.2.1.1. 显式转换运算符

关键字explicit

2.3.2.2.2. 隐式强制类型转换

2.3.3. 按是否隐式分类

2.3.3.1. 隐式转换

2.3.3.2. 显式转换

2.3.4. 按语言分类

2.3.4.1. C语言中

2.3.4.2. C++中

2.3.4.2.1. dynamic_cast

2.3.4.2.1.1. 指向基类的指针向下转型为指向派生类的指针

2.3.4.2.2. const_cast

2.3.4.2.2.1. 限定符const 和volatile之间转换

2.3.4.2.3. static_cast

2.3.4.2.3.1. 基类和派生类之间的显式转换、数值类型之间转换

2.3.4.2.4. reinterpret_cast

2.3.4.2.4.1. 指针、引用、整数三种类型之间的转换

3. 表达式与语句

3.1. 表达式

3.1.1. 值

3.1.1.1. 左值与右值

3.1.1.1.1. 分类

3.1.1.1.1.1. 左值

可修改左值

不可修改左值

特殊类左值：函数、数组

3.1.1.1.1.2. 右值

纯右值

将亡值

3.1.1.1.2. 转换

3.1.1.1.2.1. 左值->右值

运算符取地址 &

3.1.1.1.2.2. 右值->左值

运算符解引用 *

3.1.1.2. 表达式的值

3.1.2. 运算符

3.1.2.1. 运算符分类

3.1.2.2. 运算符性质

3.1.2.2.1. 运算符的优先级

3.1.2.2.2. 运算符的结合性：左、右结合性

3.1.2.2.3. 操作数个数：单、双、三目运算符

3.1.2.2.4. 可重载性

3.1.2.3. 运算符重载

3.1.2.3.1. 运算符函数

3.1.2.3.1.1. 关键字operator

3.1.2.3.1.2. 调用方法

函数表示法

运算符表示法

3.1.2.3.1.3. 作为类的成员函数还是非成员函数

3.1.2.3.2. 重载限制

3.1.3. 表达式分类

3.1.3.1. 赋值表达式

3.1.3.1.1. 赋值运算符=

3.1.3.1.2. 组合赋值运算符

3.1.3.2. 算数表达式

3.1.3.2.1. 双目加减乘除取余运算符+-*/%

3.1.3.2.2. 单目正运算符+、单目负运算符 -

3.1.3.3. 逻辑表达式

3.1.3.3.1. 逻辑运算符

3.1.3.3.1.1. 逻辑与运算符 &&、逻辑或运算符 ||、逻辑非运算符 !

3.1.3.4. 关系表达式

3.1.3.4.1. 大于运算符>、小于运算符=、小或等于运算符

3.1.3.4.2. 等于运算符==

3.1.3.4.2.1. 不等于运算符!=

3.1.3.5. 条件表达式

3.1.3.5.1. 条件运算符? :

3.1.3.6. 逗号表达式

3.1.3.6.1. 逗号运算符

3.1.3.6.1.1. 逗号分隔符（区分）

3.2. 语句

3.2.1. 复合语句{ }

4. 流程控制

5.1.1.1. 函数名

5.1.1.1.1. 函数地址

5.1.1.2. 函数声明（函数原型）

5.1.1.2.1. 默认参数

5.1.1.2.2. 函数声明与定义的关系

5.1.2. 函数参数

5.1.2.1. 实参argument与形参parameter

5.1.2.2. 局部变量

5.1.2.3. 参数的类型

5.1.2.3.1. 一维数组名

5.1.2.3.1.1. 数组表示与指针表示

5.1.2.3.1.2. 显式传递数组大小

5.1.2.3.2. 二维数组名

5.1.2.3.3. C风格字符串

5.1.2.3.3.1. C风格字符串作为返回值

5.1.2.3.4. 结构

5.1.2.3.5. 对象

5.1.2.3.5.1. string

5.1.2.3.5.2. array

5.1.2.3.6. 函数

5.1.2.3.7. 引用

5.1.2.4. 重载引用参数

5.1.2.5. 参数传递方式

5.1.2.5.1. 分类

5.1.2.5.1.1. 按值传递

5.1.2.5.1.2. 按指针传递

5.1.2.5.1.3. 按引用传递

5.1.2.5.2. 指导原则

5.1.2.5.2.1. 使用传递的值而不做修改的函数

5.1.2.5.2.2. 修改调用函数中数据的函数

5.1.2.5.2.3. 经验

5.1.3. 函数分类

5.1.3.1. 普通函数

5.1.3.2. 递归函数

5.1.3.3. 内联函数

5.1.3.3.1. 关键字：inline

5.1.3.4. 友元函数

5.1.3.4.1. 关键字friend

5.2. 深入

5.2.1. 函数重载（函数多态）

5.2.1.1. 方式：使用不同的参数列表完成相同的工作

5.2.1.2. 参数列表（函数特征标）

cin.getline(字符数组名, 长度);

cin.get(字符数组名, 长度)

getline(cin, string对象名)

6.1.1.1.1.5. cin状态标志

cin.eof()

cin.fail()

cin.good()

6.1.1.1.1.6. 输入错误

分类

类型不匹配

解决方案

条件判断

cin.clear()、cin.sync()和cin.ignor()

6.1.1.2. 输出

6.1.1.2.1. 对象cout

6.1.1.2.1.1. ostream类

6.1.1.2.1.2. 输出不同进制的整数

6.1.1.2.1.3. 输出一个字符

cout.put(char)

6.1.1.2.1.4. 控制输出的格式

cout.setf()

输出布尔值true、false

6.1.1.2.1.5. 输出char和wchar_t类型字符

6.1.1.2.1.6. 控制符endl

6.1.2. 文件

6.1.2.1. 逻辑划分

6.1.2.1.1. 文本文件

6.1.2.1.1.1. 读取

ifstream类

6.1.2.1.1.2. 写入

ofstream类

6.1.2.1.2. 二进制文件

6.1.2.1.3. 区别

6.1.2.2. 文件结尾EOF

6.1.2.3. 回车与换行

6.1.2.3.1. \r与\n与\n\r

6.2. 深入

6.2.1. 输入和输出概述

6.2.1.1. 流和缓冲区

6.2.1.1.1. streambuf类

6.2.1.2. iostream文件

6.2.1.2.1. ios_base类

6.2.1.2.1.1. ios类

ostream类

istream类

iostream类

6.2.1.2.2. 自动创建对象

6.2.1.2.2.1. 用于窄字符流

cin、cout、cerr、clog

6.2.1.2.2.2. 用于宽字符流

wcin、wcout、wcerr、wclog

6.2.1.3. 重定向

6.2.1.3.1. 标准输入和输出流通常是指键盘和屏幕

6.2.1.3.2. 可使用重定向改变流连接对象

6.2.1.3.2.1. 输入重定向

6.2.1.3.2.2. 输出重定向>

6.2.2. 使用cout进行输出

6.2.2.1. 输出基本类型和字符串

6.2.2.1.1. 插入运算符

6.2.2.2. 输出字符put( )

6.2.2.2.1. 输出字符串write( )

6.2.2.3. 刷新输出缓冲区

6.2.2.3.1. flush

6.2.2.3.1.1. endl

6.2.2.4. 输出格式化

6.2.2.4.1. 格式常量

6.2.2.4.1.1. 标准控制符

6.2.2.4.2. 浮点数的精度

6.2.2.4.3. cout成员函数precison( )、width( )、fill( )

6.2.2.4.4. 头文件iomanip

6.2.2.4.4.1. 控制符setprecision( )、setw( )、setfill( )

6.2.3. 使用cin进行输入

6.2.3.1. 输入基本类型和字符串

6.2.3.1.1. 格式化抽取方法

6.2.3.1.1.1. 抽取运算符>>

6.2.3.2. cin>>检查输入

6.2.3.2.1. 跳过空白（空格、换行、制表符）

6.2.3.3. 流状态

6.2.3.3.1. 流状态成员

6.2.3.3.1.1. 到达文件尾

eofbit

eof( )

6.2.3.3.1.2. 流被破坏

badbit

bad( )

6.2.3.3.1.3. 与预期不符

failbit

fail( )

6.2.3.3.1.4. 正常状态

goodbit

good( )

6.2.3.3.1.5. 操作

返回流状态rdstate( )

设置状态

clear( )

setstate( )

6.2.3.3.2. 流状态的影响

6.2.3.3.2.1. I/O和异常

异常

exceptions( )

exceptions(isostate ex)

6.2.3.3.3. 主动检测流状态

6.2.3.4. 其它istream类方法

6.2.3.4.1. 非格式化输入函数

6.2.3.4.1.1. 单字符输入

cin.get(char&)

cin.get( )

对比

6.2.3.4.1.2. 字符串输入：getline( )、get( )、ignore( )

6.2.3.4.2. 意外字符输入：文件尾、流被破坏、无输入、输入到达或超过指定最大字符数

6.2.3.4.3. read( )、peek( )、gcount( )、putback( )

6.2.4. 文件输入和输出

6.2.4.1. fstream族与iostream族

6.2.4.1.1. ifstream继承自istream

6.2.4.1.2. ofstream继承自ostream

6.2.4.1.3. fstream继承自iostream

6.2.4.2. 简单的文件I/O

6.2.4.3. 流状态检查

6.2.4.3.1. is_open( )

6.2.4.4. 打开多个文件

6.2.4.5. 命令行处理技术

6.2.4.5.1. int main(int argc, char *argv[ ])

6.2.4.6. 文件模式

6.2.4.6.1. 文件模式常量

6.2.4.6.2. C语言模式字符串：″r″、″w″、″a″、″r+″、″w+″

6.2.4.6.3. 对应关系

6.2.4.7. 随机存取

6.2.4.7.1. 指针移动

6.2.4.7.1.1. 输入指针移动

seekg( )

6.2.4.7.1.2. 输出指针移动

seekp( )

6.2.4.7.2. 获取指针当前位置

6.2.4.7.2.1. 对于输入流

tellg( )

6.2.4.7.2.2. 对于输出流

tellp( )

6.2.4.7.2.3. 注意

6.2.4.7.3. 使用临时文件

6.2.4.7.3.1. tmpnam( )

6.2.5. 文件类型

6.2.5.1. 文本文件

6.2.5.1.1. 插入运算符>和get( )读取

6.2.5.2. 二进制文件

6.2.5.2.1. write( )写入、read( )读取

6.2.5.2.1.1. 例子

6.2.5.3. 对比

6.2.6. 内核格式化

6.2.6.1. sstream族

6.2.6.1.1. ostringstream继承自ostream

6.2.6.1.1.1. str( )成员函数

6.2.6.1.2. istringstream继承自istream

6.2.6.1.3. stringstream继承自iostream

6.2.6.2. sstream族给格式化的文本提供了缓冲区

7. 内存模型

7.1. 单独编译

7.1.1. 文件与翻译单元

7.1.2. 头文件管理

7.1.2.1. 包含头文件命令#incude

7.1.2.1.1. "filename"与

7.1.2.2. 头文件内容

7.1.2.3. 预处理器编译命令

7.1.2.3.1. #ifndef

7.1.2.3.2. #define、#endif

7.1.3. 多个库的连接

7.1.3.1. 名称修饰

7.1.3.2. 链接错误

7.1.3.2.1. 重新编译源代码

7.2. 介绍

7.2.1. 分类：自动变量、寄存器变量（摒弃）、静态变量（包含3种）、动态变量

7.2.2. 实质：自动变量、静态变量、动态变量（动态存储）

7.3. 变量存储方式

7.3.1. 自动变量

7.3.1.1. 自动变量的初始化

7.3.1.2. 自动变量和栈

7.3.2. 静态变量

7.3.2.1. 初始化

7.3.2.1.1. 零初始化

7.3.2.1.2. 常量表达式初始化

7.3.2.1.3. 动态初始化

7.3.2.2. 特性

7.3.2.2.1. 静态持续性、外部链接性

7.3.2.2.1.1. 全局变量（外部变量）

作用域解析运算符::

7.3.2.2.2. 静态持续性、内部链接性

7.3.2.2.2.1. 局部变量（内部变量）

关键字static

关键字extern

7.3.2.2.3. 静态存储持续性、无链接性

7.3.2.2.3.1. 代码块和函数中

关键字static

7.3.3. 存储三特性

7.3.3.1. 持续性->变量在内存保留（持续）时间

7.3.3.1.1. 自动存储持续性

7.3.3.1.2. 静态存储持续性

7.3.3.1.3. 线程存储持续性

7.3.3.1.4. 动态存储持续性

7.3.3.2. 作用域->变量在文件的多大范围内可见（可被程序使用）

7.3.3.2.1. 代码块

7.3.3.2.2. 文件

7.3.3.3. 链接性->变量在哪些文件之间共享

7.3.3.3.1. 无链接性（只能在当前函数或代码块中访问）

7.3.3.3.2. 内部链接性（只在当前文件中访问）

7.3.3.3.3. 外部链接性（可在其他文件中访问）

7.3.4. 变量5种存储方式（引入命名空间前）特性总结

7.4. 存储说明符

7.4.1. auto（C++11以后不再是说明符）

7.4.1.1. register

7.4.1.1.1. static

7.4.1.1.1.1. extern

thread_local

mutable

7.5. cv-限定符

7.5.1. 关键字const

8.1.3.2.1. 作用域解析运算符::

8.1.3.2.2. 直接成员运算符.

8.1.3.2.3. 间接成员运算符->

8.2. 新的命名空间特性

8.2.1. 分类

8.2.1.1. 全局命名空间

8.2.1.1.1. 全局变量

8.2.1.2. 自定义命名空间

8.2.1.2.1. 关键字namespace

8.2.2. 特性

8.2.2.1. 外部链接性（默认）

8.2.2.2. 解决命名冲突

8.2.2.3. 开放性

8.2.2.4. 传递性

8.2.2.5. 可以嵌套

8.2.2.5.1. 可别名

8.2.2.6. 无名命名空间

8.2.3. 访问命名空间中名称的方法

8.2.3.1. 直接指定标识符

8.2.3.1.1. 作用域解析运算符::

8.2.3.2. 关键字using

9.1.2.3.1. 目标：使得使用类与使用基本的内置类型（如int）尽可能相同

9.1.2.3.2. 关键字class

9.1.2.3.3. 类规范

9.1.2.3.3.1. 类声明（蓝图）

数据成员

成员函数

9.1.2.3.3.2. 类方法定义（细节）

实现类成员函数

9.1.2.3.4. 类设计步骤

9.1.2.3.4.1. 1.提供类声明

访问控制

关键字private

默认private

数据成员常放在private部分

数据隐藏

关键字public

public和抽象

公共接口

函数成员常放在public部分

关键字protected

存储控制

关键字static

静态成员变量

静态成员函数

9.1.2.3.4.2. 2.实现类成员函数

类成员函数特点

类作用域

作用域解析运算符::

限定名

非限定名

类方法可以访问类的私有成员

内联函数

内部链接性

共享一组成员函数

const成员函数

9.1.2.4. 使用简单的类

9.1.2.4.1. 创建对象（类的实例）

9.1.2.4.1.1. 将类名视为类型名

9.1.2.4.2. 使用类函数（公有接口）

9.1.2.4.2.1. 成员运算符句点.

9.1.2.4.2.2. 客户-服务器模型

9.1.3. 类的构造函数和析构函数

9.1.3.1. 构造函数

9.1.3.1.1. 作用：初始化对象

9.1.3.1.1.1. 初始化与赋值

成员初始化列表

类内初始化

等价

9.1.3.1.2. 声明和定义构造函数

9.1.3.1.2.1. 声明构造函数

默认参数

9.1.3.1.2.2. 定义构造函数

9.1.3.1.3. 默认构造函数

9.1.3.1.3.1. 默认构造函数初始化

调用方法：显式调用、隐式调用

9.1.3.2. 析构函数

9.1.3.2.1. 作用：完成清理工作

9.1.3.2.2. 通常由编译器决定调用析构函数的时机

9.1.3.2.3. 默认析构函数

9.1.3.3. 构造函数和析构函数

9.1.3.3.1. 都没有返回类型（连void都没有）

9.1.4. this指针

9.1.5. 类作用域

9.3.1.1.1.1. 关键字static

9.3.1.1.1.2. 整型const、枚举型const静态成员可以在类声明中初始化

9.3.1.1.2. 静态类成员函数

9.3.1.1.2.1. 关键字static

9.3.1.2. 特殊成员函数

9.3.1.2.1. 分类

9.3.1.2.1.1. 默认构造函数

默认的默认构造函数

9.3.1.2.1.2. 复制构造函数

默认的复制构造函数

9.3.1.2.1.3. 赋值运算符

默认的赋值运算符

重载赋值运算符

9.3.1.2.1.4. 注意

“浅复制”与“深复制”

复制与赋值的异同

连续赋值问题

9.3.1.2.1.5. 析构函数

默认的析构函数

9.3.1.2.1.6. 移动构造函数

默认的移动构造函数

9.3.1.2.1.7. 移动赋值运算符

默认的移动赋值运算符

9.3.1.2.1.8. 注意

使用时机

9.3.1.2.2. 注意事项

9.3.1.2.2.1. 编译器自动提供的函数

自动定义

存在隐患

显式自定义

解决隐患

9.3.1.2.2.2. 启用和禁用成员函数

启用默认的方法

关键字default

禁用方法

关键字delete

伪私有方法

9.3.2. 改进后的新String类

9.3.3. 在构造函数中使用new时的注意事项

9.3.3.1. 使用new时的推荐做法

9.3.3.1.1. new的三个“统一”

9.3.3.1.2. 空指针：NULL、0、nullptr

9.4.1.1. 派生一个类

9.4.1.1.1. 继承的内容

9.4.1.1.1.1. 不能继承的内容

9.4.1.1.2. 需要增添的内容

9.4.1.1.2.1. 构造函数

9.4.1.1.3. calss 派生类名: 访问控制符 基类名

9.4.1.2. 派生类构造函数

9.4.1.2.1. 访问权限的考虑

9.4.1.2.1.1. 派生类不能直接访问基类的私有成员

9.4.1.2.2. 是否使用初始化列表

9.4.1.2.2.1. 使用基类默认构造函数

9.4.1.2.2.2. 显式调用基类构造函数

9.4.1.3. 派生类和基类之间的特殊关系

9.4.2. 继承：is-a关系

9.4.2.1. 继承方式分类

9.4.2.1.1. 公有继承

9.4.2.1.1.1. 能建立

is-a

has-a

is-implement-as-a

uses-a

导致问题

9.4.2.1.1.2. 不能建立

is-like-a

9.4.2.1.2. 保护继承

9.4.2.1.3. 私有继承

9.4.2.2. 类之间关系分类

9.4.2.2.1. is-a

9.4.2.2.1.1. 使用公有继承来处理

is-a关系的进一步抽象

9.4.2.2.2. has-a

9.4.2.2.2.1. 使用包含、私有继承和保护继承来处理

9.4.2.2.3. is-like-a

9.4.2.2.3.1. 设计共有特征的类来处理

9.4.2.2.4. is-implement-as-a

9.4.2.2.4.1. 使用隐藏数据成员来处理

9.4.2.2.5. uses-a

9.4.2.2.5.1. 使用友元函数或类来处理

9.4.3. 多态公有继承

9.4.3.1. 实现多态公有继承机制

9.4.3.1.1. 在派生类中重新定义基类的方法

9.4.3.1.2. 使用虚方法

9.4.3.1.2.1. 关键字virtual

9.4.3.2. 多态中确定调用哪个类的方法

9.4.3.2.1. 通过限定名

9.4.3.2.1.1. 类名::方法名

类名决定

9.4.3.2.2. 通过引用或指针

9.4.3.2.2.1. 不使用关键字virtual

引用类型或指针类型决定

9.4.3.2.2.2. 使用关键字virtual

引用或指针指向的对象类型决定

9.4.4. 静态联编和动态联编

9.4.4.1. 函数名联编

9.4.4.1.1. 分类

9.4.4.1.1.1. 静态联编（早期联编）

9.4.4.1.1.2. 动态联编（晚期联编）

9.4.4.2. 指针和引用类型的兼容性

9.4.4.2.1. 向上强制转换

9.4.4.2.1.1. 传递性

9.4.4.2.1.2. 允许隐式向上类型转换

9.4.4.2.1.3. 本质

is-a关系

9.4.4.2.2. 向下强制转换

9.4.4.2.2.1. 只允许显式向下强制类型转换

运算符static_cast

9.4.4.3. 虚成员函数和动态联编

9.4.4.3.1. 为什么有两种类型联编以及默认为静态联编

9.4.4.3.1.1. 效率

9.4.4.3.1.2. 概念模型

9.4.4.3.2. 虚函数

9.4.4.3.2.1. 虚函数工作原理

虚函数表

9.4.4.3.2.2. 虚函数的作用

9.4.4.3.3. 有关虚函数注意事项

9.4.4.3.3.1. 构造函数

9.4.4.3.3.2. 析构函数

9.4.4.3.3.3. 友元

9.4.4.3.3.4. 没有重新定义继承的方法

9.4.4.3.3.5. 重新定义继承的方法

注意与函数重载区别

经验规则

9.4.5. 访问控制：protected

9.4.5.1. 对外部世界：保护成员的行为与私有成员相似

9.4.5.2. 对派生类：保护成员的行为与公有成员相似

9.4.6. 抽象基类

9.4.6.1. 抽象基类（abstract base class, ABC）

9.4.6.1.1. 纯虚函数

9.4.6.1.1.1. =0

9.4.6.1.2. 具体类

9.4.6.2. 应用ABC概念

9.4.6.3. ABC理念

9.4.7. 继承和动态内存分配

9.4.7.1. 分类

9.4.7.1.1. 派生类不使用new

9.4.7.1.2. 派生类使用new

9.4.7.1.2.1. 必须自定义

复制构造函数

赋值运算符

析构函数

9.4.7.2. 使用动态内存分配和友元的继承示例

9.4.8. 类设计回顾

9.4.8.1. 编译器生成的公有成员函数

9.4.8.1.1. 默认构造函数

9.4.8.1.2. 复制构造函数

9.4.8.1.3. 赋值运算符

9.4.8.2. 其它类方法

9.4.8.2.1. 构造函数

9.4.8.2.2. 析构函数

9.4.8.2.3. 类型转换

9.4.8.2.4. 按值传递对象与传递引用

9.4.8.2.5. 返回对象和返回引用

9.4.8.2.6. 使用const

9.4.8.3. 公有继承的考虑因素

9.4.8.3.1. is-a关系

9.4.8.3.2. 为什么不能被继承

9.4.8.3.3. 赋值运算符

9.4.8.3.4. 私有成员与保护成员

9.4.8.3.5. 虚方法

9.4.8.3.6. 析构函数

9.4.8.3.7. 友元函数

9.4.8.3.8. 有关使用基类方法的说明

9.4.8.4. 类函数小结

9.4.8.4.1. 成员函数属性

9.4.8.4.1.1. 能否继承

9.4.8.4.1.2. 成员还是友元

9.4.8.4.1.3. 能否默认生成

9.4.8.4.1.4. 能否为虚函数

9.4.8.4.1.5. 是否可以有返回类型

10. C++中的代码重用

10.1. 包含对象成员的类

10.1.1. 包含（也称为：组合、层次化）

10.1.1.1. 建立has-a关系

10.1.1.1.1. A对象中的B对象：A包含B

10.1.2. 对比

10.1.2.1. 包含=A类获得了其成员B对象的实现，但不继承接口

10.1.2.2. 公有继承=获得实现（若有）+继承接口

10.1.2.3. 接口与实现

10.1.3. 接口：这里的接口是狭义上的接口，特指被public访问控制符包含的类成员，包括公有的数据成员和公有的函数成员。

10.1.4. 初始化被包含的对象（成员对象）

10.1.4.1. 构造函数使用其成员名

10.1.4.1.1. 初始化顺序

10.1.5. 使用被包含对象的接口

10.1.5.1. 被包含对象的接口不是公有的，但可以在类方法中使用它。

10.1.5.2. 对象名.数据成员 / 对象名.函数成员

10.2. 私有继承

10.2.1. 建立has-a关系

10.4.2. 使用模板类

10.4.2.1. 必须显式地指出模板类的具体类型

10.4.3. 数组模板示例和非类型参数

10.4.3.1. 指定数组大小的数组模板

10.4.3.1.1. 方案1：在类中使用动态数组和构造函数来提供数目

10.4.3.1.2. 方案2：使用模板参数来提供常规数据的大小

10.4.3.1.3. 对比

10.4.3.2. 非类型参数（表达式参数）

10.4.3.2.1. array模板类

10.4.4. 模板多功能性

10.4.4.1. 递归使用模板

10.4.4.1.1. Array, 10> twodee;

10.4.4.2. 使用多个类型参数

10.4.4.2.1. 预定义模板类pair

10.4.4.3. 默认类型模板参数

10.4.5. 模板具体化

10.4.5.1. 隐式实例化

10.4.5.2. 显式实例化

10.4.5.3. 显式具体化

10.4.5.3.1. 部分具体化

10.4.6. 成员模板

12.2.2.1. getline(cin, str)

12.2.2.2. cin >> str

12.2.3. 使用字符串

12.2.4. 其他功能

12.2.4.1. 返回C-风格字符串str.c_str( )

12.2.5. 字符串种类

12.2.5.1. 本质：模板类basic_string的具体化，然后typedef取的别名

12.2.5.2. string、wstring、u16string、u32string

12.2.6. 解读

12.2.6.1. 成员函数和运算符被多次重载

12.2.6.1.1. 参数是string对象、C-风格字符串、char值

12.3. 智能指针模板类

12.3.1. 使用智能指针

12.3.1.1. 智能指针模板类auto_ptr、unique_ptr、shared_ptr

12.3.1.2. 所有的智能指针类都是explicit构造函数

12.3.1.3. 智能指针只能指向动态分配的堆内存

12.3.2. 注意事项

12.3.2.1. auto_ptr导致的多次删除同一对象

12.3.3. 选择智能指针

12.3.3.1. 要使用多个指向同一对象的指针

模板参数与函数符概念

12.5.1.1.1.2. 注意编译器不直接检查概念

12.5.1.1.2. 改进：表示概念上的继承

12.5.1.1.3. 模型：概念的具体实现

12.5.1.2. STL组成：容器(containers)、迭代器(iterators)、空间配置器(allocator)、适配器(adapters)、算法(algorithms)、函数符(functors)六个部分

12.5.2. 迭代器

12.5.2.1. 为何使用迭代器

12.5.2.1.1. 迭代器是STL算法的接口

12.5.2.1.1.1. 模板使得算法独立于存储的数据类型

12.5.2.1.1.2. 迭代器使得算法独立于使用的容器类型

12.5.2.1.2. 基于算法的要求设计迭代器特征和容器特征

12.5.2.1.3. 超尾标记

12.5.2.2. 迭代器类型

12.5.2.2.1. 分类

12.5.2.2.1.1. 输入迭代器、输出迭代器、正向迭代器、双向迭代器、随机访问迭代器

12.5.2.2.2. 共性：可以执行解除引用操作、可以进行比较、递增、同一个类级typedef名称：iterator

12.5.2.3. 迭代器层次结构

12.5.2.3.1. 5种迭代器功能具有层次递增的包含关系

12.5.2.4. 将指针用作迭代器

12.5.2.4.1. 适配器

12.5.2.4.2. 迭代器适配器

12.5.2.4.2.1. ostream_iterator模板使ostream输出用作迭代器接口

12.5.2.4.2.2. istream_iterator模板使istream输入用作迭代器接口

12.5.2.5. 其他有用的迭代器

12.5.2.5.1. reverse_iterator

12.5.2.5.2. back_insert_iterator、front_insert_iterator、insert_iterator

12.5.3. 容器种类

12.5.3.1. 容器

12.5.3.1.1. 容器概念

12.5.3.1.1.1. 容器的要求

12.5.3.1.2. 容器类型

12.5.3.1.2.1. vector、set、map等15种容器类型

12.5.3.2. 序列容器

12.5.3.2.1. 序列的要求

12.5.3.2.1.1. 序列的可选要求

12.5.3.2.2. 分类

12.5.3.2.2.1. vector

12.5.3.2.2.2. deque

12.5.3.2.2.3. list

forward_list

12.5.3.2.2.4. 适配器类

queue

priority_queue

stack

12.5.3.2.2.5. 非STL容器

array

12.5.3.3. 关联容器

12.5.3.3.1. 有序关联容器

12.5.3.3.1.1. set

multiset

12.5.3.3.1.2. map

multimap

12.5.3.3.1.3. 模板类pair

12.5.3.3.2. 无序关联容器

12.5.3.3.2.1. unordered set、unordered multiset,、unorderedmap、unordered multimap

12.6. 函数对象（函数符）

12.6.1. 函数符概念

12.6.1.1. 函数对象是重载了( )运算符的类

12.6.1.2. 分类

12.6.1.2.1. 生成器、一元函数、二元函数

12.6.1.2.2. 谓词、二元谓词

12.6.2. 预定义的函数符

12.6.2.1. 头文件function

12.6.2.1.1. 运算符和对应的函数符

12.6.3. 包装器（适配器）

12.6.3.1. 自适应函数符和函数适配器

12.6.3.1.1. 自适应函数符

12.6.3.1.2. 函数适配器

12.6.3.1.2.1. 函数适配器类

binder1st、binder2nd

12.6.3.1.2.2. 函数适配器函数

bind1st( )、bind2nd( )

12.6.3.2. function类模板

12.6.3.2.1. 可调用类型：函数名、函数指针、函数对象、有名称的Lambda表达式

12.6.4. 函数符替代品：Lambda表达式

12.6.4.1. 比较函数指针、函数符和Lambda函数

12.6.4.2. 有名的Lambda表达式

12.6.4.3. 额外功能：访问作用域内变量

12.6.4.3.1. 例子：使用Lambda时机

12.7. 算法

12.7.1. 算法组

13.5.1.1. 位、字节、字

13.5.2. 运算方法与运算器

13.5.2.1. 运算方法

13.5.2.1.1. 数的机器码表示

13.5.2.1.1.1. 整数的表示

原码、反码、补码

13.5.2.1.1.2. 浮点数的表示

深入浅出浮点数

浮点数的表示和运算

13.5.2.1.1.3. 在线演示

13.5.2.1.2. 非数值数据的表示

13.5.2.1.2.1. Unicode

国际化策略：字符编码问题

编码类型：ASCII、UTF-8、UTF-16、UTF-32

13.5.2.1.2.2. ASCII编码

13.5.2.2. 运算器

13.5.2.2.1. 定点数四则运算

13.6. 编译原理

13.6.1. 编译原理知识汇总

13.6.2. 程序编译过程

13.6.3. 例子：gcc编译C语言程序

14. 字符函数库

14.1. ctype库

14.1.1. 字母或数字？

14.1.1.1. isalnum( )

14.1.2. 用于字母

14.1.2.1. isalpha( )

14.1.2.2. islower( )

14.1.2.3. isupper( )

14.1.2.4. tolower( )

14.1.2.5. toupper( )

14.1.3. 用于数字

14.1.3.1. isdigit( )

14.1.3.2. isxdigit( )

14.1.4. 用于空白

14.1.4.1. isspace( )

14.1.5. 用于标点

14.1.5.1. ispunct( )

14.1.6. 用于控制符

14.1.6.1. iscntrl( )

14.1.7. 用于打印符

14.1.7.1. isgraph( )

14.1.7.2. isprint( )

15. 源代码

15.1. 预编译

15.1.1. 预处理过程的代码

15.2. 编译

15.2.1. 汇编代码

15.3. 汇编

15.3.1. 目标代码

15.4. 连接

15.4.1. 可执行代码

16. C++开始的地方

16.1. C++百度百科

16.1.1. 官网

16.1.1.1. 参考手册

16.1.1.1.1. 教程

16.2. 本资源作者 袁宵

16.2.1. https://yuanxiaosc.github.io/

17. 附录F 模板类string

17.1. 模板basic_string定义了13种类型,供以后定义方法时使用

17.1.1. 常量npos

17.2. string的数据方法

17.2.1. 迭代器

18.1.1.1.3. X::iterator、X::const_iterator

18.1.1.1.4. X::different_type

18.1.1.1.5. X::size_type

18.1.1.2. 为所有容器定义的操作

18.1.1.2.1. X u、X( )、X(a)、X u(a)、X u = a

18.1.1.2.2. r = a

18.1.1.2.3. (&a)->X( )

18.1.1.2.4. begin( )、end( )、cbegin( )、cend( )

18.1.1.2.5. size( )、maxsize( )

18.1.1.2.6. empty( )

18.1.1.2.7. swap( )

18.1.1.2.8. ==、！=

18.1.1.3. 可反转容器定义的类型和操作

18.1.1.3.1. 可反转容器：vector、list、deque、array、set、map

18.1.1.3.2. X::reverse_iterator( )、**X::const_reverse_iterator( )**

18.1.1.3.3. a.rbegin( )、a.rend( )、a.crbegin( )、a.crend( )

18.1.1.4. 有序容器操作

18.1.1.4.1. 除无序set和无序map容器外都需要支持的操作

18.1.1.4.2. 、=

18.1.2. 序列容器的其它成员

18.1.2.1. 为序列容器定义的其它操作

18.1.2.1.1. 序列容器：vector、forward_list、list、deque、array

18.1.2.1.2. X(n, t)、X a(n, t)

18.1.2.1.2.1. X(i, j)、X a(i, j)

18.1.2.1.3. X(初始化列表对象)

18.1.2.1.3.1. a = 初始化列表对象

18.1.2.1.4. a.cmplace(p, args)

18.1.2.1.5. a.insert(p, t)、a.insert(p, n, t)

18.1.2.1.5.1. a.insert(p, i, j)

a.insert(p, 初始化列表对象)

18.1.2.1.6. a.resize(n)、a.resize(n, t)

18.1.2.1.7. a.assign(i, j)、a.ssign(n, t)、a.ssign(初始化列表)

18.1.2.1.8. a.erase(q)、a.erase(q1, q2)

18.1.2.1.9. a.clear( )

18.1.2.1.10. a.front( )

18.1.2.2. 为某些序列容器定义的操作

18.1.2.2.1. a.back( )

18.1.2.2.1.1. 部分序列容器：vector、list、deque

18.1.2.2.2. a.push_back(t)

18.1.2.2.2.1. 部分序列容器：vector、list、deque

18.1.2.2.3. a.pop_back( )

18.1.2.2.3.1. 部分序列容器：vector、list、deque

18.1.2.2.4. a.emplace_back(args)

18.1.2.2.4.1. 部分序列容器：vector、list、deque

18.1.2.2.5. a.push_front(t)

18.1.2.2.5.1. 部分序列容器：forward_list、list、deque

18.1.2.2.6. a.emplace_front()

18.1.2.2.6.1. 部分序列容器：forward_list、list、deque

18.1.2.2.7. a.pop_front( )

18.1.2.2.7.1. 部分序列容器：forward_list、list

18.1.2.2.8. a[n]、a.at(n)

18.1.2.2.8.1. 部分序列容器：vector、deque、array

18.1.2.3. vector的其它操作

18.1.2.3.1. a.capacity( )

18.1.2.3.2. a.reserve(n)

18.1.2.4. list的其它操作

18.1.2.4.1. a.splice(p, b)、a.splice(p, b, i)、a.splice(p, b, **i, j)**

18.1.2.4.2. a.remove(const T& t)

18.1.2.4.3. a.unique( )、a.unique(BinaryPredicate **bin_pred)**

18.1.2.4.4. a.merge(b)、a.merge(b, Compare comp)

18.1.2.4.5. a.sort( )、a.sort(Compare comp)

18.1.2.4.6. a.revese( )

18.1.2.5. forward_list操作

18.1.2.5.1. insert_after( )、erase_after( )、splice_after( )

18.1.2.5.1.1. 其它操作同list

18.1.3. 有序关联容器

18.1.3.1. 有序关联容器：set、multiset、map、multimap

18.1.3.2. 为有序关联容器定义的类型

18.1.3.2.1. X::key_type

18.1.3.2.2. X::key_compare

18.1.3.2.3. X::value_compare

18.1.3.2.4. X::mapped_type

18.1.3.2.4.1. 仅限于容器map、multimap

18.1.3.3. 为有序关联容器定义的操作

18.1.3.3.1. X(i, j, c)、X a(i, j, c)、X(i, j)、X a(i, j)

18.1.3.3.2. X(初始化列表对象)

18.1.3.3.3. a = 初始化列表对象

18.1.3.3.4. a.key_comp( )

18.1.3.3.5. a.value_comp( )

18.1.3.3.6. a_uniq.insert(t)

18.1.3.3.6.1. a_eq.insert(t)

18.1.3.3.7. a.insert(p, t)

18.1.3.3.8. a.insert(初始列表对象)

18.1.3.3.9. a_uniq.emplace(args)、a_eq.emplace(args)

18.1.3.3.10. a.emplace_hint(args)

18.1.3.3.11. a.erase(迭代器)

18.1.3.3.11.1. e.erase(q1, q2)

18.1.3.3.12. a.clear( )

18.1.3.3.13. 键值k相关操作

18.1.3.3.13.1. a.erase(k)

a.find(k)

a.count(k)

18.1.3.3.13.2. a.lower_bound(k)

a.upper_bound(k)

18.1.3.3.13.3. a.equal_range(k)

18.1.3.3.13.4. a.operator[ ](k)

仅限于map

18.1.4. 无序关联容器

18.1.4.1. 无序关联容器：unordered_set、unordered_multiset、unordered_map、**unordered_multimap**

18.1.4.2. 为无序关联容器定义的类型

18.1.4.2.1. X::key_type

18.1.4.2.2. X::key_equal

18.1.4.2.3. X::hasher

18.1.4.2.4. X::local_iterator

18.1.4.2.4.1. X::const_local_iterator

18.1.4.2.5. X::mapped_type

18.1.4.3. 为无序关联容器定义的操作

18.1.4.3.1. X(n, hf, eq)、X a(n, hf, eq)、X(i, j, n, hf, eq)、X a(i, j, n, hf, eq)

18.1.4.3.2. b.hash_function( )

18.1.4.3.3. b.key_eq( )

18.1.4.3.4. b.bucket_count( )

18.1.4.3.4.1. b.max_bucket_count( )

18.1.4.3.5. b.bucket(键值)

18.1.4.3.6. b.bucket_size(n)

18.1.4.3.7. b.begin(n)、b.end(n)、b.cbegin(n)、b.cend(n)

18.1.4.3.8. b.load_factor( )

18.1.4.3.8.1. b.max_load_factor( )、b.max_load_factor(z)

18.1.4.3.9. a.rehash(n)

18.1.4.3.10. a.reserve(n)

18.2. STL函数

18.2.1. 非修改式序列操作

18.2.1.1. all_of( )、any_of( )、none_of( )

18.2.1.2. for_each( )

18.2.1.3. find( )、find_if( )、find_if_not( )

18.2.2.10. reverse( )、reverse_copy( )

18.2.2.11. rotate( )、rotate_copy( )

18.2.2.12. shuffle( )

18.2.2.12.1. random_shuffle( )

18.2.2.13. partition( )

18.2.2.13.1. stable_partition( )

18.2.2.14. partition_copy( )

18.2.2.15. partition_point( )

18.2.3. 排序和相关操作

18.2.3.1. sort( )、stable_sort( )、partial_sort( )、**partial_sort_copy( )**

18.2.3.2. is_sorted( )

18.2.3.2.1. is_sorted_until( )

18.2.3.3. nth_element( )

18.2.3.4. lower_bound( )

18.2.3.4.1. upper_bound( )

18.2.3.5. equal_range( )

18.2.3.6. binary_search( )

18.2.3.7. merge( )

18.2.3.7.1. implace_merge( )

18.2.3.8. includes( )

18.2.3.9. set_union( )

18.2.3.9.1. set_intersection( )

18.2.3.10. set_defference( )

18.2.3.10.1. set_symmetric_difference( )

18.2.3.11. make_heap( )

18.2.3.11.1. push_heap( )

18.2.3.11.1.1. pop_heap( )

sort_heap( )

18.2.3.12. is_heap( )

18.2.3.12.1. is_heap_until( )

18.2.3.13. min( )

18.2.3.13.1. max( )

18.2.3.13.1.1. minmax( )

18.2.3.14. min_element( )

18.2.3.14.1. max_element( )

18.2.3.14.1.1. minmax_element( )

18.2.3.15. lexicographic_compare( )

18.2.3.16. next_permutation( )

18.2.3.16.1. previous_permutation( )

18.2.4. 通用数字运算

18.2.4.1. accumulate( )

18.2.4.2. inner_product( )

18.2.4.3. partial_sum( )

18.2.4.4. adjacent_difference( )

18.2.4.5. iota( )