C++1

* 基础知识
  + 数组与指针
    - 概念
      * 数组
        + 数组是用于存储多个相同类型数据的集合
      * 指针
        + 指针是一个变量，这个变量存储的事另一个变量的地址（十六进制数）
    - 关系
      * 使用数组作为参数传递时，数组名将退化为指针，指针指向数组首元素的地址
      * 指针数组
        + 存放指针变量的数组，int \* arr[8]
      * 数组指针
        + 指向数组首元素的指针，int (\*arr)[8]
    - 区别
      * sizeof
        + 使用sizeof可得数组这个连续内存块的存储空间
        + 使用sizeof可得指针变量所占的内存空间为4字节（32位）或8字节（64位）
  + 函数指针与指针函数
    - 函数指针
      * 指向函数的指针，使函数可以像参数一样被传递
      * 函数名就是一个指针，可以赋值给函数指针
      * int (\*fun)(int a,int b); int sum(int a,int b){ return a+b;} fun = sum; int s=(\*fun)(3,5);
    - 指针函数
      * 返回类型为指针的函数
        + int \* sum(int a,int b)
  + 引用和指针
    - 引用
      * 引用是充当另一个变量的**别名**的变量，常用于**函数的参数**
      * 用途
        + 主要用于在函数参数传递中，解决大块数据或对象的传递效率和空间不如意问题，提高参数传递效率
      * 引用和被引用的变量的值和地址都是相同的，因为它们共用一个内存空间，引用本身不占存储单元
      * 常量引用和引用常量
      * 使用准则
        + 只有左值可以传递给引用参数
        + 引用参数不会导致复制，而是函数直接访问被传递的变量
        + 引用一旦绑定到某个变量，就不能再指向其他对象
        + 必须在声明引用时将其初始化
        + 可以创建指针的引用
        + 不能建立数组的引用，数组是若干个变量组成的集合
      * 参数为const引用
        + 产生临时变量

实参类型正确，但不是左值（常规变量和const变量）

const int &a=6;//正确

实参类型不正确，但可以转换为正确的类型

* + - * + 使用理由

避免无意修改原有数据

可以处理const和非const类型，const只可以指向const

可以正确生成并使用临时变量

* + - * 左值引用
        + 左值：在C++11中可以取地址，有名字的就是左值
        + int a = 1;int &b=a;
      * 右值引用
        + C++11引入了右值引用的概念

实现了转移语义和精确传递

消除两个对象交互式不必要的对象拷贝，节省运算存储资源，提高效率

能够更简洁明确地定义泛型函数

* + - * + 右值：没有名称的临时变量
        + int &&b = 2；
        + 右值引用不能约束左值：int a=0;int &&b=a;//(错误)int &&b = std::move(a);//正确++b;//a=b=1;
      * 作为返回值
        + 以引用作为返回值，需要在函数名前加上&
        + 优点：在内存中不产生被返回值的副本
        + 规则

不能返回局部变量的引用

不能返回函数内部new分配的内存的引用

可以返回类成员的引用，但最好是const

* + - 联系
      * 都可以用于多态
      * 修饰函数参数
    - 区别
      * 指针是一个变量，有自己的内存空间，而引用只是个别名，不能单独存在
      * 可以有空指针，但没有空引用，引用必须初始化
      * 使用sizeof，指针大小是4或8字节，而引用则是被引用对象的大小
      * ++运算符意义不一样
      * 指针被初始化后可以指向其他对象，而指针一旦指向某个对象就不能更改
      * 指针需要解引用&，引用可以直接操作数据
      * 指针可以有多级指针，而引用只有一级，不存在引用的引用
    - 如何相互转换
      * 取地址&
      * 解引用\*
  + class和struct
    - 区别
      * 默认访问权限和继承关系不同，class是private，struct是public
      * class可以定义模板类形参，比如template <class T, int i>
    - 使用
      * 类比结构体增加了操作数据的行为，就是函数
      * struct用于数据结构集合
      * class用于对象的继承多态封装，虽然struct也可以
    - 如何计算结构体长度
      * 内存对齐问题
  + 内存动态分配和静态分配
    - 时间不同
      * 静态分配发生在程序编译和链接的时候
      * 动态分配发生在程序调用和执行的时候
    - 空间不同
      * 堆
        + 动态分配的

由程序员释放内存

* + - * 栈
        + 静态分配

编译器完成，如局部变量

* + - * + 动态分配

函数malloc分配

栈的动态分配有编译器进行释放

* + 堆heap和栈stack的区别
    - 管理方式
      * 堆中的资源由程序员控制，容易造成内存泄漏
      * 栈资源由编译器自动管理
    - 分配方式与效率
      * 堆都是动态分配；栈有动态分配和静态分配
      * 堆由C++函数库提供，机制复杂，效率低；栈是操作系统提供的数据结构，计算机底层支持，效率高
    - 空间大小
      * 堆
        + 不连续的内存区域，链表存储，堆的大小受限于虚拟内存，空间大，比较灵活
      * 栈
        + 连续的内存区域，是一个后进先出的队列，大小是操作系统设置好的，2M/1M

VS中设置大小

项目-属性-链接器-系统-堆栈保留大小

* + - 碎片问题
      * 堆频繁地new/delete会造成大量碎片，使程序效率降低；栈是队列，进出一一对应，不会产生碎片问题
    - 生长方向
      * 堆是向上生长，向高地址方向增长
      * 栈是向下生长，向低地址方向生长
  + new/delete和malloc/free的区别
    - 类型
      * new/delete是C++的操作符，调用的是赋值运算符重载，它们是保留字，不需要头文件支持
      * malloc/free是C中的函数，需要头文件stdlib.h函数支持
    - 功能
      * 都是分配、释放内存
        + new/delete在分配和释放内存时，会自动调用构造函数/析构函数
        + malloc/free只是分配释放内存
    - 使用
      * 返回值
        + 分配成功

new返回对应类型的指针

malloc返回的是void\*（无类型）指针，需要通过强制类型转换

* + - * + 分配失败

new

异常处理机制（gcc）

返回空指针（VC++6.0）

malloc

返回空指针

* + - * 所需参数
        + malloc只关心内存的总字节数，不知道申请的内存是什么类型。需要使用sizeof计算出需要分配的内存空间大小
        + new分配空间的大小由编译器自己计算，只需要说明申请内存的类型。
      * 初始化
        + new可以对分配的内存空间进行初始化（仅限于单个内存，不能初始化数组，可以使用memset初始化）
        + malloc无初始化，内存中的是随机数据（使用calloc函数会自动将内存初始化为0）
      * 配对使用
        + new和delete；new [] 和delete []
        + malloc和free
      * 内存释放
        + 无论释放多少个内存空间，free只需传递一个指针，而delete需要加[ ]
      * 内存扩缩
        + 对于malloc分配内存后，若使用过程中内存不够或太多，可以使用realloc函数对其进行扩充或缩小
        + new分配好的内存不能这样被直观的改变
  + 关键字
    - const
      * 限定变量为只读，不可修改
        + 指针

常量指针

指向常量的指针，const int \* pi;或者 int const \* pi; pi指向的值不能改变，但是pi本身的值可以改变，例如可以设置该指针指向其他const值。

指针常量

指针类型的常量，int \* const pi; 指针本身的值不能更改，pi必须指向同一个地址，但是地址上的值可以更改

常量指针常量

指针本身为const，指向的值也为const：const int \* const pi; pi不能指向其他地址，所指向的值也不能改变

* + - * + 函数参数

限定函数只能读取参数的值，而不能修改

避免无意修改数据造成错误

能够处理const和非const实参，否则只能接收非const数据

* + - * + 全局变量

暴露了数据，程序任何一部分都可更改数据，设置成const可避免这样的风险

C中常用#define宏定义，C++中常用const全局变量

在其他文件中引用

引用式声明（extern关键字）

将const变量放在一个头文件中，然后在其他文件中包含

* + - * + 成员函数

常成员函数不会修改类的非静态数据成员

const对象只能调用const成员函数

常成员函数在声明和初始化时都要在（）后加上const进行修饰，提高可读性

常成员函数不允许调用同类的其他非常成员函数（静态成员函数除外）

可以修改类中静态数据成员

两个成员函数的名字和参数表相同，但一个是 const 的，一个不是，则它们算重载

* + - * #define和const区别
        + const常量有数据类型，宏定义没有，编译器可以对const常量做类型安全检查宏定义只是简单字符替换
        + 宏定义的替换在预编译时期，const在编译时期
        + 编译器会给const变量分配内存空间，宏定义是字符串代码
        + 调试工具可以对const常量进行调试，但是不能对宏常量进行调试
      * const和函数重载
        + 修饰返回值

const返回类型只有在修饰指针或引用时才有用

无法重载仅按返回值类型区分的函数

* + - * + 修饰参数

fun(int i) 和 fun(const int i)，不算重载

fun(char \*a) 和fun(char \* const a) ，不能重载

* + - * + 修饰成员函数

两个成员函数的名字和参数表相同，但一个是 const 的，一个不是，则它们算重载

非const对象优先调用非const函数

* + - static
      * 面向对象
        + 静态成员变量

存储于静态区

属于类，所有对象共享

类中声明

包含类方法文件中初始化

如果静态成员是const或枚举类型，可以在类声明中初始化

* + - * + 静态成员函数

属于类，所有对象共享

静态成员函数只能访问静态成员变量

不能调用普通成员函数

无this指针

* + - * C语言
        + 静态全局变量

与全局变量相比

自动初始化，都会自动初始化为0

生命周期不变，存储方式，存储位置不变，静态区（全局区），生命周期与整个程序相同

文件作用域改变，静态全局变量只能在当前文件中被使用，全局变量在整个源代码中都可以被使用（extern）

* + - * + 静态函数

文件作用域改变，只能在声明它的源文件中可见

其他文件中可以定义同名函数，而不冲突

静态函数会被自动分配在一个一直使用的存储区域，直到程序结束，避免了普通函数在调用过程中进栈出栈操作，提高运行速度

注意

不要再头文件中声明static的全局函数

不要在cpp内声明非static的全局函数

如果你要在多个cpp中复用该函数，就把它的声明提到头文件里去，否则cpp内部声明需加上static修饰；

* + - * + 静态局部变量

存储位置改变，由栈区转为静态区（全局区），生命周期与整个程序相同

自动初始化为0

作用域未改变

返回函数中静态变量的地址会发生什么

* + - * 概括
        + 隐藏功能

改变了全局变量和普通函数的文件作用域，只能在本文件中使用

* + - * + 自动初始化

将变量默认初始化为0

* + - * + 延长生命周期

存储在静态区，直到程序结束才会释放内存

* + - extern
      * 使用其他源文件的全局变量和函数，extern int a;
      * extern "C"
        + C是表示编译连接规约
        + 实现C和C++混合编程
    - volatile
      * 作用
        + 告诉编译器该变量值是不稳定的，可能被更改，需要去内存中读取该值而不是读取寄存器中的备份

多个线程都要用到的某个变量，而且变量的值会被改变

中断服务子程序中访问到的非自动变量

并行设备硬件寄存器的变量（如状态寄存器）

* + - * 特点
        + 易变的
        + 不可优化

告诉编译器不要对volatile声明的变量进行各种优化保证程序员写在代码中的指令一定会被执行

volatile int a;a = 1;

如果未声明为volatile两条代码会合并成一条

* + - * + 顺序执行的（原子性）

保证volatile变量间的顺序性，不会被编译器进行乱序优化

* + - * 能否和const一起用
        + 可以，const是只读，volatile是去内存中读取
      * 指针可以是volatile
      * 可以修饰函数参数
    - explicit
      * 只修饰成员函数（具体来说是构造函数），使构造函数不提供隐式转换的功能。
      * 如何避免编译器进行隐式转换
    - inline
      * 内联函数，修饰任意函数，此关键字是一个建议，他告诉编译器应该把此函数实现为内联函数。最先进的现代编译器是优化的编译器，它们自己决定要内联的对象，所以一度被认为很重要的关键字，现在只是一个对编译器的建议。
      * 编译器会在每处调用内联函数的地方直接把内联函数的内容展开，这样可以省去函数的调用的开销，提高效率
      * 一般用于函数体的代码比较简单的函数，不能包含复杂的控制语句，while、switch，并且内联函数本身不能直接调用自身。如果内联函数的函数体过大，编译器会自动的把这个内联函数变成普通函数
      * 普通函数在被调用的时候，系统首先要到函数的入口地址去执行函数体，执行完成之后再回到函数调用的地方继续执行，函数始终只有一个复制。 内联函数不需要寻址，当执行到内联函数的时候，将此函数展开，如果程序中有N次调用了内联函数则会有N次展开函数代码
      * 特点：以空间换时间，提高函数调用的效率。
      * inline和宏定义的区别
        + 宏定义的替换在预处理时期，而内联函数在编译时期
        + 宏定义没有类型检查，只是文本替换；内联函数有类型检查，是真正的函数
        + 宏定义和内联函数使用的时候都是进行代码展开，对于短小的函数，都能提高效率
    - typedef
      * 给已存在的类型定义一个别名
      * typedef由编译器解释，不是预处理器
      * 在其受限范围内，typedef比#define更灵活。
      * 与#define不同，typedef创建的符号名之受限于类型，不能用于值。
    - sizeof
      * 以字节为单位返回所指对象所占的内存空间
        + int

4 4

* + - * + double

8 8

* + - * + long

4 4

* + - * + long long

8 8

* + - * + char

1 1

* + - * + 指针

4 8

* + - * 发生在编译期间
        + cout << sizeof(1==1) << endl;//1int a = 0;cout << sizeof(a = 3) << endl;//4cout << a << endl;//0
        + int f1() { return 0; }; double f2() { return 0.0; } void f3() {} cout << sizeof(f1()) << endl; // f1()返回值为int，因此被认为是int cout << sizeof(f2()) << endl; // f2()返回值为double，因此被认为是double cout << sizeof(f3()) << endl; // 错误！无法对void类型使用sizeof cout << sizeof(f1) << endl; // 错误！无法对函数指针使用sizeof cout << sizeof\*f2 << endl; // \*f2，和f2()等价，因为可以看作object，所以括号不是必要的。被认为是double
        + double\* (\*a)[3][6]; cout << sizeof(a) << endl; // 4cout << sizeof(\*a) << endl; // 72cout << sizeof(\*\*a) << endl; // 24cout << sizeof(\*\*\*a) << endl; // 4cout << sizeof(\*\*\*\*a) << endl; // 8
      * sizeof(1==1),C和C++结果
        + C++

1，bool

* + - * + C

4，int

* + 野指针和悬空指针
    - 野指针
      * 未初始化的指针：char \*p，但是static char \*p不是野指针
      * 如何避免
        + 没有初始化时，让它指向NULL

一方面是提醒程序员这个指针未初始化，不可直接使用

一方面是便于程序员检查和避免野指针

* + - * + 给指针赋值时，检查它已经分配了空间，如果分配了，就先释放再赋值
        + 申请内存后判空
        + 给指针释放了内存，但程序还未结束，或者以后还需要使用，就将它指向NULL
        + 使用智能指针
    - 悬空指针
      * 所指向的对象被释放或被收回，但没有让指针指向NULL
  + sizeof和strlen()的区别
    - sizeof为操作符，参数为任意类型，以字节为单位返回所指对象所占内存大小
      * sizeof ("hello")== 6strlen ("hello")== 5
      * strlen("\0") == 0sizeof("\0") == 2
    - strlen()是函数，只能计算以'\0'结尾字符串的长度
    - sizeof是在编译期计算所指对象所占内存大小
    - strlen()在程序运行的时候才能计算出来
  + 函数传参方式
    - 值传递
    - 指针传递
    - 引用传递
  + 枚举
    - enum　枚举名　{枚举元素1,枚举元素2,……};
    - 第一个枚举成员的默认值为整型的 0，后续枚举成员的值在前一个成员上加 1
  + 联合体
    - 和struct的区别
      * 结构和联合都是由多个数据类型成员组成
      * 任意时刻，联合中只存放了其中一个成员（所有成员共用一块内存空间），结构是不同成员存放的地址不同
      * 对于联合的不同成员赋值，将会对其他成员重写，原来的成员值将不存在
    - 内存大小为里面最大长度的数据类型的大小