# 1 初识 C++

## 1.1 一些基本知识

- 集成开发环境

编辑器、编译器、链接器、调试器

- 程序的运行从 main 函数开始而开始，结束而结束
- 编译器是从上到下逐行编译
- 在语法描述中，[]表示可选的。
- C++语言集结构化编程、面向对象编程、泛型编程和函数式编程于一身，特别适合大型应用程序开发。
- C++的头文件是不带.h 扩展名的
- C++的所有关键字都是小写的
- C++11，空指针：nullptr，// C 语言的空指针是 NULL（0）

## 1.2 new/delete 内存管理

- 栈 :局部变量位于栈中 ,每个函数每一次运行 ,都会自动的分配/释放栈；形参（引用类型除外）也是局部变量。
- 堆 :每个进程只有一个堆。只能手动分配和释放。只能通过指针指向堆，不能通过变量名使用堆。

- new/delete 和 malloc/free 不能混用
- new[]和 new、delete[]和 delete 是两个运算符
- new[]主要用于创建动态数组

  例：int* p = new int[变量];

  class Demo{…};

  Demo* pd = new Demo[变量];　　// 此时调用 Demo 类的无参构造函数

  int arr[3] = {1,2,3};　// 静态数组的定义和初始化

## 1.3 参数默认值

- 只能按照从右往左的顺序给出默认值
- 如果既有声明、又有实现，则只需要声明中给出默认值。
- 特别地，成员函数的类外实现不应该有参数默认值。

## 1.4 重载

- 所谓重载（Overlord）函数就是指在同一作用域内（两个函数要么是全局函数，要么是同一类的成员函数）、函数名相同但参数列表不同的函数。
- 目的：使得代码简洁。
- 参数列表不同的含义：类型不同、个数不同、类型个数都不同。不包括变量名不同。
- 不以函数类型作为重载的依据。

```
int fun(int a, int b);          // 不构成重载，会报错
float fun(int a, int b);
```

- 当使用具有默认参数的函数重载时，需注意防止调用的二义性。

### 1.5 内联函数

- inline

### 1.6 引用

- 概念：别名。本质：指针。
- 用途：函数参数、函数返回值（可以让函数作为左值）、成员变量、父类引用指向子类对象。
- 不会给变量定义相同类型的引用。

## 2 类与对象（封装）

### 2.1 面向对象的三大特征

- 封装，隐藏内部实现
- 继承，复用现有代码
- 多态，改写对象行为

### 2.2 成员

- 成员可以是自身类型的指针、自身类型的引用、其他类型的对象（不能

循环定义），但不能是自身类型的对象（造成错误的递归）。

- 三类成员函数：构造/析构函数；Get/Set 函数；其他功性能函数
- 针对某一个成员变量，往往有一对 Set/Get 函数。

```
class Demo{
public:
    void Set×××(类型 形参){
        参数有效性检查;
        ××× = 形参;
    }
    类型 Get×××() {return ×××;}
private:
    类型 ×××;
};
```

- 成员访问运算符

对象.成员变量　　对象.成员函数(参数);

指针->成员变量　　指针->成员函数(参数);　　// 前提是指针指向对象

## 2.3 访问权限

### 2.3.1 private

- 默认访问权限
- 个人财产。类内访问，子类、类外均不能访问
- 如果没有派生类，则成员变量一般设置为 private，然后设置公有的 set/get 函数。

### 2.3.2 protected

- 家族财产。类内访问，子类可以访问，类外不能访问
- 基类的成员变量一般设为 protected，方便派生类访问

### 2.3.3 public

- 公共财产。类内、派生类、类外均可访问

### 2.3.4 C++中的 struct

- 其默认访问权限是 public。

## 2.4 构造函数与析构函数

- 每创建一个对象，就必然要调用一次构造函数。
- 注意：Demo* pd[4];// 指针数组只是定义了四个指针，还没有创建对象

## 2.4.1 作用

- 构造函数用来创建和初始化对象，析构函数用于释放对象

## 2.4.2 它们的四个特点

| | 构造函数 | 析构函数 |
|---|---|---|
| 1 | 与类同名 | 与类同名，前面加~ |
| 2 | 不能有类型， void、return 都不要 | |
| 3 | 可以带参数，能够重载 | 没有参数，不能重载 |
| 4 | 一般为公有函数 | 基类一般采用虚析构函数 |

## 2.4.3 无参构造函数

- 默认构造函数，是无参构造函数
- 这四行代码都调用无参构造函数。

  A x;

  A* pa = new A;

  A* pArr = new A[size];

- 通过 new[]创建对象的动态数组时，只能调用该类的无参构造函数
- 通过参数的默认值，也能达到无参构造函数的作用

## 2.4.4 拷贝构造函数

- 形参必须是自身类型的引用（包括常引用）；形参不能是自身类型

例：class Demo{

Demo(Demo d){…;}　　// 错误，只能是 Demo& d 或者 const Demo& d

};

## 2.4.5 默认函数

一个空类 class Demo{};

- 如果程序员没有提供构造函数，则编译器会自动提供默认构造函数。
- 类似的还有：默认析构函数、默认拷贝构造函数、默认赋值运算符。
- 默认拷贝构造函数、默认赋值运算符按照“按位复制”，如果类中又封装了指针类型的成员变量，则会造成指针悬挂。

## 2.5 常成员函数，不修改对象的数据成员

```
class Demo{
    void fun() const; // 常成员函数
};
void Demo::fun() const {...;}
```

- 常对象只能调用常函数

## 2.6 静态成员函数

- 关键字：static
- 普通成员函数，编译器会为其加上默认的 this 指针，但是静态成员函数除外。参数列表中没有默认的 this 指针
- 属于类，不属于对象。也就是说，该类的所有对象拥有同一个静态成员。静态成员类似于一个类范围内的全局变量。
- 既可以通过对象调用，也可以通过类调用类名::静态成员
- 静态成员变量必须类外初始化
- 典型应用：单例模式

## 2.7 友元 friend

### 2.7.1 友元函数

- 声明全局函数为友元，则该函数可以访问所有成员变量。
- 开后门，破坏了封装。
- 友元函数不是成员函数，和访问权限没有关系。

### 2.7.2 友元类

# 3 运算符重载

运算符重载的本质是函数重载

- 主要目的是代码简洁，而不是代码复用；
- 但是不能说运算符就是函数（算数运算符是通过 CPU 的指令直接实现的）。

## 3.1 两种重载形式

- 成员函数
- 全局函数，往往（不是必须）声明为友元。

## 3.2 重载为成员函数

- 左操作数必须是自身类型
- =、()、[]、-> ，这四个符号只能重载为成员函数

## 3.3 重载为全局函数

- 流输出运算符只能重载为全局函数

```
class Demo{
    friend ostream & operator<<(ostream& out, const Demo& d);
};
ostream& operator<<(ostream& o, const Demo& d){
    ......;
    return o;
}
```

三处引用的作用：两处 ostream&，是为了确保设备的唯一性和连续使用。右操作数采用常引用，是为了避免形参到实参拷贝、避免修改实参。

- 流输入运算符只能重载为全局函数

istream& operator>>(istream& in, 引用类型) // 需要写数据，不能是常引用

# 4 继承与派生

同一个问题的两个方面

- 基类（抽象稳定）和派生类（具体变化），父类和子类
- C++支持单继承（只有一个基类）和多继承（多个基类）。
- UML 类图中，由派生类指向基类（由下而上）。

## 4.1 继承方式

- 针对基类的公有和保护成员，继承方式决定其最高访问权限
- 针对基类的私有和不可访问成员（继承自基类的基类），都是不可访问成员
- 默认的继承方式是私有继承，最常见的继承方式是公有继承。

## 4.2 派生类对象

- 派生类对象拥有基类的所有成员（但是不一定能够使用）


- 构造函数的调用顺序：基类⇨成员对象⇨派生类
- 析构函数的调用顺序：派生类⇨成员对象⇨基类

# 5 多态与虚函数

## 5.1 虚函数

- 关键字：virtual
- 实现虚函数的核心数据结构是虚函数表。

### 5.1.1 成员函数，区分三个概念：重载、隐藏、覆盖

- 同一类中：重载
- 分别在基类、派生类中
  - 同时满足原型相同；virtual，即为覆盖
  - 否则（两个条件有一个不满足），即为隐藏

### 5.1.2 同名函数的调用原则

- 在隐藏的情况下，通过指针调用函数，取决于左侧变量的类型；
- 在覆盖的情况下，通过指针调用函数，取决于对象的类型。

## 5.2 多态的概念

多态是一种：调用同名函数却因**上下文不同**会有不同实现的一种机制。

多态是指：**不同的对象**调用同名函数，会有不同的实现。

- 静态多态，通过重载在编译阶段完成
- 动态多态，通过继承和虚函数在运行阶段完成（虚函数表）

## 5.3 纯虚函数与抽象类

- 虚函数没有函数体（=0）称之为纯虚函数
- 拥有纯虚函数的类，称之为抽象类（不会创建对象）
- 抽象类不能实例化对象

## 5.4 RTTI，运行时类型识别

- typeid

type(表达式) == type(类型);

- dynamic_cast<目标类型>(表达式);

向下类型转换，父类指针指向子类对象

```
class A{
    virtual fun() = 0;
};
class B: public A{
    void fb(){};
};
class C : public A{
    void fc(){};
};
```

A* pa = new B;  B* pb = dynamic_cast<B*>(pa);  pb->fb();

# 6 异常处理

- try…catch…throw
- throw 用于（通常在子函数中）抛出异常
- try 用于检测异常，把可能出现异常的语句放在 try 块（只能有一个）中。
- 多个 catch 块依次对异常按照类型进行匹配，用于捕获并处理异常。

# 7 IO 流

## 7.1 四个预定义流对象

- 包括 cin、cout、cerr 和 clog。
- >>提取运算符、<<插入运算符

## 7.2 文件读写

- 文本文件，既可以通过>>、<<进行读写，也可以通过成员函数进行读写。
- 二进制文件，只能通过成员函数进行读写。
- 典型应用

把文本文件中的数据读入到某一个 STL 的容器中。

```
istream& operator>>(istream& in, 容器引用){…; return in}

int main() {
    ifstream ifs;                 // 定义输入文件流对象
    ifs.open("文件名", ios::in);     // 打开文件
    if(!ifs) exit(0);             // 打开失败，或者 ifs.fail()，则退出
    while(!ifs.eof())  ifs>>容器;   // 当文件未结束.....
    ifs.close();
```

}

## 7.3 字符串流

略

# 8 模版编程

- 模版编程也称为泛型编程。（c++泛型编程是通过模板实现的）
- 模版编程的主要目的是代码复用。

## 8.1 函数模版实例化为函数

template <typename T>

函数类型 函数名(参数列表) {函数体;}

// template 和 typename（也可以用 class）都是关键字，这里不能有分号;

// 函数类型、参数列表、函数体中均可以使用 T

// 可以显式实例化，也可以隐式实例化（不提供类型实参，根据参数类型自动判断）

## 8.2 类模版实例化为类

必须显式实例化。

# 9 STL

## 9.1 基本概念

- Standard Template Library，标准模版库
- 三大核心组件：容器 container、迭代器 iterator、算法 algorithm
- 六大组件，再加上：适配器、仿函数（函数对象）、分配器
- 内部实现

| | |
|---|---|
| vector | 动态数组 |
| list | 双向链表 |
| map | 红黑树 |

## 9.2 vector

- 遍历 vector 的方法：迭代器、下标、基于范围的 for 循环、for_each 函数
- 连续的动态数组，可以在尾部快速插入和删除。

## 9.3 list

- 遍历 list 的方法：迭代器、基于范围的 for 循环、for_each 函数
- 双向链表，可以在任意位置插入和删除。

## 9.4 迭代器

- 定义迭代器时，必须要指定容器类型和元素类型。

  如：vector::iterator itr; // 错误 vector<int>::iterator itr;

- 在调用一些特殊的迭代器时 ,需要包含<iterator> ,比如说 :插入迭代器、IO 流迭代器。

## 9.5 算法

- 需要包含<algorithm>

```
#include <iostream>
#include <list>
#include <algorithm>
using namespace std;


int main() {
   list<int> li;// 整数链表
   // 不使用循环，而是通过STL::algorithm实现插入10个随机数，遍历输出
   generate_n(back_inserter(li),
        10,
        []() {return rand() % 100; }   // 0~99的随机数
   );
   copy(li.begin(), li.end(), ostream_iterator<int>(cout, " "));
}
```

# 10 设计模式

## 10.1 面向对象程序设计的思想

应用程序是对现实“花花世界”的仿真，主要特征是：种类很多、数量很多。

- 在软件开发过程中唯一不变的是变化。
- 在基类指针指向派生类对象时，基类指针代表抽象和稳定的，派生类对象代表具体和变化的。只有针对稳定的抽象进行编程，才能达到“以不变应万变”的效果。
- 三大设计原则：封装变化点；针对抽象进行编程；优先使用组合。
  - 继承被称为“is-a”关系，组合被称为“has-a”关系。
  - 继承和组合均能实现代码复用，优先使用组合。

```
class Base{ void fun(){…;}};     // 基类
class Derived : public Base{      // 公有派生类
};
Derived d;
d.fun();            // 通过继承复用代码
class C{
public:
        void fun(){pb->fun();}          // 委托
```

```
private:
        Base* pb;
};
C c;
c.fun();      // 委托组合的 Base 的成员对象来实现的。
```

- 开闭原则：对扩展开放，对修改关闭。（五大设计原则之一）

## 10.2 基本概念

- 最早提出这个概念的人是，建筑设计领域的亚历山大·克里斯托弗。
- 三种设计模式：创建型模式（如：单例模式）、行为型模式（如：策略模式）、结构型模式（如：适配器模式）。

## 10.3 单例模式

## 10.4 策略模式

C++：new/delete

C：malloc/free

C++的三大特点

1、同时支持四种编程范式：面向过程、面向对象、泛型编程、函数式编程

2、同 c 语言相比，适合开发大型应用程序

3、具有可复用、可维护、可扩展、灵活性好的特点

面向进程编程的基本单位：函数

面向对象的三大特征：封装、继承、多态