简介

C++ Tutorial

这样的提纲目的只是为了方便区别知识点的重要程度，有选择性地学习C++庞大而繁杂的知识体系，减少不必要的精力消耗。

所列提纲仅个人愚见，或有不妥，会及时更正。
有什么问题。。及时问。。

第1章 开始

- 从编写简单的C++程序开始，了解编译和运行程序的流程。
- 初步认识程序的输入输出过程，了解程序中定义变量的行为和规则。
- 初步认识控制流语句，对于控制流语句的基本模板有一个大体的认知。
- 了解类和成员函数。

第I部分 C++基础

第2章 变量和基本类型

1. 数据的内存表示：
    - 掌握数据在内存中的表示方法
    - 了解内存地址和数据存储的关系
2. 基本的内置数据类型：
    - 了解C++中所有内置数据类型
    - 掌握常用数据类型（bool，char，int，float，double）
    - 掌握数据类型的无符号和有符号形式（unsigned和signed）
    - 掌握不同数据类型间的强制转换（精度损失）
    - 掌握变量的字面值含义
3. 变量和标识符：
    - 掌握变量的声明，定义，赋值，初始化
    - 掌握变量的作用域
    - 掌握基本的C++标识符的规则
4. 复合数据类型：指针和引用
   掌握C++的复合类型，着重掌握指针和引用，要将两者的区别联系烂熟于心，不混淆。
    - 引用
      区别于普通类型的变量，引用有以下几个特点：
        - 引用只是别名，定义引用时，程序把引用和它的初始值绑定在一起
        - 引用本身不是对象，编译器不会为引用分配内存空间，它只是一个别名

      牢记以上两点就很容易理解引用了：

        - 引用必须初始化（定义时要执行绑定的过程）
        - 对引用进行操作，实际是在操作引用绑定的对象（引用只是别名）
    - 指针
      和引用一样，指针也能实现对其他对象的间接访问。
        - 指针和引用的最大区别就是，指针本身是一个对象
        - 指针和普通变量的最大区别是，指针存放的是其他对象的地址
5. const限定符。
    - 了解const限定符的基本功能和const的作用域
    - 掌握const限定符和指针引用的组合使用
    - 区别顶层const和底层const
    - 熟悉常量表达式和constexpr
6. 处理数据类型
    - 熟练掌握类型别名typedef的基本功能和用法
    - 了解推断类型decltype
    - 掌握auto类型变量的使用
7. 自定义数据类型
    - 着重掌握数据结构体和数据成员
    - 着重掌握头文件的编写方式和调用方式（头文件保护符）

第3章 字符串、向量和数组

1. 掌握命名空间和using声明
2. 熟练掌握string标准库类型
    - string是可变长的字符序列
    - 定义和初始化
        - 直接初始化和拷贝初始化（用赋值符号进行的初始化）
        - 直接初始化的两种形式：

```
string valuename("value");
string valuename(n,'c');
```

    - 操作
        - 输入(is<<s)和输出(os>>s)
        - 获取行getline()
        - 拼接、赋值、相等、不等
        - 字典顺序遍历<.<=,>=,>
            - 两个规则
        - 函数s.empty()和s.size()
    - 运用cctype头文件进行处理string对象中的字符
        - 基于范围的for

```
for (declaration: expression)
    statement
```

- 下标运算符
  - 掌握下标运算符接受的参数类型和返回值
  - 严格注意下标越界问题

3. 熟练掌握vector标准库
   - vector是可变长的对象序列
   - 使用vector有何优点?
   - 定义和初始化方式

```
vector<T> v1;
vector<T> v2(v1);
vector<T> v2 = v1;
vector<T> v3(n,val);
vector<T> v4(n);
vector<T> v5{a,b,c...};
vector<T> v5 = {a,b,c...};
```

     - 要注意到vector的对象类型的匹配问题
   - 操作
     - 添加元素v.push_back(t)
     - 函数v.empty()和v.size()
     - 相等、不等、字典顺序遍历<.<=,>=,>
       - 两个规则!
   - 无法向空vector对象通过使用下标操作的方式添加元素
4. 迭代器
   - 掌握迭代器的使用

```
atuo itBegin = v.begin(), itEnd = v.end();
```

     - begin成员返回第一个元素的迭代器
     - end成员返回指向"尾元素的下一个位置"的迭代器，又称为**尾后迭代器**
     - 当容器为空时，begin和end返回同一个迭代器，都是尾后迭代器
   - 掌握迭代器的操作
5. 数组
   - 掌握数组各种定义和初始化的方式
     - 数组的大小是确定不变的，需要在定义时明确指定
     - 数组维度必须是常量表达式
     - 数组本身是不允许进行赋值和拷贝操作的
   - 掌握复合类型数组的声明
   - 掌握如何访问并操作数组元素

        - 范围for
        - 下标
        - 迭代器、指针
            - 理解掌握指针操作和下标操作，了解两者的异同和等价关系
            - 理解掌握解引用和指针运算的交互
                - 数组名的内涵
    - 学会处理C风格字符串 并 与string类型对比优劣
    - 学会C风格字符串与string类型、vector容器的转化
6. 多维数组
    - 理解“多维数组”：数组的数组
    - 掌握多维数组的初始化方式，不同初始化方式对于数组元素初始值的影响。
    - 多维数组的下标引用和操作。==P115==
        - 用指针操作多维数组
            - 必须熟悉、明辨多维数组名会被转化为指向数组首元素的指针
            - 必须熟悉、明辨指针究竟指向了多维数组的何处

第4章 表达式

1. 什么是表达式（expression）？
    - 了解什么是表达式
    - 了解什么是运算符
    - 掌握表达式中组合运算符和其运算对象
        - 熟悉表达式中的运算符对运算对象的转换（类型转换）
        - 熟悉重载运算符操作
    - 熟练掌握表达式的左值和右值并区分
        - c++中，当一个对象被用作右值的时候，用的是对象的值（内容）；当对象用作左值的时候，用的是对象的身份（在内存中的位置）
    - 熟练掌握复合表达式中的优先级和结合规律
        - 必须熟记常见运算符的优先级顺序。==P147==
    - 了解表达式的求值顺序。知晓未定义求值顺序可能产生的后果
2. 算术运算符
    - 算术运算符包含：（依照优先级顺序）（括号内为运算符的用法示例）
        - 一元正号(+ expr)、一元负号(- expr))
        - 乘法(expr1 * expr2)、除法(expr1 / expr2)、求余(expr1 % expr2)
        - 加法(expr1 + expr2)、减法(expr1 - expr2)
    - 算术运算符都满足左结合律
    - 算术运算符的运算对象和求值结果都是右值
    - 了解溢出和其他算术运算符异常问题
3. 逻辑和关系运算符

- 逻辑和关系运算符包含：（依照优先级顺序）（括号内为运算符的用法示例）
  - 逻辑非 (!epxr)
  - 小于(expr1 < expr2)、 小于等于(expr1 <= expr2)、大于(expr1 > expr2)、大于等于(expr1 >= expr2)
  - 相等(expr1 == expr2)、不相等(expr1 != expr2)
  - 逻辑与(expr1 && expr2)
  - 逻辑或(expr1 || expr2)
- 逻辑和关系运算符中，只有逻辑非是右结合。
- 熟悉逻辑与(&&)和逻辑或(||)运算符的短路求值策略

4. 赋值运算符
   - 赋值运算符的左侧必须是一个可修改的左值
   - C++11 新标准的列表初始化用作赋值的右侧运算对象
   - 掌握的赋值运算符是右结合、优先级的等级
   - 切勿混淆相等运算符和赋值运算符
   - 复合赋值运算符==P131==
5. 递增和递减运算符
   - 掌握递增运算符(++)和递减运算符(--)的两种形式：前置版本和后置版本
   - 理解这条建议：
     - 建议：除非必须，否则不用递增运算符的后置版本
6. 成员访问运算符
   - 掌握点运算符和箭头运算符
     - 各自左右值的区分
7. 条件运算符
   - 掌握并运用唯一的三元运算符——条件运算符(cond ? expr1 : expr2)
8. 位运算符
   - 掌握移位运算符的原理和操作使用（区别于IO中的重载版本）
   - 掌握位求反的原理
   - 掌握位与、位或、位异或运算符的原理
9. sizeof运算符
   - 返回一个表达式或类型名字所占的字节数，结果是size_t类型
   - 对类型使用

     ```
     sizeof (type)
     ```

   - 对表达式使用

```
sizeof expr
Sales_data data, *p;

sizeof (Sales_data);

sizeof data;

sizeof p;

sizeof *p;
```

- sizeof不实际计算运算对象的值
- sizeof运算符的结果 ==P139==

10. 逗号运算符
    - 逗号运算符的优先级最低，又常与;混淆。
    - 需熟练掌握逗号运算符的运算方式。
    - 区分,和;将对程序产生的影响
11. 类型转换
    - 熟悉隐式转换（哪些情况会引发隐式转换）==P141==
    - 算数转换中的整型提升、无符号有符号等情况下的转换结果
    - 掌握显式转换的方式

第5章 语句

1. 简单语句
    - 表达式语句的构成：表达式末尾加上分号便构成了表达式语句
    - 了解空语句的常见用途
    - 掌握复合语句（块）
        - 复合语句的定义（花括号）
        - 复合语句在编程中使用的普遍性（举例）
        - 块不以分号作为结束。块本身由花括号决定始末。
2. 语句作用域
    - 熟悉在语句的不同位置出现的变量的作用域。（句内、句外，产生的影响）
3. 条件语句
    - if语句
        - 基本模式

```
//if语句
if (condition)
    statement;

//if-else语句
if (condition)
```

```
    statement1;
else
    statement2;

//嵌套if-else语句
if (conditon1)
    statement1;
else if(conditon2)
    statement2;
    ……
```

- 在使用if语句时，要注意statement处的语句块，需要合理使用花括号控制执行
- 悬垂else问题
  - C++规定：else与离他最近的尚未匹配的if匹配。
  - 与缩进对齐的位置无关

- switch语句
  - 熟练掌握switch的使用
    - case标签必须是整形常量表达式
    - 必须明确掌握switch内部的控制流，知晓switch内部的程序的执行顺序和方式
    - 掌握通过使用break语句控制执行
    - 掌握default标签
  - 了解在switch内部变量的定义情况

4. 迭代语句
   - while语句
     - 基本模式

```
while (condition)
    statement
```

     - 要注意到定义在while条件部分和while循环体内的变量每次迭代都会经历从创建到销毁的过程
   - 传统for语句
     - 基本模式

```
for (initializer; condition; expression)
    statement
```

     - 理解掌握for循环中的执行流程。可以用简单的流程图表示出来
     - 掌握在for语句头中可以进行多重定义
     - 掌握for语句头中的省略用法
     - 熟悉for语句中各部分的变量作用域
   - 范围for语句

        - 基本模式

```
for (declaration: expression)
    statement
```

        - 掌握范围for的执行原理
    - do while语句
        - do while语句是唯一一个必须先无条件执行一次循环体的迭代语句。
        - 基本模式

```
do
    statement
while (condition)
```

        - 掌握do while语句的执行流程
        - 学会利用do while语句无条件先执行一次循环体的特性进行合理的运行
5. 跳转语句
    - break语句
        - 理解掌握这句话
          break语句负责终止离他最近的while、do while、for或switch语句，并从**这些语句之后的第一条语句**开始继续执行
    - continue语句
        - 理解掌握这句话
          continue语句功能是终止**最近的循环体中的当前迭代**并立即开始**下一次迭代**
        - continue语句只能出现在for、while、do while循环的内部，或者嵌套在此类循环里的语句或语句块中
    - goto语句
        - goto的功能是从goto语句无条件跳转到同一函数的另一条语句。（不建议使用。）
        - 了解goto的用法
6. TRY语句块和异常处理
    - 暂不要求掌握此部分

第6章 函数

1. 函数基础
    - 什么是函数?
        - 掌握函数的声明
            - 函数原型：函数的三要素
        - 掌握函数的定义
        - 理解函数的声明和定义提供了分离式编译

- 函数的构成
  掌握函数的构成要素，区别不同函数的关键要素（重载）
  - 返回类型
  - 函数名
  - 参数列表（形参列表）
  - 函数体
    注：区别函数的关键在于***函数名***和***参数列表***

```
#include <iostream>

int add(int, int); // function prototype, declare a function

int main()
{
    int param_1 = 10;
    int param_2 = 5;
    int result = function(param_1, param_2); // call the function add

    return 0;
}

int add(int num1, int num2) // function definition
{
    return num1 + num2;
}
```

- 调用函数
  了解函数调用的过程机制
  - 函数如何调用?
  - 主调函数和被调函数
- 形参和实参
  理解并掌握形参和实参的区别和联系
  - 实参是形参的初始值
  - 实参的类型必须和形参的类型一一对应
- 函数返回类型与返回值
  理解并掌握函数返回类型与返回值
  - 有返回类型的函数体中，触发return执行结束即代表该次调用执行完毕。
- 局部对象
  理解函数中的局部对象及其生命周期
  - 函数体内所声明丁义的变量为局部变量
  - 函数体内的局部变量生命周期为函数执行周期
- 局部静态对象
  理解静态对象的作用域和生命周期
  - 局部静态对象的生命周期被延长至整个程序结束之时

- 自动对象
  了解什么是函数的自动对象

2. 参数传递
   理解并掌握不同的参数传递的机理，尤其是传值传递和引用传递
   - 形参的初始化机理与变量的初始化一致
   - 值传递
     - 值传递的过程是将初始值拷贝给参数，即，将实参的值复制到形参
     - 当形参为指针类型时，进行值传递也是将实参指针的值拷贝给形参

     理解这句话

     > 值传递过程中，函数对形参所有的操作不会影响实参，不改变实参的任何内容

   - 引用传递
     - 引用的操作实际上是作用的在引用所引的对象上
     - 引用传递允许函数改变一个或多个实参的内容
     - 普通引用只接受同类型的对象作为初始值
     - 常量引用可以用同类型对象、表达式、字面值初始化
   - 如果函数需要多个返回值，可以使用引用形参来返回额外信息

     理解这两句话

     > 传递引用可以避免对象的拷贝，因此建议尽量使用引用传递
     > 如果不需要改变引用形参的值，最好使用常量引用，它能接受的实参类型比普通引用多

   - const形参和实参
     - 当形参中有顶层const时，允许传递常量对象和非常量对象
       注意以下例子

     ```
     void func(const int i) {/* func能够读取i，但无法写入i */}
     void func(int i) {/* ... */} // 错误：重复定义了func
     ```

   - 数组形参
     - 了解掌握数组形参的两个特殊性质
       - 不支持拷贝
       - 数组名会被自动转化为指针（数组名 - 参考“数组”一章节）
     - 数组形参的使用和调用

       ```
       void print(const int*);
       void print(const int[]);
       void print(const int[10]);
       ```

       - 主观上我们会以为这三个是不同的定义，然而后面两个也会自动转化为`const int *`来处理
       - 在调用时，实参可以是数组名，也可以是整形指针
     - 了解数组形参并没有传递数组的大小
     - 掌握防止数组形参在函数体内越界的三种办法：

1. 使用数组引用形参

```
void print(int (&arr)[10] );
// &arr 必须用圆括号括起来提升优先级
```

   - 不过这种定义也限制了我们只能传递维度既定的数组

2. 使用标准库规范
   - 传递首元素和尾后元素的指针
3. 显式传递一个表示数组大小的形参

- 了解传递多维数组中的细节

```
void print(int matrix[10][10], int rowSize);
// 数组会被自动转化为指针，上述函数原型等同于
void print(int (*matrix)[10], int rowSize);
// 需要rowSize来指定二维数组的第一个维度大小
```

- main函数的形参：处理命令行选项
  - 简单了解main函数的形参的功能
- 含有可变兴产的函数
  - 简单了解如何传递不定量的参数

3. 返回类型和return语句
   - 了解return语句的功能
     - return语句终止当前正在执行的函数并将控制权返回到调用该函数的地方
     - return语句有两种形式

```
return;             // 无返回值
return expression;  // 有返回值
```

   - 无返回值函数
     - 无返回值函数无需return语句，也可以含有无返回值的return语句
     - 无返回值函数可以调用带有返回值的return，但返回值必须是另一个返回void的函数
   - 有返回值函数
     - 有返回值return提供了函数执行的结果
     - 返回值类型必须和函数返回值类型一致（或能够进行隐式类型转换）
     - 确保有返回值函数只能通过一条有效的return语句退出
       - 确保函数无论哪条路径都能有return退出
     - 了解函数返回值是如何返回的
       - 返回一个值的方式和初始化一个变量或形参的方式完全一样
       - 返回值用于初始化调用点的一个临时量，该临时量就是函数调用的结果
       - ***不要返回局部对象的引用或指针***

```
const string &manip()
{
    string ret;
    if( !ret.empty() )
    {
        return ret;       // 错误：返回局部对象的引用
    }
    else
    {
        return "Empty";  // 错误：“Empty”是一个局部对象
    }
}
```

- 返回类 类型的函数和调用运算符
  - 如果函数返回指针、引用或者类的对象，可以直接在函数调用的结果访问结果对象的成员

```
const string shorterString(string, string);
...
...
...
auto sz = shorterString(s1, s2).size();
// shorterString()的结果是一个string，故可以直接调用string中的size()方法
```

- 引用返回左值
  - 了解函数返回的结果是左值还是右值
    - 调用一个返回引用的函数得到左值，其他返回类型则得到右值
  - 如果函数返回的是引用类型的对象，可以直接对函数调用的结果进行赋值
- 列表初始化返回值
  - 了解C++11中支持的“函数可以返回花括号包围的值的列表”

```
vector<string> process()
{
    return {"Hi", "bye"};
}
```

- main函数的返回值
  - 简单了解mian函数返回值的意义
- 理解并掌握如何使用递归函数
- 返回数组指针
  - 了解函数为何无法返回数组
    - 数组无法被拷贝，函数无法返回数组
    - 函数可以返回数组的指针或引用

        - 了解C++11新增的尾置返回指针

4. 函数重载
    - 掌握何为重载函数和如何进行函数的重载
        - 同一作用域内的几个函数名相同但形参列表不同的函数，称为**重载函数**

```
// print函数的3个重载
void print(const char *cp);
void print(const int *beg, const int *end);
void print(const int ia[], size_t size)
```

        - 形参列表的不同指的是***形参数量不同***或***数量相同但形参类型不同***
        - 类型别名不构成重载，顶层const不构成重载
    - 简单了解const_cast重载
    - 重载函数的调用
        - 掌握函数匹配的机制（见“函数匹配”）
5. 特殊用途与语言特性 理解并掌握默认实参、内联函数和constexpr函数
    - 默认实参
        - 默认实参作为形参的初始值出现在形参列表中
        - 如果某个形参被赋予了默认值，它后面的所有形参都必须有默认值
        - 如果在调用时省略了实参，则使用默认实参初始化形参
    - 默认实参声明
        - 允许多次声明同一个函数，给不同的形参添加默认实参
        - 在给定的作用域中，一个形参只能被赋予一次默认实参

```
string screen( int width, int height, char title = ' ' );
string screen( int width, int height, char title = '*' );       //
错误：一个形参只能被赋予一次默认实参
string screen( int width = 24, int height = 80, char title );   //
正确：可以看到它并没有再次给title设置默认实参
```

    - 默认实参初始化

```
int wd = 80;
char def = ' ';
int ht();
string screen( int width = ht(), int height = wd, char title = def );


void f2()
{
    def = '*';
    //def改变了默认实参的值
    int wd = 100;
    //wd隐藏了外层定义的wd，但没有改变默认值
    string window = screen();
}
```

下面的两句话很好的解释了这个现象

> 用作实参的名字在函数声明所在的作用域内解析
> 求值过程发生在函数调用时

也就是说

> screen只会找同一作用域内的变量作为实参，所以后来定义的局部变量wd根本是不可见的
> 默认实参只是代替了实参，但是初始化形参的时机没有变，还是发生在函数调用时。因此改变全局变量def的值后，默认实参也发生了改变

- 内联函数
  - 了解函数调用的过程和内联函数的意义
  - 一次函数调用实际包含着一系列工作：
    - 调用前要先保存寄存器，并在返回时恢复
    - 可能要拷贝实参
    - 程序转向一个新的位置继续执行
  - 把函数声明为内联函数可以避免这一系列的开销

    > 函数声明前加上inline关键字将函数转换为内联函数
    >
    > > 声明为内联函数后，在编译时将函数在每个调用点上“内联地”展开
    > >
    > > > 因为要展开，所以内容长的函数不适合内联

- constexpr函数
  了解什么是constexpr函数
  - 能用于常量表达式的函数
    - 函数的返回类型必须是字面值类型
    - 所有形参的类型必须是字面值类型
    - constexpr函数被隐式地指定为内联函数

6. 函数匹配
   - 理解并掌握函数的匹配规则

1. 选定本次调用需要的对应的重载函数集，其中的函数称为**候选函数**
    - 候选函数的两个特征：
        1. 与被调函数同名
        2. 其声明在调用点可见
2. 根据调用实参，从候选函数中选出能被实参匹配的**可行函数**
    - 可行函数的两个特征
        1. 形参数量与本次调用提供的实参数量相等
        2. 每个实参的类型与对应的形参类型相同
3. 寻找最佳匹配

- 调用重载函数时的三种可能结果
    1. 编译器找到与实参**最佳匹配**的函数
    2. 找不到函数与调用的实参匹配，编译器报错**无匹配**
    3. 找到多个可匹配的函数，但每一个都不是最佳匹配，编译器报错**二义性调用**
- 多形参的重载调用
    - 具有多个形参的重载函数在调用时，最佳匹配需同时满足以下条件
        1. 每个实参的匹配都不劣于其他可行函数的匹配
        2. 至少有一个实参的匹配优于其他可行函数的匹配
        3. 满足条件的函数只能有一个

```
void f();
void f(int);
void f(int, int);
void f(double, double=3.14);

f(1, 1.2);  // 错误：二义性调用
// 考虑第一个参数时f(int, int)胜出，考虑第二个参数时f(double, double)胜出，因此没有最佳匹配
```

- 实参类型转换
    - 前面提到实参和形参的类型越接近，匹配的越好。有以下的排序规则：
        1. 精确匹配
            - 实参和形参类型相同
            - 实参从数组或函数类型转换成对应的指针
            - 向实参添加顶层const或从实参删除顶层const
        2. 通过const转换实现的匹配
        3. 通过类型提升实现的匹配
        4. 通过算术类型转换或指针转换实现的匹配
        5. 通过类类型转换实现的匹配
    - 调用重载函数时应该尽量避免强制类型转化。如果在实际应用中需要强制类型转换，则说明设计的形参集合不合理

7. 函数指针

- 理解函数指针的声明和定义
- 掌握函数指针在形参和返回值中的应用
- 声明函数指针
  - 函数的类型由它的返回类型和形参类型共同决定，与函数名无关
    `bool lengthCompare(const string&, const string&);`
  - 该函数的类型是:
    `bool (const string&, const string&)`
  - 声明函数指针:
    `bool (*pf)( const string&, const string& );`
- 使用函数指针
  函数指针有以下特殊:
  1. 函数名会自动地转换成指针，取地址符不是必须的
  2. 可以直接使用指向函数的指针调用函数，解引用不是必须的

```
pf = lengthCompare;
pf = &lengthCompare;
```

- 调用函数指针

```
bool b1 = pf( "Hello", "goodbye" );
bool b2 = (*pf)( "Hello", "goodbye" );
bool b3 = lengthCompare( "Hello", "goodbye" );
```

- 函数指针形参

```
void useBegger( const string &s1, const string &s2, bool pf( const
string &, const string & ));

void useBegger( const string &s1, const string &s2, bool (*pf)( const
string&, const string& ));
```

上面两个函数都是合法的

> 我们在使用函数指针作形参时, 可以显示的将形参定义成指向函数的指针, 也可以直接使用函数类型，会自动转换为函数指针

第7章 类

1. 定义抽象数据类型
2. 访问控制与封装
3. 类的其他特性
4. 类的作用域
5. 构造函数
6. 类的静态成员

第II部分 C++标准库

第8章 IO库

第9章 顺序容器

第10章 泛型算法

第11章 关联容器

第12章 动态内存

第III部分 类设计者的工具

此部分为C++高阶部分。暂不列入。