1 类型转换
- 1.1 静态转换 static_cast
- 1.2 使用方式 static_cast< 目标类型>（原始数据）
- 1.3 可以进行基础数据类型转换
- 1.4 父与子类型转换
- 1.5 没有父子关系的自定义类型不可以转换
- 1.6 动态转换 dynamic_cast
- 1.7 不可以转换基础数据类型
- 1.8 父子之间可以转换
  - 1.8.1 父转子 不可以
  - 1.8.2 子转父 可以
  - 1.8.3 发生多态 都可以
- 1.9 常量转换 const_cast
- 1.10 不能对非指针或者非引用进行转换
- 1.11 重新解释转换 reinterpret_cast
  - 1.11.1 最不安全，最鸡肋 不推荐

2 异常
- 2.1 try 试图执行 try{}中的内容
- 2.2 在可能出现异常的地方 抛出异常 throw
- 2.3 try 下面 catch 捕获异常
- 2.4 catch(捕获类型) …代表 所有其他类型
- 2.5 如果不想处理异常，继续向上抛出 throw
- 2.6 如果没有任何处理异常的地方，那么成员调用 terminate 函数，中断程序
- 2.7 自定义异常类 ，可以抛出自定义的对象 ，捕获自定义的异常

3 栈解旋
- 3.1 从try开始 到 throw 抛出异常之前 所有栈上的对象 都会被释放 这个过程称为栈解旋
- 3.2 栈上对象构造顺序与析构顺序相反

4 异常的接口声明
- 4.1 如果想抛出特定的类型异常 ，可以利用异常的接口声明
- 4.2 void func() throw (int) 只能抛出 int 类型
- 4.3 throw() 不抛出任何类型异常

5 异常变量生命周期
- 5.1 如果 MyException e，会多开销一份数据 ,调用拷贝构造
- 5.2 如果 MyExcepiton *e ， 不 new 提前释放对象 new 自己管理 delete
- 5.3 推荐 MyException &e 容易些 而且 就一份数据

6 异常的多态使用
- 6.1 利用多态来实现 printError 同一个接口调用
- 6.2 抛出不同的错误对象，提示不同错误

7 使用系统标准异常
- 7.1 #incldue <stdexcept>
- 7.2 throw out_of_range（”aaa”） 。。。
- 7.3 catch(out_of_range & e) cout << e.what();

8 编写自己的异常类

8.1 自己的异常类 需要继承于 exception
8.2 重写 虚析构 what（）
8.3 内部维护以错误信息 字符串
8.4 构造时候传入 错误信息字符串，what 返回这个字符串
8.5 string 转 char * .c_str();

9 标准的输入流
9.1 cin.get 缓冲区中读取一个字符
9.2 cin.get(两个参数) 不读换行符
9.3 cin.getline () 读取换行 并且扔掉
9.4 cin.ignore 忽略 （N） N 代表忽略字符数
9.5 cin.peek 偷窥 偷看 1 个字符然后放回去
9.6 cin.putback 放回 把字符放回缓冲区

10 输入流案例
10.1 判断用户输入的是字符串还是数字 利用偷窥 或者 放回
10.2 让用户输入指定范围内的数字，如果不正确 重新输入
10.2.1 cin.fail() 看标志位 0 正常 1 不正常
10.2.2 cin.clear()重置标志位
10.2.3 cin.syne() 清空缓冲区

11 标准输出流
11.1 流对象的成员函数
```
11.1.1 int number = 99;
11.1.2 cout.width(20);
11.1.3 cout.fill('*');
11.1.4 cout.setf(ios::left); //设置格式  输入内容做对齐
11.1.5 cout.unsetf(ios::dec); //卸载十进制
11.1.6 cout.setf(ios::hex); //安装16进制
11.1.7 cout.setf(ios::showbase); // 强制输出整数基数  0  0x
11.1.8 cout.unsetf(ios::hex);
11.1.9 cout.setf(ios::oct);
11.1.10  cout << number << endl;
```
11.2 控制符
```
int number = 99;
cout << setw(20)
<< setfill('~')
<< setiosflags(ios::showbase) //基数
<< setiosflags(ios::left) //左对齐
<< hex // 十六进制
<< number
<< endl;
```

12 文件操作
12.1 写文件
12.1.1 ofstream ofs
12.1.2 open 指定打开方式
12.1.3 isopen 判断是否打开成功