stdin, stdout, stderr

- In C, three text streams are predefined, and their type is (FILE *).
- stdin: standard input stream
- stdout: standard output stream, for conventional output
- stderr: standard error stream, for diagnostic output.

Output Stream 、Error Stream

- Send contents into streams in C and C++

C

```
fprintf(stdout, "Info: ...\n", ...);
printf("Info: ... \n", ...);

fprintf(stderr, "Error: ...\n", ...);
```

C++

```
std::cout << "Info: ..." << std::endl;
std::cerr << "Error: ..." << std::endl;
```

Redirection(重定向)

pipeline(管道): 一个进程的输出可作用另一个进程的输入

- The output of a program is in a pipeline.
- The output can be redirected. You can redirect the output into a file for debugging especially when the program run a very long time.

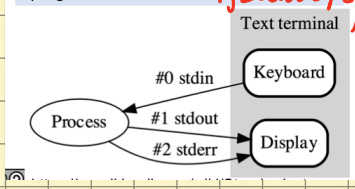
```
./program | less
./program > output.log
./program 1> output.log
./program >> output.log
./program > /dev/null
```

将stdout/cout内容直接销毁

```
./program 2> error.log
./program > output.log 2> error.log
./program &> all.log
./program > all.log 2>&1
```

所有都写到all.log中

all.log重定向为stdout.cout



将stdout/cout的结果写入log中，stderr/cerr的结果在终端中显示

将stderr/cerr的结果写入log中，stdout/cout的结果在终端中显示

less 打开文件，对大文件显示一部分前几行。
more 打开文件，对大文件显示大量前n行。
tail 打开文件，显示尾部n行（默认为10行）
cat 打开文件，能显示完后自动关闭文件。

assert

- assert is a function-like macro in <assert.h> and <cassert>.

```
#ifdef NDEBUG
# define assert(condition) ((void)0)
#else
# define assert(condition) /*implementation defined*/
#endif
```

- Do nothing if the condition is true
- Output diagnostic information and call abort() if the condition is false.
- If NDEBUG is defined, do nothing whatever the condition is.
- assert can be used only for debugging, be removed by a macro NDEBUG before releasing.

① condition == true 时什么都不做
② condition == false 时输出错误信息并调用abort()中断程序。
③ NDEBUG未定义，则什么都不做，无论condition

OpenCV 中的assert

- Many applications define their own assert macros.
- CV_Assert in OpenCV checks a condition at runtime and throws exception if it fails.

```
#define CV_Assert( expr ) do { if(!!(expr)) ; else cv::error( cv::Error::StsAssert, #expr, CV_Func, __FILE__, __LINE__ ); } while(0)
```

- cv::error() may behavior differently with different settings.

```
void cv::error ( int          _code,
                 const String & _err,
                 const char *   _func,
                 const char *   _file,
                 int            _line
               )
```

Exceptions (仅限C++, C中无异常处理)

Error Handling

① • Solution 1: Kill the program when error occurs

```
float ratio(float a, float b)
{
    if (fabs(a+b) < FLT_EPSILON)
    {
        std::cerr << "Error ..." << std::endl;
        std::abort();
    }
    return (a - b) / (a + b);
}
```

② • Solution 2: Tell the caller by the return value when error occurs
• We have to use the 3rd parameter to send the result.

```
bool ratio(float a, float b, float & c)
{
    if (fabs(a+b) < FLT_EPSILON)
    {
        std::cerr << "Error ..." << std::endl;
        return false;
    }
    c = (a - b) / (a + b);
    return true;
}
```

③ • Solution 3: Throw exceptions (C++ feature)

```
float ratio(float a, float b)
{
    if (fabs(a+b) < FLT_EPSILON)
        throw "Error ...";

    return (a - b) / (a + b);
}
```

```
try{
    z = ratio(x,y);
    std::cout << z << std::endl;
}
catch(const char * msg)
{
    std::cerr << msg << std::endl;
}
```

Handling Exceptions

```
float ratio(float a, float b)
{
    if (a < 0)
        throw 1;
    if (b < 0)
        throw 2;
    if (fabs(a+b) < FLT_EPSILON)
        throw "Error ...";

    return (a - b) / (a + b);
}
```

```
try{
    z = ratio(x,y);
}
catch(const char * msg)
{...}
catch(int eid)
{...}
```

try块后可接多个catch
内含有可能出现exception的语句.
C++中throw可以是任意对象,不一定要是std::exception子类.

Stack Unwinding (堆栈解退)

```
float ratio(float a, float b)
{
    if (a < 0)
        throw 1;
    if (b < 0)
        throw 2;
    if (fabs(a+b) < FLT_EPSILON)
        throw "Error ...";
    return (a - b) / (a + b);
}
float ratio_wrapper(float a, float b)
{
    try{
        return ratio(a, b);
    }
    catch(int eid){...}
    return 0;
}
```

error5.cp

```
try{
    z = ratio_wrapper(x,y);
}
catch(const char * msg)
{...}
```

throw异常后,将由上一级调用者处理异常,
若有异常未处理,继续抛至上一级,直至main()
若直到main()都仍有异常未处理,会报错并中止程序

Catch-all Handler

```
int main()
{
    runSomething1();
    try
    {
        runSomething2();
    }
    runSomeOthers();

    catch(...)
    {
        std::cerr << "Unrecognized Exception" << std::endl;
    }
    return 0;
}
```

三个点
↓
catch(...)能catch所有异常.

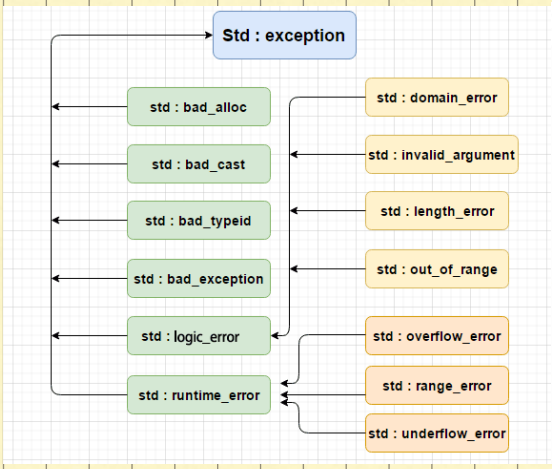
Exceptions and Inheritance

```
try
{
    throw Derived();
}
catch (const Base& base)
{
    std::cerr << "I caught Base." << std::endl;
}
catch (const Derived& derived)
{ // never reach here
    std::cerr << "I caught Derived." << std::endl;
}
```

若抛出的是某类的子类,且在catch中同时有catch该类和子类,则会进入前一个的catch块,后一个不再进入.

所以一般将子类放前面

std::exception：所有异常、错误的基类.



下表是对上面层次结构中出现的每个异常的说明：

异常	描述
std::exception	该异常是所有标准 C++ 异常的父类。
std::bad_alloc	该异常可以通过 **new** 抛出。
std::bad_cast	该异常可以通过 **dynamic_cast** 抛出。
std::bad_exception	这在处理 C++ 程序中无法预期的异常时非常有用。
std::bad_typeid	该异常可以通过 **typeid** 抛出。
std::logic_error	理论上可以通过读取代码来检测到的异常。
std::domain_error	当使用了一个无效的数学域时，会抛出该异常。
std::invalid_argument	当使用了无效的参数时，会抛出该异常。
std::length_error	当创建了太长的 std::string 时，会抛出该异常。
std::out_of_range	该异常可以通过方法抛出，例如 std::vector 和 std::bitset<>::operator[]()。
std::runtime_error	理论上不可以通过读取代码来检测到的异常。
std::overflow_error	当发生数学上溢时，会抛出该异常。
std::range_error	当尝试存储超出范围的值时，会抛出该异常。
std::underflow_error	当发生数学下溢时，会抛出该异常。

```
namespace std {
    class logic_error;
    class domain_error;
    class invalid_argument;
    class length_error;
    class out_of_range;
    class runtime_error;
    class range_error;
    class overflow_error;
    class underflow_error;
}
```

Class std::logic_error

```
namespace std {
  class logic_error : public exception {
  public:
    explicit logic_error(const string& what_arg);
    explicit logic_error(const char* what_arg);
  };
}
```

Class std::domain_error

```
namespace std {
  class domain_error : public logic_error {
  public:
    explicit domain_error(const string& what_arg);
    explicit domain_error(const char* what_arg);
  };
}
```

std::exception::what() 返回 C-style String 信息，可被重写/重载 (virtual function)

Exception Specification & noexcept

- The noexcept specifier defines a function which will not throw anything.

```
void foo() noexcept; // this function is non-throwing
```

保证该函数不会抛出异常.

new(nothrow)

- std::nothrow is a constant to select a non-throwing allocation function

```
int * p = NULL;


try {// may throw an exception
    p = new int[length];
}
catch (std::bad_alloc & ba)
{
    cerr << ba.what() << endl;
}

// not throw an exception
p = new(nothrow) int[length];
if(p==NULL)
{ ... }
```