南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

CS205

Prof. Shiqi Yu (于仕琪)

yusq@sustech.edu.cn

http://faculty.sustech.edu.cn/yusq/

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Standard Output Stream and Standard Error Stream

stdin, stdout, stderr

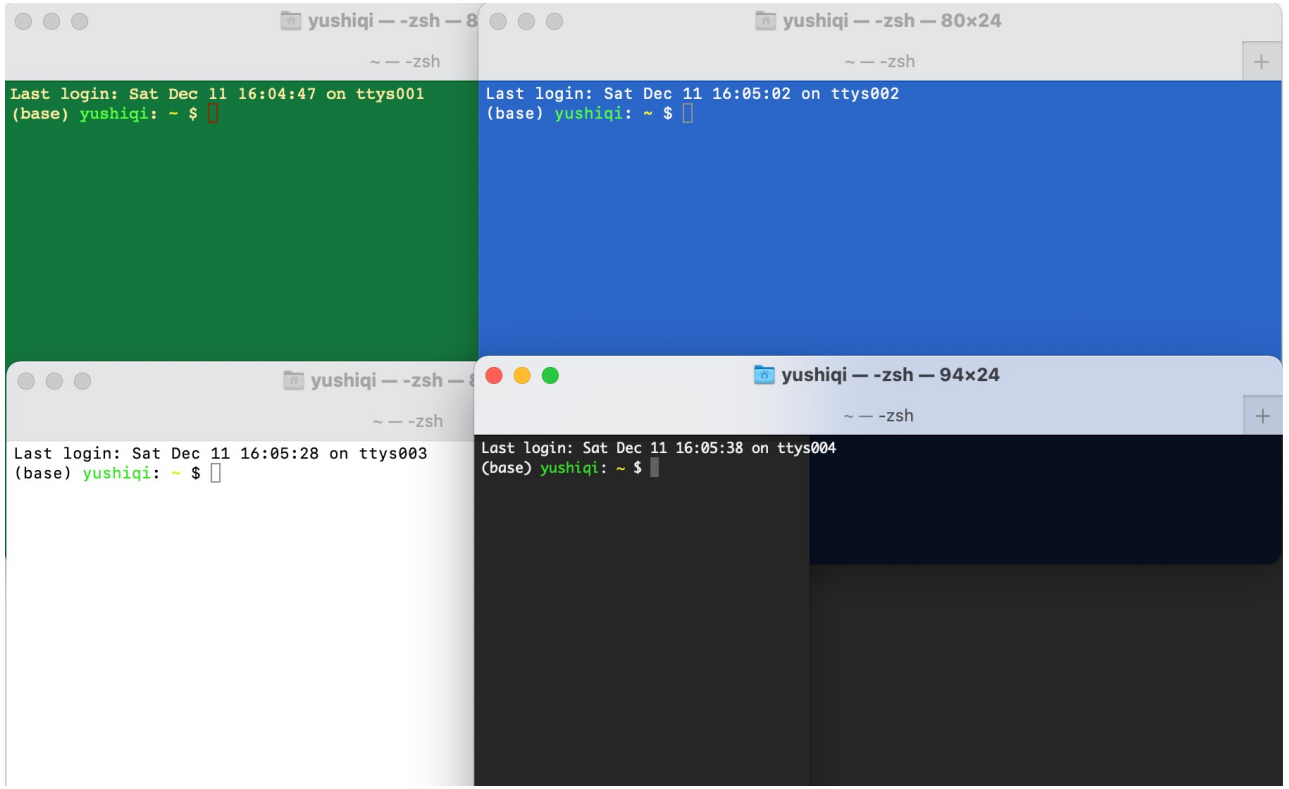
- In C, three text streams are predefined, and their type is (`FILE *`).
- `stdin`: standard input stream
- `stdout`: standard output stream, for conventional output
- `stderr`: standard error stream, for diagnostic output.

- Why do we need the "ugly" black command windows?

Command-line Interface

- The ONLY interface in the past.

The end of the HELP command output from RT-11SJ displayed on a VT100.



https://en.wikipedia.org/wiki/Command-line_interface

But We are in the 21st Centaury

- We still need them!
  - Many computers still have no GUI: severs, intelligent devices
  - Many programs do not provide GUI: HTTP servers, DB servers, ...

Output Stream and Error Stream

- Send contents into streams in C and C++

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

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

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

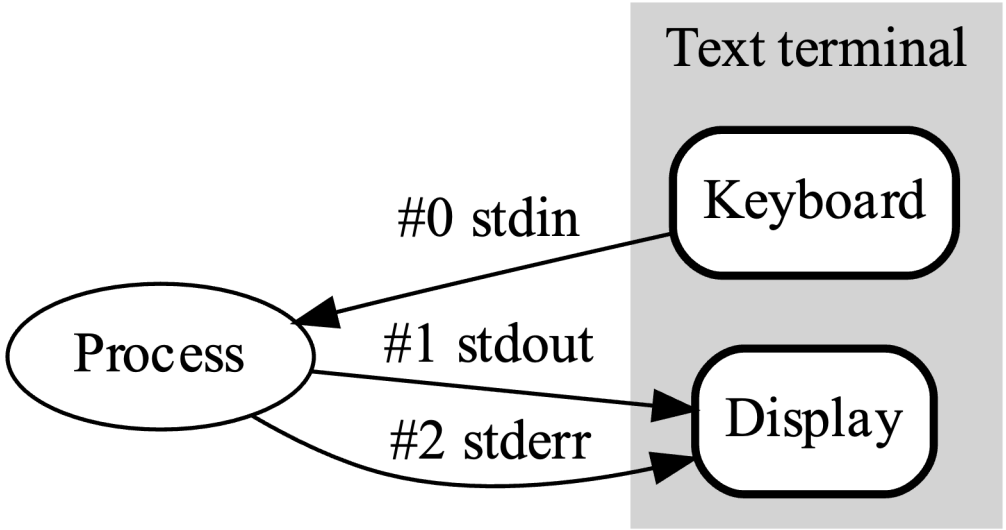

stderr.c stderr.cpp

Redirection

- 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
```

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



https://en.wikipedia.org/wiki/Standard_streams

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

assert

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.

`assert.cpp`

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
               )
```

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Exceptions

Error Handling

- 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);
}
```

- A good solution?
- If not, how to tell the caller?

error1.cpp

Error Handling

- 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;
}
```


error2.cpp

Error Handling

- 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;
}
```


error3.cpp

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

More About Exceptions

Handling Exceptions

- A try block can be followed by multiple catch blocks.

```
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)
{...}
```


error4.cpp

Stack Unwinding

- If an exception is not handled in the function, throw it to the caller.
- If the caller does not handle, throw it to the caller of the caller, or until main()

```
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;
}
```

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

error5.cpp

Catch-all Handler

- If an exception is not caught, it will reach to the top caller, and terminate the program😱
- A catch-all handler can catch all kinds of exceptions.

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

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

Exceptions and Inheritance

- If an object is thrown, and its class is derived from another class.
- An exception handler with the base class type can catch the exception.

```
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;
}
```

derived.cpp

std::exception

- `std::exception` is a class that can be a base class for any exception.
- Function `std:exception::what()` can be overridden to return a C-style string message.

```
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);
  };
}
```

Exception Specifications and `noexcept`

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

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

nothrow new

- 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)
{ ... }
```

nothrow.cpp