Cygwin

安装设置相关

安装cygwin及和g++

安装cygwin,我选择devel所有，把所有的开发工具都安装了。

cygwin中g++版本的区别

```
    cygwin32-gcc-g++ is a compiler for 32 bit cygwin
    gcc-g++ is the basic 64 bit compiler (you probably must install this one).
    mingw-gcc-g++ is a 32 bit compiler for native 32 bit Windows
    mingw64-x86_64-gcc-g++ is a 64 bit compiler for native 64 bit Windows
    mingw64-i686-gcc-g++ is a 64 bit compiler for native 32 bit Windows
```

安装完以后，运行`cygcheck -c`检查下，然后再`g++ --version`查看下版本。

see: GCC and Make
Compiling, Linking and Building
C/C++ Application

安装版本 2.877 64位

g++ -maix64

设置技巧

see: Setting up to compile C++ on Windows

先安装cygwin，然后新建一个`gccp.bat`，如下

```
        @echo off
        echo compiling C++ using -ansi -pedantic-errors -Wall
        g++ -ansi -pedantic-errors -Wall %1 %2 %3
```

编译的时候`gccp temp.cpp`（使用`gcc+`也可以，但是使用那个bat文件更规范化）

编译一般性说明

- 头文件
  - 头文件一般为`.h`，在编译c++文件时候用到
  - 主要作用：以后深入学习
  - 编译器在编译源文件的时候需要用到头文件
  - 编译器在include-path里寻找头文件，编译的时候通过`-Idir`指出（或者环境变量CPATH）
- 库文件
  - 连接器需要将库文件夹和编译以后文件变成可执行文件
  - 库文件在`-Ldir`中指出(或者环境变量`LIBRARY_PATH`)
- 连接器不仅需要知道路径，还需要知道库的名字。
  - 静态库：`.a`（unix） 或者`.lib`（window）
  - 动态库:`.so` (unix) 或者 `.dll` （window）
- 环境变量

```
- PATH: For searching the executables and run-time shared libraries (.dll, .so).

- CPATH: For searching the include-paths for headers. It is searched after paths specified in -I<dir> options. C_INCLUDE_PATH and CPLUS_INCLUDE_PATH can be used to specify C and C++ headers if the

- LIBRARY_PATH: For searching library-paths for link libraries. It is searched after paths specified in -L<dir> options.
```

可以通过系统echo来显示环境变量, 比如`echo $HOME`显示HOME环境变量。

cygwin的一些工具

- cygcheck：比如缺少了某库，导致程序无法运行，则可以使用此工具，去检测到底问题出在哪里。
- cygpath：实现Linux/Unix和Windows之间的路径转换。

常用的linux命令

- pwd 显示当前路径
- env 显示所有环境变量，`echo $HOME`显示HOME环境变量
- ls 列出当前目录下的文件
- rmdir 删除目录
- rm 栓除文件
- mv 文件改名或者目录改名
- man 联机帮助 e.g.`man ls`
- less 显示文件最后几行

g++实践

运行一个简单的程序

假设源程序为`prog1.cpp`,首先利用cygpath
将windows的路径转换成linux路径，然后cd到该目录下

```
    cygpath -au 'd:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\cpp_primer\chap1\'

    cd /cygdrive/d/mdeng/WORK/MyCode/CodePratices/workSpaceCppGcc/cpp_primer/chap1/
```

实际上直接加上单引号就可以了

```
cd 'd:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\cpp_primer\chap1\'
```

然后编译

```
 g++ -o hello.exe prog1.cpp
   // Compile and link source hello.cpp into executable hello.exe
 hello
   // Execute under CMD shell
$ ./hello
   // Execute under Bash or Bourne shell, specifying the current path (./) 在cygwin的命令窗口里输入这个
```

如果输入

```
g++ -Wall -g -o Hello.exe prog1.cpp
```

会输出警告信息（Wall），以及额外的符号调试信息。

如果将编译和连接分开进行：

```
    // Compile-only with -c option
     g++ -c -Wall -g prog1.cpp //生成了.o文件
     // Link object file(s) into an executable
     g++ -g -o Hello.exe prog1.o
```

如果要查看编译的详细信息，加上`-v`

```
g++ -v prog1.cpp -o hello.exe
```

include path

假设有`Circle.h`和`Circle.cpp`，cpp文件调用了h头文件，
如果两者位于同一文件夹，且引用的方式为`#include "Circle.h"`，则编译的时候不需要指明include路径。否则需要指定包含路径，实现方式有如下几种：

1. `-I`命令行指定法
2. 放在g++系统的include path里
3. 环境变量法

方式1

```
g++  -I 'd:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testIncludePatah\'  -c Circle.cpp
```

方式2：首先看系统有哪些include path,参考

```
g++ -E -x c++ - -v < /dev/null
```

然后放到include path里，然后运行`g++ -c Circle.cpp`,这种方式不推荐，仅测试用。

设置环境变量法

官方说明

设置方法参考

```
    CPLUS_INCLUDE_PATH='d:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testIncludePatah\'
    export CPLUS_INCLUDE_PATH
```

然后运行`g++ -c Circle.cpp`

几点说明和注意

- 设置的等号后面不能有空格
- export那行命令不可缺少
- 设置完以后可以运行`g++ -E -x c++ - -v < /dev/null`检查下是否设置好
- 设置完以后重启以后环境变量就没了，要想重启以后还有的话就需要将上面的两行命令放到`.bash_profile`文件里
- 清除include path的话，使用`CPLUS_INCLUDE_PATH=''`，要将多个目录添加为include路径应该是用冒号隔开(unix路径)，比如：

```
LIBRARY_PATH=/usr/lib/gcc/x86_64-pc-cygwin/5.4.0/:/usr/lib/gcc/x86_64-pc-cygwin/5.4.0/../../../../x86_64-pc-cygwin/lib/../lib/:/usr/lib/gcc/x86_64-pc-cygwin/5.4.0/../../../../lib/:/lib/../lib/:/usr/lib/
```

- 搜索顺序：首先搜索`-I`指定路径，然后搜索`CPLUS_INCLUDE_PATH`环境变量的路径，最后搜索系统自带的路径：
- 一般如果是下载的封装好的库文件，可以修改`.bash_profile`进行永久性设置，但是如果在g++里再设置会覆盖`.bash_profile`的设置。

生成库

生成静态库

1. 接上文，首先先将cpp编译成`.o`文件

```
g++  -I 'd:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testIncludePatah\'  -c Circle.cpp
```

2. 然后利用`ar`工具打包成静态库

```
ar -crv libCircle.a Circle.o
```

这样就生成了静态库`.a`文件。人为的将其复制到别的目录测试。比如说：

```
d:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testLibPath\
```

参考资料

生成动态库

```
g++ -fPIC   -c Circle.cpp  #生成.o文件
g++ -shared -o libCircle.so Circle.o  # 打包成动态库.so
```

`-fPIC`创建与地址无关的编译程序（pic，position independent code），是为了能够在多个应用程序间共享。

参考资料

调用库

主程序为`TestCircle.cpp`为主程序调用库

```
    /* A test driver for the Circle class (TestCircle.cpp) */
    #include <iostream>
    #include "Circle.h"   // using Circle class
    using namespace std;

    int main() {
       // Construct an instance of Circle c1
       Circle c1(1.2, "red");
       cout << "Radius=" << c1.getRadius() << " Area=" << c1.getArea()
            << " Color=" << c1.getColor() << endl;

       c1.setRadius(2.1);   // Change radius and color of c1
       c1.setColor("blue");
       cout << "Radius=" << c1.getRadius() << " Area=" << c1.getArea()
            << " Color=" << c1.getColor() << endl;

       // Construct another instance using the default constructor
       Circle c2;
       cout << "Radius=" << c2.getRadius() << " Area=" << c2.getArea()
            << " Color=" << c2.getColor()  << endl;
       return 0;
    }
```

库分为静态库(`.a`或`lib`)和动态库（共享库，`.so`或`.dll`）。不管是调用那种库都有三种调用方式:

1. `g++`命令后面直接给出库的路径和名称
2. `g++ -L后面接库路径，然后接库名`
3. 将库所在路径添加到环境变量，假如库名为libCab.a，那么添加完环境变量以后，`g++ -lCab.....`。（这种方法仅适用于库名是以lib开头的情况）

首先生成4个库，1个静态库1个动态库，两个库都采用标准命名形式以lib开头。

```
CPLUS_INCLUDE_PATH='d:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testIncludePatah\'
    export CPLUS_INCLUDE_PATH
    g++ -c Circle.cpp
    ar -crv libCircle.a Circle.o
    g++ -shared -o libCircle.so Circle.o  # 打包成动态库.so
```

动态库放入`d:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testLibPath\dynamic\`；

静态库放入`d:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testLibPath\static\`

查看不指明库名出现的错误

首先不指明库文件进行编译

```
g++  -I 'd:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testIncludePatah\'  -o TestCircle.exe TestCircle.cpp
```

会出错，`对‘Circle::Circle(double, std::string)’未定义的引用`，因此如果有类似的错误，可能说明系统没有找到库文件。

调用静态库

库文件不仅要指出路径还要指出库文件名，首先给出最直观的编译方式。下面两种写法都是正确的。第一种写法库文件的路径给出的是windows系统系统的路径，第二种写法库文件的路径是在linux中给出的。

上文中已经定义了环境变量`CPLUS_INCLUDE_PATH`，这里不需要再指明include path。

```
# 路径格式为window路径
g++  TestCircle.cpp 'd:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testLibPath\static\libCircle.a'  -o TestCircle.exe
# 路径格式为linux路径
g++   TestCircle.cpp /cygdrive/d/mdeng/WORK/MyCode/CodePratices/workSpaceCppGcc/ehchua-cpp/testLibPath/static/libCircle.a -o TestCircle.exe
```

具体说明如下：

2. `TestCircle.cpp` 源文件名
3. `'d:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testLibPath\libCircle.a'` 库文件名和路径
4. `-o TestCircle.exe` 编译生成exe文件

注意 cpp文件不能放在库文件后面，比如下面的写法编译不通过

```
g++  -I 'd:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testIncludePatah\' -L 'd:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testLibPath\Circle.a'  -o TestCircle.exe TestCir
```

设置环境变量调用静态库的过程

```
LIBRARY_PATH=/cygdrive/d/mdeng/WORK/MyCode/CodePratices/workSpaceCppGcc/ehchua-cpp/testLibPath/static
export LIBRARY_PATH
g++  TestCircle.cpp -lCircle -o TestCircle.exe
```

说明如下

- `LIBRARY_PATH` 为静态库路径的环境变量
- `-lCircle` 中 `-l` 表示连接的意思， `Circle` 表示连接的是 `libCircl.a`
- `-lCircle` 中不能有后缀名，否则编译通不过

动态库的调用

和静态库的调用类似，这里给出环境变量法调用方法：

```
g++  TestCircle.cpp 'd:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testLibPath\dynamic\libCircle.so'  -o TestCircle.exe
```

生成exe完以后需要把库拷贝到exe所在的文件夹才能运行。

下面采用环境变量法

```
LD_LIBRARY_PATH=/cygdrive/d/mdeng/WORK/MyCode/CodePratices/workSpaceCppGcc/ehchua-cpp/testLibPath/dynamic
export LD_LIBRARY_PATH
g++  TestCircle.cpp -lCircle -o TestCircle.exe
```

这种方法不需要将 `.so` 库拷贝到exe所在文件夹内运行exe。

g++ 命令总结

> 查看include path和library path

```
g++ -E -x c++ - -v < /dev/null
```

> 指定include path

```
g++  -I 'd:\mdeng\WORK\MyCode\CodePratices\workSpaceCppGcc\ehchua-cpp\testIncludePatah\'
```

> 打包静态库 (下面的命令能正常运行的前提是设置好了include path)

```
g++    -c Circle.cpp  #生成.o文件
ar -crv Circle.a Circle.o  # 打包成静态库.a
```

> 查看exe文件需要的动态库

`ldd hello.exe`

```
      ntdll.dll => /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll (0x7ffb4e260000)
        KERNEL32.DLL => /cygdrive/c/WINDOWS/System32/KERNEL32.DLL (0x7ffb4c440000)
        KERNELBASE.dll => /cygdrive/c/WINDOWS/System32/KERNELBASE.dll (0x7ffb4a7f0000)
        cygwin1.dll => /usr/bin/cygwin1.dll (0x180040000)
        cygstdc++-6.dll => /usr/bin/cygstdc++-6.dll (0x3d6a20000)
        cyggcc_s-seh-1.dll => /usr/bin/cyggcc_s-seh-1.dll (0x3e6f70000)
```

参考资料

Cygwin详解