关于今天课程的释疑

1. 为什么要将gcc所在的目录添加到Windows的搜索路径中？

在C-Free或者Dev-Cpp中，这两个IDE会自动找到gcc在哪里，请使用它来编译程序。但如果我们是在控制台中使用，那么就脱离了IDE环境，系统不知道gcc在哪里，从而产生找不到命令的错误提示。

为了让系统能够找到gcc，我们就将那两个IDE自带的gcc手工添加到系统的所有路径中。系统的搜索路径在环境变量PATH中存放。因此，我们找到PATH变量，将gcc的路径加进去。这样，就可以在控制台中直接使用gcc了。

如果大家使用的Linux，那么在安装gcc时，系统已经将其加入到搜索路径中了。

*注：以C-Free为例，gcc是安装在C:\Program Files(X86)\C-Free 5\mingw\bin这个目录中的。你的C-Free也有可能安装在其他盘上。为了看到C-Free的安装目录，你可以在它的图标上按右键，选择“属性”就可以看到了。*

1. 为什么要使用make工具？

对于很多的应用，是由多个源代码组成的。这些源代码会被分别编译，然后在由链接器组装在一起。

使用IDE的绿色三角按钮或使用gcc main.c这样的命令只能一次编译一个源文件，这使得链接器找不到其他的所需的代码，因此会报错。

为解决这个问题，就要使用make工具。这个工具可以用半自动化的方式生成应用。今后大家可以看到很多的例子。

使用make工具，需要为它提供一个依赖文件。这个文件中，写明了应用依赖的源文件，以及其他一些相关规则。我们来看一个make依赖文件的样本Makefile（这个是make工具的默认依赖文件）。

CC = gcc

flags = -Wall -g

sources = main.c list.c

all:

$(CC) $(flags) $(sources)

其中，前三行是变量定义，分别定义了使用的编译器、编译选项和源文件列表。

all:这个标号定义了一个任务(task)，又称为recipe。其下面的一行（必须用tab缩进，不能是空格），写明了make规则。在样本里，这条规则的含义是：启动编译器$(CC)，也就是gcc，使用编译选项$(flags)，分别编译$(sources)列表里面的所有源代码，然后启动链接器，将生成的中间代码组装在一起，生成最终的可执行代码。这个代码在Windows下，默认名为a.exe，在Linux下为a.out。执行可执行代码，就可以看到结果。

使用make工具生成可执行代码的过程一般称为“构建(build)”。

*注：Windows下gcc的make工具名字叫：mingw32-make，可能会有一个别名，就叫做make。如果没有，那么可以将mingw32-make.exe复制一份，然后将其改名为make.exe。这个工具和gcc安装在相同的目录中。*

1. Make的依赖文件必须是Makefile吗？

当然不是。你可以为依赖文件去任意一个名字，例如：list.mk。在使用make时，可以用这样的方式：

make –f list.mk

1. 可以不使用make来构建多源代码应用吗？

当然可以，不过需要使用IDE。一般的IDE，例如C-Free、Dev-Cpp或者Visual Studio都用一种措施，称为“工程(project)”。在使用时，首先要创建一个空的工程，然后在工程中添加源代码。注意，多个源代码中，有且仅有一个源代码含有main()函数。最后，就是启动构建工程的动作，IDE会自动编译、链接所有的源代码并生成最终的可执行代码。

今天课上的内容，是为了让同学们了解到构建工程的多种做法。我推荐使用make，这会为今后的学习提供帮助。

实际上，我也建议大家在编译单个源代码应用时，不使用IDE（虽然这较快），而是手动使用gcc来完成任务。这也是为了今后做准备。