大程序

多个.c文件

- main()里的代码太长了适合分成几个函数
- 一个源代码文件太长了适合分成几个文件
- 两个独立的源代码文件不能编译形成可执行的程序

编译单元

- 一个.c文件是一个编译单元
- 编译器每次编译只处理一个编译单元

项目

- 在Dev C++中新建一个项目，然后把几个源代码文件加入进去
- 对于项目，Dev C++的编译会把一个项目中所有的源代码文件都编译后，链接起来
- 有的IDE有分开的编译和构建两个按钮，前者是对单个源代码文件编译，后者是对整个项目做链接

头文件

函数原型

- 如果不给出函数原型，编译器会猜测你所调用的函数的所有参数都是int，返回类型也是int
- 编译器在编译的时候只看当前的一个编译单元，它不会去看同一个项目中的其他编译单元以找出那个函数的原型
- 如果你的函数并非如此，程序链接的时候不会出错
- 但是执行的时候就不对了
- 所以需要在调用函数的地方给出函数的原型，以告诉编译器那个函数究竟长什么样

头文件

- 把函数原型放到一个头文件（以.h结尾）中，在需要调用这个函数的源代码文件（.c文件）中#include这个头文件，就能让编译器在编译的时候知道函数的原型

#include

- #include是一个编译预处理指令，和宏一样，在编译之前就处理了
- 它把那个文件的全部文本内容原封不动地插入到它所在的地方
  - 所以也不是一定要在.c文件的最前面#include

“”还是<>

- #include有两种形式来指出要插入的文件
  - “”要求编译器首先在当前目录（.c文件所在的目录）寻找这个文件，如果没有，到编译器指定的目录去找
  - <>让编译器只在指定的目录去找
- 编译器自己知道自己的标准库的头文件在哪里
- 环境变量和编译器命令行参数也可以指定寻找头文件的目录

#include的误区

- #include不是用来引入库的
- stdio.h里只有printf的原型，printf的代码在另外的地方，某个.lib(Windows)或.a(Unix)中
- 现在的C语言编译器默认会引入所有的标准库
- #include <stdio.h>只是为了让编译器知道printf函数的原型，保证你调用时给出的参数值是正确的类型

头文件

- 在使用和定义这个函数的地方都应该#include这个头文件
- 一般的做法就是任何.c都有对应的同名的.h，把所有对外公开的函数的原型和全局变量的声明都放进去

不对外公开的函数

- 在函数前面加上static就使得它成为只能在所在的编译单元中被使用的函数
- 在全局变量前面加上static就使得它成为只能在所在的编译单元中被使用的全局变量

声明

变量的声明

- int i;是变量的定义
- extern int i;是变量的声明

声明和定义

- 声明是不产生代码的东西
  - 函数原型
  - 变量声明
  - 结构声明
  - 宏声明
  - 枚举声明
  - 类型声明
  - inline函数
- 定义是产生代码的东西

头文件

- 只有声明可以被放在头文件中
  - 是规则不是法律
- 否则会造成一个项目中多个编译单元里有重名的实体
  - * 某些编译器允许几个编译单元中存在同名的函数，或者用weak修饰符来强调这种存在

重复声明

- 同一个编译单元里，同名的结构不能被重复声明
- 如果你的头文件里有结构的声明，很难这个头文件不会在一个编译单元里被#include多次
- 所以需要“标准头文件结构”

标准头文件结构

```
#ifndef __LIST_HEAD__
#define __LIST_HEAD__

#include "node.h"

typedef struct _list {
    Node* head;
    Node* tail;
} List;

#endif
```

- 运用条件编译和宏，保证这个头文件在一个编译单元中只会被#include一次
- #pragma once也能起到相同的作用，但是不是所有的编译器都支持

* 前向声明

```
#ifndef __LIST_HEAD__
#define __LIST_HEAD__

struct Node;

typedef struct _list {
    struct Node* head;
    struct Node* tail;
} List;

#endif
```

- 因为在这个地方不需要具体知道Node是怎样的，所以可以用struct Node来告诉编译器Node是一个结构