No: Date:

W5. 2020.3.23. 程序结构.

○ 全局变量.

- 定义在所有函数外面的变量是全局变量.
- 全局变量具有全局的寿命存在期和作用域
- 与任何函数都无关
- 在任何函数内部都可以使用.

```
int a = 10;
int main(){
    printf("%d", a);
}
```
运用结果输出10.

○ 全局变量的初始化. (作用域大于所有函数).

- 没有明确初始化的全局变量会得到0值.
- 指针会得到NULL值.
- 只能用编译时刻已知的值来初始化全局变量.
- 初始化发生在main函数之前

```
int gAll=12;
int g2=gAll;   ×
```

```
const int gAll=12;
int g2=gAll;   ✓
```
(虽然可以但不建议这么做).

○ 被隐藏的全局变量.

- 如果函数内部存在与全局变量同名的变量，则全局变量被屏蔽

○ 静态本地变量.

- 在本地变量定义时加上 static 修饰符就成为静态本地变量.
- 当函数离开时，静态本地变量继续存在并保持其值.
- ☆ 静态本地变量的初始化只会在第一次进入这个函数时做，之后进入函数时会保持上次离开的值.
- 是特殊的全局变量
- 它们位于相同的内存区域
- 全局的生存期，函数内的局部作用域
- static 意为局部作用域 (本地可访问)

○ 返回指针的函数

- 返回本地变量的地址是危险的
- △ 返回全局变量或静态本地变量的地址是安全的.

No: Date:

- 返回在函数内malloc的内存是安全的（但易造成问题）
- 最好的传递法是返回传入的指针

o tips.

- 不要使用全局变量在函数间传递参数和结果
- 尽量避免使用全局变量
- △ 使用全局变量和静态本地变量的函数是线程不安全的

o 编译预处理指令

- #开头的是编译预处理指令（不是C语言的成分，但是C语言程序需开启的）
- #define 定义一个宏 #define <名字> <值> 注意结尾没有分号

完全的文本替换

o 宏的注意点：

- 如果一个宏的值中有其他宏的名字，也会被替换
- 如果一个宏的值超过一行，最后一行之前的行末需要加 \
- 宏的值后面出现的注释不会被当作宏的一部分

o 带参数的宏

- #define cube(x) ((x)*(x)*(x)) 宏可以带参数
- 一切都要括号（整个值，参数出现的地方） #define RADTODEG(x) ((x)*57.29578)
- 可以带多个参数，也可以组合（嵌套）使用其他宏
- 在大程序代码中使用非常普遍

错误演示例：

#define RADTODEG(x) (x*57.29578)

#define RADTODEG(x) (x)*57.29578

o 其他编译预处理指令

- 条件编译
- error ……

No: Date:

- #include

  - #include 是一个编译预处理指令，我们一样在编译之前就处理了。
  - " " 要求编译器首先到在当前目录（.c文件所在目录）寻找这个文件，
    如果没有，到指定目录去找。
    < > 让编译器只在指定的目录去找。
  - 把包含那个文本文件的全部文本内容原封不动地插入到它所在的地方
    （所以也不是一定在.c文件的最前面#include）

- 大程序

  - 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，就可以让编译器在编译时知道函数的原型。
    （在使用和定义这个函数的地方都应该#include这个头文件）
  - 在编函数前面加上static，使它成为只能在所在编译单元中被使用的函数
  - 在全局变量前面加上static就使它成为只能在所在编译单元使用的全局变量。
    （不对外公开的函数）
  - int i; → 变量的定义
    extern int i; → 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……）
    定义是产生代码的东西。
  - 头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有重名的实体。
    （某些编译器允许一个编译单元中存在同名函数，
    或者用weak修饰符来强调这种存在）。

No: Date:

- 重复声明
  - 在同一个编译单元里，同名的结构体不能被重复声明

需用到"标准头文件结构"：

```
#ifndef _LIST_HEAD_
#define _LIST_HEAD_
#include "node.h"
typedef struct _list {
    Node * head;
    Node * list;
} List;
#endif
```

← struct Node; 前向声明，告诉编译器 Node 是一个结构体

条件编译预处理，保证包含文件在一个编译单元中只会被 #include 一次