函数和作用域（2）

中山大学计算机学院

讲课人：潘茂林

目录

CONTENTS

递归（recursive）的概念

n 阶乘的两种定义：

$$n! = n * (n-1) * (n-2) * \cdots * 1$$

或

$$n! = \begin{cases} 1 & where\ n = 1 \\ n * (n-1)!\ & where\ n > 1 \end{cases}$$

前面的是**迭代**定义可以看作用一个循环去计算 n 阶乘的方法
后面的是**递归**定义是用一个分段函数表示，它包括两个部分：

- 基本情况（bottom cases），基本的事实
- 递推关系（recurrence relation），通过递推分解，最终递归到基本情况，给出问题解

递归是一种问题求解的方法，在递归过程中，**函数将自身作为子例程调用。**

递归（recursive）的实现

计算算法

```
int FacterialLoop(int n){
    int out = 1;
    for (int i=n; i>0; i--)
        out *= i;
    return out;
}

int FacterialRecursive(int n){
    if (n <= 1)
       return 1;
    else
       return n * FacterialRecursive(n - 1);
}
```

```
/*Facterial*/
#include<stdio.h>

int FacterialLoop(int);
int FacterialRecursive(int);

int main() {
    /*请使用 -1,0,1,2,5 测试程序*/
    int n = 5;
    printf("loop out
%d\n",FacterialLoop(n));
    printf("recursive out
%d\n",FacterialRecursive(n));
}
```

递归函数

递归函数（recursive function）是一种能够调用自身的函数。
这种技术叫做递归（recursion）

```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

当然，不能无休止的调用
使用条件语句等避免无休止调用，
即满足一定条件后停止自身调用

递归函数

```c
int sum(int n) {
    if (n != 0)
        // sum() function calls itself
        return n + sum(n-1);
    else
        return n;
}

int main() {
    int number, result;

    printf("Enter a positive integer: ");
    scanf("%d", &number);

    result = sum(number);

    printf("sum = %d", result);
    return 0;
}
```

一个简单的1+2+3+...+n的递归计算函数

- 最初，从main函数调用sum函数，并传递一个参数number。
- 假设number为3，则在下一次调用sum时，传递参数2。直到n等于0。
- 当n等于0时，if条件不满足，执行else语句里的返回，直到main函数。

递归函数 – 展开与回溯

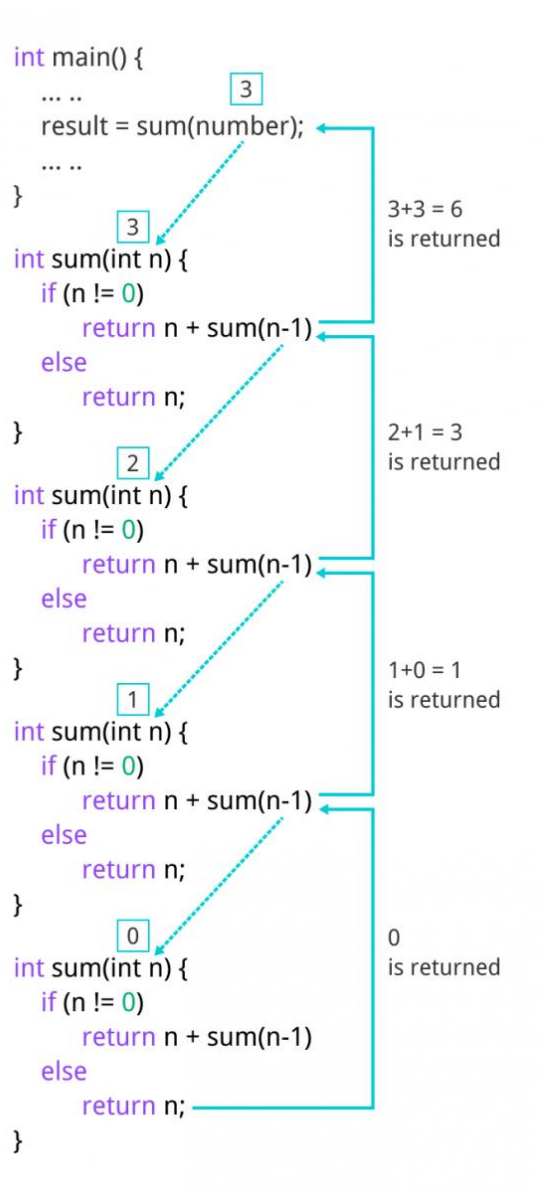
```c
int sum(int n) {
    if (n != 0)
        // sum() function calls itself
        return n + sum(n-1);
    else
        return n;
}

int main() {
    int number, result;

    printf("Enter a positive integer: ");
    scanf("%d", &number);

    result = sum(number);

    printf("sum = %d", result);
    return 0;
}
```



递归练习：小青蛙跳台阶

- 问题描述

  一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

- 问题求解

递归练习：小青蛙跳台阶

- 问题描述

  一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

- 问题求解
  - 申明问题

  int f(int n);

  - 基本事实

  f(1) = 1;

  验证 f(2) = f(1) + f(0); 应该 f(2) = 2;

  验证 f(3) = f(2) + f(1) = 3

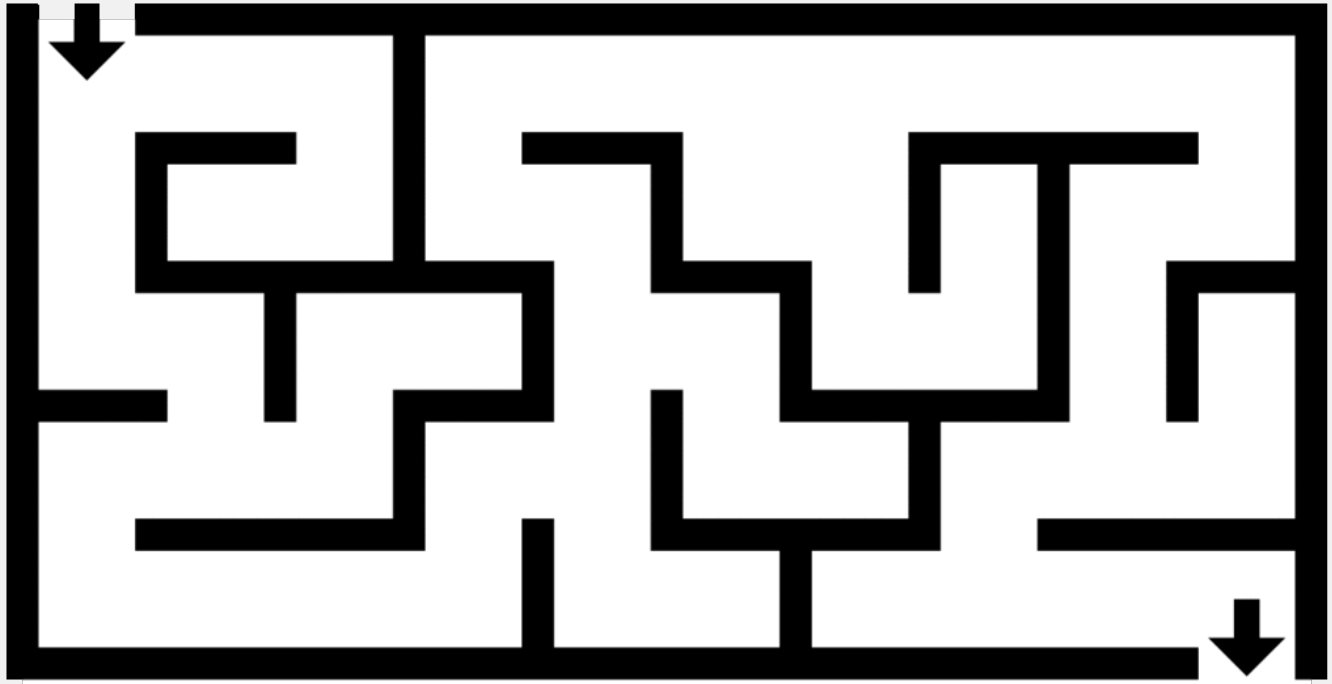
  - 分解问题

  f(n) = f(n-1) + f(n-2);

$$F(n) = \begin{cases} 1 & where\ n = 1 \\ 2 & where\ n = 2 \\ F(n-1) + F(n-2) \end{cases}$$

递归函数 – 探索迷宫

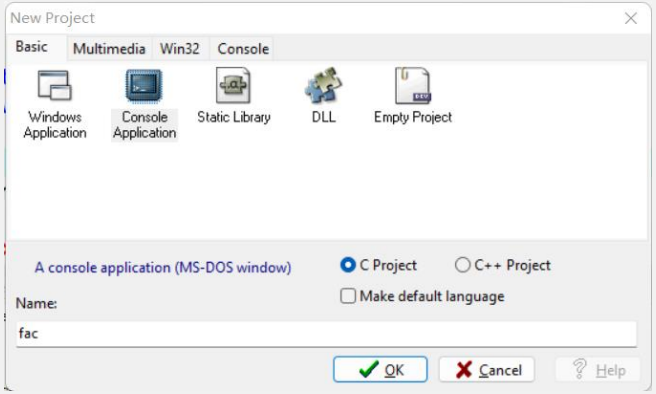
递归使程序代码优雅简洁。但是，由于调用函数的缘故，递归通常要慢得多。但是递归的概念至关重要，它经常用在数据结构和算法中，比如图的搜索与回溯



多文件编译 – 项目

项目：管理一个应用相关源代码与资源文件的集合

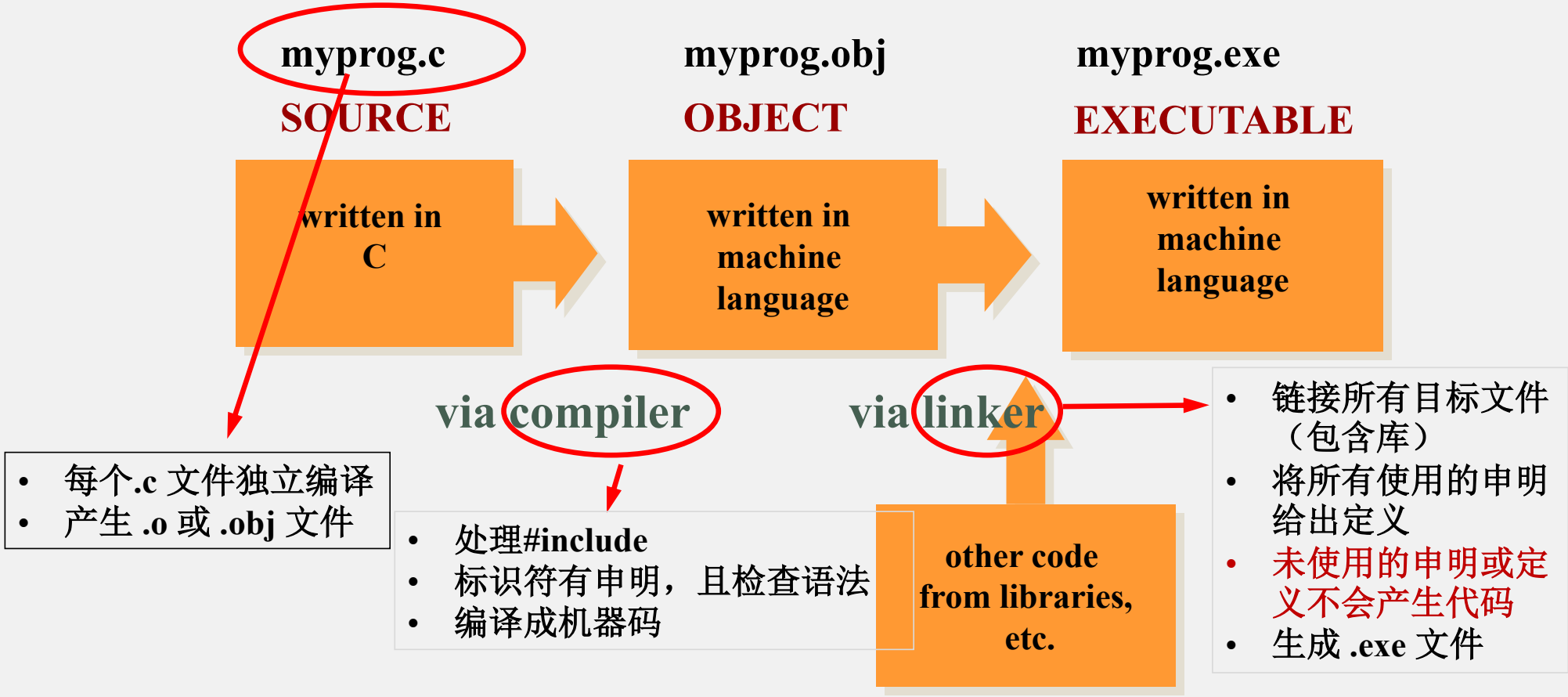
1. 创建一个新项目：
2. 添加文件 main.c 到项目
   - 按 F12 或菜单 Rebuild All
   - 产生结果
     - 编译产生 .o 文件，通过
     - 链接产生 .exe 文件，失败
       - 两个函数未定义
       - 一个错误
3. 添加文件 func.c 到项目
   - 按 F12 或菜单 Rebuild All
   - 运行，观察文件目录，产生了哪些文件




```
rm.exe -f main.o fac.exe
gcc.exe -c main.c -o main.o -I"C:/Program Files (x86)/Dev-Cpp/M
gcc.exe main.o -o fac.exe -L"C:/Program Files (x86)/Dev-Cpp/Min
main.o:main.c:(.text+0x1a): undefined reference to `FacterialLo
main.o:main.c:(.text+0x32): undefined reference to `FacterialRe
collect2.exe: error: ld returned 1 exit status
D:\pml\courses\se-xx-c\my-ppts\2022\w11\sources\Makefile.win:25
mingw32-make.exe: *** [fac.exe] Error 1

Compilation results...
--------
- Errors: 1
- Warnings: 0
- Compilation Time: 0.33s
```

Review：编译过程 – 编译 – 链接



多文件编译 – 头文件

为了确保函数**开发者**和**使用者**使用相同的申明和定义，c 语言使用头文件书写这些申明与定义，使用 **#include** 编译预处理指令将 头文件 插入即将编译的源代码中。

1. 将 main.c 修改为：

```
#include<stdio.h>
#include"fac.h"

int main() {
  …
}
```

2. 将 #include“fac.h” 添加入 func.c
3. 重新编译，运行

fac.h 头文件

```
#pragma once

int FacterialLoop(int);
int FacterialRecursive(int);
```

注意：
① 自定义的头文件使用“”号；
② 头文件第一行使用 #pragma once 指令确保编译仅展开一次

外部（external）变量

两个代码文件如何使用同一全局变量?

使用关键字 extern 申明变量。例如:
extern int ClassCount;

使用申明的意义:
- 全局变量定义由外部程序给出，**或者**
- 全局变量定义后置给出。右例输出 12

```
#include<stdio.h>

int main() {
    extern int ClassCount;
    printf("%d",ClassCount);
}

int ClassCount = 12; //定义后置
```

注意:
① 有初始化的变量申明一定是定义。
② 一个项目中，一个标识符**能仅能**定义一次。定义两次或以上，在一个源代码中是编译错误，其他代码文件中是链接错误。
③ **全局变量**申明， **extern** int ClassCount 等价于 int ClassCount。 如果全局变量未初始化，在多文件编译中，其初始值由其在其他文件中的定义决定，而不是默认零值。

静态（static）变量

如何定义仅在当前文件使用的变量?

使用关键字 static **定义**变量。例如:
static int ClassCount;

使用 static 定义的意义:

- static 修饰一定是变量定义。不能定义两次
- 没有初始值则默认零值，右例输出 0
- 静态变量**生命周期**为整个应用，离开函数值依然保留
- 静态变量是局部变量，**作用域**是一个块或一个文件
- 静态变量重名，则会**屏蔽**作用域外部的定义

```
#include<stdio.h>

int main() {
    static int ClassCount;
    printf("%d",ClassCount);
}

int ClassCount = 12; //全局定义
```

注意：本例中，静态ClassCount在main中定义是局部变量，屏蔽全局定义。如何在main外定义，则是重定义错误!

静态（static）变量 – 案例研究

最典型的应用是做一个局部的计数器

例如：

- 静态变量定义语句必须在main执行前初始化，初始化值只能是常量
- 函数退出静态变量不会销毁

```
/*static*/
#include<stdio.h>

int func(){
    static int count = 4;
    if (count) {
        printf("%d\n",count);
        return count--;
    }
    else {
        printf("you are out!\n");
        return 0;
    }
}

int main(){
    for (int i=0; i<10 ;i++)
        if (!func()) break;
}
```

Review：局部变量和全局变量

局部变量：

① 自动变量，即动态局部变量(离开函数，值就消失)。
② 静态局部变量(离开函数，值仍保留)。
③ 寄存器变量(离开函数，值就消失)。
④ 形式参数可以定义为自动变量或寄存器变量。

全局变量：

① 静态外部变量(只限本程序文件使用)。
② 外部变量(即非静态的外部变量，允许其它程序文件引用)。

注：静态变量也可以全归于局部变量

可变参函数（选讲）

如何编写向 printf 那样参数可变的函数?

尝试阅读官方文档 … …

变长实参 - cppreference.com

模块化编程基础 – 创造积木

素数是数学中的一个基础问题。*素数是 n 大于 1 且只能被除 1 和 自身整除*

判断一个数是否是素数是一个基础。让我们先写出它的函数申明

$$int\ isPrime(unsigned\ int\ n);$$

让我们先写这个函数… 参见 prime0.c，prime1.c

- 思考：prime0.c 函数体就仅一个 return 1 有价值吗?

然后用自己熟悉的素数和非素数测试一下。。。（注：prime1有BUG）

模块化编程基础 – 分解到已知问题

问题：计算 n 以内的最大素数，例如：n = 12345

问题求解：从 n 开始递减枚举，直到发现一个素数。

- 这是一个什么循环，for? while? do-while?

解法1，参见 prime2.c

优化1，请自己编写 prime3.c ,使得问题求解速度提高1倍左右。

- 如何计算才能快一倍呢?

课堂练习

Leetcode.cn，509. 斐波那契数（请使用递归法）

课后练习

Leetcode.cn，231. 2 的幂

求解过程要求

1. 题目请根据函数申明，先思考一种解法（不要看解答）
2. 自己写测试用 main 函数，用官方示例 case 测试函数
3. 阅读官方的答案
   - 请记录看了答案后的感想
   - 和同学分享或学习提升算法能力的经验，不自卑不沮丧，不断前行

中山大學
SUN YAT-SEN UNIVERSITY

谢 谢

中山大学计算机学院

编制人：课题组