程序环境和预处理

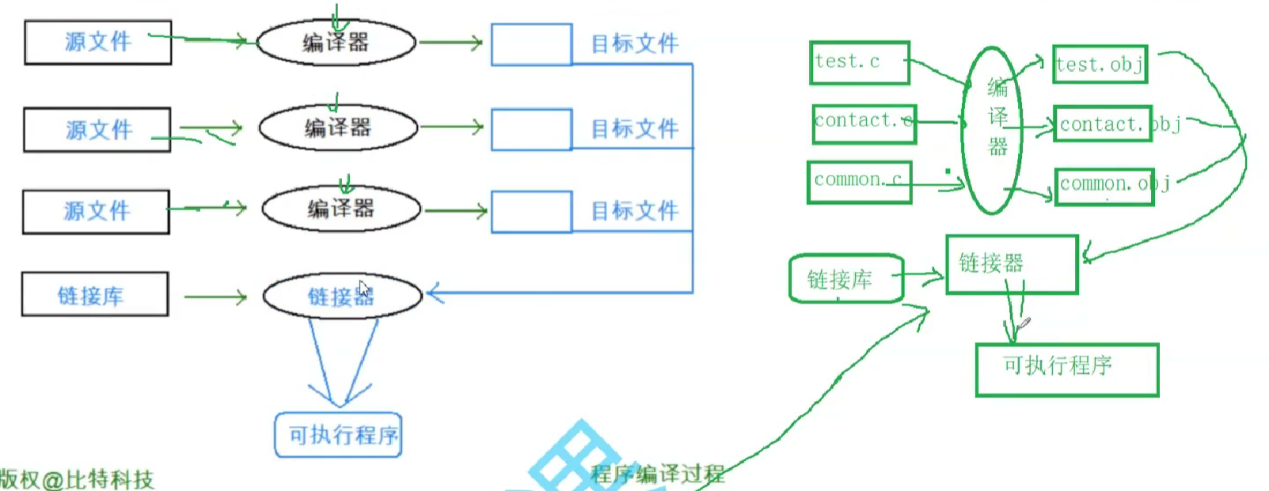
1 程序的翻译环境

任何一个标准C代码都具备两个环境，即翻译环境和执行环境。

Test.c(源程序) -> test.exe(可执行环境)即程序的翻译过程（编译+链接），依赖的是程序的翻译环境，如VS2019集成开发环境。

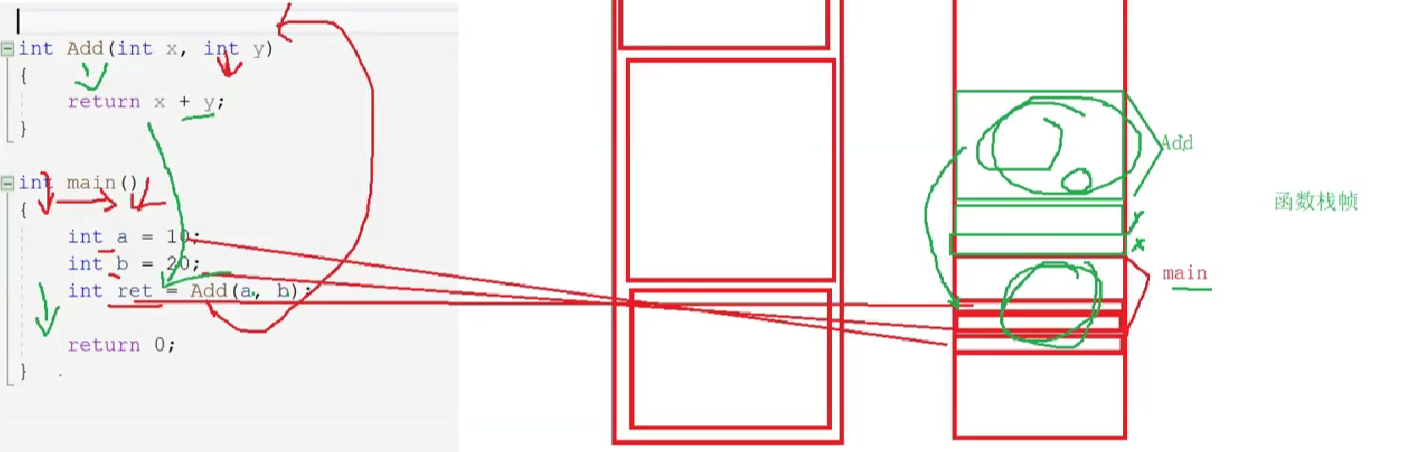


程序编译链接过程：VS中编译器cl.exe 连接器link.exe



2 程序的执行环境

执行环境是将可执行程序运行出结果通过操作系统完成的。包括：第一，将程序载入内存，如单片机烧录等；第二，从主函数开始运行；第三，开始执行程序代码，程序会使用一个运行时堆栈stack（开辟栈区），存储函数的局部变量和返回地址（函数栈帧）。程序同时也可以使用静态内存，存储于静态内存中的变量在程序的整个执行过程一直保持它的值；第四，程序完成结束或者异常结束。

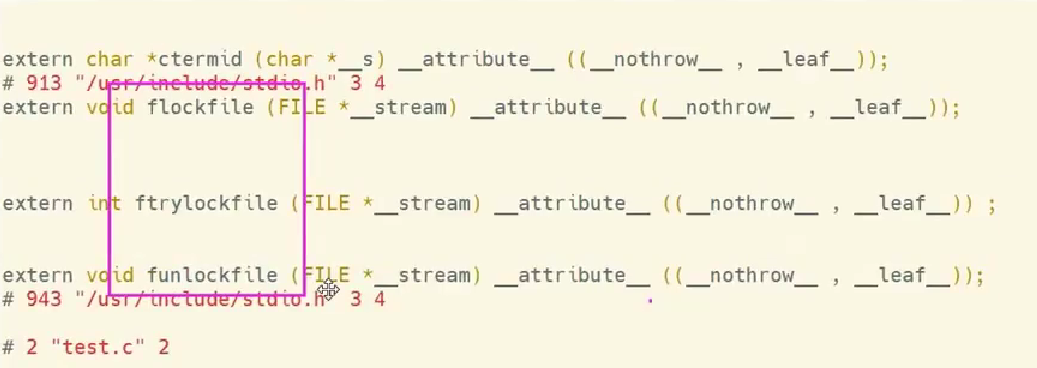


**3 C语言程序的编译和链接**

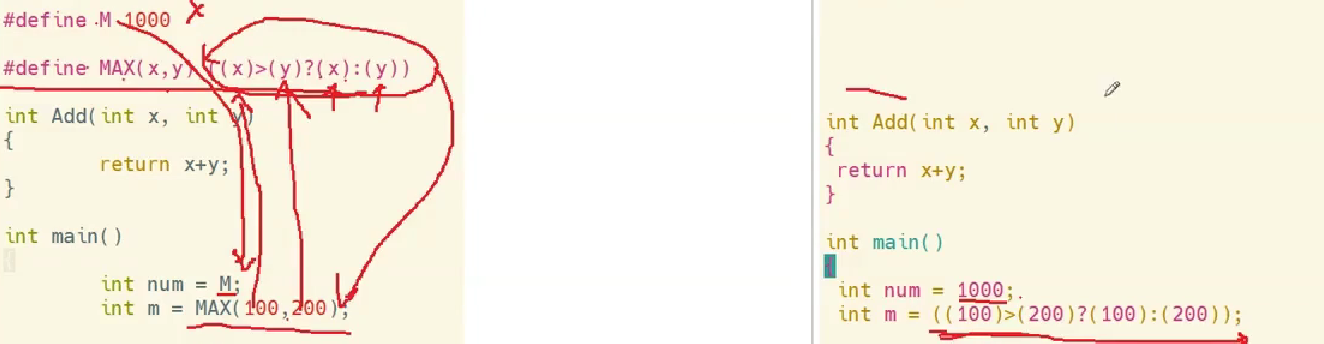
（1）编译的步骤（以Linux程序解剖）：如果直接gcc，默认生产一个a.out程序，可执行程序，即完成编译过程。./a.out执行程序。

第一，预编译（预处理）gcc test.c -E > test.i只进行预处理，完成了相关头文件的包含、完成了#define定义和宏的替换、注释删除**（文本操作）**；

头文件的包含：

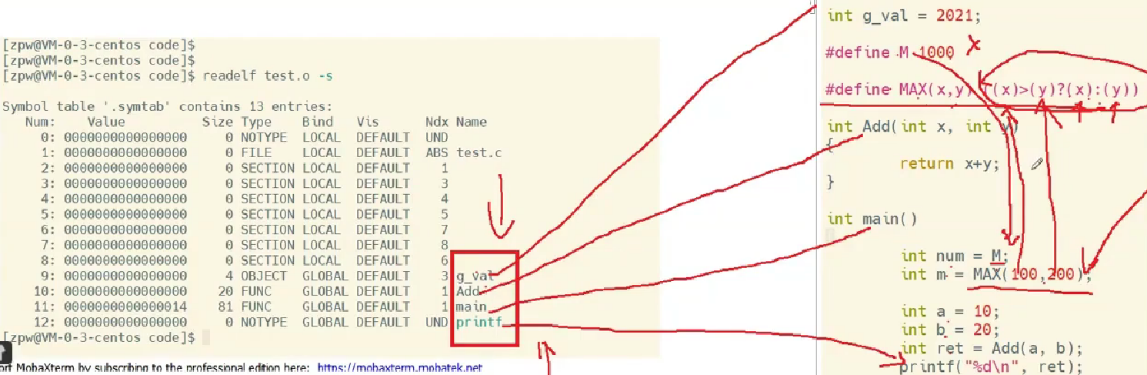


完成#define定义和宏的替换：



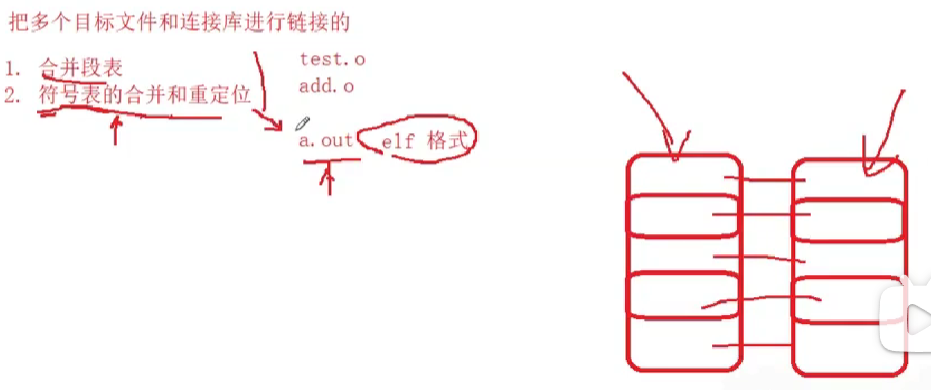
第二，编译gcc test.i -S编译生成一个文件test.s汇编代码，将C语言代码转换成汇编代码（语法分析、词法分析、语义分析、符号汇总全局的符号）；

第三，汇编gcc test.s -c生成一个test.o文件（test.obj目标文件），把汇编代码转换成二进制或者机器指令（生成符号表、转换机器指令）readelf test.o -s读二进制程序。

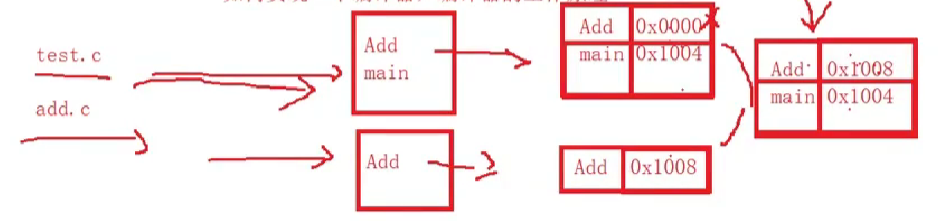


（2）链接的步骤：

第一，合并段表obj和exe格式都是elf格式存放的，段存放，多个目标文件进行段的合并；



第二，符号表的合并和重定位；



4 预处理详解

4.1 预定义符号介绍

#include<stdio.h>

//C语言已经定义好的符号 记录当前的时间地点等信息的

//\_\_FILE\_\_//进行编译的源文件的名称，包含路径

//\_\_LINE\_\_//文件当前的行号

//\_\_DATE\_\_//文件被编译的日期

//\_\_TIME\_\_//文件被编译的时间

//\_\_FUNCTION\_\_//函数

int main()

{

FILE\* pf = fopen("note.txt", "a+");

if (pf == NULL)

{

perror("fopen");

return 1;

}

int i = 0;

for (i = 0; i < 10; i++)

{

fprintf(pf, "%s %d %s %s %d\n", \_\_FILE\_\_, \_\_LINE\_\_,

\_\_DATE\_\_, \_\_TIME\_\_, i);//格式化写入

}

fclose(pf);

pf = NULL;

return 0;

}

4.2 预处理指令#define

（1）#define定义符号

由前所述，预处理是在编译之前的，#define定义的符号在编译之前已经替换。而这里的定义符号可以是多种形式，但是定义的符号不替换字符串常量。如果定义的符号内容太长，可以用反斜杠\表示续行符。**注：预处理指令#define后面不要加;号，因为害怕被误解为替换符号。**

#include<stdio.h>

#define M 100

#define reg register

#define do\_foever for(; ;)

#define CASE break;case

int main()

{

reg int a = 0;

int b = 0;

b = M;

printf("M = %d\n", b);//这里的M不替换

int n = 0;

switch (n)

{

case 0:

CASE 1:

CASE 2:

}

return 0;

}

（2）#define定义宏

#define允许把参数替换到文本中，这种实现通常称为宏或定义宏，其书写形式如：#define name(parament\_list) stuff，其中parament\_list是以逗号隔开的参数列表参数可能出现在内容中，stuff表示替换内容可能包含参数。

#include<stdio.h>

#define MAX(x, y) (((x) > (y))? (x):(y))

//#define ADD(x, y) x + y

#define ADD(x, y) ((x) + (y))

int main()

{

int a = 5;

int b = 10;

printf("%d\n", MAX(a, b));

printf("%d\n", 10 \* ADD(a, b));

return 0;

}

**注：#define定义的宏是完全替换的（包括参数和整个内容），由此我们在使用过程中尽量将参数和内容都用括号括起来。宏不能出现递归，不能自己调用自己。**

4.3 预处理操作符#和##的介绍（可能某些源码中存在）

#把参数名插入到字符串中，由于参数名不能传递给宏，所以有这样的方法。

#include<stdio.h>

#define Print(x, y) printf("The value of "#x" is "y"\n", x)//**只能在宏定义中使用**

int main()

{

int a = 5;

int b = 10;

float c = 20.0;

//The value of a is 5

Print(a, "%d");

//The value of b is 10

Print(b, "%d");

//The value of c is 20.000000

Print(c, "%f");

return 0;

}

##连接两个符号

#include<stdio.h>

#define link(x, y) x##y

int main()

{

int hello\_boy = 10;

printf("%d\n", link(hello, \_boy));

return 0;

}

带有**副作用**的预处理指令，对参数和结果都有一定的影响。如：

int a = 1;

int b = a + 1;//b=2, a=1

int b = ++a;//b=2, a=2 改变了a的值

例：下面程序就会引起误解

#include<stdio.h>

#define MAX(X, Y) ((X)>(Y)?(X):(Y))

int main()

{

int a = 5;

int b = 8;

//int m = MAX(a++, b++);

int m = ((a++) > (b++) ? (a++) : (b++));

printf("m = %d\n", m);//m=9

printf("a = %d, b = %d\n", a, b);//a=6,b=10

return 0;

}

4.4 宏和函数的区别

宏常用来执行简单的运算，比如找出两个数中的最大值。主要原因是：

（1）宏比函数在程序的规模少和运算速度快等方面更胜一筹。宏只需要预处理时替换即可，没有函数的调用和返回等过程。

（2）函数固定了参数和返回值的类型的，但宏是类型无关的，适用性更广。

而宏也有自己的劣势的

（1）对于较长的宏，在多次使用时会使得程序大幅增多，而函数调用则不会。

（2）宏是没有办法调试的，因为宏是在调试之前处理的。

（3）由于宏是类型无关的，也注定了它并不是特别严谨。

（4）宏可能会带来运算符优先级问题，容易导致程序出错。

注：如果比较简单的程序，用宏比较好，如果较为复杂的程序，用函数比较好。一般命名规定，宏名全部大写，而函数名不要全部大写。

#include<stdio.h>

#define MALLOC(num, type) (int \*)malloc(num\*sizeof(type))

int main()

{

//int \*p = (int \*)malloc(10 \* sizeof(int));

int \*p = MALLOC(10, int);

return 0;

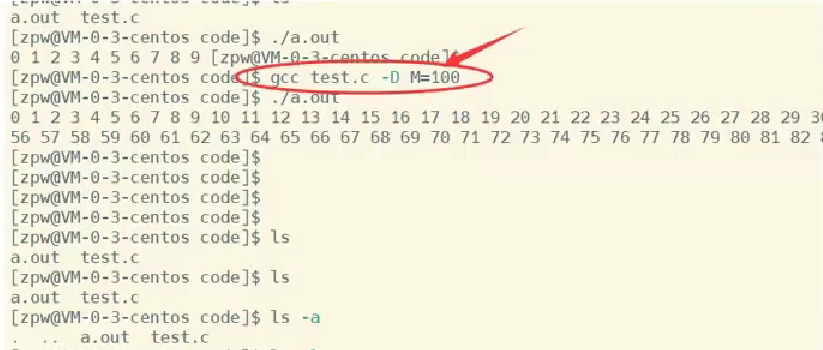
}

4.5 预处理指令#undef

取消预定义符号指令。#undef MALLOC(num, type)

4.6 命令行定义

Linux中命令行可以定义一些参数



4.7 条件编译

满足条件就编译，不满足条件就不编译。

#include<stdio.h>

#define PRINT

int main()

{

#ifdef PRINT

printf("hehe\n");

#endif

return 0;

}

常见的几种形式：

（1）#if 1-2// #if 常量表达式，非零执行，否则执行。

printf("hehe\n");

#endif

（2）#if 1==1// #if 常量表达式，非零执行，否则执行。

printf("hehe\n");

#elif 2==2// #if 常量表达式，非零执行，否则执行。

printf("haha\n");

#else

printf("heihei\n");

#endif

（3）#ifdef PRINT //是否定义 #ifndef #if !defined()

printf("hehe\n");

#endif

（4）条件编译的嵌套

4.8 预处理指令#include

**文件包含指令**，两种查找方式，双引号和尖括号。

#include “add.h”包含本地文件用双引号，自定义函数的头文件。双引号查找首先在项目目录下面查找，如果没找到则再去标准库目录取查找。

#include <stdio.h>包含库头文件用尖括号，库函数的头文件。尖括号直接去标准库目录查找。

头文件的嵌套包含，头文件多次包含，处理方式（防止头文件多次包含）：

（1）头文件的最前面写#pragma once

（2）条件编译方式

//防止头文件被多次包含的方式

//1.#pragma once

//2.

#ifndef \_\_ADD\_H\_\_

#define \_\_ADD\_H\_\_

int Add(int x, int y);//头文件内容

#endif