

实验六

分析GCC编译器

- 汇报时间 2024.1.19
- 小组：李多扬小组
- 成员：李多扬 智于行 刘栩孜 孙田塍 储伟涛 张桓嘉

任务分工

内容

1. GCC编译器的概述

GNU编译器 (GCC, GNU Compiler Collection) 是GNU工具链的关键组件，与GNU、Linux相关项目的标准编译器。它设计之初仅用来处理C语言的（也被称为GNU C编译器），紧接着扩展到C++、Objective-C/C++、Fortran、Java、Go等编程语言(如g++)。



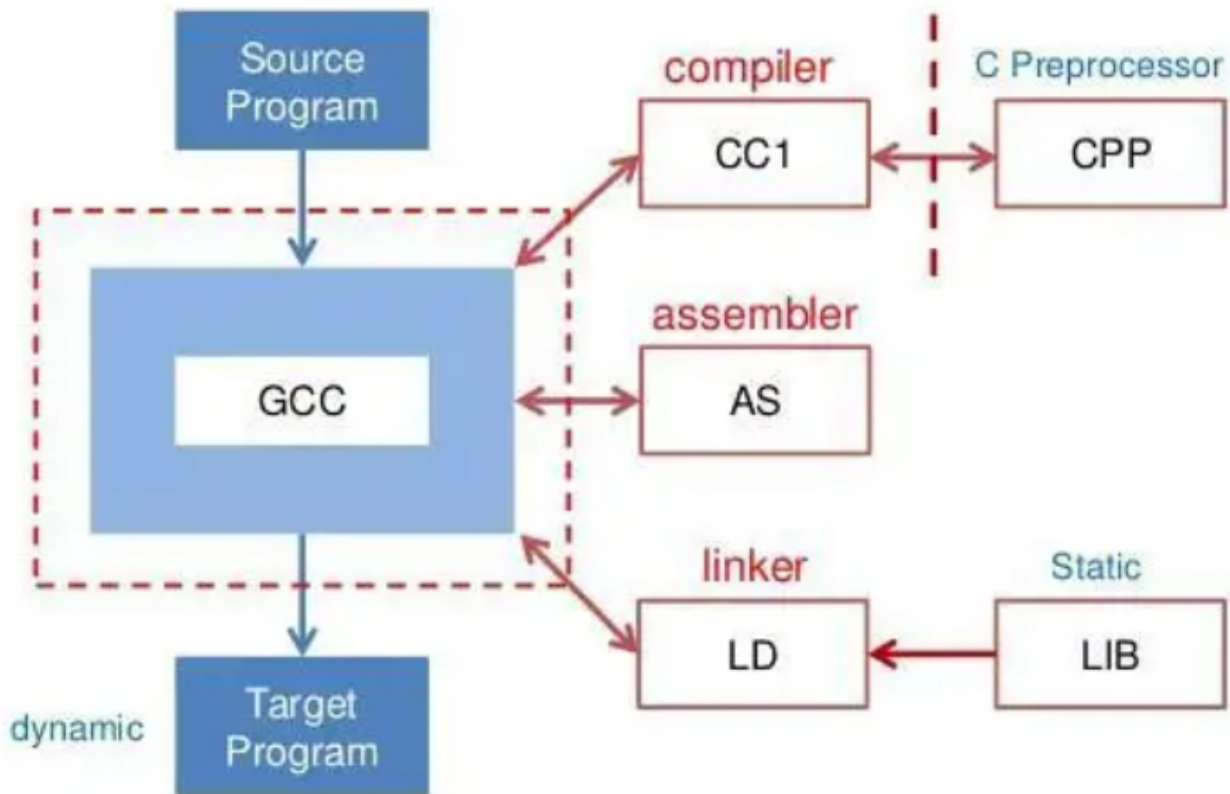
2. GNU编译器（GCC）的编译过程

2.1 从程序编译角度看GCC

GCC的编译工程可以分为四个步骤：

- 预处理（Pre-Processing）：使用CPP对C语言源程序进行预处理生成预处理.i文件，在该过程主要对源代码文件中的文件包含(include)、预编译语句(如宏定义define等)进行分析；
- 编译（Compiling）：调用CC1将预处理后的.i文件编译汇编语言.s文件，这个阶段主要是对预处理文件进行转换以生成机器语言的目标程序；
- 汇编（Assembling）：GCC调用AS对汇编语言.s文件进行汇编，生成目标文件.o；
- 链接（Linking）：GCC调用LD将各个模块的.o文件连接起来生成一个可执行程序文件，该过程将目标文件/库文件指派到可执行程序各自对应的位置。

GCC compiler

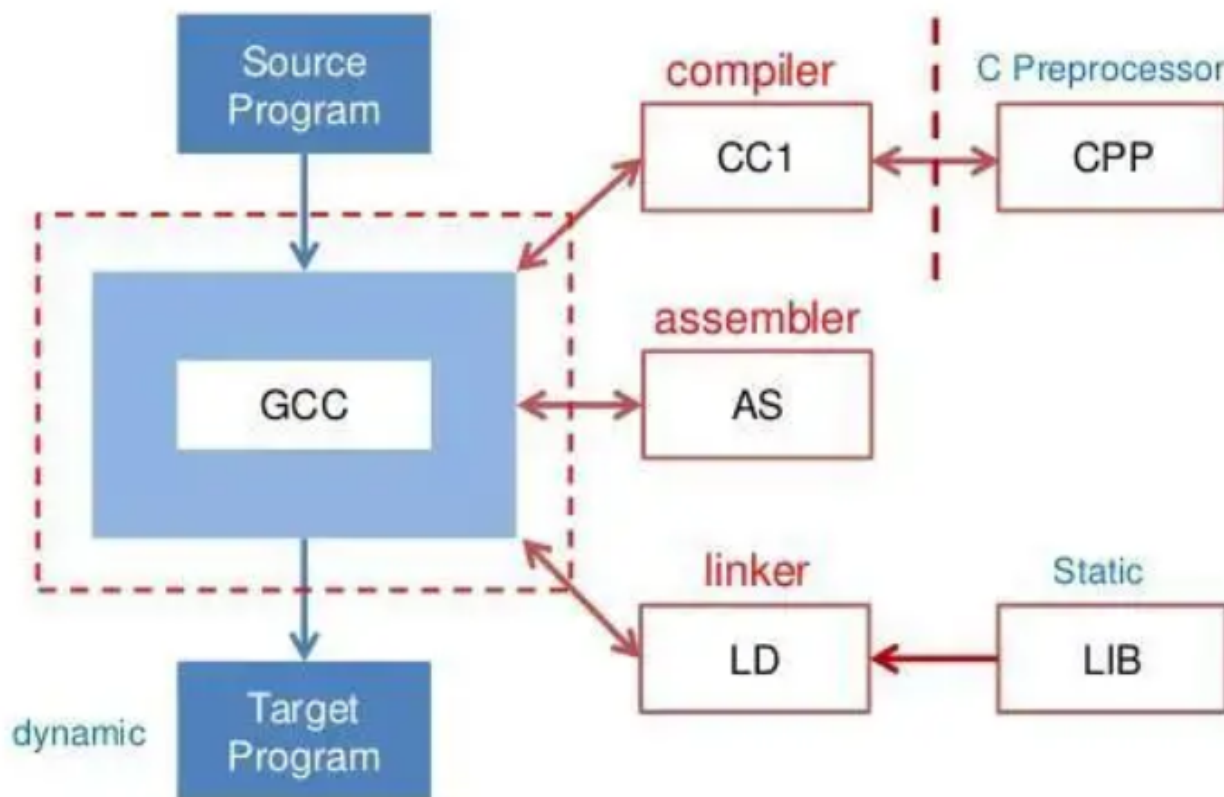


2.2 从文件角度看GCC

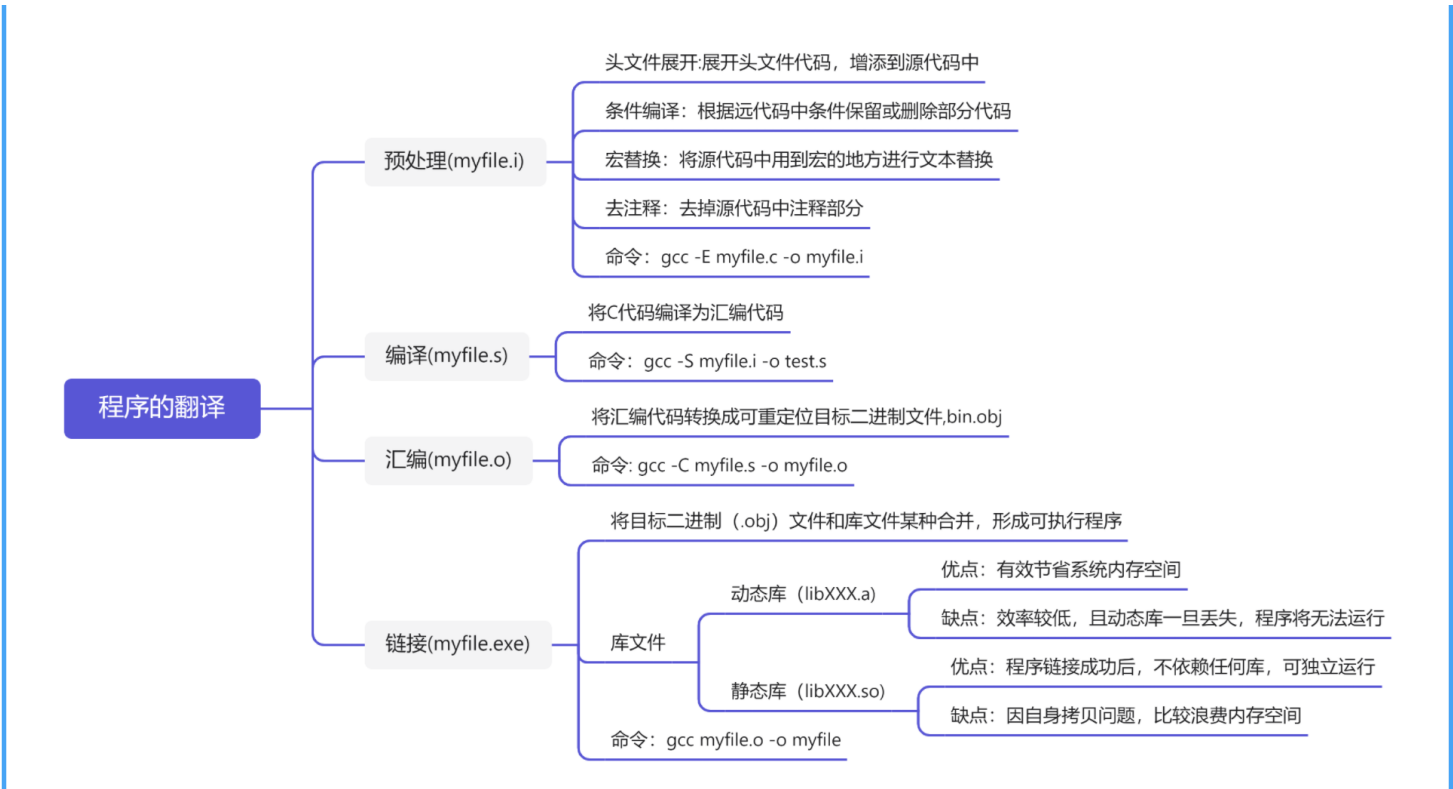
GCC编译过程中生成了许多不同种类与功能的文件：

文件后缀	文件类型	文件内容
.c	C语言源程序	源代码文件
.h	C语言头文件	源代码文件
.i	预处理后的C语言源程序	源代码文件
.s	汇编语言源程序	源代码文件
.o/.obj	目标文件	二进制文件
.so/.dll	动态链接库	二进制文件

GCC compiler



2.3 以C语言程序为例分析GCC编译过程



2.3.1 c源代码

```
//test.c
// 简单的整型变量操作与输出程序
#include <stdio.h>
#include <stdlib.h>
int main()
{

    int x = 1;
    for (int i = 0; i < 10; i++)
    {
        x = x + 1;
    }
    printf("return:%d", x);
    system("pause");
    return 0;
}
```

2.3.2 词法分析 (Lexical Analysis)

gcc中的词法分析主要发生在 预处理阶段(Preprocessing)

首先，在这个阶段，预处理器会处理所有以井号（#）开头的指令，如#include和#define等。处理宏定义，将代码中的宏替换为相应内容。展开头文件，将#include的头文件内容插入到代码中。如果有条件编译指令（如#ifdef、#ifndef、#if等），根据条件进行代码的选择性包含或排除

- 对应步骤：预处理（Preprocessing）。
- 任务：展开头文件、宏替换、条件编译等。
- 命令：gcc -E source.c -o output.i
- 示例：gcc -E test.c -o test_i.i

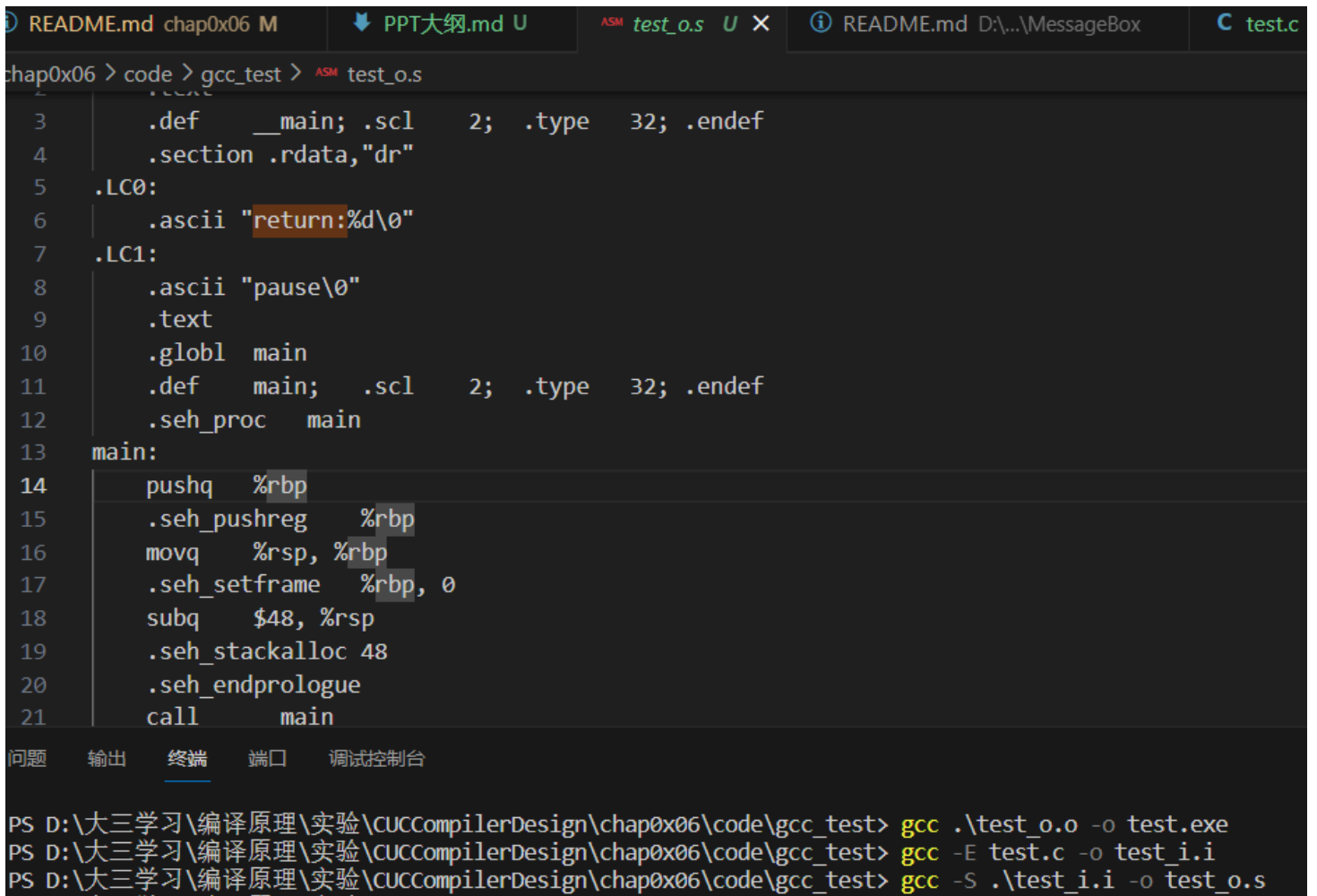
```
chap0x06 > code > gcc_test > test_i.i > main()
1470     _Ptr = (char*)_Ptr + 16;
1471 }
1472 return _Ptr;
1473 }
1474 # 163 "C:/Program Files (x86)/mingw64/x86_64-w64-mingw32/include/malloc.h" 3
1475 static __inline void __attribute__((__cdecl__)) _freea(void *_Memory) {
1476     unsigned int _Marker;
1477     if(_Memory) {
1478         _Memory = (char*)_Memory - 16;
1479         _Marker = *(unsigned int *)_Memory;
1480         if(_Marker==0xDDDD) {
1481             free(_Memory);
1482         }
1483     }
1484 }
1485 }
1486 }
1487 }
1488 }
1489 }
1490 # 209 "C:/Program Files (x86)/mingw64/x86_64-w64-mingw32/include/malloc.h" 3
1491 #pragma pack(pop)
1492 # 742 "C:/Program Files (x86)/mingw64/x86_64-w64-mingw32/include/stdlib.h" 2 3
1493 # 3 "test.c" 2
1494
1495 # 3 "test.c"
1496 int main()
1497 {
```

2.3.3 语法分析 (Syntax Analysis)

将预处理后的源代码翻译成汇编语言。这个阶段产生的文件通常被命名为*.s（汇编代码）。

- 对应步骤：编译（Compilation）。
- 任务：将预处理后的源代码翻译成汇编代码。
- 命令：gcc -S output.i -o output.s

- 示例：gcc -S .\test_i.i -o test_o.s



```
chap0x06 > code > gcc_test > ASM test_o.s
3      .def    __main; .scl    2; .type  32; .endef
4      .section .rdata,"dr"
5      .LC0:
6      .ascii  "return:%d\0"
7      .LC1:
8      .ascii  "pause\0"
9      .text
10     .globl  main
11     .def    main; .scl    2; .type  32; .endef
12     .seh_proc main
13 main:
14     pushq   %rbp
15     .seh_pushreg %rbp
16     movq    %rsp, %rbp
17     .seh_setframe %rbp, 0
18     subq    $48, %rsp
19     .seh_stackalloc 48
20     .seh_endprologue
21     call    main

问题  输出  终端  端口  调试控制台

PS D:\大三学习\编译原理\实验\CUCCompilerDesign\chap0x06\code\gcc_test> gcc .\test_o.o -o test.exe
PS D:\大三学习\编译原理\实验\CUCCompilerDesign\chap0x06\code\gcc_test> gcc -E test.c -o test_i.i
PS D:\大三学习\编译原理\实验\CUCCompilerDesign\chap0x06\code\gcc_test> gcc -S .\test_i.i -o test_o.s
```

2.3.4 语义分析 (Semantic Analysis) 与 中间代码生成 (Intermediate Code Generation)

汇编器将汇编代码翻译成机器语言，生成目标文件（通常是*.o或*.obj文件），其中包含了二进制代码和符号表信息。

- 对应步骤：汇编 (Assembly) 。
- 任务：将汇编代码翻译成目标机器码（二进制文件）。
- 命令：gcc -c output.s -o output.o
- 示例：gcc -c .\test_o.s -o test_o.o
- gcc -fdump-tree-all map.c 会生成大量中间代码

```
chapg0x06 > code > gcc_test > test_o.o
```

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded Text
000200	00	00	0A	00	00	00	04	00	38	00	00	00	13	00	00	00 8
000210	04	00	3F	00	00	00	0A	00	00	00	04	00	44	00	00	00	. . ? D . . .
000220	14	00	00	00	04	00	00	00	00	00	04	00	00	00	00	03
000230	04	00	00	00	04	00	00	00	03	00	08	00	00	00	00	0C
000240	00	00	03	00	2E	66	69	6C	65	00	00	00	00	00	00	00 f i l e
000250	FE	FF	00	00	67	01	74	65	73	74	2E	63	00	00	00	00 g . t e s t . c
000260	00	00	00	00	00	00	00	00	6D	61	69	6E	00	00	00	00 m a i n
000270	00	00	00	00	01	00	20	00	02	01	00	00	00	00	00	00
000280	00	00	00	00	00	00	00	00	00	00	00	00	2E	74	65	78 t e x
000290	74	00	00	00	00	00	00	00	01	00	00	00	03	01	53	00	t S .
0002A0	00	00	05	00	00	00	00	00	00	00	00	00	00	00	00	00
0002B0	2E	64	61	74	61	00	00	00	00	00	00	00	02	00	00	00	. d a t a
0002C0	03	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0002D0	00	00	00	00	2E	62	73	73	00	00	00	00	00	00	00	00 b s s
0002E0	03	00	00	00	03	01	00	00	00	00	00	00	00	00	00	00
0002F0	00	00	00	00	00	00	00	00	2E	72	64	61	74	61	00	00 r d a t a . . .

问题 输出 终端 端口 调试控制台

```
PS D:\大三学习\编译原理\实验\CUCCompilerDesign\chap0x06\code\gcc_test> gcc .\test_o.o -o test.exe
PS D:\大三学习\编译原理\实验\CUCCompilerDesign\chap0x06\code\gcc_test> gcc -E test.c -o test_i.i
PS D:\大三学习\编译原理\实验\CUCCompilerDesign\chap0x06\code\gcc_test> gcc -S .\test_i.i -o test_o.s
PS D:\大三学习\编译原理\实验\CUCCompilerDesign\chap0x06\code\gcc_test> gcc -c .\test_o.s -o test_o.o
```

2.3.5 代码优化 (Code Optimization)

- 对应步骤： 在整个过程中的优化步骤，包括中间代码生成后和目标代码生成前的优化。
- 在GCC中的实现： GCC内部包含多个优化器，如Gimple、RTL优化器等。

2.3.6 目标代码生成 (Target Code Generation)

链接器将各个目标文件以及可能的库文件组合在一起，解决各种符号的引用关系。
解析全局变量和函数的引用，将它们关联到相应的地址。
最终生成可执行文件，其中包含了程序的二进制代码，以及各个部分在内存中的布局信息。

- 对应步骤： 链接 (Linking) 。
- 任务： 将编译后的目标文件和可能需要的库文件链接成可执行文件。
- 命令： gcc output.o -o output
- 示例： gcc .\test_o.o -o test.exe

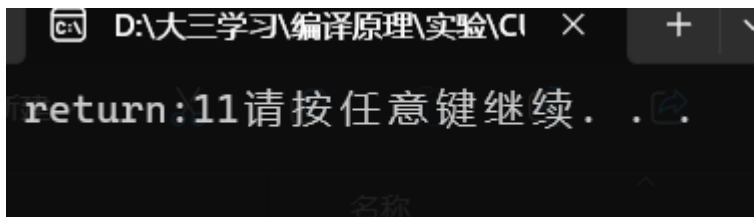
README.md chap0x06 M PPT大纲.md U test.exe U X README.md D:\...\MessageBox test.c U

chap0x06 > code > gcc_test > test.exe

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded Text
000023B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.
000023C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.
000023D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.
000023E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.
000023F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.
00002400	72	65	74	75	72	6E	3A	25	64	00	70	61	75	73	65	00	r e t u r n : % d . p a u s e .
00002410	40	75	40	00	00	00	00	00	60	70	40	00	00	00	00	00	@ u @ p @
00002420	B0	18	40	00	00	00	00	00	00	00	00	00	00	00	00	00	. . @
00002430	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.
00002440	00	A0	40	00	00	00	00	00	08	A0	40	00	00	00	00	00	. . @ @
00002450	FC	75	40	00	00	00	00	00	40	90	40	00	00	00	00	00	. u @ @ . @
00002460	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.
00002470	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.
00002480	41	72	67	75	6D	65	6E	74	20	64	6F	6D	61	69	6E	20	A r g u m e n t d o m a i n
00002490	65	72	72	6F	72	20	28	44	4F	4D	41	49	4E	29	00	41	e r r o r (D O M A I N) . A
000024A0	72	67	75	6D	65	6E	74	20	73	69	6E	67	75	6C	61	72	r g u m e n t s i n g u l a r

问题 输出 终端 端口 调试控制台

PS D:\大三学习\编译原理\实验\CUCCompilerDesign\chap0x06\code\gcc_test> gcc .\test_o.o -o test.exe
PS D:\大三学习\编译原理\实验\CUCCompilerDesign\chap0x06\code\gcc_test> gcc -E test.c -o test_i.i
PS D:\大三学习\编译原理\实验\CUCCompilerDesign\chap0x06\code\gcc_test> gcc -S .\test_i.i -o test_o.s
PS D:\大三学习\编译原理\实验\CUCCompilerDesign\chap0x06\code\gcc_test> gcc -c .\test_o.s -o test_o.o
PS D:\大三学习\编译原理\实验\CUCCompilerDesign\chap0x06\code\gcc_test> gcc .\test_o.o -o test.exe



总结

- test.c->test.i->test.s->test.o->test.exe
- 从 c 语言或者 c++语言的高级代码，转化为预处理的结果文件，之后再去头变为汇编语言程序，再然后转为二进制机器代码，最后进行库的链接，最终得到可执行文件。