ICS Seminar Week8 Prep

朱家启 徐梓越 许珈铭

2023.11.13

Rules

remainder <- ordinal number in WeChat Group % 4

for all questions do

    if question number % 4 == remainder then

        you should work on it

    end

end

Q1

6. 在链接时，对于下列哪些符号需要进行重定位？

(1) 不同 C 语言源文件中定义的函数

(2) 同一 C 语言源文件中定义的全局变量

(3) 同一函数中定义时不带 static 的变量

(4) 同一函数中定义时带有 static 的变量

A. (1)(3)　　B. (2)(4)　　C. (1)(2)(4)　　D. (1)(2)(3)(4)

(1) 函数需要重定位
(2) 全局变量需要重定位
(3) 函数中定义的非静态变量不是链接器的工作
(4) 静态变量需要重定位

C

Q2

4. 以下关于静态库链接的描述中，正确的是：

A. 链接时，链接器会拷贝静态库中的所有目标模块。

B. 使用库的时候必须把它们放在命令行的结尾处。

C. 如果库不是相互独立的，那么它们必须排序。

D. 每个库在命令行只须出现一次即可。

A：链接时，静态库中那些能解释前述目标文件中引用了但未定义的符号的目标模块才会被拷贝

B：使用库时不一定要放在结尾处，事实上我同时用两个库总有一个库不在结尾

C：如果库不是独立的，那么被依赖的库应该放在后面

D：当两个库有相互依赖关系时，至少一个库要出现两次

C

Q3

8． 下面关于链接的说法，正确的是：

A. Linux 链接器在处理多重定义的同名弱符号时，选择链接时遇到的第一个符号

B. 链接发生在源代码编译之后、可执行目标程序运行之前

C. C 程序静态局部变量和静态全局变量都在 ELF 可重定位目标文件的.data 段

D. 链接器构造可执行目标文件时，只复制静态库里被应用程序引用的目标模块

A：同名弱符号随机选择

B：链接发生的时间较自由，可以发生在编译时，加载时，运行时（见中文版P464第一段）

C：未初始化的静态变量和初始化为0的静态变量在.bss中

D：正确，同Q2中解析

D

Q4

10.在 foo.c 文件中的函数外，如果添加如下一条语句：

```
static int count = 0xdeadbeef;
```

那么它在编译为 foo.o 后，会影响到 ELF 可重定位目标文件中的除.text 以外的那些字段？（　　）

A. .rodata

B. .data,.symtab,

C. .data,.symtab,.rel.data

D. .rodata,.symtab,.rel.data

！！！这道题暂时保留，答案可能是B或C

增加了一个已经初始化了的静态变量，会存到.data中，同时改变symtab；
对.rodata，为只读数据，不会改变

有争议的是对.rel.data，按照书上所说里面只有被模块引用或定义的全局变量信息；但是实际操作中静态全局变量和静态局部变量都会

B

Q5

8. C 源文件 m1.c 和 m2.c 的代码分别如下所示，编译链接生成可执行文件后执行，结果最可能为（ ）

```
$ gcc -o a.out m2.c m1.c ; ./a.out
0x1083020 _____ _____
```

A. 0x1083018, 0x108301c　　B. 0x1083028, 0x1083024

C. 0x1083024, 0x1083028　　D. 0x108301c, 0x1083018

```
// m1.c
#include <stdio.h>

int a1 ;
int a2 = 2 ;
extern int a4 ;

void hello()
{
    printf("%p ", &a1);
    printf("%p ", &a2);
    printf("%p\n", &a4);
}
```

```
//m2.c
int a4 = 10 ;

int main()
{
    extern void hello() ;

    hello() ;
    return 0 ;
}
```

进行了链接并执行，并且链接时m2.c在前。先进入hello，输出a1的地址，为0x1083020，接着输出a2和a4的地址。我们知道链接不会为未初始化的变量分配地址，所以a1的地址是在运行时分配的。最先分配的是a4的地址，然后是a2的地址，再是运行时分配a1的地址。所以应有&a4<&a2<&a1，即D

D

Q6

7. C源文件 f1.c 和 f2.c 的代码分别如下所示：

```
// f1.c
#include <stdio.h>
void f();
int x ;
int main(void)
{
   x = 1;
   f() ;
  printf("%x\n", x) ;
  return 0;
}
```

```
//f2.c
float x ;
void f()
{
  x = 2 ;
}
```

运行下面的命令后得到的结果是：

```
$ gcc f1.c f2.c
$ ./a.out
```

A. 1　　B. 2　　C. 3f800000　　D. 40000000

这里x在两个文件中均未初始化，为弱定义，且同时被定义成了int和float。类比书本P473的例子，在main中调用了f，让x=2，此时x的IEEE浮点表示为0b01000...0，即0x40000000，之后按int的%x输出即输出D选项

D

Q7

8. 文件 f1.c 和 f2.c 的 C 源代码如下图所示：

```
//f1.c
#include<stdio.h>
extern float x;
extern void f();
int main()
{
   f();
   printf("%d\n",(int)x);
   return 0;
}
```

```
//f2.c
int x = 0;
void f()
{
   x++;
}
```

已知这两个文件在同一个目录下，在该目录下用“gcc -Og -o f f1.c f2.c”编译，然后用“./f”运行，这个过程中会出现的情况是：

A. 编译错误
B. 输出0
C. 输出1
D. 链接错误

f1.c中声明x是外部的float变量，而在f1.c中x是全局的初始化为0的int变量。所以在编译中，main函数中的x会被看作一个float类型的变量，而void函数中x会看作一个int类型的变量。
运行时，先进入void函数，x从0经++为1，由于void中x看作int类型，实际x=0b000...01，而后回到main函数中，x被看作float类型的变量，根据IEEE浮点表示，它是一个非规范化的很小的数，所以int类型转换后向0取整为0，选B。

B

Q8

11. （ ）c 源文件 f1.c 和 f2.c 的代码分别如下所示，那么运行结果为?

```
// f1.c
#include <stdio.h>
static int var;
int main() {
  extern void f(void);
  f();
  printf("%d", var);
  return 0;
}
```

```
// f2.c
extern int var;

void f() { var++; }

int var = 100;
```

A. 0　　B. 1　　C. 101　　D. 链接时出错

f1中的var是静态变量，所以它只会在自己的f1.o里面被调用，别人调用的都不是它。而f2中extern int var会寻找var的定义，在下方中int var=100中找到定义，所以将f()中的var链接到下面的全局变量var。实际运行中main函数调用f，但是修改的不是f1中的静态变量var，所以输出仍为0。

A

Q9

第四题（10 分）

考虑如下两个程序（fact1.c**和**fact2.c**）：**

```
/* fact1.c */
#define MAXNUM 12
int table[MAXNUM];
int fact(int n);
int main(int argc, char **argv) {
    int n;
    table[0] = 0;
   table[1] = 1;
   if (argc == 1) {
      printf("Error: missing argument\n");
      exit (0);
   }
   argv++;
   if (sscanf(*argv, "%d", &n) != 1 || n < 0 || n >=
   MAXNUM) {
       printf ("Error: %s not an int or out of
   range\n", *argv);
       exit (0);
   }
   printf("fact(%d) = %d\n", n, fact(n));
}
```

```
/* fact2.c */
int* table;
int fact(int n) {
    static int num = 2;
    if (n >= num) {
        int i = num;
        while (i <= n) {
            table[i] = table[i-1] * i;
            i++;
        }
        num = i;
    }
     return table[n];
 }
```

Q9

（1）对于每个程序中的相应符号，给出它的属性（局部变量、强全局变量或弱全局变量），以及它在链接后位于 ELF 文件中的什么位置？（提示：如果某表项中的内容无法确定，请画 X）（6 分）

fact1.c

变量	类型	ELF Section
table		
fact		
num		

fact2.c

变量	类型	ELF Section
table		
fact		
num		

（2）对上述两个文件进行链接之后，会对每个符号进行解析。请给出链接后下列符号被定义的模块（fact1 or fact2）。（2 分）

	定义模块
table	
fact	
num	

（3）使用 gcc（命令：gcc -o fact fact1.c fact2.c）来编译之后得到的可执行文件是否能够正确执行？为什么？（2 分）

（1）表述有点问题，问的应该是链接前在.o文件中位于ELF文件中的位置。fac1中table是长为max的未初始化的数组，为弱全局变量，ELF Section为COMMON(.bss)；fact为未定义完全的函数，为弱全局变量，ELF Section为UNDEF(X,.bss)；num没有出现，为X。fac2中table是未初始化的指针，是弱全局变量，ELF Section为COMMON(.bss)；fact为定义的函数，强全局变量，ELF Section为.text；num为初始化了的静态变量，所以是局部变量，ELF Section为.data。

（2）由上一问中分析，table是两个弱全局变量，不确定在哪里定义；fact在fac2中定义，num在fact2中定义

（3）见下方解析

Q9

fact1.c

变量	类型	ELF Section
table	**weak global**	**.bss**
fact	**weak global**	**X（或.bss）**
num	**X**	**X**

fact2.c

变量	类型	ELF Section
table	**weak global**	**.bss**
fact	**strong global**	**.text**
num	**local**	**.data**

	定义模块
table	**不确定**
fact	**fact2**
num	**fact2**

不一定能正确执行。因为 table 在两个文件中都是 weak global，因此链接器会任选一个定义来解析 table。而因为在 fact2 模块中只给 table 预留了一个单字（4 字节）空间，因此如果选择了 fact2 来解析 table 的话，会出现 segmentation fault。【该问题已经过编译检查确认。】

Q10

第四题（10 分）

考虑如下两个程序（fact1.c**和**fact2.c**）：**

```
/* fact1.c */
#define MAXNUM 12
int table[MAXNUM];
int fact(int n);
int main(int argc, char **argv) {
    int n;
    table[0] = 0;
   table[1] = 1;
   if (argc == 1) {
      printf("Error: missing argument\n");
      exit (0);
   }
   argv++;
   if (sscanf(*argv, "%d", &n) != 1 || n < 0 || n >=
   MAXNUM) {
       printf ("Error: %s not an int or out of
   range\n", *argv);
       exit (0);
   }
   printf("fact(%d) = %d\n", n, fact(n));
}
```

```
/* fact2.c */
int* table;
int fact(int n) {
    static int num = 2;
    if (n >= num) {
        int i = num;
        while (i <= n) {
            table[i] = table[i-1] * i;
            i++;
        }
        num = i;
    }
     return table[n];
 }
```

Q10

（1）对于每个程序中的相应符号，给出它的属性（局部变量、强全局变量或弱全局变量），以及它在链接后位于 ELF 文件中的什么位置？（提示：如果某表项中的内容无法确定，请画 X）（6 分）

fact1.c

变量	类型	ELF Section
table		
fact		
num		

fact2.c

变量	类型	ELF Section
table		
fact		
num		

（2）对上述两个文件进行链接之后，会对每个符号进行解析。请给出链接后下列符号被定义的模块（fact1 or fact2）。（2 分）

	定义模块
table	
fact	
num	

（3）使用 gcc（命令：gcc -o fact fact1.c fact2.c）来编译之后得到的可执行文件是否能够正确执行？为什么？（2 分）

Q10

fact1.c

变量	类型	ELF Section
table	**weak global**	**.bss**
fact	**weak global**	**X（或.bss）**
num	**X**	**X**

fact2.c

变量	类型	ELF Section
table	**weak global**	**.bss**
fact	**strong global**	**.text**
num	**local**	**.data**

	定义模块
table	**不确定**
fact	**fact2**
num	**fact2**

不一定能正确执行。因为 table 在两个文件中都是 weak global，因此链接器会任选一个定义来解析 table。而因为在 fact2 模块中只给 table 预留了一个单字（4 字节）空间，因此如果选择了 fact2 来解析 table 的话，会出现 segmentation fault。【该问题已经过编译检查确认。】

Q11

第四题（10 分） 链接

考虑如下3个文件：main.c, fib.c和bignat.c:

```
/* main.c */
void fib (int n);
int main (int argc, char** argv) {
    int n = 0;
    sscanf(argv[1], "%d", &n);
    fib(n);
}
```

另外，假设在文件 bignat.c 中定义了如下两个函数 plus 和 from_int（具体定义略）:

```
int plus (int n, unsigned int* a, unsigned int* b, unsigned
int* c);
void from_int (int n, unsigned int k, unsigned int* a);
```

```
/* fib.c */
#define N 16

static unsigned int ring[3][N];

static void print_bignat(unsigned int* a) {
    int i;
    for (i = N-1; i >= 0; i--)
        printf("%u ", a[i]); /* print a[i] as unsigned int
        */
    printf("\n");
}
void fib (int n) {
    int i, carry;
    from_int(N, 0, ring[0]); /* fib(0) = 0 */
    from_int(N, 1, ring[1]); /* fib(1) = 1 */
    for (i = 0; i <= n-2; i++) {
        carry = plus(N, ring[i%3], ring[(i+1)%3],
        ring[(i+2)%3]);
        if (carry)
            { printf("Overflow at fib(%d)\n", i+2);
        exit(0); }
    }
    print_bignat(ring[n%3]);
}
```

Q11

1. （5 分）对于每个程序中的相应符号，给出它的属性（局部或全局，强符号或弱符号）（提示：如果某表项中的内容无法确定，请画 X。）

main.c

	局部或全局？	强或弱？
fib		
main		

fib.c

	局部或全局？	强或弱？
ring		
fib		
plus		

2. （3 分） 假设文件 bignat.c 被编译为一个静态库 bignat.a，对于如下的 gcc 调用，会得到什么样的结果（请选择）？

（A） 编译和链接都正确

（B） 链接失败（原因是包含未定义的引用）

（C） 链接失败（原因是包含重复定义）

命令	结果（A，B 或 C）
gcc -o fib main.c fib.c bignat.a	
gcc -o fib bignat.a main.c fib.c	
gcc -o fib fib.c main.c bignat.a	

3. （2 分）如果在文件 fib.c 中，程序员在声明变量 ring 时，不小心把它写成了：

```
static int ring[3][N];
```

会不会影响这些文件的编译、链接和运行结果？为什么？

先看main.c，里面fib是未定义完全的函数，是全局的弱符号；main是定义完全的函数，是全局的强符号。

再看fib.c，ring是静态变量，为局部符号；fib是定义完全的函数，为全局的强符号；plus在fib.c中未定义，编译器假设它在外部被定义，所以是全局的弱符号。

对于2，由于fib.c中需要调用bignat.a中的plus函数，所以当bignat.a在fib.c之后时才能正确链接，所以答案为ABA

3的解析见下方答案。

Q11

main.c

	局部或全局？	强或弱？
fib	全局	弱
main	全局	强

fib.c

	局部或全局？	强或弱？
ring	局部	X
fib	全局	强
plus	全局	弱

命令	结果（A，B 或 C）
gcc -o fib main.c fib.c bignat.a	A
gcc -o fib bignat.a main.c fib.c	B
gcc -o fib fib.c main.c bignat.a	A

对编译、链接和执行结果都没有影响。因为 signed 和 unsigned 之间的转换不会改变整数的表示形式，因此函数调用会正常进行。

Q12

第四题（10 分） 链接

考虑如下3个文件：main.c, fib.c和bignat.c:

```
/* main.c */
void fib (int n);
int main (int argc, char** argv) {
    int n = 0;
    sscanf(argv[1], "%d", &n);
    fib(n);
}
```

另外，假设在文件 bignat.c 中定义了如下两个函数 plus 和 from_int（具体定义略）:

```
int plus (int n, unsigned int* a, unsigned int* b, unsigned
int* c);
void from_int (int n, unsigned int k, unsigned int* a);
```

```
/* fib.c */
#define N 16

static unsigned int ring[3][N];

static void print_bignat(unsigned int* a) {
    int i;
    for (i = N-1; i >= 0; i--)
        printf("%u ", a[i]); /* print a[i] as unsigned int
        */
    printf("\n");
}
void fib (int n) {
    int i, carry;
    from_int(N, 0, ring[0]); /* fib(0) = 0 */
    from_int(N, 1, ring[1]); /* fib(1) = 1 */
    for (i = 0; i <= n-2; i++) {
        carry = plus(N, ring[i%3], ring[(i+1)%3],
        ring[(i+2)%3]);
        if (carry)
            { printf("Overflow at fib(%d)\n", i+2);
        exit(0); }
    }
    print_bignat(ring[n%3]);
}
```

Q12

1. （5 分）对于每个程序中的相应符号，给出它的属性（局部或全局，强符号或弱符号）（提示：如果某表项中的内容无法确定，请画 X。）

main.c

	局部或全局？	强或弱？
fib		
main		

fib.c

	局部或全局？	强或弱？
ring		
fib		
plus		

2. （3 分） 假设文件 bignat.c 被编译为一个静态库 bignat.a，对于如下的 gcc 调用，会得到什么样的结果（请选择）？

（A） 编译和链接都正确

（B） 链接失败（原因是包含未定义的引用）

（C） 链接失败（原因是包含重复定义）

命令	结果（A，B 或 C）
gcc -o fib main.c fib.c bignat.a	
gcc -o fib bignat.a main.c fib.c	
gcc -o fib fib.c main.c bignat.a	

3. （2 分）如果在文件 fib.c 中，程序员在声明变量 ring 时，不小心把它写成了：

```
static int ring[3][N];
```

会不会影响这些文件的编译、链接和运行结果？为什么？

Q12

main.c

	局部或全局？	强或弱？
fib	全局	弱
main	全局	强

fib.c

	局部或全局？	强或弱？
ring	局部	X
fib	全局	强
plus	全局	弱

命令	结果（A，B 或 C）
gcc -o fib main.c fib.c bignat.a	A
gcc -o fib bignat.a main.c fib.c	B
gcc -o fib fib.c main.c bignat.a	A

对编译、链接和执行结果都没有影响。因为 signed 和 unsigned 之间的转换不会改变整数的表示形式，因此函数调用会正常进行。

3. 在int 类型值为32 位、用补码表示、采用算术右移，unsigned 类型值也为32 位的机器上，声明

```
int x, y;
unsigned ux = (unsigned)x;
unsigned uy = (unsigned)y;
```

则下列表达式中，当x 与y 取遍所有可能值时，可能为假的是

A. (x<y)==(-x>-y)

B. ~x+~y+1==~(x+y)

C. (ux-uy)==-(unsigned)(y-x)

D. ((x>>3)<<3)<=x

"$-x = \sim x + 1$"

$x = 10\cdots0_2 = -2^{32}+1$

$-x = 10\cdots0_2 = -2^{32}+1$

A: $x = Tmin$ $-x = Tmin$

B: 给左右都+1, $\sim x + \sim y + 1 + 1 = \sim x + 1 - y = -x - y = -(x+y) = \sim(x+y)+1$

C: 整型具交换、结合律

D: 后3位都向零舍入了, ___xxx => ___000 ∴ N一样

A

A

```
3.  int x = -2019;
    int a = (x / 8) * 8;
    int b = ((x + 7) >> 3) << 3;
    int c = -(((~x)+1) >> 3) << 3;
    a, b, c 大小关系是什么?
    A. a=b, b=c
    B. a=b, b>c
    C. a<b b<c
    D. a<b, b=c
```

除法向"0"舍入；右移向下舍入

在负数情况下

$x >> 3 = x/8 - 1$

$\therefore (x+7) >> 3 = x/8$

$\Rightarrow a = b$

$(\sim x+1) = -x$

正数情况下

$x >> 3 = x/8$

$\therefore a = c$

$\Rightarrow a = b = c$

A

A

0. 下面程序的输出是（　　）

```
int main() {
    int x = 0xbadbeef >> 3;
    char y = (char)(x);
    unsigned char z = (unsigned char)(x);
    printf("%d %u\n", y, z);
    return 0;
}
```

A. -35 221

B. -35 35

C. -221 221

D. -221 35

原数 1011 …… 1110 1110 1111

>>3后：1111 …… 110| 110|

(char)(x) = 110| 110| $= -2^7+2^6+2^4+2^3+2^2+1 = -35$

(unsigned x) = 110| 110| $= 2^7+2^6+2^4+2^3+2^2+1 = 221$

A

2. 假设有下面 x 和 y 的程序定义

```
int x = a >> 2;
int y = (x + a) / 4;
```

那么有多少个位于闭区间[-8,8]的整数 a 能使得 x 和 y 相等？（　　）

A. 12

B. 13

C. 14

B

a	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8
x=a>>2	-2	-2	-2	-2	-1	-1	-1	-1	0	0	0	0	1	1	1	1	2
(向下取整)																	
x+a	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
(x+a)/4	-2	-2	-2	-1	-1	-1	0	0	0	0	0	0	1	1	1	2	2
向"0"取整	√	√	√		√	√		√	√	√	√	√	√	√	√		√

B

1、对于 IEEE 浮点数，如果减少 1 位指数位，将其用于小数部分，将会有怎样的效果？答：（　　）

C

A．能表示更多数量的实数值，但实数值取值范围比原来小了。

B．能表示的实数数量没有变化，但数值的精度更高了。

C．能表示的最大实数变小，最小的实数变大，但数值的精度更高。

D．以上说法都不正确。

1 8 23

S exp frac

frac 增加 精度提高

exp 减小. 范围 $-2^{exp-1}+2 \sim 2^{exp-1}-1$ 减小

A.B 数量范围 $\cdot 2^{frac}$

$(2^{exp}-2)\cdot 2^{frac}$

$(2^8-2)\cdot 2^{23}$

↓

$(2^7-2)\cdot 2^{24} = (2^8-4)\cdot 2^{23}$

∴表示范围减少

C 最大实数：

最大实数

$1\ \underbrace{1\cdots10}_{7}\ \underbrace{1\cdots\cdots1}_{24}$　bias $= 2^6-1=63$　$2^{63}\cdot(2-\frac{1}{2^{24}})$

$1\ \underbrace{1\cdots10}_{8}\ \underbrace{1\cdots\cdots1}_{23}$　bias $=2^7-1=127$　$2^{127}\cdot(2-\frac{1}{2^{23}})$

最大实数变小；最小实数（S=0 ⇒ S=1）变大.

C

2． 浮点数的阶码没有设计成补码的形式的原因是：

B

A．为了加快四则运算　　B．为了方便排序

C．为了保存更多数值　　D．为了表示特殊值

为了连续性

A. 阶码不影响运算过程.

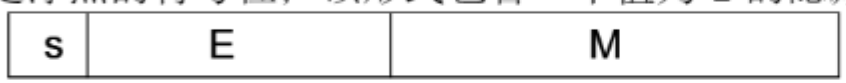
C. 是阶码的位数决定数值多少. 而不是形式.

D. 无特殊值

B

2. 现有一个二进制浮点的表示规则，其中 E 为指数部分，3 比特，且 bias 为 3；M 为小数部分，5 比特，采用二进制补码表示形式，且取值（ ½ ≤ |M|<1）；s 是浮点的符号位；该形式包含一个值为 1 的隐藏位。

D



如果用该形式表示$+5_{10}$，s、E 和 M 分别是：

A. 0, 100, 01100
B. 0, 101, 00100
C. 0, 110, 11010
D. 0, 111, 10101

A. $E = 4-3 = 1$ $\frac{1}{2}+\frac{1}{4}$ $+1$ $2\cdot\frac{7}{4}=\frac{7}{2}$

B. $E = 5-3 = 2$ $\frac{1}{4}$ 不满足 $+1$

C. $E = 6-3 = 3$ $-1+\frac{1}{2}+\frac{1}{8}$ 不满足 $+1$

D. $E = 7-3 = 4$ $-1+\frac{1}{4}+\frac{1}{16}$ $+1$ $16\cdot\frac{5}{16}=5$ ✓

D

D

1. 下面关于 IEEE 浮点数标准说法正确的是（　　）

A. 在位数一定的情况下，不论怎么分配 exponent bits 和 fraction bits，所能表示的数的个数是不变的

B. 如果甲类浮点数有 10 位，乙类浮点数有 11 位，那么甲所能表示的最大数一定比乙小

C. 如果甲类浮点数有 10 位，乙类浮点数有 11 位，那么甲所能表示的最小正数一定比乙小

D. "0111000"可能是 7 位浮点数的 NAN 表示

A. $2 \times (2^{exp}-2) \cdot 2^{frac}$ 表达数的个数

↑ 符号

D. s exp frac

0 | 11 | 1000

NaN

B. 10位：最大数与 exp 大小有关 $2^{(2^{exp}-1)} \cdot \left(2 - \frac{1}{2^{frac-1}}\right)$

C. 最小正数与 exp 有关 $2^{-(2^{exp-1}-2)} \cdot 2^{-frac}$

D