武汉大学计算机学院 2019-2020 学年第一学期期末考试试卷

课程名称：《 系统级程序设计 》（ A 卷） 授课教师：_____

年级：_____专业：_____层次：本科

姓名：_____学号：_____

说明：1、答案一律书写在答题纸上，书写在试卷上或其他地方一律无效。

2、请准确规范书写姓名和学号，否则视为答卷作废。

1. 在 GCC 编译环境下，一个 C 语言源程序文件编译为可执行目标文件需要经过哪些阶段？简述执行各个阶段的程序及其作用。(10 分)

答案：

GCC 编译过程可分为四个阶段（每个阶段 2 分）：

1、预处理阶段。预处理器（cpp)根据以字符#开头的命令，修改原始的 C 程序，得到了另一个 C 程序，通常是以.i 作为文件扩展名。

2、编译阶段。编译器(ccl)将文本文件 hello.i 翻译成文本文件 hello.s，它包含一个汇编语言程序。

3、汇编阶段。接下来，汇编器(as)将 hello.s 翻译成机器语言指令，把这些指令打包成一种叫做可重定位目标程序（relocatable object program)的格式，并将结果保存在目标文件 hello.o 中。

4、链接阶段。链接器(ld)负责将 hello 程序调用的其他函数以某种方式合并到 hello.o 程序中，结果就得到 hello 文件，它是一个可执行目标文件(或者简称为可执行文件），可以被加载到内存中，由系统执行。

2. 假设在一个 int 类型为 32 位长度的机器上运行程序。整型值以补码形式表示，而且只支持算术右移。unsigned 类型的值也是 32 位的。我们产生随机数 x 和 y，并且把它们转换成无符号数，如下左图所示。

对于下右图中每个 C 表达式，你要指出表达式是否总是为 1。如果它总是为 1，那么请描述其中的数学原理。否则，列举出一个使它为 0 的参数示例。注：其中符号==>的含义为蕴含关系，即 A==>B 表示 A=1 时 B=1。(12 分)

```
/* Create some arbitrary values */
int x = random();
int y = random();

/* Convert to unsigned */
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
```

```
A. ( x < y  ) == ( -x > -y )
B. ( x >= 0 ) || ( x < ux  )
C. x > 0 && y > 0 ==> x + y >0
D. ( ( x >> 1 ) << 1 ) <=  x
E. ( x | -x ) >> 31 == -1
F. ux - uy   ==  - ( y - x )
```

答案:

A. No, Let x = TMin32, y = 0. (TMin 的 negative 还是 TMin)
B. No. 当 x>=时表达式值为 1；当 x=Tmin 时，取 ux= 0U，则 x 被 casting 为 Tmax+1>0，所以 x<ux 也未假，整个表达式取值为 0
C. No. x+y 可能发生正溢出（加法溢出）
D. Yes. 有符号数算术右移向负无穷舍入，导致结果变小. (有符号数除法舍入问题)
E. No. 考察 negative 的特性，当 x=0 时, x | -x = 0，右移 31 位仍为 0 （0 的 negative 特性）
F. **Yes**

3. 下面两个基于 IEEE 浮点格式的 9 位表示：

格式 A	格式 B
1. 有一个符号位 2. 有 k=5 个阶码位。阶码偏置值是 15. 3. 有 n=3 个小数位	1. 有一个符号位 2. 有 k=4 个阶码位。阶码偏置值是 7. 3. 有 n=4 个小数位

请将下面给出的格式 A 的位模式，转换为最接近的格式 B 的值。如果需要舍入，你要向 +∞舍入。另外，给出用格式 A 和格式 B 表示的位模式对应的值：要么是整数，要么是小数（例如 17/64）。（共 8 分）

格式 A		格式 B	
位	值	位	值
1 01110 001	-9/16	1 0110 0010	-9/16
1 00111 110			

解答：

格式 A		格式 B	
位	值	位	值
1 01110 001	-9/16	1 0110 0010	-9/16
1 00111 110	**-7/1024**	**1 0000 0111**	**-7/1024**

4. 我们在一个 int 类型值为 32 位的机器上运行程序，这些值以补码形式表示，而且它们都是算术右移的。MAX_INT 是最大的整数，MIN_INT 是最小的整数。假设 x 和 y 都是有符号整数，x=100, y= -89。请填写下表，指明各个表达式的字节值 。注：在该机器上整数值为 1 的字节值表示为 0x00000001。（共 10 分，每空 2 分）

表达式	字节值
x & y	
!x || !y	
x && ~y	
1 + (x << 3) + ~x	
~(~x |(y ^ (MIN_INT+ MAX_INT)))	

解答： X = 0x00000064, y= 0xFFFFFFA7

表达式	字节值
x & y	0x00000024
!x || !y	0x00000000
x && ~y	0x00000040
1 + (x << 3) + ~x	7*x = 700 = 0x000002BC
~(~x |(y ^ (MIN_INT+ MAX_INT)))	x & y = 0x00000024

5. 假设某些内存地址和寄存器中的值如下表所示。填写下表中问号位置，给出对应操作数的值：（共 10 分，每空 2 分）

内存地址	值	寄存器	值
0x200	0x7F	%rax	0x200
0x206	0x47	%rcx	0x2
0x208	0xAB	%rdx	0x3
0x20D	0x39	---	---

操作数	值
%rax	?
$0x208	?
6(%rax)	?
515(%rcx,%rdx)	?
0x5(%rax, %rcx, 4)	?

解答：

操作数	值	注释
%rax	0x200	寄存器
$0x208	0x208	立即数
6(%rax)	0x47	地址 0x206
515(%rcx,%rdx)	0xAB	地址 0x208
0x5(%rax, %rcx, 4)	0x39	地址 0x20D

6. 如下左图是 C 语言代码，右图是该代码对应的汇编代码。（10 分）

（1）哪条指令完成了 P[i][j]的地址的计算？

（2）哪条指令完成了 Q[j][i]的地址的计算？

（3）假设右图中指令前的数字是该指令的存储地址，那么全局变量 P 和 Q 的地址分别是多少？

（4）M、N 的值分别是多少？

解答:

（1）地址为 60 的指令，存于%esi

（2）地址为 79 的指令，存于%rcx+%rax*4

（3）59+167，72+238

（4）N=28/4=7；M=12/4=3。

```
int P[M][N];
int Q[N][M];

int sub(int i, int j)
{
    return P[i][j] - Q[j][i];
}
```

```
40: pushq   %rbp
41: movq    %rsp, %rbp
44: movl    %edi, -4(%rbp)
47: movl    %esi, -8(%rbp)
4a: movslq  -4(%rbp), %rax
4e: imulq   $28, %rax, %rax
52: leaq    167(%rip), %rcx
59: addq    %rax, %rcx
5c: movslq  -8(%rbp), %rax
60: movl    (%rcx,%rax,4), %esi
63: movslq  -8(%rbp), %rax
67: imulq   $12, %rax, %rax
6b: leaq    238(%rip), %rcx
72: addq    %rax, %rcx
75: movslq  -4(%rbp), %rax
79: subl    (%rcx,%rax,4), %esi
7c: movl    %esi, %eax
7e: popq    %rbp
7f: retq
```

7. 一个 C 语言的函数原型如下左图，用 GCC –O1 -S 编译实际的函数得到的汇编代码如下右图。请分析每一条汇编语句的作用，并填写出 C 代码中缺失的部分。（共 10 分）

```
long func(long a, long b,long *u)
{

    long k,n= ____;
    *u = ____;
    if(a<b)
    {
         for(k  = a; ____; k++)
         {
            n ____;
            *u____;
         }
     }
    return n;
}
```

```
// 假设a in %rdi, b in %rsi, u in %rdx, n in %rax)
func:
    movq    $1, (%rdx)
    movl    $0, %eax
    cmpq    %rsi, %rdi
    jge     .L1
    addq    $1, %rsi
    movl    $1, %r8d
.L3:
    movq    %rdi, %rcx
    imulq   %rdi, %rcx
    leaq    (%rcx,%rcx,2), %rcx
    leaq    (%rax,%rcx,2), %rax
    leaq    3(%rdi), %rcx
    imulq   %rcx, %r8
    addq    $1, %rdi
    cmpq    %rsi, %rdi
    jne     .L3
    movq    %r8, (%rdx)
.L1:
    ret
```

解答： long func(long a, long b,long *u)
{
long k,n=0;
*u = 1;
if(a<b)
for(k = a;k <= b; k++)
{
n += 6 * k * k;
*u *= 3 + k;
}
return n;
}

8. 下面左图是一个 C 代码，右图是该代码对应的输出结果。提示：字符'0'和'9'的 ascii 码分别为 0x30 和 0x39。（12 分）

（1）依次分析第 0-4、5-9、10 次循环迭代的输出结果。

（2）解释为什么循环总共只迭代了 11 次。

（3）画出该函数的栈帧结构（包含 buf，i，j 等局部变量的位置关系即可）。

```
int main()
{
    int i = 0;
    int j = 0x39393939;
    char buf[6] = "hello";

    for(; i < 20; ++ i)
    {
        buf[i] = '0';
        printf("%d:\t%s\n",i, buf);

    }
    return 0;
}
```

```
0:      0ello
1:      00llo
2:      000lo
3:      0000o
4:      00000
5:      0000009999
6:      0000000999
7:      0000000099
8:      0000000009
9:      0000000000
48:     00000000000
Process exited with status 0
```

解答:

buf[6]-buf[9]修改了 j；buf[10]修改了 i。

9. 多项式$a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$求值的直接计算函数如下：

```
double poly(double a[], double x, long degree){
    long i;
    double result = a[0];
    double xpwr=x;
    for (i =1;i<= degree; i++){
        result += a[i] *xpwr;
        xpwr= x*xpwr;
    }
    return result;
}
```

假设我们忽略计算顺序对浮点运算结果的影响。试着用我们讲过的并行累积技术（2x2：教材图 5-21）写出这个函数更快的版本。（8 分）

```
double poly2(double a[], double x, long degree){
  long i;
  double result0 = 0;
  double result1 = 0;
  double x2 = x*x;
  double xpwr = x;
  for (i = 1; i< degree; i+=2){
        double nxpwr = x2 * xpwr;
        result0 += a[i] * xpwr;
        result1 += a[i+1] * xpwr;

                xpwr= nxpwr;
```

```
    }
    result0 += x*result1;
    for (; i<=degree; i++){
        result0 += a[i]*xpwr;
        xpwr = x* xpwr;
      }
    return result0;
}
```

10. 考虑下面的程序，它由两个目标模块组成：

```
4.  /*foo.c*/
5.  #include <stdio.h>
6.  void p2(void);
7.  int a=3;
8.  int b=5;
9.  int main()
10. {
11.     p2();
12.    prinf("%d,%d\n",a,b);
13.     return 0;
14. }
```

```
4.  /*bar6.c*/
5.  #include <stdio.h>
6.
7.  double a;
8.
9.  void p2()
10. {
11.      a=9.8;
12.      printf("%lf \n",a);
13. }
```

当在 x86-64 Linux 系统中编译和执行这个程序时，foo.c 文件的第 9 行输出语句输出的变量 a、b 的值不是 3、5，请分析原因是什么？在不修改两个文件的全局变量命名的前提下，如何修改可以改正错误？（10 分，每问 5 分）

解答：bar6.c 中定义的全局变量 a 是和 foo.c 的强符号同名的，因此 bar6.c 中第 8 行的 9.8 实际可能被赋给 foo.c 中的变量 a 和 b ，从而产生错误。

修改方法：bar6.c 第 4 行改为

static double a;