四川大学期末考试试题（闭卷）

（2017~2018 学年第 1 学期） A 卷

课程号：311143040 课程名称：系统级编程 任课教师：____________

适用专业年级：软件工程 2015 级 学号：____________ 姓名：____________

考生承诺

我已认真阅读并知晓《四川大学考场规则》和《四川大学本科学生考试违纪作弊处分规定（修订）》，郑重承诺：

1、 已按要求将考试禁止携带的文具用品或与考试有关的物品放置在指定地点；

2、 不带手机进入考场；

3、 考试期间遵守以上两项规定，若有违规行为，同意按照有关条款接受处理。

考生签名：

题　　号	一(40%)	二(10%)	三(13%)	四(12%)	五(15%)	六(10%)
得　　分						
卷面总分		教师签名		阅卷时间		

注意事项：1. 请务必将本人所在学院、姓名、学号、任课教师姓名等信息准确填写在试题纸和添卷纸上；

2. **请将答案全部填写在本试题纸上；**

3. 考试结束，请将试题纸、添卷纸和草稿纸一并交给监考老师。

评阅教师	得分

一、单项选择题（本大题共 20 小题，每小题 2 分，共 40 分）

提示：在每小题列出的四个备选项中只有一个是符合题目要求的，请将其代码填写在下表中。错选、多选或未选均无分。

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20

1. Which one of the following languages is hardware dependent?
   A. C
   B. Assembly
   C. C++
   D. JAVA

2. Which expression is false?
   A. !(-5)
   B. ~0x01
   C. ~0x00
   D. !0

3. Given the following code segments, which statement is NOT true?
   Statement 1: char str1[]="abc";
   Statement 2: char *str2 ="abc";

Statement 3: char *str3 = “abc”

I. statement 1 requires 3 bytes from stack
II. statement 2 requires 4 bytes from stack
III. expression (str2 = =str3) is true
IV. expression (str1 = =str2) is true

A. II III and IV　　B. II and III
C. I and IV　　D. II and IV

4. The machine code generated from source code by a compiler
A. executes more quickly than the source code
B. does not preserve all the information given in the source code
C. can be easily inspected to check the correctness of the compiler
D. associates variable values with their names

5. After execution of the following code in a 32 bits system, which statement is TRUE?

```
char a[8]={‘1’,’2’,’3’,’4’,’5’,’6’,’7’,’8’};
double *p = (double *)&a;
*p= -1.5f;
```

A. a[0] is ‘1’　　B. a[7] is 0
C. a[8] is 1　　D. None of the above

6. In C, local variables allocated inside functions are allocated
A. on the stack　　B. in static storage
C. in the heap　　D. in a fifo

7. Which statement will copy the value in register BX to AX?
A. LDS AX,BX　　B. LEA AX,BX
C. MOV AX,BX　　D. STOS AX,BX

8. Consider the following segment of C source code.

```
int a = 8;
int b = *&a;
```

What is the value of variable b at the end of execution of the segment?
A. a　　B. (int) &a
C. (int) &b　　D. &a

9. A memory leak is caused by a
A.failure to free allocated memory
B.bug in which too much memory is allocated, causing internal fragmentation
C.bug in the memory allocator that fails to free memory
D.function that allocates a large amount of memory from the heap

10. What properties of a variable are specified by the static keyword in c?
i.the variable will be statically allocated.
ii.the variable name will be visible only to functions defined within the same file.
iii.the variable's value does not change very often. the compiler uses this fact to focus optimizations on other variables.

A. i only　　　　B. i and iii only.
C. i and ii only.　　　　D. iii only

11. We use dynamic memory because:
A. The heap is significantly faster than the stack.
B. The stack is prone to corruption from buffer overflows.
C. Storing data on the stack requires knowing the size of that data at compile time.
D. None of the above.

12. Which one of the following statements is NOT true of storage allocators?
A. In the best case, coalescing with boundary tags is linear in the number of free blocks.
B. Segregated free list typically approximates best fit search.
C. Payloads must be aligned to some boundary.
D. Explicit lists are typically faster than implicit list.

13. A garbage collector starts from some "root" set of places that are always considered "reachable", such as
i. CPU registers
ii. stack
iii. global variables
A. i & ii　　　　B. i & iii
C. iii　　　　D. all of them

14. Which on the following is a reason next-fit might perform better than first-fit?
A. If a large number of small blocks are stored at the beginning of the free list, next-fit avoids walking through those small blocks upon every allocation.
B. First fit requires a traversal of the entire free list, but next-fit does not.
C. First-fit requires that both allocated and unallocated blocks be examined, and next-fit examines only free blocks.
D. Next-fit is an approximation of best-fit, so it reduces internal fragmentation compared to first-fit.

15. Which of the following cannot be used to estimate time consumption of a program?
A. RDTSC
B. QueryPerformanceFrequency
C. GetTickCount
D. clock() / CLOCKS_PER_SEC

16. from the time when a c program is written, to the time when it is running as a process on windows, what should be done?
i.compile
ii.link
iii.load.
A.i and ii only.
B.i and iii only.

C.i, ii and iii.
D.ii only.

17. a lock is a software mechanism that
A.temporarily makes memory read-only.
B.limits access to a critical section.
C.implements password protection to data.
D.prevents execution except in debug mode.

18. which of the following is not optimization technique?
A.code motion
B.loop unrolling
C.constant folding
D.memory aliasing

19. in ia32 or x86, which exception is used to implement system call
A. interrupt
B. trap
C. fault
D. Abort

20. which facts about the cache can be determined by calling the following function?

```
int data[1 << 20];
void callee(int x) {
    int i, result;
    for (i = 0; i < (1 << 20); i += x) {
        result += data[i];
    }
}
```

i cache line size
ii cache size
iii cache speed
A. i and ii only
B. i only
C. i and iii only
D. i, ii, and iii

评阅教师	得分

二、bit operation（本大题共 10 分）。

Now complete the following functions according to the following rules.
Each "Expr" is an expression using ONLY the following:
1. Integer constants 0 through 0xFFFFFFFF inclusive.
2. Function arguments and local variables (no global variables).
3. Some of the problems restrict the set of allowed operators.
You are expressly forbidden to:
1. Use any control constructs such as if, do, while, for, switch, etc.

2. Define or use any macros.
3. Define any additional functions in this file.
4. Call any functions.
5. Use any other operations, such as &&, ||, -, ?, or [] :
6. Use any form of casting.

You may assume that your machine:
1. Uses 2s complement, 32-bit representations of integers.
2. Performs right shifts arithmetically.
3. Has unpredictable behavior when shifting an integer by more than the word size.

```
/* NegativeNum using only ~ and & , ignore 0
* Example: NegativeNum(-5) retrun -5 , NegativeNum(5) retrun -5, Negative(0) can
return any value
* Legal ops: ~ &
* Max ops: 8 */
int NegativeNum (int x)
{




}
```

```
/*
 * SetByte - Set byte n from word x to 0xFF
 *    Bytes numbered from 0 (LSB) to 3 (MSB)
 *    Examples: SetByte(0x12345678,1) = 0x1234FF78
 *    Legal ops: ! ~ & | << >>
 *    Max ops: 6*/
int SetByte (int x,int n)
{




}
```

评阅教师	得分

三、stack discipline（本大题共 13 分）。

Consider the following C code and assembly code for two mutually recursive functions:

```
int even(unsigned int n)       0x080483e4 <even+0>:    push   %ebp
{                              0x080483e5 <even+1>:    mov    %esp,%ebp
    if(!n)                     0x080483e7 <even+3>:    sub    $0x8,%esp
    {                          0x080483ea <even+6>:    cmpl   $0x0,0x8(%ebp)
        return 1;              0x080483ee <even+10>:   jne    0x80483f9 <even+21>
    }                          0x080483f0 <even+12>:   movl   $0x1,-0x4(%ebp)
                               0x080483f7 <even+19>:   jmp    0x804840a <even+38>
    return odd(n - 1);         0x080483f9 <even+21>:   mov    0x8(%ebp),%eax
}                              0x080483fc <even+24>:   sub    $0x1,%eax
                               0x080483ff <even+27>:   mov    %eax,(%esp)
                               0x08048402 <even+30>:   call   0x804840f <odd>
                               0x08048407 <even+35>:   mov    %eax,-0x4(%ebp)
                               0x0804840a <even+38>:   mov    -0x4(%ebp),%eax
                               0x0804840d <even+41>:   leave
                               0x0804840e <even+42>:   ret

int odd(unsigned int n)        0x0804840f <odd+0>:     push   %ebp
{                              0x08048410 <odd+1>:     mov    %esp,%ebp
    if(!n)                     0x08048412 <odd+3>:     sub    $0x8,%esp
    {                          0x08048415 <odd+6>:     cmpl   $0x0,0x8(%ebp)
        return 0;              0x08048419 <odd+10>:    jne    0x8048424 <odd+21>
    }                          0x0804841b <odd+12>:    movl   $0x0,-0x4(%ebp)
                               0x08048422 <odd+19>:    jmp    0x8048435 <odd+38>
    return even(n - 1);        0x08048424 <odd+21>:    mov    0x8(%ebp),%eax
}                              0x08048427 <odd+24>:    sub    $0x1,%eax
                               0x0804842a <odd+27>:    mov    %eax,(%esp)
                               0x0804842d <odd+30>:    call   0x80483e4 <even>
                               0x08048432 <odd+35>:    mov    %eax,-0x4(%ebp)
                               0x08048435 <odd+38>:    mov    -0x4(%ebp),%eax
                               0x08048438 <odd+41>:    leave
                               0x08048439 <odd+42>:    ret
```

Imagine that a program makes the procedure call **even(3)**. Also imagine that prior to the invocation, the value of ESP is 0xffff1000 - that is, 0xffff1000 is the value of ESP immediately before the execution of the **call** instruction.

1. Note the the call even(3) will result in the following function invocations: even(3), odd(2), even(1), and odd(0). Full in the stack diagram with the values that would be present immediately before the execution of the ret instruction for odd(0). Cross out each blank for which there is insufficient information to complete.
2. What are the values of ESP and EBP immediately before the execution of the ret instruction for odd(0)?

ESP= ____________________________

EBP=_____________________________

```
+--------------------------------+
|                                |  0xffff1004
+--------------------------------+
|                                |  0xffff1000
+--------------------------------+
|                                |  0xffff0ffc
+--------------------------------+
|                                |  0xffff0ff8
+--------------------------------+
|                                |  0xffff0ff4
+--------------------------------+
|                                |  0xffff0ff0
+--------------------------------+
|                                |  0xffff0fec
+--------------------------------+
|                                |  0xffff0fe8
+--------------------------------+
|                                |  0xffff0fe4
+--------------------------------+
|                                |  0xffff0fe0
+--------------------------------+
|                                |  0xffff0fdc
+--------------------------------+
|                                |  0xffff0fd8
+--------------------------------+
|                                |  0xffff0fd4
+--------------------------------+
|                                |  0xffff0fd0
+--------------------------------+
|                                |  0xffff0fcc
+--------------------------------+
|                                |  0xffff0fc8
+--------------------------------+
|                                |  0xffff0fc4
+--------------------------------+
|                                |  0xffff0fc0
+--------------------------------+
```

评阅教师	得分

四、Performance Optimization（本大题共 12 分）。

1． Consider the following functions:

```
int min(int x, int y) { return x < y ? x : y; }
int max(int x, int y) { return x < y ? y : x; }
void incr(int *xp, int v) { *xp += v; }
int square(int x) { return x * x;   }
```

Here are three code fragments that call these functions

```
A.  for (i = min(x, y); i < max(x,y); incr(&i, 1)) { t += square(i);   }
B.  for (i = max(x, y) - 1; i >= min(x,y); incr(&i, -1)) { t += square(i);   }
C.  int low = min(x, y);
     int high = max(x, y);
    for (i = low; i < high; incr(&i, 1)) { t += square(i); }
```

assume x = 10 and y = 100. Fill in the table below indicating the number of times each of the four functions is called for each of these code fragments.

Code	min	max	incr	square
A				
B				
C				

2． The following problem illustrates the way memory aliasing can cause unexpected program behavior. Consider the following procedure:

```
// Swap value x at xp with value y at yp
void Swap(int *xp, int *yp)
{
      *xp = *xp + *yp; // x + y
      *yp = *xp - *yp; // x+y-y=x
      *xp = *xp - *yp; // x+y-x=y
}
```

Can this procedure be used to swap two values? And why?

评阅教师	得分

五、Cache（本大题共 15 分）。

1． Analyse the locality the following code with nested loop. Note：This is the pseudo-code.

```
  integer data(M, N);      （5 分 ）
  for (i = 0;   i < M;   i=i+1) {
     for (j = 0 ;   j < N;   j=j+1) {
          sum += data( i ,   j );
     }
  }
```

2.You are evaluating a system's cache performance on a machine with a 1024-byte direct-mapped data cache with 16-byte blocks (B =16). You are given the following definitions: （10 分 ）

```
struct position {
    int x;
    int y;
    int z1;
    int z2;
};
struct position grid[16][16];
int total_x = 0, total_y = 0, total_z = 0;
int i, j, k;
```

You should also assume:

- sizeof(int) == 4.
- grid begins at memory address 0.
- The cache is initially empty.
- The only memory accesses are to the entries of the array grid. Variables i, j , total_x, total_y and total_z are stored in registers.

Determine the cache performance of the following code:

```
for (i = 0; i < 16; i++)
{
      for (j = 0; j < 16; j++)
      {
          total_x += grid[i][j].x;
          total_y += grid[i][j].y;
          total_z += grid[i][j].z1+ grid[i][j].z2;
      }
}
```

A. What is the total number of reads? _______.
B. What is the total number of reads that miss in the cache? _______ .
C. What is the miss rate? _______.
D. What would the miss rate be if the cache were twice as big?
And explain every answer briefly.

评阅教师	得分

六、Answering Question（本大题共 10 分）。

1. What is the mean of our textbook of CSAPP? Please give your understanding about it, such as what you have learned from this course or this textbook.（5 分）

2. What is exception?　Please list the four types of exception in IA32, and draw a figure to show the different.（5 分）