CS 211
Spring 2018
Midterm
3/06/2018
Time limit: 80 Minutes

Name: Answer
NetID: _____

Instructor: Abhishek Bhattacharjee

This exam contains 10 pages (including cover page) and 11 questions, for 100 points total.

Points Table (for grader use only)

Question	Points	Score
1	5	
2	4	
3	20	
4	10	
5	10	
6	15	
7	10	
8	6	
9	6	
10	6	
11	8	
Total:	100	

By signing below, I pledge that I have not violated the terms of Rutgers University's academic integrity policy[1]. I have not consulted other students during the exam, I have not relied on notes or textbooks, I have not used a calculator, and I have not used any other manner of study and/or digital aids. Furthermore, I have not provided any aid to other students taking the exam during the examination period. I understand that violating the terms of this pledge will mean that I will be subject to disciplinary action at the university-level.

Student's signature: _____

[1] http://academicintegrity.rutgers.edu

1. (5 points) First, consider the following declarations:

```
// declarations
int i,j,k[10]={2,4,6,8,10,12,14,16,18,20};
int *ip;
char a,b,c[10]={11,12,13,14,15,16,17,18,19,20};
char *cp;
void *vp;

typedef struct stst {
    int i;
    char *p;
} st;

// assignment statements
st x;
ip = &k[4];
vp = ip;
cp = &c[0];
x.p = cp+3;
x.i = 27;
```

Evaluate the following expressions (if the expression is an error, indicate that with "X"):

(a) (1 point) $*(ip+2)+c[3];$ 28

(b) (1 point) $(*cp+1);$ 12

(c) (1 point) $*(ip)+6;$ 16

(d) (1 point) $vp = \&x; ((st*)vp)->p;$ X or 14

(e) (1 point) $*(ip-2);$ 6

2. (4 points) In the programming assignment 1, fourth problem(hash table), what was the strategy to deal with conflict resolution?

A. Double hashing

B. Linear probing

C. Separate chaining with linked lists

D. Cuckoo hashing

3. (20 points) For the following items, write **T** if the sentence is *True*, **F** if it is *False*, or **X** if you don't know the answer. You get +1 point for each correct answer, 0 points for an item answered with an X, −1 point for each wrong answer.

   Assume x86 32-bit architecture.

   T The stack grows towards lower addresses.

   F *Two's complement* notation has two different representations of zero, this is the reason that its name is *two's complement.*

   F Accessing data from memory (*RAM*) is faster than from registers.

   T Dynamic memory regions, i.e., regions of the memory allocated by *malloc*(), are located in the heap.

   T The line of C code: *val = val << 4;* multiplies *val* by 16.

   T $0xDEADBEEF$ = 1101 1110 1010 1101 1011 1110 1110 1111 b

   T The C *for* loop: for(i = 0; i ≤10; i++) iterates over the loop 11 times

   F In two's complement notation, the most significant bit of a negative number is zero.

   T IEEE floating-point representation uses signed magnitude format.

   F Arithmetic operations on floating point numbers are faster than on integers.

   T The mantissa field of a floating point number can be determined for normalized representation by assuming a single 1 at the left of the decimal point.

   F A Von Neumann machine stores data in memory and instructions on registers.

   T ISA stands for Instruction Set Architecture. It is the contract between software and hardware on what instructions are supported by the machine.

   F *c = (char **)malloc(sizeof(char*) * 4);* allocates 4 bytes of memory in the heap;

   T A pointer in C is a number that stores a memory address.

   F C is an interpreted language.

   F %ebp and %esp are registers that always point to the stack and %ebp ≤ %esp.

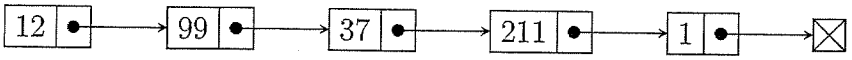
   F General purpose registers in a 32-bit machine are 32-bit in size, meaning that instructions' operands are also 32-bit in size. The maximum amount of memory that a 32-bit machine can address is, then, 32GB.

   T Two's complement −1 in binary is: 1111 1111 1111 1111 1111 1111 1111 1111 b

   F char, int, long, float, double, void, and string are data types in C.

4. (10 points) Consider the code below. You will see a struct, and also a function, *printList*, that prints the linked list from its head to its tail. Your task is to implement *printReverseList*, a function that prints the list from its tail to its head.

Example, for the following linked list:



printList prints: $12 \rightarrow 99 \rightarrow 37 \rightarrow 211 \rightarrow 1 \rightarrow NULL$

printReverseList should print: $1 \rightarrow 211 \rightarrow 37 \rightarrow 99 \rightarrow 12 \rightarrow NULL$

```
typedef struct node_ {
    int data;
    struct node_ *next;
} node;
void printList(node *head) {
    while(head) {
        printf("%d-->", head->data);
        head = head->next;
    }
    printf("NULL\n");
}
void printReverseList(node *head){
    if (head == NULL)
        return;
    printReverseList (head → next);   ← Recursive
    printf ("%d-->", head → data);
```

- Actually reversing the list is also okay!
- Using array is also okay!
- Other... if it works I gave you points and also you've got some partial credit

```
}
```

5. (10 points) What is the output of the following program?

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE (5)
void matTr(int **mat, int n) {
    int i, j;
    for(i = 0; i < n; i++) {
        for(j = i+1; j < n; j++) {
            int tmp = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = tmp;
        }
    }
}
int main(int argc, char** argv) {
    int **mat, i, j;
    mat = (int **) malloc(sizeof(*mat) * SIZE);
    for(i = 0; i < SIZE; i++){
        mat[i] = (int *) malloc(sizeof(**mat) * SIZE);
        for(j = 0; j < SIZE; j++)
            mat[i][j] = (i*2 + j) % SIZE;
    }
    matTr(mat, SIZE);
    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }
    for(i = 0; i < SIZE; i++)
        free(mat[i]);
    free(mat);
    return 0;
}
```

Output:

0 2 4 1 3
1 3 0 2 4
2 4 1 3 0
3 0 2 4 1
4 1 3 0 2

6. (15 points) For the following questions, assume the numbers are 16 bits.

   1. Convert two's complement $0xC521$ to:
      (a) (2 points) Decimal: -15071
      (b) (2 points) Octal: 142441
      (c) (2 points) Binary: 1100 0101 0010 0001
   2. Convert -512 (decimal) into hexadecimal with
      (a) (3 points) Sign and magnitude representation: 0x B200
      (b) (3 points) One's complement: 0xFDFF
      (c) (3 points) Two's complement: 0x FE00

7. (10 points) Implement a function that prints the hexadecimal representation of a decimal number (16-bit, two's complement). You are not allowed to use any library API or *printf*() specifier.

```
#include <stdio.h>

void dec2hex(short int num) {
    char hexa[5];
    int i, j = 0;
    while (num != 0) {
        int temp = 0;
        temp = num % 16;
        if (temp > 9)
            hexa[i] = temp - 10 + 'A';
        else
            hexa[i] = temp + '0';
        num = num/16
        i++
    }
    for (j=0, --i; j < i; j++, i--) {
        char tmp = hexa[i];
        hexa[i] = hexa[j];
        hexa[j] = tmp;
    }
    printf("Hexa of %d is 0x%s.\n", num, hexa);
}
```

8. (6 points) As an application of the property that $a \oplus a = 0$ ($\oplus$ is symbol for XOR) for any bit vector a, consider the following program:

```
void inplace_swap(int *x, int *y) {
    *y = *x ^ *y;   // step 1
    *x = *x ^ *y;   // step 2
    *y = *x ^ *y;   // step 3
}
```

As the name implies, the effect of this procedure is to swap the values stored at the locations denoted by the pointer variables x and y. Note that unlike the usual technique for swapping two values, we do not need a third location to temporarily store one value while we are moving the other.

Starting with values a and b in the locations pointed by x and y, respectively, fill the table that follows, giving the values stored at the two locations after each step of the procedure. Use the properties of XOR to show that the desired effect is achieved. Recall that every element is its own additive inverse (that is, $a \oplus a = 0$).

step	*x	*y
Initially	a	b
step 1	a	a^b
step 2	(a^a)^b = b	a^b
step 3	b	a^b^b = a

9. (6 points) Consider the content of three files below.

File1

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5   printf("CS211\n");
6   return 0;
7 }
```

File2

```
1 #ifdef __CS211__
2 #define __CS211__
3
4 #define ABC (100)
5 typedef struct abc {
6     int a, b, c;
7 };
8
9 #endif
```

File3

```
C=gcc
CFLAGS=-I.

hello: hello.o
 $(CC) -o hello ↩
     hello.o -I.
```

Identify which category each one of the files belong to:

(a) (2 points) Header file: File 2

(b) (2 points) Source file: File 1

(c) (1 point) Makefile: File 3

(d) (1 point) What is the simplest command-line statement to compile this code?

make

10. (6 points) Write bellow the function $CS211$ that takes integer n as input and iterates over all the numbers from 1 up to n(inclusive). If the number is divisible by 3, then print "*Rutgers*". If the number is divisible by 5, then print "$CS211$". If the number is divisible by 3 and 5, then print "*Rutgers* $CS211$". Otherwise, print the number. For example, $CS211(16)$ prints as follows:

1
2
Rutgers
4
CS211
Rutgers
7
8
Rutgers
CS211
11
Rutgers
13
14
Rutgers CS211
16

```
void CS211(int n) {
    for (int i=1; i<=n; i++) {
        if (i%3==0 && i%5==0)
            printf("Rutgers CS211\n");
        else if (i%3==0)
            printf("Rutgers\n");
        else if (i%5==0)
            printf("CS211\n");
        else
            printf("%d\n");
    }
}
```

11. (8 points) Considering the following assembly code, answer the questions below.

```
1  .section .text
2  .globl main
3  main:   push %ebp          # stack frame setup
4          mov %esp, %ebp     # stack frame setup
5          mov $0x5, %ecx
6          mov %ecx, %edx
7  loop:   cmp $0x1, %ecx
8          jle end
9          dec %ecx
10         imul %ecx, %edx
11         jmp loop
12 end:    mov %edx, %eax
13         mov %ebp, %esp   # stack frame cleanup
14         pop %ebp         # stack frame cleanup
15         ret
```

We haven't covered in class yet how to setup and cleanup the stack frame (activation record) of a function, you won't need this information to answer this question. You can safely ignore the lines $1-4$, and $13-15$ to answer these questions.

Answer the questions:

(a) (2 points) Content of *%eax* (in decimal) after executing the code: 120

(b) (2 points) Number of times line 7 is executed: 5

(c) (2 points) Number of memory references(ignore lines $1-4, 13-15$): 25 or 30

(d) (2 points) Concisely explain what this code does: factorial(5), 5!, fact(5)