ECE220: Computer Systems and Programming

Spring 2017 - Midterm Exam 2

March 27, 2017

1. This is a closed book exam except for 1 sheet of hand-written notes
2. You may not use any personal electronic devices, such as cellphone and calculator
3. Absolutely no interaction between students is allowed
4. Illegible handwriting will be graded as incorrect

Select the Lecture Section You Will Attend to Pick Up Booklet

[ ] Prof. Mitra (BL4, 9:30 am)

[ ] Prof. Patel (BL2, 11:00 am)

[ ] Prof. Chen (BL, 12:30 pm)

[ ] Prof. Hu (BL3, 3:00 pm)

Name:______________________________

NetID:______________________________

Room:______________________________

Question 1 (20 points): ________________

Question 2 (30 points): ________________

Question 3 (20 points): ________________

Question 4 (30 points): ________________

Total Score: ________________

Problem 1 (20 points): Find Max Profit

For this problem, write a function maxProfit that takes as input an array of integers prices, and an integer n that indicates size of prices. The i-th element in the array is the price of a stock on the i-th day. The function should return the maximum profit achievable by buying one share of stock and then selling that share, possibly on the same day or a later day.

Example 1:
Input: [2 7 3 1 2 8], n = 6
Return value: 7
// Max profit = Buy at 1 on 4^{th} day, Sell at 8 on 6^{th} day

Example 2:
Input: [5 4 3 2 1], n = 5
Return value: 0
// Max profit = Buy at 1 on 5^{th} day, Sell at 1 on 5^{th} day

Example 3:
Input: [4 3 2 3 6 4 2 1 3 4], n = 10
Return value: 4
// Max profit = Buy at 2 on 3^{rd} day, Sell at 6 on 5^{th} day

Hint: you can use the following algorithm to solve this problem.

1. Assume buying the stock on the 1^{st} day (first element) and then traverse the array to find max profit by selling on the same day or a later day.
2. Perform step 1 on each subsequent day (the rest of the elements in the array).

```
int maxProfit(int *prices, int n)
{
/* YOUR CODE STARTS HERE */










/* YOUR CODE ENDS HERE */
}
```

Problem 2 (30 points): C to LC-3

You will provide the data values that appear on the run-time stack during the execution of simple C program. Part of the stack frame for function main is shown in the memory table in Part A and Part B. R6 is the stack pointer and R5 is the frame pointer.

To ensure consistent answers:
a) after main's callee setup, R5 is 0xBCDB. Derive the value of R6 from this information
b) place `n` above `accum` in the stack.

Part A: In the memory table, draw the stack at the point that JSR/JSRR begins transferring control to find_ngon during the execution of the statement "z = find_ngon(x, n);". **Also indicate the values of R5 and R6 at this point of program execution.**

Part B: In the memory table, draw the stack right before "n = 0;" is executed; that is, after find_ngon has set up the stack but before it has begun executing. **Also indicate the values of R5 and R6 at this point of program execution.**

Part C: Convert the find_ngon function from C to an LC-3 subroutine with correct use of the run-time stack. Be aware of the requirements on code length (4 instructions each for callee setup and teardown) and execution (every local variable write is reflected in the stack frame).

```
int find_ngon(int perimeter, int side_len){
  int accum, n;
  n = 0;
  accum = perimeter;
  while (accum > 0) {
    accum -= side_len;
    n++;
  }
  return n;
}

int main() {
  int perim = 15;
  int side = 3;
  int z;

  z = find_ngon(perim, side);

  return 0;
}
```

Part A (10 points): fill the stack with name (value is not needed)

xBCC8	
xBCC9	
xBCCA	
xBCCB	
xBCCC	
xBCCD	
xBCCE	
xBCCF	
xBCD0	
xBCD1	
xBCD2	
xBCD3	
xBCD4	
xBCD5	
xBCD6	
xBCD7	
xBCD8	
xBCD9	z
xBCDA	side
xBCDB	perim

Indicate the values of R5 and R6 at this point of program execution

R5 = ________________________; R6 = ________________________

Part B (10 points): fill the stack with name (value not needed)

xBCC8	
xBCC9	
xBCCA	
xBCCB	
xBCCC	
xBCCD	
xBCCE	
xBCCF	
xBCD0	
xBCD1	
xBCD2	
xBCD3	
xBCD4	
xBCD5	
xBCD6	
xBCD7	
xBCD8	
xBCD9	z
xBCDA	side
xBCDB	perim

Indicate the values of R5 and R6 at this point of program execution

R5 = _______________________; R6 = _______________________

Part C (10 points):

```
;; FIND_NGON Subroutine
; callee setup – push bookkeeping info and local variables
; This section shall be no longer than 4 instructions




; function logic.
; Every variable write MUST be reflected in memory.




; callee tear-down – pop local variables, bookkeeping info
; This section shall be no longer than 4 instructions,
; INCLUDING any control flow transfers
```

Problem 3 (20 points): Permutation using swap function

The goal of this problem is to print all possible permutations of a given string with no duplicate characters. For example, if the given string is "ABC", your program should print out ABC, ACB, BCA, BAC, CAB and CBA.

In Lab 7 we did permutation using backtracking with a mask array of chosen elements. This time, you will use a different backtracking method which changes the characters in place in the string to generate permutations. The permutation function will use the familiar swap function, so let us start with that. The function declarations are provided; complete the functions.

Part A (5 points): Write the swap function to swap the character pointed by a and b.

```
void swap(char *a, char *b)
{
/* YOUR CODE STARTS HERE */









/* YOUR CODE ENDS HERE */
}
```

Part B (10 points): The permutation function takes three parameters: char* s is the input string; *int left* and *right* are indices that define the substring that is being permuted. Here is the idea of the permutation function:

The base case occurs when the length of the substring is 1. In this case, there is nothing to permute and the input string is just printed out.

In the recursive case, for each character *c* in the range [*left*, *right*] the following three things are done: *(i)* first the *left* char is swapped with c, (ii) the permutation function is recursively called on the smaller substring defined by [*left+1, right*], and finally, (iii) the swap in (i) is backtracked (i.e., undone).

Your implementation should not make copies of the input string. Our implementation used total 8 lines of C code.

```
void permutation(char *s, int left, int right)
{
   int i;
   if (______________________________)/* Base case here */
     printf("%s\n", s);
   else
   {
       for (i______________; ______________; i++)
       {
         ______________________________/* Write your code */

         ______________________________/*                */

         ______________________________/*                */
       }
   }
}
```

Part C (5 points): If the given string is "DEF", what will be the output. Assume your implementation for the permutation is correct. (5 points)

```
int main()
{
    char str[] = "DEF";
    permutation(str, 0, 2);
    return 0;
}
```

Your Answer:

1	
2	
3	
4	
5	
6	

Problem4: Concepts (30 points)

Part A (10 points): print the addresses of an array
Danny created an array(arr1) that simply contains 5 integers. He wants to print out the addresses of arr1 by first assigning the addresses of arr1 to arr2 and then printing out using arr2 to see the arr1's addresses. However, the program does not work since he does not understand the concept of address, pointer, and array at all. As a good friend, you need to help him out by modifying his code. Note: (1) you can assume %p is the correct format to print address; (2) line 12 must not use arr1 after preprocessing.

```
line
0   #include <stdio.h>
1   int main() {
2       int arr1[5]={1,2,3,4,5};
3       int arr2[5];
4
5       int i;
6       for(i=0; i<5; i++){
7               arr2[i] = &(arr1+i);
8       }
9
10      for(i=0; i<5; i++){
11              /*assume %p is correct format to print address*/
12              printf("%p\n", &(arr2[i]) ); /*must not use arr1 after preprocessing*/
13      }
14
15  }
```

Please provide your answer below as shown in the example. (Must write the full syntax of the changed line)
Example:
line 12 should be changed to
printf("%p\n", x);

Your answer:

Part B (10 points): Binary Search

An engineer is implementing a recursive binary search in C to use on an integer array sorted in **descending order** (largest to smallest). The code is as follows. Please help this engineer finish the function by filling in the correct parameters in line 5 and 6.

```
line

1    int binarySearch(int arr[], int start, int end, int item){

2        if (end >= start){

3            int mid = (end + start)/2;

4            if (arr[mid] == item)  return mid;

5            if (arr[mid] > item) return binarySearch(__________,__________,__________,__________);

6            return binarySearch(__________,__________,__________,__________);

7            }

8        return -1;

9    }
```

Part C (5 points): How is a 2-D array stored in memory?

Your Answer (no more than 30 words):

Part D (5 points): What would happen if there's a bug in a recursive implementation, in which the code will never reach the base case?

Your Answer (no more than 30 words):

Table E.2 The Standard ASCII Table

ASCII			ASCII			ASCII			ASCII		
Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex
nul	0	00	sp	32	20	@	64	40	`	96	60
soh	1	01	!	33	21	A	65	41	a	97	61
stx	2	02	"	34	22	B	66	42	b	98	62
etx	3	03	#	35	23	C	67	43	c	99	63
eot	4	04	$	36	24	D	68	44	d	100	64
enq	5	05	%	37	25	E	69	45	e	101	65
ack	6	06	&	38	26	F	70	46	f	102	66
bel	7	07	'	39	27	G	71	47	g	103	67
bs	8	08	(	40	28	H	72	48	h	104	68
ht	9	09	)	41	29	I	73	49	i	105	69
lf	10	0A	*	42	2A	J	74	4A	j	106	6A
vt	11	0B	+	43	2B	K	75	4B	k	107	6B
ff	12	0C	,	44	2C	L	76	4C	l	108	6C
cr	13	0D	-	45	2D	M	77	4D	m	109	6D
so	14	0E	.	46	2E	N	78	4E	n	110	6E
si	15	0F	/	47	2F	O	79	4F	o	111	6F
dle	16	10	0	48	30	P	80	50	p	112	70
dc1	17	11	1	49	31	Q	81	51	q	113	71
dc2	18	12	2	50	32	R	82	52	r	114	72
dc3	19	13	3	51	33	S	83	53	s	115	73
dc4	20	14	4	52	34	T	84	54	t	116	74
nak	21	15	5	53	35	U	85	55	u	117	75
syn	22	16	6	54	36	V	86	56	v	118	76
etb	23	17	7	55	37	W	87	57	w	119	77
can	24	18	8	56	38	X	88	58	x	120	78
em	25	19	9	57	39	Y	89	59	y	121	79
sub	26	1A	:	58	3A	Z	90	5A	z	122	7A
esc	27	1B	;	59	3B	[	91	5B	{	123	7B
fs	28	1C	<	60	3C	\	92	5C	|	124	7C
gs	29	1D	=	61	3D	]	93	5D	}	125	7D
rs	30	1E	>	62	3E	^	94	5E	~	126	7E
us	31	1F	?	63	3F	_	95	5F	del	127	7F

NOTES: RTL corresponds to execution (after fetch!); JSRR not shown

ADD	0001 | DR | SR1 | 0 | 00 | SR2	ADD DR, SR1, SR2	DR ← SR1 + SR2, Setcc
ADD	0001 | DR | SR1 | 1 | imm5	ADD DR, SR1, *imm5*	DR ← SR1 + SEXT(imm5), Setcc
AND	0101 | DR | SR1 | 0 | 00 | SR2	AND DR, SR1, SR2	DR ← SR1 AND SR2, Setcc
AND	0101 | DR | SR1 | 1 | imm5	AND DR, SR1, *imm5*	DR ← SR1 AND SEXT(imm5), Setcc
BR	0000 | n | z | p | PCoffset9	BR{nzp} *PCoffset9*	((n AND N) OR (z AND Z) OR (p AND P)): PC ← PC + SEXT(PCoffset9)
JMP	1100 | 000 | BaseR | 000000	JMP BaseR	PC ← BaseR
JSR	0100 | 1 | PCoffset11	JSR *PCoffset11*	R7 ← PC, PC ← PC + SEXT(PCoffset11)
TRAP	1111 | 0000 | trapvect8	TRAP *trapvect8*	R7 ← PC, PC ← M[ZEXT(trapvect8)]

LD	0010 | DR | PCoffset9	LD DR, *PCoffset9*	DR ← M[PC + SEXT(PCoffset9)], Setcc
LDI	1010 | DR | PCoffset9	LDI DR, *PCoffset9*	DR ← M[M[PC + SEXT(PCoffset9)]], Setcc
LDR	0110 | DR | BaseR | offset6	LDR DR, BaseR, *offset6*	DR ← M[BaseR + SEXT(offset6)], Setcc
LEA	1110 | DR | PCoffset9	LEA DR, *PCoffset9*	DR ← PC + SEXT(PCoffset9), Setcc
NOT	1001 | DR | SR | 111111	NOT DR, SR	DR ← NOT SR, Setcc
ST	0011 | SR | PCoffset9	ST SR, *PCoffset9*	M[PC + SEXT(PCoffset9)] ← SR
STI	1011 | SR | PCoffset9	STI SR, *PCoffset9*	M[M[PC + SEXT(PCoffset9)]] ← SR
STR	0111 | SR | BaseR | offset6	STR SR, BaseR, *offset6*	M[BaseR + SEXT(offset6)] ← SR

End of ECE 220 Midterm Exam 2