CMPT 125, Fall 2019

Final Exam - solutions
December 7, 2019

Name_____

SFU ID: |__|__|__|__|__|__|__|__|__|

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. Write your name and SFU ID **clearly**.
2. This is a closed book exam, no calculators, cell phones, or any other material.
3. The exam consists of four (4) problems.
4. Write your answers in the provided space.
5. There is an extra page at the end of the exam. You may use it if needed.
6. Explain all your answers.
7. Really, explain all your answers.

Good luck!

Problem 1 [20 points]

a) [4 points]Write the function str_reverse that gets a string and reverses it in place.

```
void str_reverse(char *str) {

    char tmp;

    int n = strlen(str);
        //  for loop i=0...n/2
    for (int i=0; i< n/2; i++)
    { // swap str[i] with str[n-1-i]
      tmp = str[i];
      str[i] = str[n-1-i];
      str[n-1-i] = tmp;
    }
}
```

b) [3 points] What will be the output of the following program?

```
#include <stdio.h>

enum colors {RED, GREEN, BLUE};

void foo(int* x, int *y, int z) {
    *x = z;
    x = y;
    *x = z;
    z = RED;
}

int main() {
    int a = RED, b = GREEN, c = BLUE;
    foo(&a, &b, c);
    printf("a = %d, b = %d, c = %d", a, b, c);
    return 0;
}
```

ANSWER: a = 2, b = 2, c = 2

c) [3 points] Will the code below compile?
If yes, what will be the output? If no, explain why.

```
#include <stdio.h>

int main() {
  char str[10] ={'a','b','c',0,'1','2','3','0','\0'};
  char* ptr = str;
  printf("%s\n", ptr);
  return 0;
}
```

ANSWER: It will compile and print abc. 0 denotes the end of the string

d) [4 points] What will be the output of the following code? Explain your answer.

```
int bar(int n) {
   if (n <= 1)
     return 1;

   int sum = 0;
   for (int i=1; i < n; i=i*2) {
     printf("%d ", i);
     sum += i;
   }
   printf("\n");

   return sum + bar(n/2);
}

int main(void) {
    printf("bar(8) = %d", bar(8));
    return 0;
}
```

ANSWER: 1 2 4
1 2
1
bar(8) = 12

Explanation: bar(1) =1
bar(2) = 1 + bar(1) = 2
bar(4) = 3 + bar(2) = 5
bar(8) = 7 + bar(4) = 12

[4 points] Use the big-O notation to express the running time of bar(n) as a function of n. Explain your answer.

ANSWER: The for loop runs for O(log(n)) iterations. Therefore the runtime can be written as T(n) = O(log(n)) + T(n/2)
How many recursive calls will be made? Well, each time n is divided by 2, so there are O(log(n) recursive calls.
Therefore, the total runtime will be $O((\log(n))^2)$

e) [2 points] Explain what is void* in C.
ANSWER: It represents a pointer of unspecified type (just address in the memory).
In order to use need to cast (e.g. cast to int*)

Problem 2 [30 points - 3 points for each question]

Implement the ADT *Linked List of ints* so that the runtime of each operation is O(1).

a) Declare the type LLnode_t here:

```
struct LLnode {
  int data;
  struct LLnode* next;
};
typedef struct LLnode LLnode_t;
```

b) Declare LL_t here:

```
typedef struct {
  node_t* head;
} LL_t;
```

c)
```
LL_t* create_LL() {

  LL_t* list = malloc(sizeof(LL_t));
  if (list == NULL)
     return NULL;

  list->head = NULL;
  return list;
}
```

d)
```
void add_to_head(LL_t* list, int item) {

  LLnode_t* newNode = (LLnode_t*) malloc(sizeof(LLnode_t));
  if (newNode == NULL)
      return NULL;

  newNode->data = value;
  newNode->next = list->head;
  list->head = newNode;
}
```

e)
```
int remove_from_head(LL_t* list) {

  LLnode_t* prev_head = list->head;
  int ret_value = prev_head->data;

  list->head = prev_head->next;
  free(prev_head);
  return ret_value;
}
```

Suppose you are given an ADT Doubly *Linked List of ints* `DLL_t` with the following operations.

```
void add_to_head(DLL_t* list, int item);
void add_to_tail(DLL_t* list, int item);
int remove_from_head(DLL_t* list);
int remove_from_tail(DLL_t* list);
bool is_empty(DLL_t* list);
```

Suppose that the runtime of each operation is O(1).
Use DLL_t to implement stack of ints that also supports the stack_reverse operation.
The runtime of each operation must be O(1).

f)
```
typedef struct stack {
    int flag_head;
    DLL_t list;
} stack_t;
```
```
// idea: use DLL inside the stack,
// and use flag_head=0/1 to decide whether
// to push/pop to the head or to the tail

// create_stack() creates an empty list
// and sets fla_head = 1;
```

g)
```
void stack_push(stack_t* stack, int item) {
    if (flag_head)
        add_to_head(list, item);
    else (flag_head)
        add_to_tail(list, item);
}
```

h)
```
int stack_pop(stack_t* stack) {
    if (flag_head)
        return remove_from_head(list);
    else (flag_head)
        return remove_from_tail(list);
}
```

i)
```
bool stack_is_empty(stack_t* stack) {
    return is_empty(list);
}
```

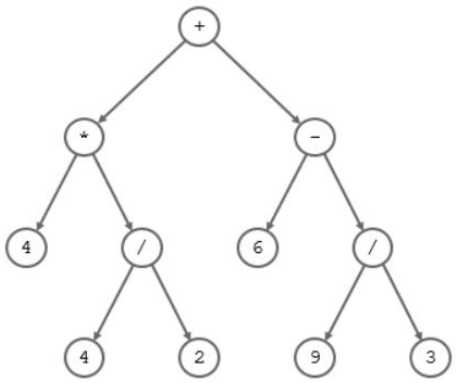
j)
```
void stack_reverse(stack_t* stack) {
    flag_head = 1 - flag_head;
}
```

Problem 3 [25 points]

In this problem use the following struct for Binary Tree of ints.

```
struct BTnode {
  int value;
  struct BTnode* left;
  struct BTnode* right;
  struct BTnode* parent;
};
typedef struct BTnode BTnode_t;
```

a) [7 points] Write an algorithm that gets a Binary Tree representing an arithmetic expression, and returns the evaluation of the expression. For example, for the tree below the function will return 11. This is because (4 * (4/2)) + (6 - (9/3)) = 8 + 3 = 11.



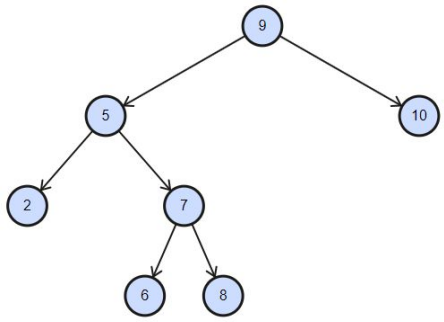
You may assume that the operations are implemented as

```
enum operators {PLUS='+', MINUS='-', MULTIPLY='*', DIVIDE='/'};

int evaluate(BTnode_t* expression} {
  if (expression->left == NULL && expression->right) == NULL)
    return expression->value;

  int tmp1 = evaluate(expression->left);
  int tmp2 = evaluate(expression->right);
  if (expression->value == PLUS) return tmp1 + tmp2;
  if (expression->value == MINUS) return tmp1 - tmp2;
  if (expression->value == MULTIPLY) return tmp1 * tmp2;
  if (expression->value == DIVIDE) return tmp1 / tmp2;


}
```

[4 points] Use the big-O notation to express the running time of evaluate. Explain your answer.

O(size of the tree) - we make O(1) operation on each node of the tree

b) [10 points] Write a function that gets a pointer to the root of a Binary Search Tree, and returns the sum of the two smallest elements. For example, the smallest elements in the tree below are 2 and 5, and so the function should output 7. You may assume that the tree has at least 2 nodes.



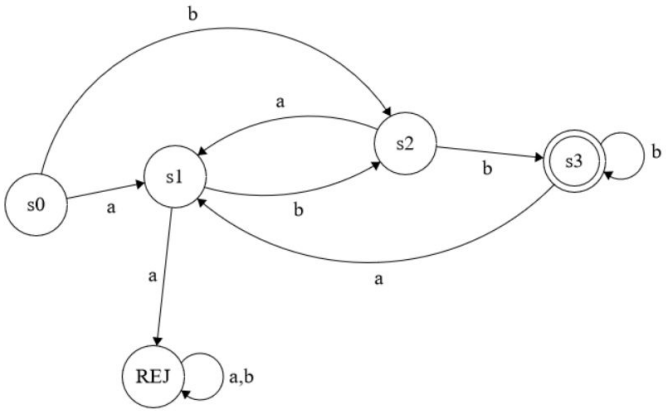

```
int sum_of_min2(BTnode_t* root) {

 // idea: find min first
 // if min has a right child, find second min in right subtree of min
 // otherwise, return the parent of min
 BTnode_t* min, *min2;
 min = root;
 while (min->left != NULL)
   min = min->left;

 if (min->right) {
   min2 = min->right;
   while (min2->left != NULL)
     min2 = min2->left;
 }
 else
   min2 = min->parent;

 return min->value + min2->value;



}
```

[4 points] Use the big-O notation to express the running time of sum_of_min2. Explain your answer.

O(depth of the tree) - we go down the tree exactly once + some O(1) operations.

Problem 4 [25 points]

a) [4 points] Describe in words the language defined the following DFA. Explain your answer.



The language contains all words that
- end with bb, and
- do not contain to aa.

b) [5 points] Write a function in C that decides the language accepted by the DFA above.

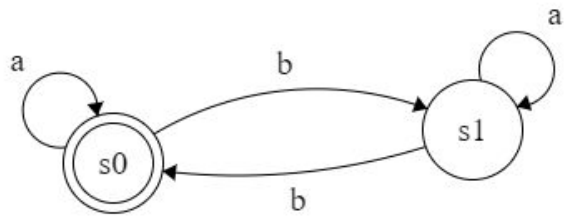
```
bool decide_lang(char* str) {

 int n = strlen(str);
 for (int i = 0; i < n-2; i++) {
   if (str[i] == 'a' && str[i+1] == 'a')
     return false;
 }
 return (str[n-2]== 'b' && str[n-1]== 'b');




}
```

b) Consider the following regular expression: a*(ba*ba*)*

[4 points] Describe in words the language defined by the regular expression above.

The language consists of all words that have an even number of b's.

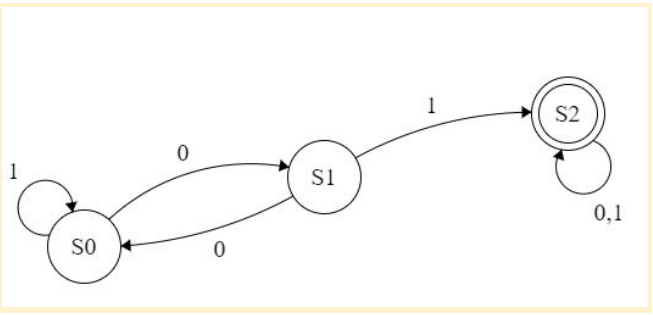
[4 points] Draw a DFA that accepts the language defined by the regular expression.



c) Consider the following description of DFA:

$\sum = \{0,1\}$ $S = \{s_0, s_1, s_2\}$ $F = \{s_2\}$	$\delta(s_0,0) = s_1$ $\delta(s_0,1) = s_0$ $\delta(s_1,0) = s_0$ $\delta(s_1,1) = s_2$ $\delta(s_2,0) = s_2$ $\delta(s_2,1) = s_2$

[4 points] Draw the corresponding DFA.



[4 points] Describe in words the language accepted by the DFA.

The language consists of all words that have 1 in some position k such that before k there is an odd number of zeros.

Extra page