CMPT 125, Spring 2022 - Solution

Final Exam
April 26, 2022

Name_____

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

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. Duration of the exam is 180 minutes.
2. Write your full name and SFU ID **clearly**.
3. This is a closed book exam, no calculators, cell phones, or any other material.
4. The exam consists of four (4) problems. Each problem is worth 25 points.
5. Write your answers in the provided space.
6. There is an extra page at the end of the exam. You may use it if needed.
7. Explain all your answers if the question asks you to.
8. Really, explain all your answers.

Good luck!

Problem 1 [25 points]

a) [6 points] What will be the output of the following code? Explain your answer.

```
#include <stdio.h>
int what(int n) {
    if (n<=0)   return 0;
    int sum = 0;
    for (int i=1; i<=n; i=i+2) {
      printf("%d ", i);
      sum += i;
    }
    printf("\n");
    return sum + what(n/2);
}
int main(void) {
     printf("what(6) = %d", what(6));
     return 0;
}
```

Answer:

- what(1) prints "1", calls what(0), and returns 1+what(0)=1
- what(3) prints "1 3", calls what(1) and return 1+3+what(1)=5
- what(6) prints on "1 3 5", calls what(3), and returns 1+3+5+what(3)=14

The output will be:

1 3 5
1 3
1
what(6) = 14

b) [6 points] Will the code below compile?
If yes, describe what the code does? If not, explain why.

```
#include <stdio.h>
#include <stdlib.h>
char* bar() {
  char* str = malloc(2);
  str[0] = 'A';
  char* ret = str+1;
  ret[1] = 'B';
  return ret;
}
int main() {
    char* a = bar();
    printf("%s\n", a);
    free(a);
    return 0;
}
```

Answer: it will compile. but the behavior is undefined because:
(1) ret[1] writes out of bounds (2) printf might crash as there is no '\0' at the end of the string, (3) will probably crash because free is not applied on the beginning of the allocated memory.

c) [8 points] A string is said to be a good password if it satisfies the following conditions:

- its length is at least 8
- contains at least one digit
- contains at least one lower case letter
- contains at least one upper case letter
- contains at least one of these three symbols: !@#

Write a function that gets a string, and checks if it is a good password.

```
bool is_good_password(const char* str) {
  int n = strlen(str);
  if (n<8)
      return false;
  bool digit = false;
  bool lower_case = false;
  bool flag_upper_case = false;
  bool symbol = false;

  for(int i=0; i<n; i++) {
      if (str[i]>='0' && str[i]<='9')
        digit = true;
      if (str[i]>='a' && str[i]<='z')
        lower_case= true;
      if (str[i]>='A' && str[i]<='Z')
        flag_upper_case = true;
      if (str[i]=='!' || str[i]=='@' || str[i] == '#')
        symbol = true;
  }
  return digit && lower_case && flag_upper_case && symbol;


}
```

d) [5 points] Explain the differences in C++ between the following:

- passing a variable to a function by value.
- passing a variable to a function using a pointer.
- passing a variable to a function using a reference.

Explain how each of them can be used. Provide examples if that helps the explanation.

Answer: passing by value means the passed variable cannot change. We just copy the value into the called function.
Passing by reference/pointer allows changing the value of the passed variable. For example can be used for swap
One difference between pointer and reference that pointer can be NULL, but reference always refers to an existing variable

Problem 2 [25 points]

An array A[0...n-1] of ints is called *a mountain* if there exists an index 0≤k≤n-1 such that
- arr[0] < arr[1] < ... < arr[k-1] < arr[k]
- arr[k] > arr[k+1] > ... > arr[n-1]

The index k is called the peak.
Note that if k=0 then A is a decreasing array, and k=n-1 corresponds to an increasing array.

a) [10 points] Write a function in C that solves the following problem.
Input: A mountain array of ints A of length n>0 with all values distinct
Output: the index of the peak.
The running time: must be O(log(n)).
Examples:
- For A = [1, 4, 5, 6, 7, 3, 1], the output should be 4 (because, A[4] = 7).
- For A = [-5, 1, 3], the output should be 2 (because A[2] = 3).
- For A = [10, 4], the output should be 0 (because A[0] = 10).

Explain your idea before writing code.

Idea: use a binary search, each time checking A[mid-1] A[mid] A[mid+1].
If they are all increasing, decreasing, or A[mid] is the peak

```
int find_peak(const int* A, int n) {
  if (n==1)
      return 0;

  if (A[0] > A[1]) // A is decreasing
      return 0;
  if (A[n-1] > A[n-2])  // A is increasing
      return n-1;

  int start = 0;
  int end = n-1;
  int mid;
  while (start<end) {
      mid = (start+end)/2;
      if (A[mid-1] < A[mid] && A[mid] < A[mid+1])
          start = mid;
      else if (A[mid-1] > A[mid] && A[mid] > A[mid+1])
          end = mid;
      else // (A[mid-1] < A[mid] && A[mid] > A[mid+1])
          return mid; // found peak
  }
  return -1;  // never reaches here


}
```

b) [15 points] Write a function that gets a mountain array of length n, and the index k representing the peak, and sorts the array in the increasing order in O(n) time.
Explain your idea before writing code.

Idea: use the merge procedure from merge sort.

```
void sort_mountain(int* A, int n, int k) {
  int* tmp = malloc(n*sizeof(int)); // used for the sorted values

  int start=0; // pointer to the A[0…k] part
  int end=n-1; // pointer to the A[k…n-1] part
  int tmp_ind=0; // index of the sorted array tmp

  // in each iteration we choose either A[start] or A[end]
  while (start<=k && end>=k) {
     if (A[start] < A[end]) {
           tmp[tmp_ind] = A[start];
           start++;
     }
     else {
           tmp[tmp_ind] = A[end];
           end--;
     }
     tmp_ind++;
  }

  if (end<k) { // only the elements from A[0…k] are left
     while (start<k) {
       tmp[tmp_ind] = A[start];
       start++;
       tmp_ind++;
     }
  }
  else { //  // only the elements from A[k…n-1] are left
     while (end>k) {
       tmp[tmp_ind] = A[end];
       end--;
       tmp_ind++;
     }
  }
  for (int i=0; i<n; i++) // copy from tmp to A
     A[i] = tmp[i];

  free(tmp); // don't forget to free tmp

}
```

Problem 3 [25 points]

In the problem a linked list of ints is represented as follows.

```
struct LL_node {
  int data;
  struct LL_node* next;
};
typedef struct LL_node LL_node_t;

typedef struct {
  LL_node_t* head;
  LL_node_t* tail;
} LL_t;
```

a) [13 points] Write a function in C that gets a linked list of ints and checks if all negative numbers in the list come before all non-negative numbers. (zero is non-negative)
The running time of the function must be O(length of list).

For example, the function returns true on the following inputs:

-2 → -4 → 10 → 0 → 11
4 → 3 → 5
-2 → -4
1
-6
empty list

The function needs to return false on the following inputs:

-3 → 4 → 6 → -8
7 → -1
1 → 0 → -1 → 1

```
bool negative_first(LL_t* list) {
// idea: we first skip all negative numbers,
// when we see the first number >=0, we don't expect more negatives
// if we see a negative number after that, we return false

  LL_node_t* cur = list->head;
  while (cur && cur->data<0) // skip all negatives
      cur = cur->next;

  // here either cur==null or is cur->data>=0
  while (cur) {
      if (cur->data<0)
          return false;
      cur = cur->next;
  }
  // if reached here, return true
  return true;

}
```

b) [12 points: 6 points each] Implement the following standard functions on a linked list with pointers to head and tail, using the struct above. Each of the functions must run in O(1) time.

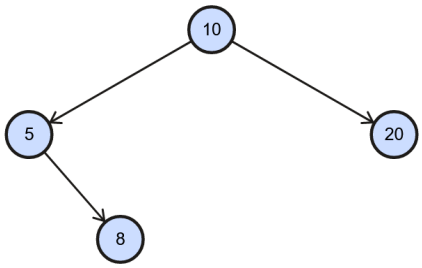
```
// adds a node with the given value to the end of the list
void add_to_tail(LL_t* list, int val) {

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

      newNode->data = value;
      newNode->next = NULL;

      if (list->head==NULL) {  // edge case: list was empty
        list->head = newNode;
        list->tail = newNode;
      }
      else {
        list->tail->next = newNode;
        list->tail = newNode;
      }



}

// removes the first node from the list, and returns its value
// assumption: list is not empty
int remove_from_head(LL_t* list) {


  node_t* prev_head = list->head;
  int ret = prev_head->data; // save value to return

  list->head = list->head->next;

  if (list->head==NULL) // edge case: list becomes empty
      list->tail = NULL;

  free(prev_head);
  return ret;



}
```

Problem 4 [25 points]

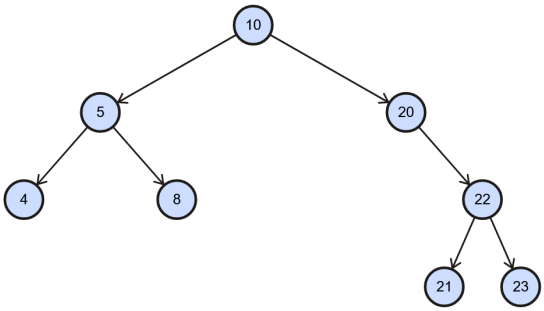
In this problem use the following struct for Binary Tree of ints.

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

a) Consider the following Binary Search Tree.



[3 points] Add the list of numbers to the Binary Search Tree in the given order: 4, 22, 21, 23. What will be the result in the end? Draw the resulting tree with the eight nodes.



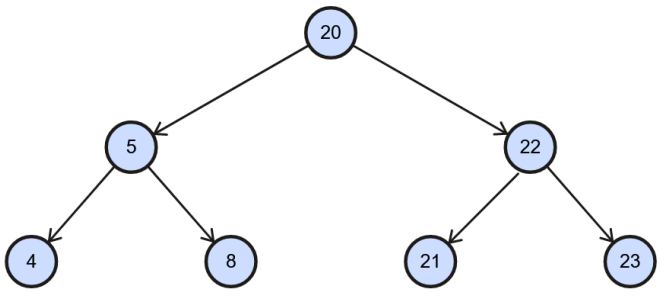
[4 points] Find four more permutations (reorderings) of the numbers {4,21,22,23}, such that for each of the permutations if we add the numbers in this order to the original BST, the result will be the same as in your answer above. Explain your answer.

ANSWER: total 7 more such sequences: 4 can be anywhere, 22 must be before 21 and 23

(1) 4, 22, 23, 21 (2) 22, 4, 21, 23 (3) 22, 4, 23, 21

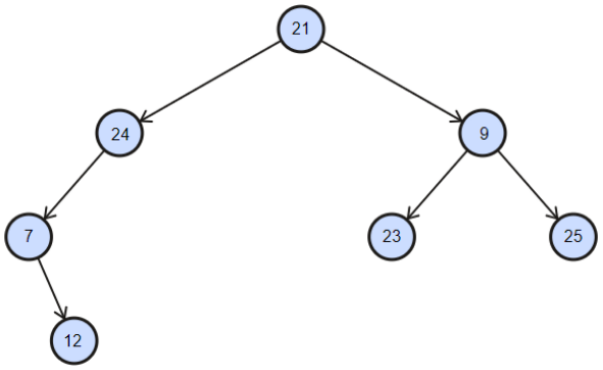
(4) 22, 21, 4, 23 (5) 22, 23, 4, 21 (6) 22, 21, 23, 4 (7) 22, 23, 21, 4

[3 points] Remove 10 from the BST obtained above (having the values 4,5,8,10,20,21,22,23). What will be the result?



b) [15 points] Write a function that gets a node in a binary tree (not necessarily the root), and returns the next node in the *in-order traversal order*.
The running time must be O(depth of three).
For example, in the tree below:

- on input 7 the output should be 12
- on input 24 the output should be 21
- on input 21 the output should be 23
- on input 25 the output should be NULL



Explain your answer before writing the code.

```
BTnode_t* next_inorder(BTnode_t* node) {

  // if node has right child, we find min in node->right
  if (node->right) {
      BTnode_t* cur = node->right;
      while (cur->left)
            cur = cur->left;
      return cur;
  }

  // if we are here, node doesn't have a right child
  // we need to go to the parents of node
  // if node is the left child of the parent, return the parent
  // if node is the right child of the parent, go higher up the tree
  BTnode_t* cur = node;
  while (cur->parent && cur->parent->right == cur) {
      cur = cur->parent;
  }
  return cur->parent;




}
```

Extra page

Empty page

Empty page