UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION, APRIL 2012

CSC 190 H1S — ALGORITHMS AND DATA STRUCTURES

Calculator Type: 4 (*none*)

Examination Type: D (*course textbook and printouts of lecture slides and code*)

Examiner: François Pitt

Student Number: |_|_|_|_|_|_|_|_|_|

Family Name(s): _____

Given Name(s): _____

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully.*

This final examination paper consists of 8 questions on 20 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy is complete and fill in the identification section above.*

Answer each question directly on this paper, in the space provided. If you need more space for one of your solutions, use one of the extra "blank" pages at the end of the exam and *indicate clearly the part of your work that should be marked.*

Write up your solutions carefully! Part marks *will* be given for showing that you know the general structure of an answer, even if your solution is incomplete. In particular, when writing code, comments are **not** required *except where indicated*, although they may help us understand your answers (and may be worth part marks).

MARKING GUIDE

# 1:	_____/18
# 2:	_____/ 8
# 3:	_____/ 8
# 4:	_____/ 8
# 5:	_____/ 8
# 6:	_____/ 8
# 7:	_____/ 8
# 8:	_____/ 9
TOTAL:	_____/75

Good Luck!

OVER...

Question 1. SHORT ANSWERS [18 MARKS]

Part (a) [2 MARKS]

The following code contains a bug that could cause runtime errors. Explain what this bug is.

```
#include <stdlib.h>

struct node { int value; struct node *next; };

void destroy(struct node *first)
{
    struct node *node;
    for (node = first; node; node = node->next)
        free(node);
}
```

Part (b) [2 MARKS]

The following code contains a bug that could cause runtime errors. Explain what this bug is.

```
#include <stdlib.h>
#include <assert.h>

struct node { int value; struct node *next; };

struct node *create(int value)
{
    struct node *first = malloc(sizeof(struct node *));
    assert(first);
    first->value = value;
    first->next = NULL;
    return first;
}
```

Question 1. (CONTINUED)

Part (c) [2 MARKS]

Write down the output produced by the following code. *Be careful to write down the* ***exact*** *output.*

```
char *names[] = {"Jim", "Jack", "Joe"};
int i;

for (i = 0; i < 3; ++i)
    printf(" %s", names[i]);
```

Part (d) [2 MARKS]

Write down the output produced by the following code. *Be careful to write down the* ***exact*** *output.*

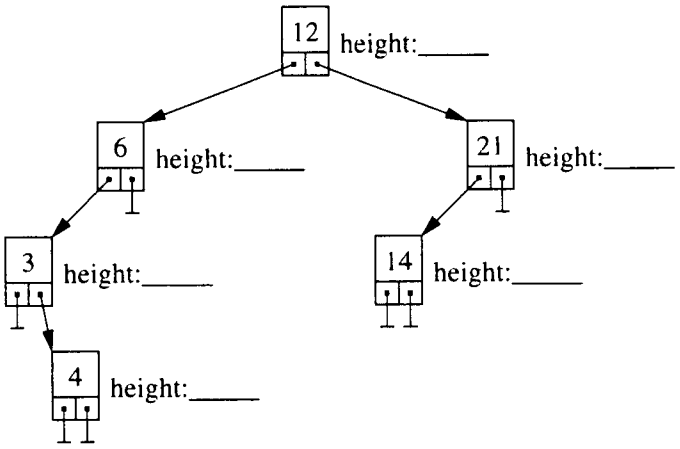
```
int list[] = {3, 2, 1, 0 , 1, 2, 3};
int *p = list;

while (*p++)  printf("%d\n", *p);
```

Question 1. (CONTINUED)

Part (e) [2 MARKS]

For each node in the tree pictured below, fill in the height of the subtree rooted at that node, as computed for an AVL tree.



Part (f) [2 MARKS]

Is the tree in Part (e) a BST? Explain.

Part (g) [2 MARKS]

Is the tree in Part (e) an AVL tree? Explain.

Question 1. (CONTINUED)

Part (h) [2 MARKS]

Draw the tree structure for the complete binary tree stored in array $[-, 25, 14, 9, 11, 12, 10, 6, 6, 3, 7, 6, 8]$.

Part (i) [2 MARKS]

Is the array in Part (h) a proper max-heap? Explain.

Question 2. STRINGS [8 MARKS]

Write the body of the function below so that it satisfies its description.

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>

/* Return a pointer to a new string (allocated dynamically) that contains a copy of
 * each of the strings in the array 'strs', one after the other, with a copy of 'sep'
 * between every two strings.  For example, if strs = {"hi", "there", "you"}, then
 * join(3, strs, "; ") returns "hi; there; you".  Preconditions: n = length(strs),
 * strs != NULL, strs[0] != NULL, ..., strs[n-1] != NULL, sep != NULL.
 */
char *join(int n, const char *strs[], const char *sep)
{
    // Hint: use assert() to verify the success of each memory allocation.




}
```

Question 3. LINKED LISTS [8 MARKS]

Write the body of the function below so that it satisfies its description.

```
#include <stdbool.h>

/* A node in a _doubly_ linked list of longs. */
typedef struct list_node {
    long value;
    struct list_node *prev; /* the node just before this one in the list */
    struct list_node *next; /* the node just after this one in the list */
} list_node_t;

/* Search for elem in the linked list starting at *first, and return true if it is
 * found (false otherwise).  If elem is found, MOVE THE NODE THAT CONTAINS ELEM TO THE
 * FRONT OF THE LIST.   Preconditions: first != NULL (though *first may be NULL)
 */
bool find(long elem, list_node_t **first)
{

}
```

Question 4. BINARY SEARCH TREES [8 MARKS]

Write the bodies of both functions below so that they satisfy their descriptions.

```
#include <stdlib.h>
#include <assert.h>

typedef struct bst_node {
    long value;
    struct bst_node *left;
    struct bst_node *right;
} bst_node_t;

/* Return the maximum value stored in the BST rooted at node 'root' (root != NULL). */
long bst_max(const bst_node_t *root)
{
    // NOTE: FOR FULL MARKS, DO **NOT** USE RECURSION FOR THIS FUNCTION!









}

/* Make a copy of every node in the tree rooted at 'root' and return a pointer to the
 * root of the new tree.  (If root == NULL, return NULL.) */
bst_node_t *clone(const bst_node_t *root)
{
    // Hint: use assert() to verify the success of each memory allocation.
    // (You may use recursion for this function.)









}
```

Question 5. HASHING [8 MARKS]

Part (a) [5 MARKS]

Complete the function below so that it satisfies its description.

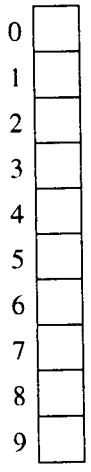
```
/* Return the hash function computed from 'value' in the following way: add up the
 * individual digits of 'value', in base 10; if the result contains more than one
 * digit, do this again until the final result contains exactly one digit (i.e., is
 * between 0 and 9).  For example, when value == 55, then 5 + 5 = 10 and (since 10
 * contains more than one digit) 1 + 0 = 1, so the final result is 1.
 */
unsigned hash10(unsigned value)
{
    // NOTE: FOR FULL MARKS, YOU **MUST** USE RECURSION FOR THIS FUNCTION!




}
```

Part (b) [3 MARKS]

Draw the state of the hash table below after *inserting* the following values: $190, 55, 180, 42, 9, 1234$ (use hash function `hash10` above and *open* hashing, *i.e.*, store the values in singly-linked lists at each slot in the table; assume the size of the table is *fixed*.)



Question 6. HEAPS [8 MARKS]

Part (a) [4 MARKS]

Consider the max-heap stored in array $[-, 31, 25, 15, 11, 20, 12, 14, 9, 8, 7, 6]$. Write the contents of the **array** *after* inserting value 22. Show the intermediate steps generated by the insertion algorithm.

Question 6. (CONTINUED)

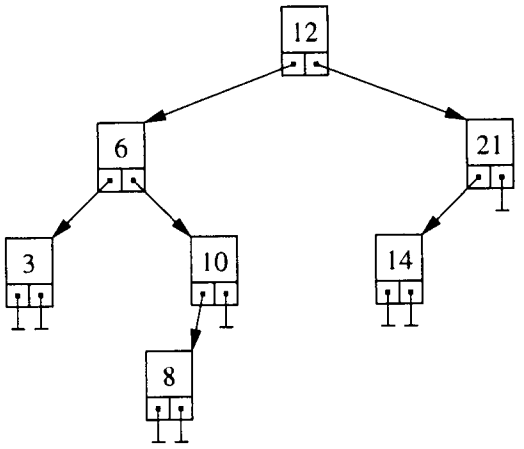
Part (b) [4 MARKS]

Consider the max-heap stored in array $[-, 31, 25, 15, 11, 20, 12, 14, 9, 8, 7, 6]$. Write the contents of the **array** *after* removing the maximum value. Show the intermediate steps generated by the removal algorithm.

Question 7. AVL TREES [8 MARKS]

Part (a) [4 MARKS]

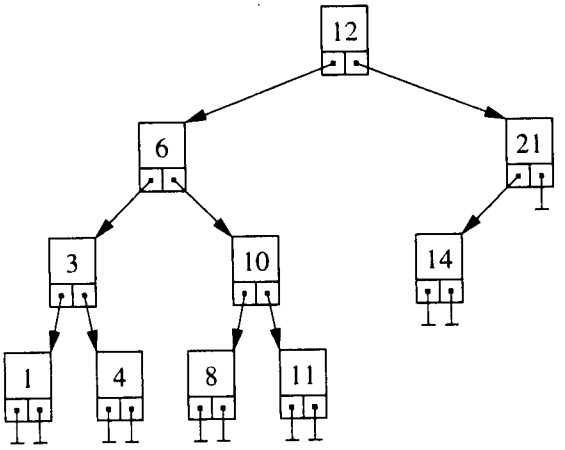
Consider the AVL tree pictured below. Draw the AVL tree that results from *inserting* value 9 into the tree. Show the intermediate steps carried out by the insertion algorithm, and include height information for each node.



Question 7. (CONTINUED)

Part (b) [4 MARKS]

Consider the AVL tree pictured below. Draw the AVL tree that results from *removing* value 21 from the tree. Show the intermediate steps carried out by the removal algorithm, and include height information for each node.



Question 8. DESIGN [9 MARKS]

Consider the following header file `deque.h`.

```
/* A double-ended queue (or "deque") allows insertion and removal of elements (of type
 * const void *) at either end of the dequeue. */
#include <stdbool.h> /* for type bool   */
#include <stdlib.h>  /* for type size_t */

/* The type of a double-ended queue -- to be fully defined in the source file. */
typedef struct deque deque_t;

/* DEFAULT PRECONDITION: for all functions, all parameters must be non-NULL. */

/* Return a pointer to a newly allocated empty deque; NULL in case of error. */
deque_t *deque_create(void);

/* Free all the memory allocated for deque q. */
void deque_destroy(deque_t *q);

/* Return the current size of deque q. */
size_t deque_size(const deque_t *q);

/* Return an element d in deque q such that (*eq)(d,e), if q contains such an element;
 * return NULL if q contains no such element. */
const void *deque_contains(const deque_t *q, const void *e,
                           bool (*eq)(const void *, const void *));

/* Insert element e at the front of deque q. */
void deque_insert_front(deque_t *q, const void *e);

/* Insert element e at the back of deque q. */
void deque_insert_back(deque_t *q, const void *e);

/* Remove and return the element at the front of deque q; return NULL if q is empty. */
const void *deque_remove_front(deque_t *q);

/* Remove and return the element at the back of deque q; return NULL if q is empty. */
const void *deque_remove_back(deque_t *q);
```

On the next page, describe **two** different data structures that could be used to implement the deque ADT. For each data structure:

- describe how the deque's elements are stored in memory (*i.e.*, how you organize the data) — *your design must be able to handle any number of elements* (limited by the computer's memory, of course);
- state the worst-case running time of each function — *it's okay if your implementation is inefficient*;
- describe one main *advantage* of your data structure;
- describe one main *disadvantage* of your data structure.

Do NOT write any code for this question!

Question 8. (CONTINUED)

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each answer with the appropriate question and part number.

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each answer with the appropriate question and part number.

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each answer with the appropriate question and part number.

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each answer with the appropriate question and part number.