UNIVERSITY OF TORONTO MISSISSAUGA
APRIL 2013 FINAL EXAMINATION
CSC 148H5S
Introduction to Computer Science
Arnold Rosenbloom
Duration — 3 hours
Aids: *None*

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

Last Name: _____

First Name: _____

Signature: _____

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This final examination consists of a two page appendix AND this exam book, with 7 questions on 12 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the examination is complete.* Answer each question directly on the examination paper, in the space provided.

Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero (0) so write legibly.

The University of Toronto Mississauga and you, as a student, share a commitment to academic integrity. You are reminded that you may be charged with an academic offence for possessing any unauthorized aids during the writing of an exam, including but not limited to any electronic devices with storage, such as cell phones, pagers, personal digital assistants (PDAs), iPods, and MP3 players. Unauthorized calculators and notes are also not permitted. Do not have any of these items in your possession in the area of your desk. Please turn the electronics off and put all unauthorized aids with your belongings at the front of the room before the examination begins. If any of these items are kept with you during the writing of your exam, you may be charged with an academic offence. A typical penalty may cause you to fail the course.

Please note, you **CANNOT** petition to **re-write** an examination once the exam has begun. exam has begun.

# 1:	_____/ 5
# 2:	_____/ 5
# 3:	_____/ 5
# 4:	_____/ 5
# 5:	_____/ 5
# 6:	_____/ 5
# 7:	_____/15
TOTAL:	_____/45

Good Luck!

Question 1. [5 MARKS]

To determine if a number is divisible by three, use the following trick: A number is divisible by 3, if and only if the sum of its digits is divisible by 3. For example: 12123 (1+2+1+2+3=9) 9 is divisible by 3, therefore 12123 is as well!

Write recursive routines **sum_digits** and **is_divisible_by_3** below. No for loops and while loops allowed.

```
def sum_digits(n):
    ''' return the sum of the digits in non-negative integer n
        sum_digits(0) returns 0
        sum_digits(5) returns 5
        sum_digits(17) returns 8
        sum_digits(5762) returns 20 '''
    # Hint: n%10 is the last digit of n, so 5%10==5, 17%10==7, 5762%10==2
```

```
def is_divisible_by_3(n):
    ''' return True, if non-negative integer n is divisible by 3, and False otherwise '''
    # You must use the divisibility by 3 trick and only assume 0, 3, 6 and 9 are divisible by 3.
```

Question 2. [5 MARKS]

In the space on the right, write the output of the following:

```
class ExceptionT(Exception): pass
class ExceptionW(Exception): pass
def f(n):
    try:
        print("n=", n)
        if n==0:
            t=1/0
        elif n=='T':
            raise ExceptionT("Causing Exception T")
        elif n=='W':
            raise ExceptionW("Causing Exception W")
        else:
            print("Not raising any exceptions")

    except ExceptionT as e:
        print("Handling excecption T")

    except ExceptionW as e:
        print(w)

    except Exception as e:
        print('Handle Exception', e)

    else:
        print("else")

    finally:
        print("finally")

    print("back in main block")

f(5)
f(0)
f('T')
f('W')
```

Question 3. [5 MARKS]

Draw a UML Class diagram for the following scenario. Do not write python code. A Calendar has a year and a collection of Events. Each Event has a month, day, description and a list of people involved in the event. Each Person has a name and email. A Person can be a Student, in which case, they also have a student number. A Person can be an Instructor, in which case, they have an office. The string representation of a Person includes their name and email. The string representation of a Student additionally includes their student number. An Instructor's string representation additionally includes their office location. Use inheritance if you can, to shorten your code.

A sample execution of your code follows:

```
cal=Calendar(2012)

s1=Student("Sid", "sid@utoronto.ca", 987654321)
s2=Student("Jane", "jane@utoronto.ca", 998877665)
p1=Person("Jill", "jill@hill.com")
p2=Person("Jack", "jack@hill.com")
i1=Instructor("Arnold", "arnold@cs.toronto.edu", "CCT3067")

e=Event("Feb", 13, "CSC148 Midterm")
e.add_person(s1)
e.add_person(s2)
e.add_person(i1)
cal.add(e)

e=Event("Jul",3, "The Amazing Spiderman")
cal.add(e)
e.add_person(i1)
e.add_person(s1)
e.add_person(p1)
e.add_person(p2)

print(str(cal))
# OUTPUT ##########################################
2012
Feb 13:CSC148 Midterm
        Sid sid@utoronto.ca 987654321
        Jane jane@utoronto.ca 998877665
        Arnold arnold@cs.toronto.edu CCT3067

Jul 3:The Amazing Spiderman
        Arnold arnold@cs.toronto.edu CCT3067
        Sid sid@utoronto.ca 987654321
        Jill jill@hill.com
        Jack jack@hill.com
```

Question 4. [5 MARKS]

Re-write the `LinkedList` code below so that it implements a constant time `add_at_end` method. You may have to make many changes to this code! Max 4/5 if your `add_at_end` takes more than constant time.

```
class LinkedListNode(object):
    def __init__(self, element, next_node):
        self._element=element
        self._next=next_node

class LinkedList(object):
    def __init__(self):
        self._start=None

    def add_at_start(self, element):
        ''' add element in a new node at the front of the LinkedList '''
        self._start=LinkedListNode(element, self._start)

    def __str__(self):
        s="["
        current=self._start
        while current is not None:
            s=s+str(current._element)+','
            current=current._next
        s=s+']'
        return s
```

Question 5. [5 MARKS]

Modify the `BSTree` in the appendix to this exam so that it supports a `shortest_path` method. This returns a list of keys on a shortest root to leaf path. If there is more than one such path, you are free to return the keys on any one of them. For example:

```
t=BSTree()
for v in [65, 2, 99, 22, 132, 5, 6]:
    t.insert(v)
print t.shortest_path()
# OUTPUT:  [99, 65]

t=BSTree()
for v in [8, 4, 12, 2, 6, 10, 14, 1, 3, 5, 7, 9, 11, 13, 15]:
    t.insert(v)
print t.shortest_path()
# OUTPUT: [15, 14, 12, 8]
```

Question 6. [5 MARKS]

Using the same notation as in lecture, draw the B+Tree with capacity 3, that results from the following sequence of inserts. We will only mark the final B+Tree.

```
t = BPTree()
for v in [10,5,7,20,15,2,1,0,6,21,30,35,32]:
    t.insert(v)
```

Question 7. [15 MARKS]

Part (a) [5 MARKS]

Let $T_1(n)$ be the worst case running time for $f1$ on input n. Let $T_2(n)$ be the worst case running time for $f1$ on input n (etc.). In the space provided, write a best $O(?)$ running time for each of python functions below. You do not need to show your work.

```
def f1(n): # T_1(n) is in O(__________)
    i=j=0
    while i<n:
        j=j+2*i+1
        i=i+1
    return j

def f2(n): # T_2(n) is in O(__________)
    i=j=0
    while i<10000: # this is not a typo
        j=j+1
        i=i+1
    return j


def f3(n): # T_3(n) is in O(__________)
    i=j=0
    while j<n:
        i=i+f1(n) # see f1, above
        j=j+1
    return i

def f4(n): # T_4(n) is in O(__________)
    s=0
    for i in range(n):
        if i>(2*n)/3:
            for j in range(n):
                s=s+i
        else:
            s=s+3
    return s

def f5(n): # T_5(n) is in O(__________)
    s=0
    for i in range(n):
        if i==0 or i==7 or i==5020:
            for j in range(n):
                s=s+i
    return s
```

Part (b) [5 MARKS]

Let $t(n)$ be the number of milliseconds that $f7(L)$ takes to run on random lists of size $n = len(L)$. On my laptop, I found that $t(1000) = 300, t(2000) = 822, t(3000) = 1600, t(10000) = 18000, t(20000) = 71500$. I am interested in estimating $t(100000)$, using the numbers in this example, explain how I would do this. For 5 marks, setup the equations that need to be solved, **do not solve them.**

```
def f7(L): # T_7(n) is in O(n^2), where n=len(L)
    # handle the special case of a list with only one distinct value
    all_same=True
    for i in range(len(L)):
        if L[0]<>L[i]:
            all_same=False
    if all_same:
        return len(L)

    # handle the general case
    m=0
    for i in range(len(L)):
        d=0
        for j in range(len(L)):
            if L[i]==L[j]:
                d=d+1
        if d>m:
            m=d
    return m
```

Part (c) [5 MARKS]

State what `f7` computes and then re-write it so that it is much more efficient. State the O(?) runtime for your algorithm. If you use interesting python functions, you should talk about their behaviour.

BSTree.py:

```
class BSTree:
    ''' Note: keys appear at most one time in this tree '''
    def __init__(self):
        self._tree=BSTreeNull()

    def delete(self, key):
        self._tree=self._tree.delete(key)

    def insert(self, key):
        self._tree=self._tree.insert(key)

    def is_empty(self):
        return(self._tree.is_empty())

    def get_max(self):
        return self._tree.get_max()

    def __str__(self):
        return str(self._tree)

class BSTreeNode:
    def __init__(self, key):
        self._left=BSTreeNull()
        self._right=BSTreeNull()
        self._key=key

    def delete(self, key):
        if key<self._key:
            self._left=self._left.delete(key)
            return self
        if key>self._key:
            self._right=self._right.delete(key)
            return self
        # key == self._key
        if self._left.is_null() and self._right.is_null():
            return self._left # any BSTreeNull will do

        if self._left.is_null():
            return self._right

        if self._right.is_null():
            return self._left

        m=self._left.get_max()
        self._key=m._key
        self._left=self._left.delete(self._key)
        return self

    def get_max(self):
        m=self._right.get_max()
        if m.is_null():
                return self
        else:
```

```
            return m

    def is_null(self):
        return False

    def insert(self, key):
        ''' insert key in my subtree if not already there, return the root of my subtree '''
        if key<self._key:
            self._left=self._left.insert(key)
        elif key>self._key:
            self._right=self._right.insert(key)
        else:
             pass
        return self

    def __str__(self):
        return str(self._left)+" "+str(self._key)+" "+str(self._right)

    def is_empty(self):
        return False

class BSTreeNull:
    def __init__(self):
        pass

    def delete(self, key):
        return self

    def is_empty(self):
        return True

    def is_null(self):
        return True

    def get_max(self):
        return self

    def insert(self, key):
        ''' insert key in my subtree if not already there, return the root of my subtree '''
        return BSTreeNode(key)

    def __str__(self):
        return ""
```