MARKING KEY

The University of British Columbia

MARKING KEY

Computer Science 260

Midterm #2 Examination

12:30 noon, Thursday, March 15, 2012

Instructor: K. S. Booth **Time: 70 minutes (one hour ten minutes)** **Total marks: 70**

First _____ Last _____ Student No _____
Printed first name **Printed last name**

Signature _____ Lecture Section **201** Lab Section _____

☛ **This examination has 11 pages. Check that you have a complete paper.**

This is a closed book exam. Notes, books or other materials are not allowed.

Answer all the questions on this paper. The marks for each question are given in { braces }. Use this to manage your time.

Good luck.

READ AND OBSERVE THE FOLLOWING RULES:

1. Each candidate should be prepared to produce, upon request, his or her Library/AMS card.

2. No candidate shall be permitted to enter the examination room after the expiration of 10 minutes, or to leave during the first 10 minutes of the examination.

3. Candidates are not permitted to ask questions of the invigilators, except in cases of supposed errors in examination questions.

4. **CAUTION** — Candidates guilty of any of the following, or similar dishonest practices shall be immediately dismissed from the examination and shall be liable to disciplinary action.

   a. Making use of any books, papers or memoranda, calculators or computers, audio or visual cassette players, or other memory aid devices, other than those authorized by the examiners.
   b. Speaking or communicating with other candidates.
   c. Purposely exposing written papers to the view of other candidates. The plea of accident or forgetfulness shall not be received.

Page	Mean	Max
3	7.23	10
4	5.63	10
5	7.29	10
6	5.97	10
7	2.56	6
8	4.94	8
9	8.02	9
10	6.10	7
Total	47.75	**70**

1. Multiple choice questions { 30 marks — 1 mark per question }

On the next page is a series of short fill-in-the-blanks questions. All of your answers are to be selected from the list below. You may find it convenient to remove this page from the answer booklet so you can look at it while you answer the questions that follow.

1)	abstract	14)	for	27)	leak	40)	reference
2)	activation	15)	forward	28)	level	41)	shallow
3)	backward	16)	function	29)	literal	42)	signature
4)	base	17)	garbage	30)	max-heap	43)	stack
5)	BST	18)	global	31)	min-heap	44)	static
6)	class	19)	heap	32)	overload	45)	stream
7)	constant	20)	helper	33)	override	46)	template
8)	deep	21)	heterogeneous	34)	postorder	47)	trichotomy
9)	default	22)	homogeneous	35)	preorder	48)	v-pointer
10)	depth	23)	implicit	36)	private	49)	v-table
11)	derived	24)	initializer	37)	protected	50)	vector
12)	do	25)	inorder	38)	public	51)	virtual
13)	explicit	26)	iteration	39)	recursion	52)	width

Each statement will have one ■■■■■■■■ within it, which is where the missing term or phrase would appear. Choose the best answer from among those above and write its number in the space provided in the first column. Do not write the term or phrase. It may be a good idea to read over the list of terms and phrases before you start answering. Some of the terms listed may not appear in any of the statements, some may appear in more than one statement, and some many appear in exactly one statement.

Continue on to the next page…

You may remove this page from the exam booklet.

Read the instructions on the previous page. Enter the number for your answer in the first column. Do not write words!

(a)	**11**	A subclass is a ■■■■■■■■ class of a superclass. derived
(b)	**4**	A superclass is a ■■■■■■■■ class for a subclass. base
(c)	**32**	A method in a subclass that has the same name but has a different signature than a method in a superclass will ■■■■■■■■ the method in the superclass. overload
(d)	**33**	A method in a subclass that has the same name and the same signature as a method in a superclass will ■■■■■■■■ the method in the superclass. override
(e)	**1**	A pure virtual method is ■■■■■■■■. abstract
(f)	**1**	A class that has one or more pure virtual methods is ■■■■■■■■. abstract
(g)	**1**	The method declared here is ■■■■■■■■. `virtual void foo() = 0;` abstract
(h)	**51**	Every object in a class has a v-pointer if the class has one or more ■■■■■■■■ methods. virtual
(i)	**51**	A class has exactly one v-table if and only if the class has one or more ■■■■■■■■ methods. virtual
(j)	**9**	A constructor that has no parameters is the ■■■■■■■■ constructor. default

(This question is continued on the next page.)

Put a slash through every wrong answer, and an X through every blank answer. Total the number of correct answers and write at top of page AND on first page.

(This question is continued from the previous page.)

(k)	**22**	Unlike structs, arrays are [redacted] data types. homogeneous
(l)	**44**	The keyword [redacted] is used to indicate that the scope of a function declared outside of a class declaration only provides visibility within the compilation unit in which it is declared. static
(m)	**44**	The keyword [redacted] is used to indicate that a function declared inside of a class declaration does not have an implicit **`this`** parameter. static
(n)	**44**	The keyword [redacted] is used to indicate that the scope of a variable declared outside of any class declaration or block only provides visibility within the compilation unit in which it is declared. static
(o)	**44**	The keyword [redacted] is used to indicate that the scope of a variable declared inside a block has local scope but does not have automatic extent. static
(p)	**44**	The keyword [redacted] is used to indicate that a variable declared inside a class declaration is not an instance variable. static
(q)	**42**	Whether a function is a method or regular function and the class it belongs to if it is a method are all part of the [redacted] for the function. signature
(r)	**24**	The [redacted] list in the following constructor definition ensures that the member variable **`count`** has a value after the object has been constructed. `InSet::InSet() : count(0) {}` initializer
(s)	**20**	The private methods **`copy()`** and **`cleanup()`** that we often declare for a class are examples of [redacted] methods. helper
(t)	**16**	The type of the parameter in the following function declaration is [redacted]. `void foo( void (*bar) (int))` function

(This question is continued on the next page.)

(This question is continued from the previous page.)

(u)	**35**	Traversing the nodes of a tree in ■■■■■■■■■ visits the parent before the children. preorder
(v)	**34**	Traversing the nodes of a tree in ■■■■■■■■■ visits the parent after the children. postorder
(w)	**31**	In any ■■■■■■■■■ the value stored in any parent is never greater than the values stored in any of the parent's children. min-heap
(x)	**30**	In any ■■■■■■■■■ the value stored in any parent is never less than the values stored in any of the parent's children. max-heap
(y)	**5**	In any ■■■■■■■■■ the value stored in any parent is never less than the value stored in any left child of the parent. BST
(z)	**5**	In any ■■■■■■■■■ the value stored in any parent is never greater than the value stored in any right child of the parent. BST
(aa)	**31**	A priority queue in which the lowest value is always the next to be removed is easily implemented using the ■■■■■■■■■ data structure. min-heap
(bb)	**39**	A function or method that invokes itself is an example of ■■■■■■■■■. recursion
(cc)	**2**	" ■■■■■■■■■ record" is another name for "stack frame" when we talk about the run time behavior of a program. activation
(dd)	**15**	The ■■■■■■■■■ declaration of a struct **`Foo`** preceding the declaration of a struct **`Bar`** allows an instance of **`Foo`** to have a field pointing to an instance of **`Bar`** and an instance of **`Bar`** to have a pointer to an instance of **`Foo`**. forward

2. Class declarations and definitions { 10 marks }

Refer to the following class declaration that might appear in a header file.

```
class A
{
  public:
    static int foo(); // Always returns one hundred.
    friend char bar(); // Always returns 'a'.
    static double e; // Always is 2.71828
    static const double pi; // Always is 3.14159
    static const int two = 2; // Always is 2
};
```

Write a complete set of definitions as they should appear in the corresponding source file so that all of the declared members are defined at run time and they behave as they are described by the comments in the header file.

ANSWER:

½ mark off for each part that is incorrect in each of the five parts (maximum -2 per part).

```
double A::e = 2.71828;
```

- static cannot be used as a qualifier (it has already been used)
- A:: is required or the compiler will not know it belongs to the class A

```
const double A::pi = 3.14159;
```

- static cannot be used as a qualifier (it has already been used)
- A:: is required or the compiler will not know it belongs to A
- const is required

```
int A::foo()
{
  return 100;
}
```

- static cannot be used as a qualifier (it has already been used)
- A:: is required or the compiler will not know it belongs to A
- a value must be returned

```
char bar()
{
  return 'a';
}
```

- friend cannot be used as a qualifier (it has already been used; what would it be a friend of?)
- a value must be returned

Subtract 2 marks if there is ANYTHING for "two"
- NO DEFINITION REQUIRED OR ALLOWED FOR "two" because the declaration has already defined it.

3. Class declarations and method definitions with dynamic memory { 14 marks }

Refer to the following class declaration for all parts of this question. It is part of a complete class declaration for **`IntSet`** that is similar to what we have seen in lectures and on the first assignment. In your answers use only methods and variables that are show here (not others that may exist).

```
class IntSet {
  private:
    static const int CAPACITY = 5;
    static const int RATIO = 2;
    void grow();          // Expands capacity by a factor of RATIO
    int capacity_;
    int *data_;
    int count_;
      . . .
  public:
      . . .
    bool find( int entry ) const;
    int get( int index ) const;
    int size() const;     // returns the number items in the IntSet
};
```

(a) { 6 marks } Write a complete definition for the helper method **`grow()`** based on the information in the declaration above and the example in the first assignment. When **`grow()`** has finished its work, the object should be able to hold more items than it did previously.

ANSWER:

```
void IntSet::grow()
{
   int* temp = new int[RATIO*capacity_];
   for (int i=0; i<count_; i++ )
   {
     temp[i] = data_[i];
   }
   delete [] data_;
   data_ = temp;
   capacity_ *= RATIO;
}
```

The code you were asked to write came verbatim from code that was given to you on Assignment 1.

½ mark off for each missing or wrong item

- return value type must be void
- IntSet:: is required
- grow() with no parameters
- must get a new array of int
- size of new array should be RATIO * size of old array
- for loop to copy values from data_ to temp
- look limits should be zero to capacity_ or count_
- must delete old data_
- delete must have []
- must set data_ to point to new array
- must set new value to capacity_
- sometimes order makes a different (check for this)

Continue on to the next page…

(b) { 6 marks } A set is strictly contained in another set if all of its items are contained in the other set but the other set has at least one item that is not in the first set.

Write a C++ function definition that compares two **`IntSet`** objects and returns **`true`** if the first is strictly contained in the second, otherwise it returns **`false`**. The first line of the function definition is provided for you. Note that the function is not a friend of the class.

ANSWER:

```
bool operator<( const IntSet& A, const IntSet& B )
{
  int count = A.size();
  if ( A.size() >= B.size() ) return false;
  for ( int i=0; i<count; i++ )
  {
    if ( !B.find( A.get( i ) ) ) return false;
  }
  return true;
}
```

RANT

The function returns a bool. Don't use 0 and 1 to represent false and true. Use the built-in constants "true" and "false". This is a course in C++, not in C.

-2 if the code tries to look at member variables. This is function, not a method, and it is not a friend of the class. So it has to use accessors because it cannot look at member variables. The objects are const, to mutators cannot be used.

-1/2 for each return case that is wrong (three of them are usually needed.)

-1/2 if size of sets not checked properly

-1/2 if loop iteration count is not right

-1 if loop does not do a sensible job of checking to see if A is strictly contained in B. What many students missed was getting the case A==B correct.

-1/2 if there are other problems

(c) { 2 marks } The running time of the function you wrote is probably $O(m \cdot n)$ when the first set has m items and the second has n items. Suggest how a more efficient solution could be achieved that has a running time of $O(n \log \cdot n)$ (or explain why your solution is already that fast). **Do not write any code**.

ANSWER:

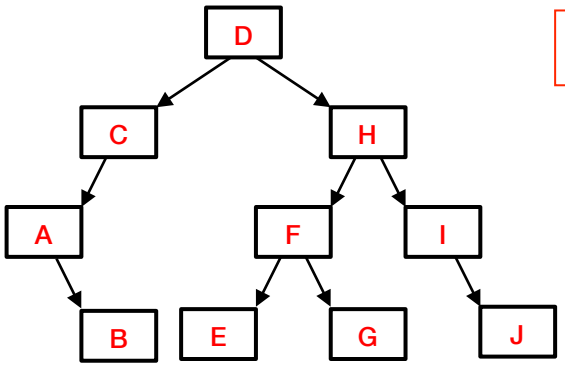
If we get the m items from the first set and the n items in the second set we can sort each of them in $O(m \log m)$ and $O(n \log n)$ respectively. We can then do a linear time comparison to see if all of the items in the first ordered set are in the second ordered set. Because it must be the case that $m<n$ (otherwise we can provide an answer in constant time) the total time is $O(n \log n)$.

An unacceptable answer was to change the IntSet class to use a BST, heap, or other structure. That is a fine thing to do if you are the designer of the IntSet class, but the question was how to improve the function in part (b), not how to re-design the class. All good solutions would need to extract all of the integers from the two sets and then (using sorting or something equivalent) efficiently figure out if A is contained in B. There is a O(n log m) solution, is if $m << n$ (much less than) would be faster than O(n log n).

2 for getting most of it right
1 for an OK answer
0 if you wrote code unless there was a complete answer that did not rely on the code

4. Binary Search Trees { 9 marks }

(a) { 5 marks } The ten letters **A-B-C-D-E-F-G-H-I-J** are inserted into a binary search tree (BST) in some order. The tree shown below is the result. Write the ten letters in their correct positions.



1/2 mark for each node label

(f) { 1 mark } How many leaves are there in the BST?

4 (B, E, G, J)

All we asked for is the numbers. The list of nodes is to help you understand the solutions.

(g) { 1 mark } How many nodes are there in the BST whose height is two?

2 (C, H)

(h) { 1 mark } How many nodes are there in the BST whose depth is three?

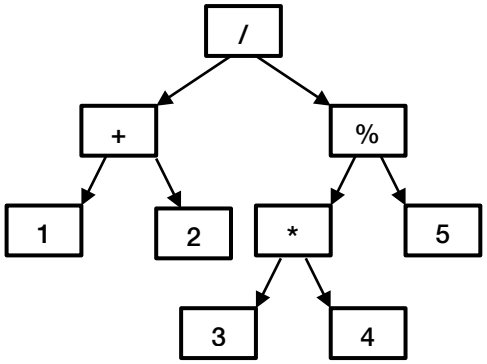
4 (B, E, G, J)

(i) { 1 mark } What is the height of the BST?

3

5. Binary expression trees { 7 marks }

Consider the following binary expression tree when answering each part of this question.



(f) { 2 mark } What is the preorder expression for the tree?

```
/ + 1 2 % * 3 4 5
```

(g) { 2 mark } What is the postorder expression for the tree?

```
1 2 + 3 4 * 5 % /
```

RANT

Preorder and postorder (both of which are Polish notation) are used precisely because they require no parentheses!

Solutions with parentheses got zero marks.

(h) { 2 mark } What is the inorder expression for the tree that has the least parentheses assuming integer arithmetic and the precedence rules for C++?

```
(1 + 2) / (3 * 4 % 5)
```

Additional parentheses inside the denominator are not required, although we will accept (3 * 4).
Why do we know that these extra parentheses are not needed?

(i) { 1 mark } What is the value of the expression represented by the tree?

```
1 or 1.5
```

This was supposed to again assume integer arithmetic, but we will allow 1.5 for this exam, but not for future exams.

½ mark for 1 ½ or 3/2.

RANT

Given the questions on the Pre-Test at the start of the term, and the wording in part (h) of this question, everyone should have understood that the question was about integers. None of the numbers had decimal points. In future exams the rules for arithmetic expressions will always be the same those in C++ unless otherwise stated. So the only correct answer is "1".

Continue your answers here – make sure to identify the questions whose answers are here!

More rant (in case you haven't had enough yet)

The fill-in-the-blank multiple choice questions had a couple of questions that had the same answer. These questions always carry with them the possibility of one or more terms being used multiple times, and also of some terms not being used at all.

The term "static" was the big winner on this exam. It was the answer to five consecutive questions. There is a reason for this. The term "static" has many meanings in C and even more in C++. The questions explored your knowledge of these meanings.

The term "abstract" was the next most common answer (three times) and virtual was close behind (two times). In at least one of the questions "virtual" would have been an OK answer except that "abstract" was a much better answer. If you don't know the distinction between these two terms, the seven questions will actually help you figure it out.

The question about grow() asked you to remember the general idea behind code that you were given on Assignment 1. In general, when you are given source code you should read it enough to understand how it works, especially when it is for a class that we have been using most of the term as an example of features in C++.

The questions about the < operator for IntSet is an example of a whole genre of questions where you are asked to write one or more functions that use the basic methods provided by the class to build additional functionality. In some cases you might be asked to implement new methods in the class, or a function that is a friend of the class. In those cases you can access the member variables. But in this case the question explicitly told you that the function was not a friend of the class, so you have to keep away from the private parts of an object. This means you can only use accessors and (if the objects are not const) mutators. In this question both parameters were call-by-const-ref (why?), so you could only use accessors, no mutators.

An example of a similar type of question would be to implement the body of the following non-friend function for the IntSet class.

```
IntSet operator*( const IntSet& A, const IntSet& B )
```

An example of a similar type of question would be to implement the body of the following non-friend function for the IntSet class. For sets the obvious meaning for + is "intersection". The implementation would have a lot in common with the < operator you saw on this exam. In both cases you need to "get inside" the two sets and figure out which elements are in common. For the intersection operator you would need to keep track of the common elements and build a new IntSet that had just those elements and then return that set as the result of the operator.

It is not easy to do this efficiently without access to the member variables. It is fairly easy to figure out the elements of the new set in $O(n \log n)$ time. It harder to know how to get them "into" a new IntSet in less than $O(n^2)$ steps without knowing the details of the class implementation (which we are not supposed to know or rely on!).