### The University of British Columbia Computer Science 252

**MARKING KEY** 

# 2nd Midterm Exam

6:30 PM, Monday, November 8, 2004

Instructors:	K. Booth & N.	Hutchinson	Time: 90 minutes	Total marks: 90
Name (PRINT)	(Last)	(First)	Student N	No
Signature			Lecture S	ection

### **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, but you may use **one** piece of  $8-1/2 \times 11$  paper (both sides) with your own notes either hand-written, photocopied, or both.

Answer all the questions on this paper. Give **short but precise** answers. Always use point form where it is appropriate. 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 twenty minutes, or to leave during the first twenty minutes of the examination.
- 3. Candidates are not permitted to ask questions of the invigilators, except in cases of supposed errors or ambiguities 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	Marks	Max
3		10
4		10
5		10
7		15
8		17
9		10
10		18
Total		90

## 1. Multiple choice questions { 30 marks -- 10 marks per page }

On the next three pages there are a series of short fill-in-the-blanks statements. All of your answers are to be selected from the list below.

1)	abstract	15)	height	29)	root
2)	circular	16)	leaf	30)	segmentation fault
3)	class	17)	level	31)	singly
4)	constructor	18)	memory leak	32)	source
5)	current directory	19)	midnight	33)	static
6)	dangling pointer	20)	noon	34)	subclass
7)	default constructor	21)	overload	35)	superclass
8)	depth	22)	override	36)	template
9)	destructor	23)	polygamy	37)	title page
10)	doubly	24)	polymorphism	38)	triply
11)	dummy	25)	private	39)	virtual
12)	dynamic	26)	protected	40)	VPTR
13)	exit	27)	public	41)	VTABLE
14)	function	28)	recursive	42)	wooden chair

Each statement will have one **manufacture** within it, which is where the missing term or phrase would appear. Choose the <u>best</u> answer from among those above and write its <u>number</u> in the space provided in the <u>first</u> 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 <u>may not appear</u> in any of the statements, some may appear in <u>more than one</u> 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) [		Signad handgony of the <b>Second Second</b> must be submitted for Assignments 1.6		
(a)	37	Signed hardcopy of the		
(b)	20	All signed hardcopy for an assignment is due before <b>Management</b> on the Monday after the assignment due date for Assignments 1-6.		
(c)	19	All electronic copies of files for an assignment are due before <b>HEADER</b> on the Friday that is the assignment due date for Assignments 1-6.		
( <b>d</b> )	18	The code int* iPtr = new int[10]; iPtr = new int;		
		will result in a <b>management</b> in the program.		
(e)	6	The code int* jPtr = new int; delete jPtr;		
		will result in a <b>manufacture</b> and thus <b>jPtr</b> should not be dereferenced.		
( <b>f</b> )	30	The code int* kPtr; *kPtr = 6;		
		will often result in a when kPtr is dereferenced.		
(g)	11	Some linked lists have a <b>manufacture</b> node with <u>no content</u> as the <u>first</u> node in the list to make insertion and deletion easier because there are no special cases when the list is empty.		
( <b>h</b> )	2	In a linked list the <u>last</u> node in the list has a pointer to the <u>first</u> node in the list.		
(i)	31	In a linked list each node has a pointer to the node that <u>follows</u> it but <u>no</u> pointer to the node that <u>precedes</u> it.		
(j)	10	In a linked list each node has a pointer to the node that <u>follows</u> it <u>and</u> a pointer to the node that <u>precedes</u> it.		

## (continued on the next page)

### (continued from previous page)

( <b>k</b> )	22	A method in a derived class that has the <u>same</u> signature as a method in the base class is said to <b>EXECUTE</b> the method in the base class.
(1)	21	A method in a derived class that has a <u>different</u> signature but the <u>same name</u> as a method in the base class is said to <b>Exercise</b> the method in the base class.
( <b>m</b> )	34	A derived class is a <b>Example of</b> its base class.
( <b>n</b> )	35	A base class is a <b>Here Here</b> of any of its derived classes.
(0)	26	A member variable of a base class is <u>always</u> accessible (readable and writeable) in any derived class but is <u>never</u> accessible to clients of the base class.
( <b>p</b> )	27	A member variable of a base class is <u>always</u> accessible (readable and writeable) in any derived class <u>and</u> to all clients of the base class.
(q)	25	A member variable of a base class is <u>never</u> accessible (readable or writeable) to any derived class <u>or</u> to clients of the base class.
( <b>r</b> )	12	A call to a member function of a base class will use <b>Definition</b> binding if the method is invoked through a <u>pointer</u> to an object and the object is an instance of the <u>base</u> class.
(s)	12	A call to a member function of a base class will use <b>members</b> binding if the method is invoked through a <u>pointer</u> to an object and the object is an instance of a <u>derived</u> class.
( <b>t</b> )	33	A call to a member function of a base class will use <b>derived</b> binding if the method is invoked <u>on an object</u> that is an instance of a <u>derived</u> class.

## (continued on the next page)

## (continued from previous page)

<b>(u)</b>		In binary trees every <b>and the set of the se</b>
	16	
( <b>v</b> )	29	In binary trees the <b>EXAMPLE</b> node has no parent.
(w)	15	In binary trees the <b>Example 1</b> of a node is the length of the longest path from that node to some other node in the tree.
(x)	8	In binary trees the <b>EXECUTE</b> of a node is the length of the path from the root to that node.
( <b>y</b> )	17	In binary trees the <b>Description</b> of a node is the length of the path from the root to that node. (The answer to this must be <u>different</u> from the answer to the previous question!)
(z)	15	In binary trees the <b>Example of</b> a tree is the length of the longest path from the root to some other node in the tree.
(aa)	9	The method for a base class is almost always declared to be virtual because subclasses may add member variables that point to dynamically allocated memory that the base class will not know about.
( <b>bb</b> )	1	A parameter-less mutator returning no value is declared using this syntax in a class declaration, which makes it a <u>pure virtual</u> or <b>method( void ) = 0;</b>
(cc)	3	In a template declaration the keyword <b>means</b> means <u>exactly</u> the same thing as the keyword <b>typename</b> .
( <b>dd</b> )	42	The King (Elvis Presley) says "if you can't find a partner use a

Below is a declaration for the **Queue** class from a **header file**, **Queue**.**h**, from Assignment 1, followed by the definition for the **dequeue** () method from a **source file**, **Queue**.**cpp**. These will be useful when you answer the <u>next two questions</u> on this exam.

```
header file
typedef int Item type;
                                        // type for items in the Queue
class Queue
{
   public:
      Queue();
      bool enqueue( const Item type& item );
      Item type dequeue();
      bool empty() const;
      bool full() const;
      void print() const;
   private:
      static const int CAPACITY = 50; // maximum number of items
      int count;
                                        // number of items in the Queue
                                        // index of first item
      int front;
                                        // array of items
      Item type data[CAPACITY];
};
```

\_\_\_\_\_ source file

Item\_type Queue::dequeue()
// Remove item from front of Queue
// PRE: (none)
// POST: An item is removed from the Queue and returned, unless the
// Queue is empty, in which case the outcome is undefined.
{
 count--;
 return data[front++];
}

# Continue on to the next page... You may remove this page from the exam booklet.

## 2. Templatized class declarations { 15 marks }

Write a <u>complete</u> declaration for the <u>templatized</u> version of the **Queue** class where:

- a) the declaration is <u>templatized</u> with a parameter **Item\_type** for the type of items to be stored;
- b) declarations for the "Big Three" are added to the class declaration;
- c) the **full()** method is <u>replaced</u> with a **size()** method in the class declaration;
- d) the internal data structure is changed to a <u>dynamic circular array</u> that expands as required;
- e) the <u>default</u> size of the dynamic circular array is 25 items;
- f) a growth factor for enlarging the dynamic circular array is declared to be 1.5;
- g) an additional <u>member variable</u> is added to keep track of the size of the dynamic array.

### No other changes are required or permitted in the class declaration.

Be sure to include an appropriate <u>invariant</u> for the class. This is missing in the example on the previous page but is **required** in your solution below!

### ANSWER:

```
// data points to an array of size maxCount
// 0 <= count <= maxCount</pre>
// front is the index (mod maxCount) of the head item
// items are in FIFO order from [front] to [front+count} mod maxCount
template<typename Item type>
class Queue
{
   public:
      Queue();
      ~Queue();
      Queue( const Queue& other );
      Queue& Operator= ( const Queue& other );
      bool enqueue( const Item type& item );
      Item type dequeue();
      bool empty() const;
      int size() const;
      void print() const;
   private:
      static const int CAPACITY = 25; // default array size
      static const float GROWTH = 1.5; // growth factor
                                        // maximum number of items
      int maxCount;
      int count;
                                        // number of items in the Queue
                                        // index of first item
      int front;
      Item type* data;
                                        // pointer to array of items
```

## **3.** Template method definitions with exception handling { 17 marks }

(a) **{8 marks}** Write a <u>complete</u> definition for the <u>templatized</u> version of the **dequeue()** method where **Item\_type** specifies the type of the items stored in a **Queue**. Be sure to include appropriate <u>preconditions</u> and <u>postconditions</u>. The method should <u>throw</u> a **logic\_error** when the preconditions are not satisfied.

#### ANSWER:

```
template<typename Item_type>
Item_type Queue<Item_type>::dequeue() throw ( logic_error )
// Mutator
// PRE: The queue is not empty.
// POST: Remove and return the first item in the queue.
{
    if ( empty() ) throw logic_error( "Empty queue" );
    count--;
    int index = front;
    front = (front++)%maxCount;
    return data[index];
}
```

(b) **{9 marks}** Write a <u>complete</u> definition for the <u>templatized</u> version of the default constructor. Be sure to include appropriate <u>preconditions</u> and <u>postconditions</u>. The method should throw a **runtime\_error** when the preconditions are not satisfied or the **Queue** cannot be initialized.

#### ANSWER:

```
template<typename Item type>
Queue<Item type>::Queue() throw ( runtime error )
// Constructor
// PRE:
        (none)
// POST: Queue is initialized unless there is no memory left, in which
11
         case an exception is thrown.
{
   maxCount = CAPACITY;
   count = 0;
   front = 0;
   data = NULL;
   try
   {
      data = new Item type[ maxCount ];
   }
   catch ( bad alloc )
   ł
           throw runtime error ( "Cannot create Queue" );
   }
}
```

## 4. Exception handling { 10 marks }

The following program will produce <u>exactly</u> five lines of <u>output</u>, one for each of the five sets of try-catch blocks in the main program, when it is compiled under g++ and run on the undergrad servers.

```
#include <iostream>
#include <exception>
#include <stdexcept>
using namespace std;
void throwException( void )
{ throw ( logic error( "logic error!" ) ); }
int main() {
   try { throwException(); }
   catch ( | exception& | e1) { cout << e1.what() << endl; }</pre>
   catch (\ldots) \{ cout << "What?" << endl; \}
   try { throwException(); }
   catch ( exception
                         e2) { cout << e2.what() << endl; }</pre>
   catch (\ldots) { cout << "What?" << endl; }
   try { throwException(); }
   catch ( logic_error ) e3) { cout << e3.what() << endl; }</pre>
   catch (\ldots) { cout << "What?" << endl; }
   try { throwException(); }
   catch ( runtime error e4) { cout << e4.what() << endl; }
   catch (\ldots) { cout << "What?" << endl; }
   try { throwException(); }
   catch (
            runtime error&
                             e5) { cout << e5.what() << endl; }
}
```

In the spaces below, write the five (5) lines of <u>output</u> produced by <u>executing</u> the above program.

### ANSWER:

Line 1: logic\_error! Line 2: St9exception Line 3: logic\_error! Line 4: What? Line 5: abort

## 5. Tree traversal { 18 marks }

(a) For the binary tree below, label each node with its <u>height</u> in the tree.



Continue your answers here – make sure to identify the question(s) whose answer(s) are here!