

CMPT 117 (02)
Instructor: Kevin Grant

Midterm Examination
February 22, 2006

Name:

Craig Bloch-Hansen

Student Number:

147742

Rules:

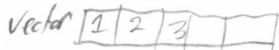
- 1) You have 50 minutes to complete the exam. You have approximately one minute per mark, so allocate your time wisely
 - 2) This is a closed book examination. You may not use any reference material (electronic or non-electronic) of any kind.
 - 3) You may not communicate with others
 - 4) Cheating is not tolerated, and will result in a forfeiture of the exam, and a 0% grade.
-

Good luck!!

Part I: Theory (10 Marks)

1. (4 Marks) In class, we discussed four advantages that linked lists have over vectors. Name two of them, and explain with a small example/illustration.

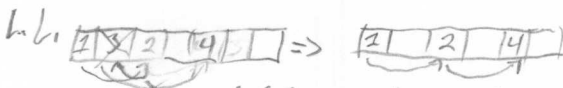
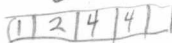
① Do not need large blocks of contiguous memory in heap.
 • Vectors are arrays and need to have sequential memory
 — whereas linked lists have pointers to the next part of the list. Each item in the list is stored as a node made up of data and a pointer to the next node in the list.



② - Do not need to copy and shift information when deleting.



- need to copy 4 into 3's position; this is a linear time operation



- deletes node and pointer and changes pointer of previous node.
 constant time operation

2. (3 marks) When a class uses heap memory, what three methods must we always include in order to ensure correct functionality?

3. (3 Marks) Compare and contrast our *container* class with our *vector* class, using our following table:

	Insertion	Deletion	Search
Container	$O(c)$	$O(c)$	$O(n)$
Vector	$O(n)$	$O(n)$	$O(n)$

Part II: Code Tracing (5 Marks)

Often in debugging, many programmers (who do not have access to Visual Studio) will leave little *cout* commands to help them find errors in code. In the following code segments, *cout* commands have been left in order to follow what the code is doing.

a) (3 marks) What would the output of this code be?

```
int f1(int x) {
    cout << "Calling f1(int)" << endl;
    return x * x;
}

double f1(double x) {
    cout << "Calling f1(double)" << endl;
    return x * x;
}

float f1(float x) {
    cout << "Calling f1(float)" << endl;
    return x * x;
}

int main() {

    f1(2.0f);
    f1(2);
    f1(2.0);

    f1(f1(2.0f) * f1(2.0));

}
```

Output:

Calling f1(float)
Calling f1(int)
Calling f1(double)
Calling f1(float)
Calling f1(double)
Calling f1(double)

b) (2 marks) If there were no *cout* commands in our *f1* functions, write a function template for *f1* that would take the place of all three functions above.

template <Item> ^{typename}

Item f1(Item x)
{ return x * x; }

3

1.5

Part III: Linked Lists (10 marks)

NOTE: The header file for the node class and linked list toolkit has been included at the back of your exam. It is not necessary to use a function from the toolkit in the following questions, but feel free if you like.

- a) (5 marks) Write a function `list_count`, that takes a pointer to the head of a linked list and a data item, and returns the number of times that data item occurs in the list. Be sure your function is template'd, and that you use an appropriate parameter type in the header.

4.5

```
template <Item>
size_t list_count(const Node<Item> *head, const Item &entry) const
{
    size_t result = 0;
    for (Node<Item> *i = head; i != NULL; i = i->Next)
    {
        if (i->data == entry)
            result++;
    }
    return result;
}
```

- b) (5 marks) Write a function `list_max`, that takes a pointer to the head of a linked list, and returns the item that occurs the most frequently inside the list. Be sure your function is template'd, and that you use an appropriate parameter type in the header. (HINT: You may use your `list_count` function).

6.5

```
template <Item>
Item list_max(const Node<Item> * &head)
```

Part IV: (5 Marks) Complexity

(a) (3 marks) Give the order notation (Big O) for the following functions:

(i) $3n^3 + 6n + 4$ $O(n^3)$

(ii) $(n^2)(2n^3)$ $O(n^5)$

(iii) $(3n^2 + 2 \log n)(4 \log n + 2)$ $O(n^2)$

(b) (2 marks) What is the complexity of the following algorithm? Give a brief explanation for your answer.

```
int bisector(int *a, size_t n) {  
    int min = a[0];  
    int max = a[0];  
    for (size_t i = 1; i < n; i++)  
        if (a[i] < min)  
            min = a[i];  
    for (size_t i = 1; i < n; i++)  
        if (a[i] > max)  
            max = a[i];  
    return (min + max)/2;  
}
```

Complexity is linear.

linear because there are 2 for loops with n number of operations, but since one is not imbedded in another then each takes a linear amount of time as such they are added giving the total complexity of $3+2n$, which is linear.

Part V: (20 Marks) Object-Oriented Programming and Data Types

Numbers are represented in a computer in *binary (base 2)*. Each digit is either a 0 or a 1.

In contrast, humans are used to dealing with *decimal (base 10)* numbers, where each digit comes from the set {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. We could store a number using a linear array of single-valued digits. So we could store the number 316 as an array of three values.

[6,1,3]

Notice that the higher-order digits are stored in higher-ordered indices. Here are some other examples:

62 : [2,6]

729: [9,2,7]

4513: [3,1,5,4]

Your job will be to write a class called *Number*. *Number* will store an array of numbers between 0 and 9, as per the description above.

Your number class will have two constructors. The default constructor will take a *size_t* variable indicating the number of digits the number can hold. The other constructor is a copy constructor.

Your number class will provide access to its digits through the subscript [] operator.

On the following page is a *partially* completed number class. Note that the implementation for the copy constructor has been deliberately omitted.

Here is how we would build the number “240” using our number class.

Number p(3); // indicates three digits

p[0] = 0;

p[1] = 4;

p[2] = 2;

You may use the following function in your code:

max(a, b) – returns the maximum value between a and b.

//number.h

```
class Number {
public:
    Number(const size_t &sz = 1);
    Number(const Number &n);
    ~Number();

    // Mutators
    int & operator[](const size_t &i);
    Number & operator = (Number &n);

    // Accessors
    size_t size() const {return num_digits;}
    Bool odd (Number n1) const;

private:
    int *data;
    size_t num_digits;
};
```

//number.cpp

```
Number::Number(const int &sz) {
    data = new int[sz];
    for (size_t i = 0; i < sz; i++)
        data[i] = 0;
    num_digits = sz;
}

int& Number::operator[](const size_t &i) {
    assert( (i >= 0) && (i < num_digits) );
    return data[i];
}
```

