

Computer Science II — CSci 1200

Test 1 Questions

Overview

- Test 1 will be held **Tuesday, February 12, 2008, 2:00-3:30pm, Darrin 308**. No make-ups will be given except for emergency situations, and even then a written excuse from the Dean of Students office will be required.
- Purpose: Check your understanding of the basics of (a) C++, (b) solving small computational problems, (c) the standard library (streams, strings and vectors), and (d) memory management.
- Coverage: Lectures 1-7, Labs 1-4, HW 1-3.
- Closed-book and closed-notes. At the start of the test, we will provide a handout summary of important C++ syntax, including standard library classes.
- Below are **many** sample questions. Solutions to most of the problems will be posted on-line.
- The test questions will be **drawn from these questions**, often with minor modifications, and from the lecture, homework and lab problems.
- *How to study?*
 - Work through the sample questions, writing out solutions! You are welcome to do this with other students.
 - Review and re-do lecture exercises, lab and homework problems.
 - Identify the problems that cause you difficulty and review lecture notes and background reading on these topic areas.

Questions

1. Write a code segment that copies the contents of a string into a vector of char in reverse order.
2. Write a function that takes a vector of strings as an argument and returns the number of vowels that appear in the string. For the purposes of this question, a vowel is defined as an 'a', 'e', 'i', 'o' or 'u'. For example, if the vector contains the strings

```
abe
lincoln
went
to
the
white
house
```

Your function should return the value 12.

You may assume that all letters are lower case. Here is the function prototype:

```
int count_vowels( const vector<string>& strings )
```

3. Write a **recursive** function to multiply two non-negative integers using only addition, subtraction and comparison operations. No loops are allowed in the function. The function prototype should be

```
int multiply( int m, int n)
```

4. Write a function called `less_string` that mimics the effect of the `<` operator on strings. In other words, given strings `a` and `b`, `less_string(a, b)` should return `true` if and only if `a < b`. Of course, you may use `<` on individual characters in the string. Start by getting the function prototype correct.

Here are examples of pairs for which your function should return true:

```
a = "abc", b = "abd"
a = "cab", b = "cabbage"
a = "christine", b = "christopher"
```

5. What is the output from the following program? We strongly suggest that you draw the contents of the vectors to help you visualize what is happening.

```
void more_confused( vector<int> a, vector<int> & b )
{
    for ( unsigned int i=0; i<2; ++i )
    {
        int temp = a[i];
        a[i] = b[i];
        b[i] = temp;
    }
    cout << "1: ";
    for ( unsigned int i=0; i<a.size(); ++i )
        cout << a[i] << " ";
    cout << endl;
    cout << "2: ";
    for ( unsigned int i=0; i<b.size(); ++i )
        cout << b[i] << " ";
    cout << endl;
}

int
main()
{
    vector<int> a, b;
    a.push_back(1); a.push_back(3); a.push_back(5);
    b.push_back(2); b.push_back(4);

    more_confused( a, b );
    a[0] = 7; a[1] = 9;
    more_confused( b, a );

    cout << "3: ";
    for ( unsigned int i=0; i<a.size(); ++i )
        cout << a[i] << " ";
    cout << endl;

    cout << "4: ";
    for ( unsigned int i=0; i<b.size(); ++i )
        cout << b[i] << " ";
    cout << endl;

    return 0;
}
```

6. Write a function that takes a vector of doubles and copies its values into two vectors of doubles, one containing only the negative numbers from the original vector, the other containing only the positive numbers. Values that are 0 should not be in either vector. For example, if the original vector contains the values

-1.3, 5.2, 8.7, -4.5, 0.0, 7.8, -9.1, 3.5, 6.6

then the resulting vector of negative numbers should contain

-1.3, -4.5, -9.1

and the resulting vector of positive numbers should contain

5.2, 8.7, 7.8, 3.5, 6.6

Start this problem by writing the function prototype as you think it should appear and then write the code.

7. Give an order notation (“O”) estimate of the worst-case number of operations required by the following sorting function. Briefly justify your answer:

```
void
ASort( vector<double>& A )
{
    int n = A.size();
    for( int i=0; i<n-1; ++i )
    {
        // Find index of next smallest value
        int small_index = i;
        for ( int j=i+1; j<n; ++j )
        {
            if ( A[j] < A[small_index] )
                small_index = j;
        }

        // Swap with A[i]
        double temp = A[i];
        A[i] = A[small_index];
        A[small_index] = temp;
    }
}
```

8. Write a function that takes an array of ints and copies the **even integers** into a dynamically-allocated array of ints, storing the values in **in reverse order**. For example, given an array containing the 12 values

5, -1, 4, 0, 13, 27, 98, 17, 97, 24, 98, 89

the resulting array should contain the values (in order)

98, 24, 98, 0, 4

after the function is completed. When doing dynamic allocation, you may only allocate enough space to store the actual numbers, and no more. Also, you may only allocate the array once, not in a loop.

The return type of the function must be `void`. Start by specifying the function prototype. Think carefully about the parameters needed and their types. Repeat the problem using (a) pointers instead of array subscripting and (b) returning the even integers in a vector instead of an array. Give an “O” estimate for each.

9. Consider the following declaration of a `Point` class, including an associated non-member function:

```
class Point {
public:
    Point();
    Point( double in_x, double in_y, double in_z );
    void get( double & x, double & y, double & z ) const;
    void set( double x, double y, double z );
    bool dominates( const Point & other );
private:
    double px, py, pz;
};

bool dominates_v2( const Point & left, const Point & right );
```

- (a) Provide the implementation of the default constructor — the constructor that takes no arguments. It should assign 0 to each of the member variables.
- (b) The member function `dominates` should return `true` whenever each of the point's coordinates is greater than or equal to each of the corresponding coordinates in the other point. The function should return `false` otherwise. For example, given

```
Point p( 1.5, 5.0, -1 );
Point q( 1.4, 5.0, -3 );
Point r( -3, 8.1, -7 );
```

Then

```
p.dominates( q )
```

should return `true` but the function calls

```
p.dominates( r )    r.dominates( q )    q.dominates(r)
```

should each return `false`. Write member function `dominates`.

- (c) The function `dominates_v2` should be a non-member function version of `dominates`. Its behavior should be essentially the same as the member function version, so that for the `Point` objects defined in (b),

```
dominates_v2( p, q )
```

should return `true` but the function calls

```
dominates_v2( p, r )    dominates_v2( r, q )    dominates_v2( q, r)
```

should each return `true`. Write `dominates_v2`.

10. Consider the following start to a class declaration:

```
class bar {
public:
    bar( int in_x, double in_y, const vector<string>& in_z )
        : x(in_x), y(in_y), z(in_z) {}

private:
    int x;
    double y;
    vector<string> z;
};
```

This class is incomplete because no member functions are defined.

- (a) Write a member function of class `bar` called `OrderZ` that sorts the member variable `z` into increasing order. Show both its prototype in `bar` and its implementation, outside of the declaration for `bar`.
- (b) Write a function that takes a vector of `bar` objects and re-arranges them so that the objects are ordered by decreasing value of `x`, and by increasing `y` for `bar`'s with equal values of `x`. Give two different solutions. One solution uses an `operator<` on `bar` objects and the other uses a non-operator comparison function on `bar` objects. In both cases, you will need to add member functions to `bar`. Be sure to include these in your solution.

11. Given an array of integers, `intarray`, and a number of array elements, `n`, write a short code segment that uses **pointer arithmetic and dereferencing** to add every second entry in the array. For example, when `intarray` is

0	1	2	3	4	5	6	7	8
1	16	4	-3	2	76	9	3	6

and `n==9`, the segment should add $1 + 4 + 2 + 9 + 6$ to get 22. Store the result in a variable called `sum`.

12. Show the output from the following code segment.

```
int x = 45;
int y = 30;
int *p = &x;
*p = 20;
cout << "a:  x = " << x << endl;

int *q = &y;
int temp = *p;
*p = *q;
*q = temp;
cout << "b:  x = " << x << ", y = " << y << endl;

int * r = p;
p = q;
q = r;
cout << "c:  *p = " << *p << ", *q = " << *q << endl;
cout << "d:  x = " << x << ", y = " << y << endl;
```

13. Write a `Vec<T>` class member function that creates a new `Vec<T>` from the current `Vec<T>` that stores the same values as the original vector but in reverse order. The function prototype is

```
template <class T>
Vec<T> Vec<T>::reverse() const;
```

Recall that `Vec<T>` class objects have three member variables:

```
T* m_data;           // Pointer to first location in the allocated array
size_type m_size;   // Number of elements stored in the vector
size_type m_alloc;  // Number of array locations allocated, m_size <= m_alloc
```

14. Here is a program that is *supposed* to detect and output each distinct integer that occurs 2 or more times in an input sequence. Unfortunately, there is a small problem with this program causing it to work incorrectly for some inputs.

```

int main() {
    vector<int> input;          // stores all the numbers
    vector<int> duplicates;   // stores all numbers that appear more than once

    // read in the input & search for duplicates
    int x;
    while (cin >> x)
    {
        input.push_back(x);
        for (int i = 0; i < input.size()-1; i++)
        {
            if (x == input[i])
                duplicates.push_back(x);
        }
    }

    // print out the duplicates
    if (duplicates.size() > 0)
    {
        cout << "These numbers appeared more than once in the input: " << endl;
        for (int i = 0; i < duplicates.size(); i++)
            cout << duplicates[i] << " ";
        cout << endl;
    }
}

```

- (a) First let's look at a test case for which the program works correctly. What is printed on the screen when the user inputs this sequence:

```
1 2 1 3 4 3 5 2
```

- (b) Now, give an example sequence of integers where the program behaves incorrectly. What is the behavior/output for your test case, and what should the output be?
- (c) Describe how to modify the code to fix the problem. Rewrite the part of the program with the bug so that it works correctly for all sequences of integers.

15. What is the output of the following code?

```

int * a = new int[4];
a[0] = 5; a[1] = 10; a[2] = 15; a[3] = 20;

cout << "A: ";
for( unsigned int i=0; i<4; ++i ) cout << a[i] << " ";
cout << endl;

for( int * b = a; b != a+4; b += 2 ) *b = b-a;

cout << "B: ";
for( unsigned int i=0; i<4; ++i ) cout << a[i] << " ";
cout << endl;

int * c = a;
c[3] = 14;
c[1] = -2;
cout << "C: ";

```

```
for( unsigned int i=0; i<4; ++i ) cout << a[i] << " ";  
cout << endl;
```

16. In this problem you will implement a simple class named `Major` to store information about students and their declared majors. Since students often change their minds, your class will need to handle these changes and keep track of how many times a student changed majors. Please carefully read through all of the information below before working on the class design.

Here are several examples of how we will create and initialize `Major` objects:

```
Major sally("Sally");  
Major fred("Fred");  
Major bob("Bob");  
Major alice("Alice");
```

Initially each student's major is listed as undeclared, but they can change their major with the `declareMajor` member function:

```
sally.declareMajor("Economics");  
fred.declareMajor("Information Technology");  
bob.declareMajor("Chemistry");  
sally.declareMajor("Information Technology");  
bob.declareMajor("Psychology");  
bob.declareMajor("Biology");
```

We also need various accessor functions to get the student's name, their major, and the number of times they changed their major. Here's an example use of these member functions:

```
cout << bob.getName() << " changed majors " << bob.numChanges()  
    << " time(s) and is currently majoring in " << bob.getMajor() << "." << endl;
```

Which will result in this output to the screen:

```
Bob changed majors 3 time(s) and is currently majoring in Biology.
```

We can also store multiple `Major` objects in a `vector` to organize the student registration database:

```
vector<Major> all_students;  
all_students.push_back(sally);  
all_students.push_back(fred);  
all_students.push_back(bob);  
all_students.push_back(alice);
```

In the third part of this problem you will write code to sort and output this database with the students grouped by their current major and sorted alphabetically:

```
Biology  
  Bob  
Information Technology  
  Fred  
  Sally  
Undeclared  
  Alice
```

- (a) Using the foregoing sample code as your guide, write the class declaration for the `Major` object. That is, write the *header file* (`major.h`) for this class. You don't need to worry about the `#include` lines or other pre-processor directives. Decide how you are going to store the information for each object. Focus on getting the member function prototypes correct. Use `const` and call by reference where appropriate. Make sure you label what parts of the class are `public` and `private`. Don't include any of the member function implementations here (even if they are just one line). Save the implementation for the next part.
- (b) Now implement the constructor and member functions you declared in the previous part, as they would appear in the corresponding `major.cpp` file.
- (c) Now we need to prepare the student lists for each department. Given the `all_students` vector that stores all of the students, your task is to write code that sorts the students into groups by major and then alphabetically within each group. This can be done with a single call to the `sort` function for vectors. First write the helper function you'll need for the sort operation.
- (d) Finally, write the code that uses this helper function to sort and then output the registration database in this format:

```

Biology
  Bob
Information Technology
  Fred
  Sally
Undeclared
  Alice

```

17. **Very briefly** state two reasons why vectors are preferable to arrays.
18. Show the output of the following code. Assume that all necessary header files have been included.

```

void order( vector<double> a, double min, double & max )
{
    sort( a.begin(), a.end() );
    min = a[0];
    max = a[ a.size()-1 ];
}

int main()
{
    vector<double> b(4);
    b[0]=10; b[1]=21.1; b[2]=2.3; b[3]=7.9;
    double first=0, last=0;
    order( b, first, last );
    cout << "first = " << first << ", last = " << last << '\n';
    for ( unsigned int i=0; i<b.size(); ++i )
        cout << b[i] << ' ';
    cout << '\n';
    return 0;
}

```

19. Write a function that takes a vector of doubles as an argument and returns an integer giving the length of the longest run of consecutive, increasing values. As one example, if the vector contains

```
3.4, 7.5, 1.1, 11.9, 17.3, 27.1, 16.9, 15.4, 29.1
```

Then the length of the longest run of consecutive increasing values is 4 — the values 1.1 through 27.1. As a second example, if the values are all decreasing (or equal), then the length of the longest run is 1. Start by providing the function prototype.

20. Here is a short declaration of one version of the string class. Its structure is very similar to the `Vec<T>` class we worked on in Lecture 7 and in Lab 4:

```
class cs2_string {
public:
    cs2_string( );
    cs2_string( unsigned int n, char c ); // problem (a)
    const char& operator[] ( unsigned int i ) const { return m_chars[i]; }
    char& operator[] ( unsigned int i ) { return m_chars[i]; }
    unsigned int size() const { return m_size; }
    void push_back( char c );
    void append( const cs2_string & other );
    cs2_string substr( unsigned int i, unsigned int len ) const;
private:
    char * m_chars; // pointer to dynamically allocated array of chars
    unsigned int m_size; // number of chars stored in the cs2_string
    unsigned int m_alloc; // number of locations allocated in the array (m_size <= m_alloc)
};
```

- (a) Write the implementation of the class constructor labeled (a) above. For example, after the line

```
cs2_string parta( 10, 'a' );
```

`parta` should be size 10, with each location from 0 to 9 containing the value 'a'.

- (b) Write the implementation of the function `append`. For example, after the code

```
cs2_string parta( 10, 'a' );
cs2_string partb( 12, 'b' );
parta.append( partb );
```

`parta` should be size 22, with the first 10 values being 'a' and the last 12 being 'b'. **You may not use `push_back` and you may not use `std::string`.**

- (c) Write the implementation of the `substr` member function. This function should create and return a new string that is a copy of the current string starting at location `i` and containing `len` characters. If `i+len` is longer than the current string then `substr` should return a new string that starts at `i` and contains all remaining chars. For example,

```
cs2_string s("it is snowing"); // length is 13
cs2_string s1 = s.substr(6,4);
cs2_string s2 = s.substr(6,500);
```

then `s1 == "snow"` and `s2 == "snowing"`.

21. Write code that (a) uses pointers to dynamically allocate an array of `n` doubles, (b) reads `n` doubles into the array from `cin`, and (c) then, in a separate loop, outputs the differences between consecutive doubles, all on one line. For example, if `n==5` and the input is

```
4.5 11.8 7.4 3.1 13.1
```

then the output should have the 4 values

```
7.3 -4.4 -4.3 10
```

In the loop for part (b) you may use subscripting. In the loop for part (c) no subscripting may be used and no integer counting variables may be employed; in this part everything must be done using pointers.