University of Calgary
Department of Electrical and Computer Engineering
ENCM 339: Programming Fundamentals
Lecture Instructors: Steve Norman, Mahmood Moussavi

Fall 2016 Final Examination
Monday, December 19 — 8:00 am to 11:00 am
Location: KN AUX (Auxiliary Gym, in Kinesiology A)

NAME (printed):

U of C ID NUMBER:

SIGNATURE:

SECTION:

(Section 01 was MWF, 1:00pm, with S. Norman; Section 02 was MWF, noon, with M. Moussavi.)

Please don't write anything within this box.

1	/ 10
2	/ 10
3	/ 25
4	/ 10
5	/ 14
6	/ 10
7	/ 16
8	/ 10
9	/ 10
TOTAL	/ 115

Instructions

- Please note that the official University of Calgary examination regulations are printed on page 1 of the *Examination Regulations and Reference Material* booklet that accompanies this examination paper. All of those regulations are in effect for this examination, except that you must write your answers on the question paper, not in the examination booklet.
- You may **not** use electronic calculators or computers during the examination.
- The examination is **closed-book**. You may not refer to books or notes during the examination, with one exception: you may refer to the *Examination Regulations and Reference Material* booklet that accompanies this examination paper.
- Some problems are relatively **easy** and some are relatively **difficult**. Go after the easy marks first.
- Write all answers on the question paper and hand in the question paper when you are done. Please do *not* hand in the *Examination Regulations and Reference Material* booklet.
- Please print or write your answers **legibly**. What cannot be read cannot be marked.
- If you write anything you do not want marked, put a large X through it and write "rough work" beside it.
- You may use the backs of pages for rough work.

SECTION 1 (*total of 10 marks*)**.** For all of Questions 1–10 within this section, you may assume that any needed library header files are included, followed by

```
using namespace std;
```

in the case that the given code is C++.

Indicate answers by drawing a circle around one of (a), (b), (c), etc.

1. Consider the following code fragment, and select the best answer.

```
vector <int> v1 = {11, 23, 99};
vector <int> v2 = {3};
v2 = v1;
for(int i = 0; i < v2.size(); i++)
    cout << v2.at(i) << " ";
```

The output will be:

(a) `11 23 99`
(b) `3 23 99`
(c) `3`
(d) There will be no output because `v2 = v1` is an improper assignment.
(e) None of the above.

2. Consider the following code fragment and select the best answer:

```
string s1 = "893";
s1 += "99";
cout << s1.at(2) - s1.at(4);
```

(a) This code gives a compilation error on the second line.
(b) This code prints `6`.
(c) This code prints `-6`.
(d) None of the above.

3. Which of the following statements is NOT true about a global integer variable declaration:

(a) If it does not have an initializer, it will be automatically initialized to zero.
(b) Its memory space will be allocated in the static area.
(c) Its value cannot be changed, because of static allocation.
(d) Its lifetime ends when its program stops running.

4. Which one of the following statements is NOT true?
A class constructor

(a) can be overloaded
(b) must have a `void` return type
(c) can have several arguments
(d) must have the name of its class

5. Which line in the following C++ (not C) code fragment will cause a compilation error?

```
char s1[ ] = "\0\0\0";
const char *s2 = s1;
const char *&s3 = s2;
*s1 = 'M';
```

(a) First line
(b) Second line
(c) Third line
(d) Fourth line
(e) None of the lines

6. Consider this code fragment:

```
vector <int> v1 = {11, 23, 99};
vector <int> v2 = {3};
v2 = v1;
for(int i = 0; i < v2.size(); i++)
    cout << v2.at(i) << " ";
```

The output will be:

(a) `444444`
(b) `333333`
(c) `33333333`
(d) `222222222222`
(e) None of the above

(Hint: The *Reference Material* has information about `std::vector` constructors.)

7. Consider the following program:

```
class Foo {
public:
    Foo(int x = 100, int y = 200) {a = x; b = y;}
    void set_b(int x) {b *= x;}
    int get_a() {return a;}
    int get_b() {return b;}
private:
    int a, b;
};

int main() {
    Foo f(300);
    f.set_b(5);
    cout << f.get_a() << ' ' << f.get_b();
}
```

What is the output?

(a) `0 0`
(b) `100 1000`
(c) `5 200`
(d) `100 5`
(e) `300 200`
(f) None of the above.

8. Consider the following code fragment:

```
const char *s[ ] = {"ON TIME", "LATE"};
const char** m = s;
```

Which one of the following C++ statement would output `N` on the screen?

(a) `cout << (*(*m+1)+1);`
(b) `cout << *(*(m+1));`
(c) `cout << **m + 1;`
(d) `cout << m[0][1];`
(e) None of the above

9. Consider the following code to answer the next question:

```
class Point {
public:
    Point(double x = 0, double y = 0 ) { this -> x = x; this -> y = y;}
    double getx() {return x;}
    double gety() {return y;}
private:
    double x, y;
};
```

Which one of the following declarations are correct to create an object of class `Point`:

(a) `Point a(23, 44);`
(b) `Point b(55);`
(c) `Point c;`
(d) All of the above declarations are correct.

10. Consider the following program.

```
void fun(int n);

void fun(int n) {

    if (n > 0) {
        cout << n;
        fun (n-1);
    }
    cout << n;
}
```

```
int main()
{
    fun (3);
    return 0;
}
```

The output is:

(a) `3210123`
(b) `3210`
(c) `210123`
(d) None of the above.

SECTION 2 (*total of 10 marks*)**.** For this section, assume the required header files are all included, followed by: `using namespace std;`

Question 1 (*3 marks*)**.** Consider the following partial definition of function `print_reverse` and complete the blank spaces. Assuming that the argument of the function, `s`, points to the start of a valid C-string , the function should print the string in reverse order. For example, if the argument `s` points to the `'A'` in `"ABCDE"`, the output should be `EDCBA`.

```
void print_reverse(const char* s)
{
    if(____________________)
        return;
    else{
        print_reverse(____________________);
    }
    cout << ____________________;
}
```

Question 2 (*4 marks*)**.** Assume that member functions of the `Vec` class are all properly defined. Consider the definition of `main` on the right and answer the questions given below the program listing.

```
class Vec {
public:
    Vec();
    Vec(int n);
    ~Vec();
    Vec(const Vec& src);
    Vec& operator= (const Vec& v);
private:
    int *storageM;
    int sizeM;
};
```

```
int main() {
    Vec v0;
    Vec *p = new Vec;
    {   // Beginning of inner scope
        Vec v1(1);
        Vec v2(2);
        Vec v3(3);
        Vec v4 = v1 = v2 = v3;
        v0 = v4;
    }   // End of inner scope
    delete p;
    return 0;
}
```

How many times are each of these members of class `Vec` called ...

- Default constructor? _____
- Constructor with parameter of type `int`? _____
- Copy constructor? _____
- Copy assigment operator? _____
- Destructor? _____

Question 3 (*3 marks*)**.** What is the output of the following `main` function? Write your answer in the space beside the program listing.

```
int main() {
    vector<string> v(3);
    for(int i = 0; i < 3; i++)
        for(int j = i; j >=0; j--)
            v.at(i).push_back('F' + i);
    for(int i = 0; i < 3; i++)
        cout << v.at(i) << endl;
    return 0;
}
```

Write your answer here ...

SECTION 3 (*total of 25 marks*).

Part a (*6 marks*). In the space below the function prototype and function interface comment, write a C++ function definition for `no_duplicates`. The return value should be `true` if the vector `v` contains no duplicate element values, and `false` otherwise.
So for the following vector, the return value would be `true` . . .

4	66	99	1	2

But for these vectors, the return value would be `false` . . .

4	66	99	4	2

0	0	99	4	2

4	4	0	4	0

```
bool no_duplicates (const std::vector<int>& v);
// PROMISES: Return value is true if v is empty or does not contain
//     duplicate values, is false otherwise.
```

Part b (*6 marks*). In the space below the function prototype and function interface comment, write a C function definition for `range`. This function should return the difference between the maximum and minimum values in an array of `int`s. For example, the return value for the following array should be $89 = 99 - 10$ . . .

40	60	99	10	20

But for the array below, the return value should be $79 = (-20) - (-99)$ . . .

-99	-40	-66	-20	-25

```
int range(const int *x, int n);
// REQUIRES: n >=1, and x[0] ... x[n-1] exist.
// PROMISES: Return value is the difference between the largest and
//    smallest values among x[0] ... x[n-1].
```

Part c (*8 marks*). In the space below the function prototype and function interface comment, write a C function definition for `append_all`. This function should allocate an appropriate array with `malloc`, then build a C-string within the array by concatenating strings found via an array of pointers. For example if `n` is 3 and `sp[0]`, `sp[1]`, and `sp[2]` point to `"ABC"`, `"WXYZ"`, and `"KL"`, the new string should match `"ABCWXYZKL"`.

If you wish, you may have your code call `<string.h>` library functions.

```
char* append_all(const char** sp, int n);
// REQUIRES: n >= 1.
//    Each of sp[0] ... s[n-1] points to the start of a C string.
// PROMISES: Return value points the beginning of an array
//    in the heap. That array contains a C string made from
//    concatenating sp[0], sp[1] ... s[n-1], in that order.
```

Part d (*5 marks*). Complete the partial definition of `circular_shift` by filling in the blank spaces. The function should shift each element of `v`, except the last element, by one index. In particular, the first element should be moved to the second element, the second element to the third element, and so on. The last element will then be moved to the first element. For example, the function should convert the array below on the left to the array below on the right:

40	60	99	10	20

20	40	60	99	10

```
void circular_shift(std::vector<int>& v)
{
    if (v.size() <= 1)
        return;
    int length = int(v.size());

    int last_value = ____________________;

    for (int i = ____________________; ____________________; i--)
    {
        v.at(i) = ____________________;
    }

    v.at(0) = ____________________;
}
```

SECTION 4 (*total of 10 marks*)**.** Consider the partial definition of the following class called `ExamString`; it is similar to a class used in a Lab 8 exercise. Answer the questions asked below the code listing.

```
class ExamString {
public:
    ExamString(): storageM(new char[1]), lengthM(0) { storageM[0] = '\0'; }

    ExamString(const char *s);
    // REQUIRES: s points to the first char of a C-string.
    // PROMISES:
    //   ExamString object is initialized to have the same sequence
    //   of chars as the C-string, including a '\0' terminator.
    //   length() == strlen(s).

    ExamString(const ExamString& src);
    ExamString& operator=(const ExamString& rhs);
    ~ExamString();

    int length() const {return lengthM;}
    char at(int i) const { return storageM[i]; }
    char& at(int i) { return storageM[i]; }
private:
    char *storageM;
    int lengthM;
};
```

Part a (*5 marks*)**.** In the following space write a definition for the `ExamString` constructor that has a parameter of type `const char *`. In this part, you are *not allowed* to call C library functions.

Part b(*5 marks*)**.** In the following space write a definition for the `ExamString` copy constructor. In this part, you are again *not allowed* to call C library functions.

SECTION 5 (*total of 14 marks*)

Part a (*8 marks*). In the space beside the program listing, make a memory diagram for ***the second time*** the program gets to `point one`.

```
class Vector {
public:
  Vector() : sizeM(0), capM(0), dataM(0) { }
  Vector(int count, int v);
  Vector(const Vector& src);
  ~Vector() { delete [ ] dataM; }
  Vector& operator =(const Vector& rhs);

  int size() const { return sizeM; }
  const int& at(int i) const { return dataM[i]; }
  int& at(int i) { return dataM[i]; }
  void push_back(int v);
  void resize(int new_size);
private:
  void grow(int new_cap);
  int sizeM;
  int capM;
  int *dataM;
};

Vector::Vector(int count, int v)
  : sizeM(count), capM(count),
    dataM((count > 0) ? new int[count] : 0)
{
  for (int i = 0; i < count; i++)
    dataM[i] = v;
}

void Vector::push_back(int v) {
  if (sizeM == capM)
    grow((capM == 0) ? 3 : (2 * capM));
  dataM[sizeM] = v;
  sizeM++;
}

void Vector::grow(int new_cap) {
  int *old = dataM;
  dataM = new int[new_cap];
  for (int i = 0; i < sizeM; i++)
    dataM[i] = old[i];
  capM = new_cap;

  // point one

  delete [ ] old;
}

int main() {
  Vector v(2, 42);
  v.at(1) = v.at(0) + 3;
  v.push_back(11);
  v.push_back(22);
  v.push_back(33);
  return 0;
}
```

Part b (*6 marks*). Write a definition for the `resize` member function of `Vector`. It should have the same effect for `Vector` as the `resize` function has for `std::vector<int>`.

SECTION 6 (*total of 10 marks*).

Part b (*4 marks*). In the space beside the code, make a memory diagram for `point one`.

```
char x[ ] = "ABCD";
char y[ ] = "EFGH";

void f(char **w)
{
  w[1][2] = *w[2];

  // point one
}

int main(void)
{
  char z[ ] = "PQR";
  char *a[3] = {x, y, z};
  f(a);
  return 0;
}
```

Part b (*6 marks*). In the space beside the code, make a memory diagram for `point one`.

```
int glob[4];

int main() {
  int *aa[3];
  int bb[ ] = { 10, 20, 30 };
  int **cc;
  aa[0] = glob;
  aa[1] = bb;
  aa[2] = new int[3];
  (*aa)[1] = 44;
  *aa[1] = 55;
  aa[2][1] = 66;
  cc = aa + 1;
  *cc += 2;
  **cc += 3;

  // point one

  return 0;
}
```

SECTION 7 (*total of 16 marks*) All three parts of this problem concern a simple C++ class for managing matrices of `double`s.

Part a (*8 marks*)**.** In the space beside the program listing, make memory diagrams for `point one` and `point two`.

```
class Matrix {
public:
  Matrix() : nrowM(0), ncolM(0), dataM(0)
  { }
  Matrix(int nr, int nc);
  Matrix(const Matrix& src);
  ~Matrix();
  Matrix& operator =(const Matrix& rhs);

  int nrow() const { return nrowM; }
  int ncol() const { return ncolM; }
  const double& at(int r, int c) const {
    return dataM[ncolM * r + c];
  }
  double& at(int r, int c) {
    return dataM[ncolM * r + c];
  }
private:
  int nrowM;
  int ncolM;
  double *dataM;
};

Matrix::Matrix(int nr, int nc) : nrowM(nr), ncolM(nc)
{
  dataM = new double[nr * nc];

  // point one

  for (int i = 0; i < nr * nc; i++)
    dataM[i] = 0.0;
}

int main() {
  Matrix m(4, 2);
  m.at(0, 1) = 0.25;
  m.at(3, 0) = -1.5;

  // point two

  return 0;
}
```

Part b (*2 marks*)**.** Write a definition for the destructor of `Matrix`.

Part c (*6 marks*)**.** Complete the definition of the copy assignment operator.

```
Matrix& Matrix::operator =(const Matrix& rhs) {
```

SECTION 8 (*total of 10 marks*).

Part a (*5 marks*). Recursion isn't an efficient way to copy a C string, but it can be used to get a correct result.

Make a memory diagram for the *second time* the program reaches `point one`. (Hint: There will be a total of four ARs on the stack at that point.)

```
void copy(char *d, const char *s)
{
  if (*s != '\0')
    copy(d + 1, s + 1);
  *d = *s;

  // point one

  return;
}

char x[ ] = "efghi";

int main(void)
{
  copy(x, "abcd");
  return 0;
}
```

Part b (*5 marks*). An *arithmetic sequence* of numbers is one in which the differences between consecutive numbers in the sequences are all the same. For example, $\{10, 20, 30, 40\}$ and $\{8, 6, 4\}$ are arithmetic sequences, but $\{10, 20, 30, 41\}$ is not.

Write a recursive, divide-and-conquer C++ function definition to match the given function interface.

```
bool is_arith_seq(const std::vector<int>& v, int lo, int hi);
// REQUIRES: lo <= hi
// PROMISES
//   If hi - lo <= 2, return value is true.
//   If hi - lo > 2, return value is true if v.at(lo) ... v.at(hi-1)
//   form an arithmetic sequence, and is false otherwise.
```

SECTION 9 (*total of 10 marks*)

Part a (*4 marks*). Assume that all the library function calls in the following program are successful. What output will appear in the terminal window when it runs?

```
#include <stdio.h>

int a[ ] = {10, 11, 12, 13, 14, 15};
int b[ ] = {200, 201, 202, 203, 204, 205};

int main(void)
{
  FILE *fp = fopen("x.dat", "wb");
  fwrite((void*) a, sizeof(int), 6, fp);
  fclose(fp);
  fp = fopen("x.dat", "rb");
  fread((void *) b, sizeof(int), 4, fp);
  fread((void *) (a + 1), sizeof(int), 2, fp);
  fclose(fp);
  for (int i = 0; i < 6; i++)
    printf("a[%d] is %d, b[%d] is %d\n", i, a[i], i, b[i]);
  return 0;
}
```

Part b (*6 marks*). Write a C++ program to write the integers 1 to 100 to a text file called `NineB.txt`. There should be five numbers per line (so the first line gets 1 to 5, the second line gets 6 to 10, and so on). Numbers on a line should be separated by spaces, but you don't need to get numbers to lined up in columns.

If the program fails to open the file, the program should print an appropriate error message and call `exit`.