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

Fall 2015 Final Examination
Saturday, December 12 — 8:00 am to 11:00 am
Location: KN AUX (Auxiliary Gym, in Kinesiology A)

NAME (printed):

U of C ID NUMBER:

SIGNATURE:

Please don't write anything within this box.

1	/ 10
2	/ 8
3	/ 12
4	/ 12
5	/ 10
6	/ 10
7	/ 13
8	/ 12
9	/ 9
10	/ 8
TOTAL	/ 104

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++.

1. Consider this code fragment:
```
char a[ ] = {'K', 'P', '\0'};
char *b[3] = {a, &a[2], 0};
int diff = *b - b[1];
```
What would be the value of `diff`, when this code segment is executed?

   (a) There is no way to know.
   (b) 2
   (c) -2
   (d) 3
   (e) None of the above.

2. Consider this code fragment:
```
/* 1 */  char s [ ] = "Red";
/* 2 */  const char* cs = &s[1];
/* 3 */  cs = cs + 1;
/* 4 */  cs[-1] = 'U';
```
What does the code fragment produce?

   (a) Compilation error on line 3.
   (b) Compilation error on line 4.
   (c) Runtime error on line 3.
   (d) Runtime error on line 4.
   (e) None of the above.

3. What is the rule of the Big Three that should often be applied when designing C++ classes?

   (a) If a C++ class needs a destructor, it should also have a copy assignment operator and a copy constructor.
   (b) If a C++ class needs a copy constructor, it should also have a destructor and copy assignment operator.
   (c) If a C++ class needs a copy assignment operator, it should also have a destructor and copy constructor.
   (d) All of the above combined.

4. Consider this code fragment:
```
const char *s[ ] = {"ENCM339", "CPSC 300"};
const char** m = s;
```
Which one of the following statements displays the character `'N'` on the screen?

   (a) `cout << *((*m+1)+1) << endl;`
   (b) `cout << *(*m+1) << endl;`
   (c) `cout << **m << endl;`
   (d) All of the above.
   (e) None of the above.

5. A global variable is declared in a C or C++ program as

```
int i;
```

It will be allocated in the ...

   (a) static memory area; its initial value will be zero, and its value cannot be changed.
   (e) static memory area; its initial value will be unknown, and its value can be changed.
   (c) static memory area; its initial value will be zero, and its value can be changed.
   (d) heap memory area; its initial value will be unknown, and its value cannot be changed.

6. What will be the output of the program listed to the right?

   (a) `1112`
   (b) `1212`
   (c) `1122`
   (d) None of the above.

```
void fun(int n) {
  if (n <= 0) return;
  cout << "1";
  fun (n-2);
  cout << "2";
}

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

7. What will be the output from the following code fragment?

```
char s1[6] = "Apple";
char s2[4] = "Red";
char s3[12] = "*";
printf("\n%s\n", strcat(strcat(strcat(s3, s1), "-"), s2));
```

   (a) `*Red-Apple`
   (b) `*Apple-Red`
   (c) `Red-Apple*`
   (d) `*Apple-Red`
   (e) None of the above.

8. What will be the output of the program listed to the right?

   (a) `810`
   (b) `89`
   (c) `1810`
   (d) `189`
   (e) None of the above.

```
#define doubled(NUMBER) (NUMBER * 2)
int main ()
{
  int x, y = 10;
  x = doubled(y-1);
  cout << x << y;
  return 0;
}
```

Consider the following class definition, then answer questions 9 and 10.

```
class Point {
public:
  Point(double x = -99, double y = -99) {
    this -> xM = x;
    this -> yM = y;
  }
  double getx() {return xM;}
  double gety() {return yM;}
  void setx(double x) {this -> xM = x;}
  void sety(double y) {this -> yM = y;}
private:
  double xM, yM;
};
```

9. What is the output of the following code fragment?

```
Point p1(100);
cout << p1.getx() << "  " << p1.gety();
```

   (a) `-99  -99`
   (b) `100  100`
   (c) `-99  100`
   (d) `100  -99`
   (e) None of the above. The code won't compile because the constructor of `Point` needs two arguments.

10. How many times does the constructor of class `Point` get called by the following code fragment?

```
Point a(100, 200);
Point b[6];
Point *c = new Point;
Point *d = new Point(300, 400);
```

   (a) Four times
   (b) Three times
   (c) Nine times
   (d) Two times
   (e) None of the above.

SECTION 2 (*total of 8 marks*)**.** For the questions in this section you should assume that the size of a pointer is 8 bytes, and that any necessary header file is included.

Questions 1–3 all refer to the following C code fragment. Documentation for the `fwrite` function can be found in the *Reference Material* that accompanies this exam.

```
const char* array[3] = {"12345.6789", "0.66666666666", "999999999999"};
FILE* fp = fopen("output.bin", "wb");
size_t nwrite = fwrite(array[1], sizeof(array[1]), 1, fp);
fclose(fp);
```

Question 1 (*1 mark*)**.** What would be the size of the file `output.bin` after the code fragment runs?

Question 2 (*1 mark*)**.** What would be the value of `nwrite` after the code fragment runs?

Question 3 (*1 mark*)**.** What characters would be written into `output.bin`?

Question 4 (*3 marks*)**.** Consider the following C program:

```
char * copystr (const char* source) {
  char *dest;
  strcpy(dest, source);
  return dest;
}
int main(void) {
  const char *s1 = "ABCD";
  char *s2 = copystr(s1);
  // more code, as needed ...
  return 0;
}
```

The function `copystr` is defective; calling it will most likely result in a runtime error.

`copystr` is supposed to make a copy of the found via the `source` string and return a pointer to the beginning of the copy. Write a code fragment that can be inserted into `copystr` to fix the defect. You may call whatever C library functions you find useful.

Question 5 (*2 marks*)**.** Consider the following C code fragment.

```
int *d[3];
for(int j = 0; j < 3; j++) {
  d[j] = malloc(sizeof (int));
  **(d+j) = sizeof(d[j]) + j;
  printf("%d\n", *d[j]);
}
```

Assuming that all the calls to `malloc` succeed, what is the output?

SECTION 3 (*total of 12 marks*)**.**

Part 1 (*7 marks*)**.** Write a definition for the following C++ function:

```
bool up_then_down(const int* arr, int n);
// REQUIRES: n >= 1; elements arr[0] ... a[n-1] exist.
// PROMISES:
//   Returns true if the sequence of element values is strictly
//   increasing from a[0] to the first appearance of the maximum
//   value, then strictly decreasing from the first appearance of the
//   maximum value to a[n - 1].
//   Otherwise, returns false.
// EXAMPLES:
//   The return value would be true for all of these sequences ...
//      {10}, {10, 20}, {20, 10}, {10, 20, 30, 25}
//   ... but would be false for all of these ...
//      {10, 20, 10, 15}, {10, 10, 20, 15}, {10, 20, 20, 15}.
```

Part 2 (*5 marks*)**.** Write a definition for the following C++ function. In this part, you may *not* make any calls to library functions.

```
bool all_diff(const char *left, const char *right);
// REQUIRES: left and right each point to the beginnings of C strings.
// PROMISES:
//   Return value is true if none of the characters in the left string
//   also appear in the right string.  ('\0' characters are not
//   included in the comparison.)
//   If there is at least one match, return value is false.
```

SECTION 4 (*total of 12 marks*)**.** Below is a complete `.h` file for a class that is similar to a class used in Lab 8, along with a `.cpp` file that has some but not all of the member function definitions needed for the class.

```
// File: intVector.h
#include <cstddef>  // for size_t
using std::size_t;

class IntVector {
public:
  IntVector() : storeM(0), end_validM(0), end_storeM(0) { }
  IntVector(const int *begin, const int *end);
  // REQUIRES: begin points to an array element, such that
  //   *begin, *(begin+1), ..., *(end-1) are all elements of the
  //   same array.
  // PROMISES:
  //   size() = end - begin.
  //   Vector elements 0 ... size()-1 are initialized using values of
  //   *begin, *(begin+1), ..., *(end-1)

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

  size_t size() const { return end_validM - storeM; }
  size_t capacity() const { return end_storeM - storeM; }
  const int& IntVector::at(size_t i) const { return storeM[i]; }
  int& IntVector::at(size_t i) { return storeM[i]; }

  void push_back(int el_val);
  // PROMISES: Size of vector is increased by one element.
  // Last element of vector is equal to el_val.
  void remove_all(int val);
  // PROMISES: If one or more elements match val, all of those
  // elements are removed from the vector, remaining elements are
  // moved to new locations as needed, size() is reduced, but
  // capacity() remains unchanged. If no elements match val, there is
  // no change to the vector.
private:
  int *storeM;
  int *end_validM;
  int *end_storeM;
};
```

```
// File: intVector.cpp
#include "intVector.h"

IntVector::IntVector(const int *begin, const int *end)
  : storeM(0), end_validM(0), end_storeM(0)
{
  if (begin == end)
    return;
  size_t count = end - begin;
  storeM = new int[count];
  end_storeM = end_validM = storeM + count;
  for (size_t i = 0; i < count; i++)
    storeM[i] = begin[i];
}

void IntVector::push_back(int el_val) {
  if (end_validM == end_storeM) {
    size_t old_size = size();
    size_t new_cap = (capacity() == 0) ? 2 : 2 * capacity();
    int *new_store = new int[new_cap];
    for (size_t i = 0; i < size(); i++)
      new_store[i] = storeM[i];
    delete [ ] storeM;
    storeM = new_store;
    end_validM = new_store + old_size;
    end_storeM = new_store + new_cap;
  }
  *end_validM = el_val;
  end_validM++;
}
```

Please answer the questions on the next page ...

All function definitions below must be consistent with the code and comments in the files given on the previous page.

Question 1 (*4 marks*)**.** Write a definition for the copy constructor function of `IntVector`.

Question 2 (*2 marks*)**.** Write a definition for the destructor function of `IntVector`.

Question 3 (*6 marks*)**.** Write a definition for the `remove_all` function of `IntVector`.

SECTION 5 (*total of 10 marks*)**.** Below is a definition of class `ExamList`; it's a linked list type similar to one used in two exercises in Lab 9.

```
typedef int ListItem;

class ExamList {
private:
  struct Node {
    ListItem itemM;
    Node *nextM;
  };
  Node *headM;
  void destroy();  // helper function
  // PROMISES: All nodes belonging to the list have been deleted.

  void copy(const ExamList& source);  // helper function
  // PROMISES:
  //   If source is an empty list, headM == 0.
  //   Otherwise 'this' list has the same number of nodes as
  //   the source list, with the same items in the same order.

  void recursive_print(const Node* p) const;  // helper function

public:
  ExamList(); // PROMISES: creates an empty list.

  ExamList(const ListItem* arr, int n);
  // REQUIRES:
  //    n > 0. arr[0] ... a[n - 1] exist.
  // PROMISES:
  //    List with n nodes is created; first node item will be equal
  //    to arr[0], second will be equal to arr[1], and so on ...

  ExamList(const ExamList& source);
  ExamList& operator =(const ExamList& rhs);
  ~ExamList();

  void print() const { recursive_print(headM); }
  // PROMISES:  If list is not empty, items in the list
  //   are printed on cout separated by 2 spaces, starting with the
  //   head node and finishing with the last node.
  //   If list is empty, nothing is printed on cout.
};
```

Question 1 (*3 marks*)**.** Write a definition for the copy assignment operator. It should make calls to some of the helper functions of the class.

This section continues on the next page ...

. . . section continued from previous next page . . .

Question 2 (*5 marks*)**.** Write a definition for the helper function `copy`.

Question 3 (*2 marks*)**.** Write a definition for the helper function `recursive_print` that is compatible with the specification for the function `print`. `recursive_print` must be recursive—no marks will be given for non-recursive implementations.

SECTION 6 *(10 marks)*. In the code below, the class `ExamString` is missing many features that would needed by a practical string type. However, the given features are enough to allow the `main` function to run correctly.

Make a memory diagram for the first (and only) time the program gets to `point one`.

```
#include <iostream>
using std::size_t;

class ExamString {
public:
  ExamString(const char *s = "");
  size_t length() const { return lenM; }
  const char& at(size_t i) const {
    return dataM[i];
  }
  char& at(size_t i) {
    return dataM[i];
  }
  void push_back(char c);
private:
  void grow(size_t new_cap);
  char *dataM;
  size_t lenM;
  size_t capM;
};

ExamString::ExamString(const char *s)
{
  size_t slen = 0;
  while (s[slen] != '\0')
    slen++;
  capM = (slen < 4) ? 4 : slen + 1;
  dataM = new char[capM];
  for (size_t i = 0; i <= slen; i++)
    dataM[i] = s[i];
  lenM = slen;
}

void ExamString::grow(size_t new_cap)
{
  char *new_data = new char[new_cap];
  for (size_t i = 0; i < lenM; i++)
    new_data[i] = dataM[i];

  // point one

  delete [ ] dataM;
  capM = new_cap;
  dataM = new_data;
}

void ExamString::push_back(char c)
{
  if (lenM + 1 == capM)
    grow(2 * capM);
  dataM[lenM] = c;
  lenM++;
  dataM[lenM] = '\0';
}

int main()
{
  ExamString s1;
  s1.push_back('A');
  s1.push_back('B');
  s1.push_back('C');
  s1.push_back('D');
  s1.push_back('E');
  s1.at(4)++;
  for (int i = 0; i < s1.length(); i++)
    std::cout << s1.at(i);
  std::cout << std::endl;
  return 0;
}
```

SECTION 7 (*total of 13 marks*).

Part a (*5 marks*). Draw a memory diagram for `point one`.

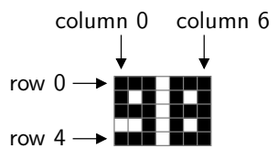
```
int foo[ ] = {1, 4, 9};
int bar[ ] = {16, 25, 36, 49};

int main(void) {
  int *x[2];
  x[0] = bar;
  x[1] = foo;
  *x[1] = (*x)[1];

  // point one

  return 0;
}
```

Part b (*8 marks*). Draw a memory diagram for `point one`.

```
const char *most_a(const char **x, int n) {
  int i, j, c, max = 0;
  const char *r = x[0];
  for (i = 0; i < n; i++) {
    c = 0;
    for (j = 0; x[i][j] != '\0'; j++)
      if (x[i][j] == 'a')
        c++;
    if (c > max) {
      max = c;
      r = x[i];
    }
  }

  // point one

  return r;
}

int main(void) {
  const char *y[] = {
    "aardvark",
    "banana",
    "quux"
  };
  const char *p;
  p = most_a(y, 3);
  return 0;
}
```

SECTION 8 (*total of 12 marks*)

Computer systems have many different ways to represent images. A very simple (but not very space-efficient) approach is to represent the image as a 2-dimensional array of *pixels*. To the right is a tiny image with some white pixels and some black pixels ...



For each pixel, three numbers between 0 and 255 are used to indicate its colour and brightness. In C++ these types are useful:

```
typedef unsigned char uchar;
struct Pixel { uchar r; uchar g; uchar b; };
```

`r`, `g` and `b` stand for red, green, and blue. On most modern computers, the possible values for `unsigned char` range from 0 to 255, so it makes sense to use that type for `r`, `g` and `b`.

This section continues on the next page ...

. . . section continued from previous next page . . .

Here are some example uses of the `Pixel` type:

```
const Pixel BLACK = {0, 0, 0};
const Pixel BRIGHT_RED = {255, 0, 0};
const Pixel BRIGHT_GREEN = {0, 255, 0};
```

And here is some C++ code that uses the `Pixel` type to create an `Image` type:

```
const Pixel MID_GRAY = {128, 128, 128};
const size_t DEFAULT_HEIGHT = 100;
const size_t DEFAULT_WIDTH = 200;
class Image {
public:
  Image(size_t nr = DEFAULT_HEIGHT, size_t nc = DEFAULT_WIDTH);
  Image(const Image& src);
  ~Image();
  Image& operator=(const Image& rhs);

  size_t nrow() const { return nrowM; }
  size_t ncol() const { return ncolM; }

  const Pixel& get_pixel(size_t r, size_t c) const {
    return storeM[ncolM * r + c];
  }
  void set_pixel(size_t r, size_t c, const Pixel& p) {
    storeM[ncolM * r + c] = p;
  }
private:
  size_t nrowM;
  size_t ncolM;
  Pixel *storeM;
};

Image::Image(size_t nr, size_t nc)
  : nrowM(nr), ncolM(nc), storeM(new Pixel[nr * nc])
{
  for (size_t i = 0; i < nrowM * ncolM; i++)
    storeM[i] = MID_GRAY;
}
```

Part a (*6 marks*)**.** For the purposes of this problem, let's define the *brightness* of a pixel to be the sum of its red, green, and blue components, divided by 765.0 to produce a value between 0.0 and 1.0. (Note that $255 + 255 + 255 = 765$.) Write a C++ function definition to match the given prototype.

```
double max_brightness(const Image& im);
// REQUIRES: im.nrow() > 0 && im.ncol() > 0.
// PROMISES: Return value is the maximum brightness among all the pixels of im.
```

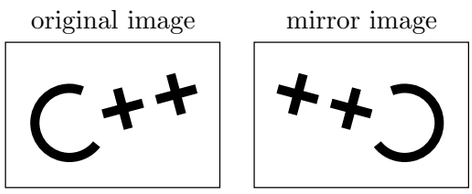
This section continues on the next page . . .

... section continued from previous next page ...

Part b (*6 marks*)**.** For convenience, here is a repeat of the class definition from the prevous page:

```
class Image {
public:
  Image(size_t nr = DEFAULT_HEIGHT, size_t nc = DEFAULT_WIDTH);
  Image(const Image& src);
  ~Image();
  Image& operator=(const Image& rhs);

  size_t nrow() const { return nrowM; }
  size_t ncol() const { return ncolM; }

  const Pixel& get_pixel(size_t r, size_t c) const {
    return storeM[ncolM * r + c];
  }
  void set_pixel(size_t r, size_t c, const Pixel& p) {
    storeM[ncolM * r + c] = p;
  }
private:
  size_t nrowM;
  size_t ncolM;
  Pixel *storeM;
};
```

Consider the problem of generating an image B that is the "mirror image" of an original image A, as shown below.



Write a C++ function definition to match the given prototype.

```
Image mirror(const Image& im);
// REQUIRES: im.nrow() > 0 && im.ncol() > 0.
// PROMISES: Return value is an Image object that is the mirror image of im.
```

SECTION 9 (*total of 9 marks*)**.**

Part a (*4 marks*)**.** A recursive function is not the most efficient way to find the length of a C-style string, but it works unless the string is incredibly long.

Draw a memory diagram for the first time the given program reaches `point one`.

```
int len(const char *s)
{
  int r = 0;
  if (*s != '\0')
    r = 1 + len(s + 1);

  // point one

  return r;
}

int main(void)
{
  int k;
  k = len("YZ");
  return 0;
}
```

b **Part b** (*5 marks*)**.** Consider this function interface:

```
int first_match(const int *a, int lo, int hi, int key);
// REQUIRES:
//   lo >= 0, hi > lo, and elements a[lo] ... a[hi-1] exist.
// PROMISES:
//   If none of a[lo] ... a[hi-1] are equal to key, return value is -1.
//   If at least one of a[lo] ... a[hi-1] is equal to key, return value
//   is the smallest i from lo to hi-1 such that a[i] == key.
// EXAMPLES:
//   Suppose we have int x[ ] = { 10, 20, 10, 21, 10};
//   Then first_match(x, 0, 5, 15) == -1, first_match(x, 0, 5, 10) == 0,
//   and first_match(x, 0, 5, 21) == 3.
```

There is a super-easy non-recursive implementation of `first_match`, but this question is about recursion. Write a recursive, divide-and-conquer function definition for `first_match`.

SECTION 10 *(8 marks)*. This problem concerns a hypothetical binary file format for images called `imgX`. Here is an incomplete specification for the file format:

The first four bytes of the file must be the ASCII codes for `'i'`, `'m'`, `'g'`, and `'X'`, in that order.
The next two bytes are the number of rows in the image, encoded as a 16-bit unsigned integer.
The next two bytes are the number of columns in the image, encoded as a 16-bit unsigned integer—for this problem, you can assume that the corresponding C++ type is `uint16_t`.
The remaining bytes are image data—the details don't matter for this problem.

Consider the design of a C++ program to read only the first 8 bytes of an `imgX` file. If the first 8 bytes of the file make sense, the terminal output should report the number of rows and columns, as in the following, in which the numbers 1000 and 2000 are just examples ...

```
File seems to be OK, with 1000 rows and 2000 columms.
```

If the first four bytes of the file can't be read or are incorrect, or if there is an error trying to read the number of rows or number of columns, an appropriate error message should be displayed.

Complete the following C++ program:

```
#include <iostream>
#include <fstream>
#include <cstdlib>  // for exit
#include <cstdint>  // for uint16_t
using namespace std;
int main(int argc, char **argv) {
  if (argc != 2) {
    cerr << "Error: need exactly one command-line argument.\n";
    exit(1);
  }
  ifstream ifs(argv[1], ios::binary);
  if (ifs.fail()) {
    cerr << "Error: could not open " << argv[1] << " for input.\n";
    exit(1);
  }
  char first4[4];
  uint16_t n_row, n_col;
  ifs.read(first4, 4);
```