SCHULICH School of Engineering

UNIVERSITY OF CALGARY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

**ENCM 339: Programming Fundamentals**
Instructors: Mahmood Moussavi and Steve Norman

**Fall 2015 MIDTERM TEST**
Thursday, October 22 — 7:00pm to 9:00pm

**Name (printed):**

**Signature:**

**First letter of last name (which helps us sort papers):**

**Please also write your name and U of C ID number in the spaces provided at the bottom of the last page.**

## General Instructions

- Attempt all questions.
- Write all answers on the question paper and hand in the question paper when you are done.
- To minimize distraction for your fellow students, you may not leave the exam room during the last ten minutes of the exam.
- The test is **closed-book**. You may not refer to books or notes during the test.
- No electronic calculators or computers may be used during the test.
- Some problems are relatively **easy** and some are relatively **difficult**. Go after the easy marks first.
- 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: Multiple choice** (*total of 4 marks: 1 per part*). To answer the questions in this section you may make the following assumptions:

- Library header files such as `<stdio.h>` have been included before any code that calls library functions such as `printf` and `scanf`.
- The size of the `int` data type is 4 bytes, and the sizes of the `double` data type and of all pointer types are 8 bytes.

In each question, **circle** your choice of a, b, c, d or e.

1. What is the output from the following code fragment, if the line of text entered by the user is `78 45 xyz`? (Reminder: When processing `%s`, `scanf` consumes characters up to but not including a space or a newline.)

```
int a = 10, b = 11, c = 12;
char d[ ] = "abcdefg";
printf("Enter a line of text:\n");
a = scanf("%s%d%d", d, &b, &c);
printf("%d %d %d %s", a, b, c, d);
```

   **a.** `0 abcdefg 10 11`
   **b.** `3 xyz 78 45`
   **c.** `2 abcdefg 78 45`
   **d.** `2 45 12 78`
   **e.** `1 abcdefg 45 11`

2. What is the output from the following code fragment?

```
char *a = "Banff";
char b[ ] = "Edmonton";
double c[ ] = {0, 3.141592653589793, 6.283185307179586};
int d[ ] = {sizeof(a), sizeof(b), sizeof(b[2]), sizeof(c)};
int i;
for (i = 0; i < 4; i++)
  printf("%d ", d[i]);
```

   **a.** `8 8 1 24`
   **b.** `8 9 1 24`
   **c.** `6 9 2 24`
   **d.** `8 9 1 3`
   **e.** `6 9 1 3`

3. Which line(s) in the following code fragment will cause a compilation error?

```
int a[4] = { 11, 22, 33 }; // line (1)
const int *b = a;          // line (2)
*b += 1;                   // line (3)
b += 2;                    // line (4)
```

   **a.** Line (1).
   **b.** Line (2).
   **c.** Line (3).
   **d.** Line (4).
   **e.** Two or more lines will cause compilation errors.

4. What is the output from the following code fragment?

```
char s[ ] = "university";
char *p = s;
int i;
for (i = 0; i < 4; i++) {
  (*p)++;
  fputc(*p, stdout);
  p++;
}
```

   **a.** `univ`
   **b.** `nvri`
   **c.** `uies`
   **d.** `nive`
   **e.** `vojw`

**SECTION 2: Short answer questions** (*total of 10 marks*).

**Part a.** *(1 marks.)* What is the output from the following code fragment?

```
int a[5] = { 2, 1, 0, 1, -2 }, i;
for (i = 0; i < 5; i++) {
  if (a[i])
    printf("Y");
  else
    printf("N");
}
```

**Answer:**

**Part b.** *(2 marks.)* An executable file is made from the two rather bizarre files below. What is the output of the program?

`main.c`

```
#include <stdio.h>
int main(void)
{
#include "stuff.h"
#include "stuff.h"
  return 0;
}
```

`stuff.h`

```
printf("hello!\n");
#ifndef GOODBYE
#define GOODBYE "bye!"
printf("GOODBYE %s\n", GOODBYE);
#endif
```

**Answer:**

**Part c.** *(2 marks.)*
What is the output of the program if the size of an `int` is 4 bytes and the size of a pointer is 8 bytes?

What is the output of the program if the sizes of `int`s and pointers are both 4 bytes?

```
#include <stdio.h>
void func(int y[5]);
int main(void) {
  int x[5] = {10, 8, 6, 4, 2};
  printf("main says %zu\n",
         sizeof(x)/sizeof(x[0]));
  func(x);
  return 0;
}
void func(int y[5]) {
  printf("func says %zu\n",
         sizeof(y)/sizeof(y[0]));
}
```

**Part d.** *(2 marks.)* What is the output from the following program? For full credit, you must show how you got your answer.

```
#include <stdio.h>
#define MAC1(x) x * x
#define MAC2(y, z) MAC1(y) - MAC1(z)
int main(void) {
  printf("%d\n", MAC2(3 + 1, 2 + 3));
  return 0;
}
```

**Answer:**

**Part e.** *(3 marks.)* What is the output from the following code fragment? For full credit, you must show how you got your answer.

```
  char x[10] = {
    'A', 'B', 'C', '\0', '\0',
    '\0', '\0', '\0', '\0', '\0'
  };
  strcat(x, "UV");
  x[8] = 'W';
  printf("%zu %zu %zu %zu\n",
         strlen(x), strlen(x+2), strlen(x+5), strlen(x+8));
```

**Answer:**

**SECTION 3: Functions to process strings and arrays** (*total of 25 marks*).

**Part a.** *(10 marks.)* Write two function definitions to implement these two interfaces. In this part, you are *not allowed* to write calls to library functions such as `strlen`.

```
// These functions are similar but not exactly the same. The call
// find_1st("Canadian", 'a') would return the address of the 'a' that
// is next to the 'C', but find_last("Canadian", 'a') would return the
// address of the 'a' that is beside the final 'n'.

const char *find_1st(const char *s, int c);
// REQUIRES: s points to start of a C string, c != '\0'.
// PROMISES: If c matches a character in the string, return
//   value points to the the location of the first match.
//   Otherwise, return value is NULL.

const char *find_last(const char *s, int c);
// Same as find_1st, except when a match is found, return value
// points to the location of the last match.
```

**Part b.** *(6 marks.)* Let's define a "square sequence" of numbers as follows: each number except the first is the square of the previous number. So $2, 4, 16, 256$ qualifies, but $2, 4, 15, 225$ does not. Given that, write a function definition for `is_square_seq`.

```
int is_square_seq(const int *a, int n);
// REQUIRES: n > 0 and elements a[0] ... a[n-1] exist.
// PROMISES:
//   Return value is 1 if n == 1.
//   Return value is 1 if for each i > 0 and i < n, a[i] is the square
//   of a[i-1].  Otherwise, return value is 0.
```

**Part c.** *(9 marks.)* Write a function definition to match the function interface.

```
int same_letters(const char *s, const char *t);
// REQUIRES: s and t point to beginnings of strings.
// PROMISES:
//   Return value is 1 if the two strings contain the same sequence of
//   letters in the same order; characters that are not letters are
//   ignored. Otherwise, return value is 0.
// EXAMPLES:
//   same_letters("ENEL", ".E..N...E....L...") == 1
//   same_letters("A B   C D", "  ABC  ") == 0, because there is no
//     match for 'D' in the second string.
//   same letters("__A_B_c", ",,A,B,C,,") == 0, because 'c' is not the
//     same letter as 'C'.
```

Hint #1: The library function

```
int isalpha(int c);
```

returns a non-zero value if `c` is the character code for a letter and a zero value otherwise.

Hint #2: Here's some **imprecise** pseudocode for an algorithm:

```
while (1) {
    loop through first string to find a letter or '\0'
    loop through second string to find a letter or '\0'
    if (characters do not match)
        break
    if (characters are both '\0')
        break
}
```

**SECTION 4: Structure types** (*total of 12 marks*).

**Part a.** *(6 marks.)* In the space to the right of the program listing, make a memory diagram for point one.

```
struct smaller {
  char s[3];
  int i;
};
struct larger {
  struct smaller sm;
  char t[4];
};

void foo(struct larger *p)
{
  int *q;
  q = &(p->sm.i);
  (*q)++;

  // point one

  return;
}

int main(void)
{
  struct larger x = {
    { "AB", 42 }, "CAN"
  };
  foo(&x);
  return 0;
}
```

**Part b.** *(6 marks.)* Assume that the type `struct larger` is as defined in **part a**. Write a function definition according to the following function interface. You may *not* call library functions such as `strlen` or `strcmp`.

```
int match_t(const char *s, const struct larger *y);
// REQUIRES:
//   s points to a C string and y points to an object whose member t
//   contains a C string.
// PROMISES:
//   Return value is 1 if the strings found through s and y->t match
//   exactly (same length, and same sequence of characters).  Otherwise
//   return value is 0.
```

**SECTION 5: Arrays, strings, and pointer arithmetic** (*12 marks*). In the space to the right of the program listing, make memory diagrams for point one and the *second time* the program gets to point two. Be clear about which items are on the stack and which are in static storage.

```
int f1(const char *s, const char *t);
int f2(const char *u);
int f3(const int *a, int n);

int main(void)
{
  char *a = "AB";
  char b[ ] = "LAB";
  int c[ ] = { 10, 20, 30, 5, 40 };
  int d, e;
  d = f3(c, 5);
  e = f1(a, b);
  return 0;
}

int f1(const char *s, const char *t)
{
  int xs, xt, i, j;
  xs = f2(s);
  xt = f2(t);
  if (xs > xt)
    return 0;
  for (i = 0, j = xt - xs; i < xs; i++, j++)
    if (s[i] != t[j])
      return 0;
  return 1;
}

int f2(const char *u)
{
  const char *v = u;
  while (*v != '\0')
    v++;

  // point two (after loop has finished)

  return v - u;
}

int f3(const int *a, int n)
{
  const int *p = a;
  while (n > 0 && *p >= *a) {
    p++;
    n--;
  }

  // point one

  return p - a;
}
```

| NAME (PRINTED) | U OF C ID # |
|---|---|
| | |

| SECTION 1 | SECTION 2 | SECTION 3 | SECTION 4 | SECTION 5 | TOTAL |
|---|---|---|---|---|---|
| / 4 | / 10 | / 25 | / 12 | / 12 | / 63 |