UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

APS 105 — Computer Fundamentals
Midterm Examination
February 25, 2020
1:10 p.m. – 2:50 p.m.
(100 minutes)

Examiners: P. Anderson, B. Li, B. Korst, M. Rezanejad

This is a "closed book" examination; no aids are permitted. No calculators or other electronic devices are allowed.

All questions are to be answered on the examination paper. If the space provided for a question is insufficient, you may use the last page to complete your answer. If you use the last page, please direct the marker to that page and indicate clearly on that page which question(s) you are answering there.

You must use the C language to answer programming questions. You are not required write `#include` directives in your solutions. The code provided to you may not have `#include` directives either. You may use functions from the `math` library as necessary.

Note that the summary of the total marks for the questions and the marks given for the questions is on the last page.

The examination has 12 pages, including this one.

First Name: _____ Last Name: _____

Your Student Number: _____

Your Lab Section: (ID or day/time) _____

Question 1 [4 Marks]

Write a single C statement using no curly brackets that rounds a double-type variable named `value` to its nearest hundredths place, and assign the result to a new double-type variable named `approximateValue`. For example, rounding `0.843` to the nearest hundredth would give `0.84`. You can use any of the C math library functions. Write your solution in the box below.

```
double approximateValue = rint(value * 100) / 100;
```

Question 2 [4 Marks]

What is the output of the following program?

```
#include<stdio.h>

int main(void) {
    int n;
    for (n = 9; n != 0; n -= 2)
        printf("%d\n", n);
    return 0;
}
```

It enters an infinite loop. Output is 9 7 5 3 1 -1 -3 etc.

Question 3 [4 Marks]

If `useCount` is declared as a variable of the `int` type, and `x` and `y` are both declared as variables of the `int` pointer type, which of the following statements will cause **compile-time** warnings or **compile-time** errors?

Statements	Answer
`x = *useCount;`	illegal (error)
`int *p = x;`	legal
`int *q = &y;`	illegal (warning)
`x = y = &useCount;`	legal

Question 4 [4 Marks]

Consider each of the sets of code below embedded in the following

```
int i = 3, j = 7;
// each set of embedded code below goes here
printf("%d\n", i);
```

In each case, what is printed by the printf statement? Note that each of these pieces of code compiles correctly.

code	printf output
`i++;`	4
`--i;`	2
`i = j / 10;`	0
`i = (int) (9.72);`	9
`if (i = 6) j = 17;`	6
`for (int i = 1; i < 6; i++) j = j + 3;`	3

Question 5 [4 Marks]

Write the output of the following program.

Program	*Program Output*

```
#include <stdio.h>

void skipSpace(int n) {
    for (int i = 1; i <= n; i++)
        printf(" ");
}

void printLeft(int n) {
    for (int i = 1; i <= n; i++)
        printf("/");
}

void printRight(int n) {
    for (int i = 1; i <= n; i++)
        printf("\\");
}

int main(void) {
    const int TSize = 5;
    int i;

    skipSpace(TSize);
    printf("*\n");

    for (i = 0; i < TSize - 2; i++) {
        skipSpace(TSize - 1 - i);
        printLeft(i + 1);
        printf("|");
        printRight(i + 1);
        printf("\n");
    }

    skipSpace(TSize);
    printf("|");
    return 0;
}
```

```
    *
   /|\
  //|\\
 ///|\\\
    |
```

Question 6 [8 Marks]

The value of the mathematical constant e can be expressed using the infinite series:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

Write a complete C program that approximates e by approximating the value of

$$1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

Rather than adding an infinite number of terms, your program should continue adding terms until the value of a term is less than 0.001. Your program should print the approximation to e and the *number* of terms used to determine the approximation. The terms in the series are 1, $\frac{1}{1!}$, $\frac{1}{2!}$, and so on.

```
int main(void) {
   const double TOLERANCE = 0.001;
   double sum = 0.0, term = 1.0;
   int n = 0;

   while (term >= TOLERANCE)
   {
       sum = sum + term;     // accumulate the term
       n = n + 1;            // determine next term
       term = term / n;
   }

   printf("The value of e is approximately %f.\n", sum);
   printf("The number of terms in the sum is %d.\n", n);

   return 0;
}
```

Question 7 [10 Marks]

An `int` in C is represented and stored with a certain number of bits inside the memory of a computer, and this number can differ depending on the actual computer. Say that number of bits is `n`, and the bits are indexed from `0` to `n - 1`. The right-most bit on a binary representation is bit `0`, and it is the least significant bit (LSB). Conversely, the left-most bit is bit `n - 1`, and is the most significant bit (MSB). In an n-bit integer, bit `n - 1` is used to represent the sign: when bit `n - 1` is a `1`, the integer is negative; otherwise, the integer is positive.

Write a complete C program to determine and print out how many bits are used to represent a signed integer (the size of n) on a given computer. For example, on a computer that uses 32 bits to represent an `int`-type integer, your C program will print:

```
n = 32
```

Note: You are not allowed to use the `sizeof()` operator. Rather, your program should make use of loops.

Solution:

```
int main(void) {
    int k = 1;
    int count = 0;

    while (k > 0) {
      k *= 2; // shift
      count++; // count steps
    }

    printf("n = %d\n", count + 1);
  }
```

Question 8 [10 Marks]

Given an array of 6 integers, write a function that prints them in order of the second digit from the right in the manner shown. For example, if the array held `269, 324, 62, 5, 111, 193`, then the output would be either

`5, 111, 324, 269, 62, 193` or `5, 111, 324, 62, 269, 193`

There are two solutions since both 62 and 269 have a 6 as the second digit from the right. Either solution would be acceptable.

You should print out the commas (,) as shown. The function need only work for arrays of exactly size 6 and for only the second digit.

Start with the following:

```
void orderArrayByDigit2(int array[]) {
```

Solution:

```
void orderArrayByDigit2(int array[]) {
    char numDone = 0;
    for (int j = 0; j <= 9; j++){
        int *p = &array[0];
        for(int i = 0; i < 6; i++){
            if ((int) ((*p % 100) / 10) == j) {
                if (numDone != 0)
                    printf(",");
                numDone++;
                printf(" %d", *p);
            }
        p++;
        }
    }
    printf("\n");
}
```

Question 9 [10 Marks]

A pythagorean triple is a triple of integers (x, y, z) such that $x^2 + y^2 = z^2$. Complete this function that takes a single positive integer `x` as an argument and prints three positive integer values x, y and z such that:

1. $x^2 + y^2 = z^2$.
2. $y > 0$ and $y < 100$
3. $y < z$

If there is no triple that satisfies these conditions print "no solution exists." Start with the following definition:

```
void pythagoreanTriples (int x) {
```

Solution:

```
void pythagoreanTriples(int x) {
    for (int y = 1; y < 100; y++){
        for (int z = y + 1; z < x * x + y * y; z++) {
            if (x * x + y * y == z * z) {
              printf("x = %d, y = %d, z = %d\n", x, y, z);
              return;
            }
        }
    }
    printf("no solution exists.\n");
}
```

Question 10 [10 Marks]

Complete the definition of a C function `largestSum` whose prototype is shown below. The function takes an `int`-type array `list` with `count` elements as input, and finds the contiguous subarray (containing at least one element in the array) which has the largest sum. It returns the sum it found.

For example, in the figure below, if the list passed to the array is `{-2, 1, -3, 4, -1, 2, 1, -5, 4}`, the function returns `6`, as `{4, -1, 2, 1|` has the largest sum, which is `6`.

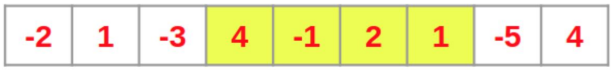

Figure 1: A contiguous subarray with the largest sum: an example.

Solution:

```
// Kadane's algorithm

int largestSum(int list[], int count) {
  int globalLargestSum = -(int) pow(2, 31);

  int localLargestSum = 0;

  for (int i = 0; i < count; i ++) {
    if (localLargestSum + list[i] > list[i]) {
        localLargestSum = localLargestSum + list[i];
    } else {
        localLargestSum = list[i];
    }

    if (localLargestSum > globalLargestSum) {
        globalLargestSum = localLargestSum;
    }
  }

  return globalLargestSum;
}

// Brute force approach O(n^2)
int largestSum(int list[], int count) {
  int globalLargestSum = -(int) pow(2, 31);


  for (int i = 0; i < count; i ++) {
    int localLargestSum = -(int) pow(2, 31), localSum = list[i];
```

```
    for (int j = i + 1; j < count; j ++) {
      localSum += list[j];
      if (localSum > localLargestSum) {
        localLargestSum = localSum;
      }
    }

    if (localLargestSum > globalLargestSum) {
       globalLargestSum = localLargestSum;
    }
  }

  return globalLargestSum;
}
```

Question 11 [12 Marks]

The *Sieve of Eratosthenes* is an ancient algorithm for finding prime numbers. To use this algorithm to find all prime numbers less than a given integer, say 100, we start by making a list of consecutive integers less than 100. We first take $p = 2$, the smallest prime number, and print it. We then eliminate all multiples of p less than 100 in the list, $(2p, 3p, 4p, \ldots)$, from the list, since they are multiples of p and are therefore not prime numbers. After eliminating the multiples of p, we find the first number after p that has not yet been eliminated, as it must be the next prime number. We assign this new prime number to p, print it, and eliminate its multiples from the list, and so on. We repeat this procedure until p^2 is greater than or equal to 100. The numbers that remain in the list are prime numbers, and we finish by printing them out.

Write a complete C program that uses the *Sieve of Eratosthenes* algorithm to print all prime numbers less than 100. Your implementation must not use the `%` (modulo) operator. The output of your program should be:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Hint: Use an array of size 100 to keep track of whether an integer has been eliminated or not.

Solution:

```
#include <stdio.h>
#include <math.h>
int main(void) {
    int sieve[100] = {0}, i, p = 2;
    while (p < sqrt(100)) {
        printf("%d ", p);
        // eliminate multiples of p
        i = 2;

        while (i * p < 100) {
            sieve[i * p] = 1;
            i ++;
        }
        p++;

        while (sieve[p] == 1) {
            p++;
        }
    }
    for (; p < 100; p++)
        if (sieve[p] == 0)
            printf("%d ", p);

    // Optional newline
    printf("\n");
}
```

MARKS

1	2	3	4	5	6	7	8	9	10	11	Total
/4	/4	/4	/4	/4	/8	/10	/10	/10	/10	/12	/80

The balance of this page has been left blank intentionally. You may use it for answers to any question in this examination.