UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

APS 105 — Computer Fundamentals
Final Examination
December 21, 2010
2:00 p.m. – 4:30 p.m.

Examiners: J. Anderson, T. Fairgrieve, B. Li, M. Papagelis

Exam Type A: This is a "closed book" examination; no aids are permitted.

Calculator Type 4: 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 programming language to answer programming questions. You are not required to write `#include` directives in your solutions. You may use any math function you have learned, as necessary.

The examination has 18 pages, including this one.

Circle your lecture section (**one mark deduction** if you do not correctly indicate your section):

L0101	or	**L0102**	or	**L0103**	or	**L0104**	or	**L0105**
Anderson		Papagelis		Li		Fairgrieve		Fairgrieve
Monday 2 PM		Monday 9 AM		Monday 11 AM		Monday 11 AM		Monday 4 PM

Full Name: ______________________________

Student Number: ______________________ ECF Login: ______________________

MARKS

1	2	3	4	5	6	7	8	9	10	11	Total
/28	/6	/6	/6	/6	/8	/8	/8	/8	/8	/8	/100

Question 1 [28 Marks]

1.1 [3 Marks] Write a single C statement that prints the following message followed by a newline character:

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

1.2 [3 Marks] Write a single C statement that declares an `int` array named `values` that has 5 elements. Each of the 5 elements should be initialized to `99`.

1.3 [3 Marks] Write a single C statement that declares a `char` type variable named `encoded` and initializes it. The initial value for `encoded` depends on the value of another `char` type variable named `c` that has already been declared and assigned an upper case letter. The value of `encoded` should be initialized according to the following scheme:

If `c` is `'Z'`, `encoded` should be initialized to `'A'`.
If `c` is `'Y'`, `encoded` should be initialized to `'B'`.
...
If `c` is `'A'`, `encoded` should be initialized to `'Z'`.

1.4 [3 marks] The following C statement has been executed:

```
typedef struct stuRec
{
    int    stuNumber;
    char   name[100];
    double score;
} StudentRecord;
```

Write a single C statement that declares and uses `malloc` to allocate an array of type `StudentRecord` that has 42 elements. The array should be named `sRecords`.

1.5 [4 Marks] Consider the following C code fragment:

```
int list[4] = {0};

int *p = list;
p++;
*p = p - list;
p++;
*p = p - list;
p = &list[3];
*p = 4;
```

Give the values of the array elements:

Array element	Answer
`list[0]`	
`list[1]`	
`list[2]`	
`list[3]`	

1.6 [4 Marks] The following implementation of quicksort has errors in two statements. Fix the errors in the code so that the implementation of quicksort is correct. The parameter `values` is an `int` type array that must be sorted into ascending (nondecreasing) order.

```
void quickSort (int values[], int low, int high)
{
  int left = low;
  int right = high;

  if (left >= right)
    return;

  int pivot = values[left];

  while (left < right)
  {
    while ((values[right] >= pivot) && (left < right))
      right--;

    values[left] = values[right];

    while ((values[left] <= pivot) && (left < right))
      left++;

    values[right] = values[left];
  }
  values[left] = pivot;

  quickSort(values, low, left + 1);
  quickSort(values, high, right + 1);
}
```

1.7 [4 Marks] Add C statements to the following C function to implement the bubble sort algorithm. The parameter `values` is an `int` type array that is to be sorted into ascending (nondecreasing) order. The parameter n gives the number of elements in the array `values`.

```
void bubbleSort (int values[], int n)
{
  bool sorted = false;
  int i, top;

  for (top = n - 1; top > 0 && !sorted; top--)
  {
    sorted = true;
    for (i = 0; i < top; i++)
    {
      if (values[i] > values[i+1])
      {








      }
    }
  }
}
```

1.8 [4 Marks] What output does the following program produce?

```
#include <stdio.h>

int aps105 (int n)
{
  int val;
  printf("Entering %d\n", n);
  if (n == 1)
    val = 1;
  else
    val = n * aps105(n-1);
  printf("Leaving %d\n", n);
  return val;
}

int main (void)
{
  int i;
  i = aps105(2);
  printf("%d\n", i);
  return 0;
}
```

Question 2 [6 Marks]

Write a C function named `reverse`, the prototype of which is given below, that has two parameters. The parameter `list` is an integer array that has already been sorted into ascending (nondecreasing) order, and the parameter `length` gives the number of elements in `list`. The function changes the order of elements in `list` from ascending to descending order.

For example, if `list` is initially {`1, 3, 3, 5, 7, 11, 13, 22, 34`}, then, after the function call has completed, `list` should contain {`34, 22, 13, 11, 7, 5, 3, 3, 1`}.

```
void reverse (int list[], int length)
{



}
```

Question 3 [6 Marks]

Write a C function named append, the prototype of which is given below, that returns the result from placing string `s2` at the end of string `s1`. For example, if `s1` contains `"Toronto"` and `s2` contains `"Canada"`, the function `append` returns the string `"TorontoCanada"`. You are not allowed to use the library function `strcat` from `string.h` in your solution.

Note: Your function may not change string `s1` or `s2`.

```
char *append (const char *s1, const char *s2)
{







}
```

Question 4 [6 Marks]

Write a C function having prototype `void moveRight (int arr[], int length, int k)` that moves the first `(length - k)` elements in array `arr` `k` places to the right and wraps the last `k` elements in array `arr` around to the left end of the array.

Assume that `0 <= k < length` and that array `arr` has `length` elements.

For example, if `arr[] = {11, 22, 33, 44, 55, 66, 77, 88},`
the call `moveRight(arr, 8, 3)` changes `arr` to `{66, 77, 88, 11, 22, 33, 44, 55}`.

```
void moveRight (int arr[], int length, int k)
{




}
```

Question 5 [6 Marks]

Complete **one** statement in the `printPermutationsHelper` function below so that the given program prints all permutations of the digits between `1` and `n`. For example, when the user enters the number `3`, the program should output:

```
123 132 213 231 312 321
```

```
#include <stdbool.h>
#include <stdio.h>
bool digitDoesNotAppear (int digit, int number)
{
    bool result = true;
    while (number > 0 && result)
    {
        if (number % 10 == digit)
            result = false;
        number = number / 10;
    }
    return result;
}
void printPermutationsHelper (int a, int b, int c)
{
    if (c == 0)
        printf("%d ", a);
    else
        for (int i = 1; i <= b; i++)
        {
            if (digitDoesNotAppear(i, a))
                // complete the statement in the box below
               -----------------------------------------------------------
              | printPermutationsHelper( 10*a +      ,          ,        );|
               -----------------------------------------------------------
        }
}
void printPermutations (int n)
{
    printPermutationsHelper(0, n, n);
}
int main (void)
{
    int n;
    scanf("%d", &n);
    printPermutations(n);
    return 0;
}
```

Question 6 [8 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 then take $p = 2$, the smallest prime number, and print it. We can eliminate all multiples of p less than 100, $(2p, 3p, 4p, \ldots)$ from the list. Since they are multiples of p, they are not prime numbers. We then find the first number after p that has not yet been eliminated, as it must be the next prime number. We assign this prime number to p, print it, and eliminate its multiples from the list. 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 C program that uses the *Sieve of Eratosthenes* algorithm to print all prime numbers less than 100. Your implementation must not use the `%` 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: You may need to implement a "bitmap" of size 100 to keep information on whether an integer is prime or not. A "bitmap" is an array of integers containing only 0s and 1s.

```
int main(void)
{








}
```

Question 7 [8 Marks]

Write a complete C function named `mergeSortedStrings`, the prototype of which is given below, that receives two alphanumerically ordered character strings (`str1` and `str2`) as parameters. The function must create and return a new string that contains the characters from `str1` and `str2` and is alphanumerically ordered as well. For example, if `str1` were `"CEFHIJ"` and `str2` were `"ABDG"` the function must create and return a new string containing `"ABCDEFGHIJ"`. You may assume that characters in the strings `str1` and `str2` are distinct (i.e., the same character cannot appear in both strings).

You may use library functions from `string.h` in your solution. Your function may not change string `str1` or `str2`.

Note: An *alphanumeric* character is a character that is either a letter or a number. If a string `str` contains an alphanumerically ordered character string, then `str[i] < str[j]` whenever `i < j`.

```
char *mergeSortedStrings (const char *str1, const char *str2)
{









}
```

Question 8 [8 Marks]

Write a complete C function named `minElementSort`, the prototype of which is given below, that receives an integer array `list` and its size `length` as its parameters. The function sorts the given array into ascending (nondecreasing) order, following a slight variation of the selection sort algorithm, called the *minimum element sort*. The algorithm works as follows: It goes through the list, finds the smallest element and swaps it with the first element of the list. Then it finds the smallest element in the rest of the list and swaps it with the second element of the list. And so on. An example follows.

If the input integer array `list` is:

```
10 5 20 40 50 12 434 21 9 232
```

then after calling the `minElementSort` function, the integer array `list` is:

```
5 9 10 12 20 21 40 50 232 434
```

```
void minElementSort (int list[], int length)
{

}
```

Question 9 [8 Marks]

The sequence 2, 6, 18, 54, 162, . . . is said to be *geometric* since the result from dividing each term by the term that precedes it is always the same. In the given sequence, the common ratio is 3. A sequence with fewer than 2 terms is **not** geometric.

Write a function named `isGeometric` that uses **recursion** to determine whether or not a length `len` sequence of integer values that are given in a type `int` array is a geometric sequence.

Your solution **must** use recursion to solve the problem. You **may** use a "helper" function in your solution.

```
bool isGeometric (int arr[], int len)
{











}
```

Question 10 [8 Marks]

In this question you are to write a function that examines a linked list of type `Node` variables, where the `Node` type has been defined as follows:

```
typedef struct node
{
    char        *studentName;
    int          grade;
    struct node *link;
} Node;
```

Write a C function having prototype `double averageGrade (Node *head)` that returns the average of the `grade` elements in the linked list. The value `0.0` should be returned if the linked list is empty.

```
double averageGrade (Node *head)
{









}
```

Question 11 [8 Marks]

Assume that you have implemented two linked lists to maintain in memory information about the players on a soccer team. One list maintains information on the players currently playing on the field, and the other list maintains information on the players currently sitting on the bench, waiting to go out onto the field (if the coach decides so). In both linked lists a player is represented by a node of type `Node`:

```
typedef struct node
{
  char        *familyName;
  char        *firstName;
  char         position;     // One of 'G', 'D', 'M', 'S'
  int          value;        // in thousands of dollars
  struct node *link;
} Node;
```

It has been decided that the team would be more successful if players with high values are on the field and players with low values are on the bench.

Write a complete C function named `replacementPlayer`, the prototype of which is given on the next page. The function receives three parameters: a pointer to the head of a linked list representing the players currently playing on the field (called `headField`), a pointer to the head of a linked list representing the players waiting on the bench (called `headBench`), and a character representing the player position to consider for replacement (one of `'G'`, `'D'`, `'M'`, `'S'`, indicating the position of a Goalkeeper, a Defender, a Midfielder, or a Striker, respectively). Your function should return a pointer to the node in the **bench list** that contains information about the **best** replacement player for the given position at this moment or `NULL` if there is no suitable replacement. For a given position, a player with a higher value is preferred to a player with a lower value. You can assume that the values of any two players (either currently playing on the field or on the bench) are different.

Note: In Lab 8, the linked list was ordered by player position. But for this question, to make things simpler, do not assume that the list is in any particular order.

```
Node *replacementPlayer (Node *headField, Node *headBench, char position)
{







}
```

This page has been left blank intentionally. You may use it for your answer to any of the questions in this examination.