Solutions

UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

APS105 — Computer Fundamentals
Final Examination
April 28, 2014

Examiner: Khoman Phang

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

A **Table of Common C functions and the ASCII Character Table** can be found on the **last page of this exam**.

All questions are to be answered on the test 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.

The test has 11 pages, including this one.

The marks allocated to the questions, out of a total of 65, are shown in the question headings.

Name: ______________________ Student No.: ______________________

MARKS

1	2	3	4	5	6	7	Total
/10	/10	/10	/10	/10	/10	/10	/70

Question 1 [10 Marks]

(a) **True** or [✓ **False**]: To *cast* in C means to set the value of a constant.

(b) [✓ **True**] or **False**: The following declaration is legal: `int list[5] = {4,7,3};`

(c) **True** or [✓ **False**]: The value of the expression, `(int) 1.8*0.6`, is 1.

(d) **True** or [✓ **False**]: The statement `i *= j + 2;` is equivalent to `i = i*j + 2;`

(e) **True** or [✓ **False**]: Whenever a `for` loop is executed, the statement in the body of the loop is always executed at least once.

(f) [✓ **True**] or **False**: In a function call, the type of the argument does not have to be identical to the type of the parameter.

(g) [✓ **True**] or **False**: In C, the only form of parameter passing is call by value.

(h) [✓ **True**] or **False**: If `list` has been declared as an array, then `list+3` is a synonym for `&list[3]`.

(i) [✓ **True**] or **False**: The maximum value of m modulo n, or `m%n`, is `n-1`.

(j) **True** or [✓ **False**]: If `malloc()` successfully allocates memory, the function will return the number of bytes it has allocated.

Question 2 [10 Marks]

(a) [**4 Marks**] State whether the following relational expressions are true or false:

Expression	Answer
`'1' > 'a'`	*false*
`'A' - 'Z' == 'a' - 'z'`	*true*
`'C' == 3 + 'A'`	*false*
`'0' == 0`	*false*

(b) [**2 Marks**] Let `colour` be the following structure:

```
struct colour
{
    int red;
    int blue;
    int green;
};
```

Write a single C statement that declares a `const` variable named `MAGENTA` of type `struct colour` whose members are initialized to the values 255, 0, 255, respectively. That is, the member red is initialized to 255, blue is initialized to 0, and green is initialized to 255.

const struct colour MAGENTA = {255,0,255};

(c) [**2 marks**] When executing the binary search algorithm on a list of sorted data, would it be preferable to have the list stored in an array or in a linked list, or does it not matter? Explain your response.

In an array since it is easier to determine the midpoint of an array than it is to determine the midpoint of a linked list.

(d) [**2 marks**] Consider a C-language identifier `values` that points to a dynamically allocated array of N `double`-type variables. Write a *single* C statement to deallocate the memory consumed by the array. You may assume that `#include <stdlib.h>` appears at the top of the C file.

free (values);

Question 3 [10 Marks]

The Node structure in a linked list has been defined as follows:

```
typedef struct node
{
   double real, img;
  struct node *link;
} Node;
```

Write a C function, `double FindAverage (Node *head)`, that calculates and returns the average of the magnitudes of all the complex numbers stored in the linked list pointed to by pointer parameter, `head`.

```
double findAverage(Node *head)
{
  Node *current;
  int count;
  double sum = 0;
  current = head;

  while(current != NULL)
  {
    sum += sqrt(pow(current->real,2) + pow(current->img,2));
    count++;
    current = current->link;
  }

  return sum/count;
}
```

Question 4 [10 Marks]

An NxN square matrix is called a **diagonal** matrix if all its entries off the main diagonal are zero. Example:

$$\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

Write a C function, `bool isDiagonal (int matrix[N][N])`, that takes an `N x N` matrix pointed to by parameter, `matrix`, and returns `true` if the matrix is diagonal and returns `false` otherwise. Assume that constant N is a positive integer, and note that a 1x1 matrix is diagonal by definition.

```
bool isDiagonal(int matrix[N][N])
{
  bool result = true;

  if (N == 1) return true;

  int i,j;

  for(i = 0; i < N; i++)
    for(j = 0; j < N; j++)
    {
      if(i != j && matrix[i][j] != 0)
     result = false;
    }

  return result;
}
```

Question 5 [10 Marks]

Write a complete C program that adds up the command-line arguments and prints out the resulting sum. You can assume that all the command line arguments are signed integers.
Assuming the name of the executable file is `sum`, here is a sample program compilation, execution, and the resulting printed output:

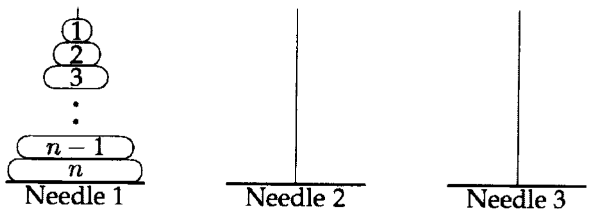
```
% gcc -o sum sum.c
% sum 1 2 3 4
The sum is 10.
```

Start of source code, `sum.c`:

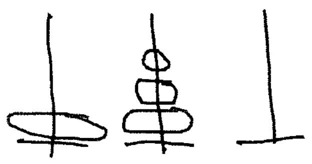
```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
   int i,sum = 0;

   for(i = 1; i<argc; i++)
     sum += atoi(argv[i]);

   printf("The sum is %d\n", sum);
   return 0;
}
```

Question 6 [10 Marks]

Trace through the Towers of Hanoi program provided below and write out the first 7 lines that are printed out by the program. Describe briefly at what point the algorithm has reached after these 7 moves, and draw the corresponding state of the three needles given an initial stack of n=4 disks on Needle 1.



First step of the recursive algorithm, moveTower (3, 1, 2) has been completed after 7 moves.



```
void moveTower (int height,int start,int finish)
{
   if (height == 1)
       printf("%d ---> %d\n",start,finish);
   else
   {
       int spare = 6 - (start + finish); // identify the spare needle

      // Move the (N-1)-disk tower from start Needle to spare Needle
       moveTower(height-1,start,spare);

      // Move the bottom disk from start Needle to finish Needle
       printf("%d ---> %d\n",start,finish);

      // Move the (N-1)-disk tower from spare Needle to finish Needle
       moveTower(height-1,spare,finish);
   }
}
```

```
  moveTower(4, 1, 3)

  moveTower(3, 1, 2)       moveTower(1, 1, 3)       moveTower(3, 2, 3)

moveTower(2, 1, 3)       moveTower(1, 1, 2)       moveTower(2, 3, 2)
                              1 ---> 2
                                 4

moveTower(1, 1, 2)       moveTower(1, 1, 3)       moveTower(1, 2, 3)
 1 ---> 2                 1 ---> 3                 2 ---> 3
    1                        2                        3

               moveTower(1, 3, 1)       moveTower(1, 3, 2)       moveTower(1, 1, 2)
                3 ---> 1                 3 ---> 2                 1 ---> 2
                   5                        6                        7
```

```
1 ---> 2
1 ---> 3
2 ---> 3
1 ---> 2
3 ---> 1
3 ---> 2
1 ---> 2
```

Question 7 [10 Marks]

Suppose that an array contains the initial values: **4 16 21 14 13 18 8**.

Show the array as it would appear after each full pass of the **selection** sort.

Pass 1: 4 16 8 14 13 18 21

Pass 2: 4 16 8 14 13 18 21

Pass 3: 4 13 8 14 16 18 21

Pass 4: 4 8 13 14 16 18 21

Pass 5: 4 8 13 14 16 18 21

Pass 6 (final result): 4 8 13 14 16 18 21

Now write the C function, `void selectSort (int list[], int length)`, to implement the above selection sort. The function receives a non-empty array, `list`, containing positive integers and the size of the array, `length`, as its parameters.

```
void selectSort (int list[], int length)
{
  int top, largest, i;

  for (top = length - 1; top > 0; top--)
  {
    // find largest entry from 0 to top
    largest = 0;
    for (i = 1; i <= top; i++)
      if (list[i] > list[largest])
        largest = i;

    // put largest entry at top
    int temp = list[top];
    list[top] = list[largest];
    list[largest] = temp;
  }
}
```

1 REFERENCE: Some Common C functions

Math functions – `#include <math.h>`	**Value returned**
`sqrt(x)`	$\sqrt{x}$
`pow(x,y)`	x^y
`log(x)`	$\ln x$
`exp(x)`	e^x
`fabs(x)`	$\|x\|$
`fmax(x,y)`	maximum of arguments
`fmin(x,y)`	minimum of arguments
`floor(x)`	largest integer $\leq x$ (as a `double`)
`ceil(x)`	smallest integer $\geq x$ (as a `double`)
`sin(x)`	$\sin x$ (in radians)
`cos(x)`	$\cos x$
`atan(x)`	$\arctan x$ (x in radians)
String functions – `#include <string.h>`	
`int strlen( char *s )`	Returns length of string.
`int strcmp( char *s1, char *s2 )`	Compare.
`char *strcpy( char *s1, char *s2 )`	Copy s2 to s1. Returns pointer to s1.
`char *strcat( char *s1, char *s2 )`	Concatenate s2 to s1. Returns pointer to s1.
`char *strstr( char *s1, char *s2)`	Find leftmost occurence of s2 in s1.
Standard utility functions – `#include <stdlib.h>`	
`double atof( char *nvalstr )`	Convert numeric string to double.
`int atoi( char *nvalstr )`	Convert numeric string to int.
`long atol( char *nvalstr )`	Convert numeric string to long.
`int rand()`	Generates pseudorandom integer.

2 ASCII Character Table (Excerpt)

Char	Decimal
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
...	...

Char	Decimal
A	65
B	66
C	67
...	...
X	88
Y	89
Z	90
...	...
a	97
b	98
c	99
...	...
x	120
y	121
z	122