UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

APS 105 — Computer Fundamentals
Final Examination
December 14, 2012
2:00 p.m. – 4:30 p.m.
(150 minutes)

Examiners: J. Anderson, B. Li, M. Sadoghi, D. Sengupta, G. Steffan

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 that you have learned, as necessary.

The examination has 22 pages, including this one.

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

L01	**L02**	**L03**	**L04**	**L05**	**L06**
Li	Li	Anderson	Steffan	Sengupta	Sadoghi
Monday 2 PM	Monday 9 AM	Monday 11 AM	Monday 11 AM	Monday 4 PM	Monday 4 PM

Full Name: _____

Student Number: _____ UTORID: _____

MARKS

2-3	4	5-6	7	8	9	10	11	12	14	15	16	18	20	Total
/6	/6	/8	/4	/4	/8	/8	/8	/8	/8	/8	/8	/8	/8	/100

Question 1 [3 Marks]

How would the following sorting algorithms perform when sorting the example arrays given? In particular, for each algorithm, count the number of times that two numbers from the array would be compared, and give that number as an answer.

Array 1: `5 4 3 2 1`

Solution:

Algorithm	Answer
Selection sort	10
Insertion sort	10

Array 2: `1 4 2 5 3`

Solution:

Algorithm	Answer
Selection sort	10
Insertion sort	7

Array 3: `1 2 3 4 5`

Solution:

Algorithm	Answer
Selection sort	10
Insertion sort	4

Question 2 [3 Marks]

Consider the following C program:

```
#include <stdio.h>

void visitGrid(int x, int y)
{
  if (x < 10 && y < 10)
  {
    visitGrid(x + 1, y);
    visitGrid(x, y + 1);
    printf("%d,%d\n", x, y);
  }
}

int main(void)
{
  visitGrid(0, 0);
  return 0;
}
```

In the space below, write only the first three lines that would be printed by its execution.

Solution:

```
9,9
9,8
9,7
```

Question 3 [3 Marks]

Write a single C statement that declares a `int` type variable named `r` and initializes it to a random number between 50 and 1500 inclusive, where the random number is divisible by 10.
Solution:

```
int r = (rand() % 146 + 5) * 10;
```

Question 4 [3 Marks]

Circle and correct the errors in the following complete C program. You should cross out any lines with errors and rewrite them.

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

int main(void)
{
  char *str = 'pass the exam';
  char *str2 = (char)malloc(strlen(str)*sizeof(char));

  strcpy(str2, str);

  printf("%s %s\n", str, str2);

  return 0;
}
```

A correct execution of the program should print:

```
pass the exam pass the exam
```

Solution:

```
1) malloc result should be cast to char *
2) we need to malloc strlen(str)+1 [for the NULL character]
3) should be " " around the string instead of ' '
```

Question 5 [4 Marks]

What output does the following C program produce?

```
#include <stdio.h>

int foo(int *num)
{
  int result = *num * 2;
  num = &result;
  (*num)++;
  return *num + result;
}

int main(void)
{
  int num = 3;
  printf("num = %d, foo(num) = %d, num = %d\n", num, foo(&num), num);
  return 0;
}
```

Solution:

```
num = 3, foo(num) = 14, num = 3
```

Question 6 [4 Marks]

What output does the following program produce?

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

void foo(char *s, char *d)
{
  if(s != d)
  {
    foo(s + 1, d - 1);
    printf("%c %c\n", *s, *d);
  }
}

int main(void)
{
  char *hello = "Hello World";
  foo(hello, hello + strlen(hello) - 1);
  return 0;
}
```

Solution:

```
o W
l o
l r
e l
H d
```

Question 7 [4 Marks]

What is the output of the following program?

```
#include <stdio.h>

void quicksort(int list[], int low, int high)
{
  printf("%d,%d\n", low, high);
  if (low <= high)
  {
    double pivot = list[low];
    int left = low;
    int right = high;

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

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

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

    quicksort(list, low, left - 1);
    quicksort(list, right + 1, high);
  }
}

int main(void)
{
  int list[] = {5, 7, 2};
  quicksort(list, 0, 2);
}
```

Solution:

```
0,2
0,0
0,-1
1,0
2,2
2,1
3,2
```

Question 8 [4 Marks]

Given the following C function:

```
void theNineAgesOfMan()
{
  char words[5][8] = {"not",
                      "old",
                      "enough",
                      "to know",
                      "better"};

  for (int i = 0; i < 9; i++)
  {
    int start = i % 2;
    int stop = 5 - i/2;
    for (int j = start; j < stop; j++)
      printf("%s ", words[j]);
    printf("\n");
  }
}
```

In the space below, write the output that would be printed if this function were called (a poem by Emerson Andrews).

Solution:

```
not old enough to know better
old enough to know better
not old enough to know
old enough to know
not old enough
old enough
not old
old
not
```

Question 9 [8 Marks]

Write a C function called `borderSum`, the prototype of which is given below, that returns the sum of the values in the border of a `n x n` array containing integers. The sum should include values in the top and bottom rows, and the leftmost and rightmost columns, but include the corner values in the sum only once. For example, if the input array `a` is:

```
8   6   9  5
1   5   2  4
9  12  16  8
13 10  11  16
```

with `n = 4`, the function `borderSum(4, a)` should return `100`.

Assume that the value of `N` is greater than `0`.

Hint: Consider three different cases of `n`: `n = 1`, `n = 2` and `n > 2`.
Solution:

```
int borderSum(int n, int **a)
{
  int sum = 0, col, row;

  if(n == 1)
    sum = a[0][0];

  else if (n == 2)
    sum = a[0][0] + a[0][1] + a[1][0] + a[1][1];

  else
  {
    for (col = 0; col < n; col++)
      sum += a[0][col] + a[n-1][col];

    for (row = 1; row < n-1; row++)
      sum += a[row][0] + a[row][n-1];
  }

  return sum;
}
```

Question 10 [8 Marks]

Write a complete C function named `findLongestSequence`, the prototype of which is given below, that receives two parameters: a non-empty array `list` containing unique positive integers (without duplicates), and the size of the array `size`. The function returns the length of the longest sequence of increasing numbers in the array. For example if the input array `list` is:

```
12  14  16  8  9  10  11 21 6
```

The function `findLongestSequence(list, 9)` should return `5`, since the longest sequence of increasing numbers is `8, 9, 10, 11, 21`.

Important: You are **not** allowed to use any additional array within the function.
Solution:

```
int findLongestSequence(int list[], int size)
{
  int start = 0, end = 0, length = 0;

  for (int i = 1; i < size; i++)
  {
    if (list[i] > list[i-1])
      end++;
    else
    {
      if (end - start + 1 > length)
        length = end - start + 1;

      start = end = i;
    }
  }

  if (end - start + 1 > length)
      length = end - start + 1;

  return length;
}
```

Question 11 [8 Marks]

Write a complete C function called `produceRandomNumbers`, the prototype of which is given below, that receives an array `list` and the size of the array `length` as its parameters. The function fills all the elements in the array with randomly generated integer values in the range between 1 and `length * 2` inclusive, with the requirement that all integer values in the array must be unique, *i.e.*, there are no duplicates.

Solution:

```
void produceRandomNumbers(int list[], int length)
{
  int *status = (int *) malloc(sizeof(int) * (length * 2 + 1));
  int i;

  for (i = 1; i <= length * 2; i ++)
    status[i] = 0;

  for (i = 0; i < length; i ++)
  {
    do
      list[i] = rand() % (length * 2) + 1;
    while (status[list[i]]);

    status[list[i]] = 1;
  }
  free(status);
}
```

Question 12 [8 Marks]

Write a C function called `checkPlagiarism`, the prototype of which is given below, that returns `true` if two suspected input codes (`code1`) and (`code2`) have high similarity. *High similarity* is defined as matching exactly, but ignoring any spaces (`' '`) or return characters (`'\n'`). For example, the function `checkPlagiarism()` returns `true` when comparing the example strings `c1` and `c2` below. Do not use recursion in your solution. You can assume that `code1` and `code2` are null-terminated strings. **Hint:** Your code should return `false` as soon as it finds evidence of a mis-match.

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

bool checkPlagiarism(char *code1, char *code2);

int main(void)
{
 char c1[] = "int main(void){\n int x = 10;\n int z = x + 5;\n return 0;\n}\n";
 char c2[] = "int main(void){\n  int x=10;\n  int z=x+5;\n\n\n  return 0;\n}\n";

 printf("%d\n", checkPlagiarism(c1, c2));
}
```

Solution:

```
bool checkPlagiarism(char *code1, char *code2)
{
  while (*code1 && *code2)
  {
    if (*code1 == *code2)
    {
      code1++;
      code2++;
    }
    else
    {
      if (*code1 == '\n' || *code1 == ' ')
        code1++;
      else if (*code2 == '\n' || *code2 == ' ')
        code2++;
      else
        return false;
    }
  }

  if (!*code1 && !*code2)
    return true;
```

```
  else
    return false;
}
```

Question 13 [8 Marks]

Given an unsorted array of size `length`, insertion sort builds the sorted array by visiting each element from index `1` forward to the end of the array (index `length - 1`). A variation of insertion sort visits each element in the unsorted array from index `length - 2` backward to the start of the array. At any given iteration, if the index of the visited element is `i` then one has to find the correct position of the element in the already sorted list starting from index `i + 1` to `length - 1`. For example, if the input unsorted array is:

`9 16 7 2 5 4 3`

then the content of the array after each iteration is shown below:

`9 16 7 2 5` **`3 4`**

`9 16 7 2` **`3 4 5`**

`9 16 7` **`2 3 4 5`**

`9 16` **`2 3 4 5 7`**

`9` **`2 3 4 5 7 16`**

`2 3 4 5 7 9 16`

The sorted sublist after each iteration is shown in **bold** font. Note that the array is to be sorted in ascending order.

Write a C function called `newInsertionSort`, the prototype of which is given below, that receives a non-empty array `list` containing positive integers and the size of the array `length` as its parameters and sorts the elements in the array in ascending order using the above technique.

Important: You are **not** allowed to use any additional arrays within the function. Solutions using additional arrays within the function will receive 0 marks.
Solution:

```
void newInsertionSort(int list[], int length)
{
  int top, i;
  int item;

  for (top = length - 2; top >= 0; top--)
  {
    item = list[top];
    i = top;
    while (i < length - 1 && item > list[i+1])
    {
      list[i] = list[i+1];
      i++;
```

```
    }

    list[i] = item;
  }
}
```

Question 14 [8 Marks]

The following sentences are examples of *palindromic sentences* because, when spaces and punctuation are removed, the remaining words are palindromes (palindromes read the same in both directions).

never odd or even.
a nut for a jar of tuna.
no lemon, no melon.

Write a recursive C function called `isPalindromicSentence` that determines if a string (passed in as a parameter) is a palindromic sentence. You may assume the string contains only lower-case characters, and that the only punctuation marks used are `'.'` and `','`. You may write a helper function (and have that helper function be recursive rather than the `isPalindromicSentence` function). You may only use the `strlen` function from the `string.h` library, and may not use any other functions from `string.h`. Solutions that do not use recursion will receive 0 marks. You are not allowed to allocate any new arrays in your solution.

Solution:

```
bool isPalindromicSentenceHelper(char *start, char *end)
{
  while ((start <= end) && (*start == '.' || *start == ',' || *start == ' '))
    start++;

  while ((end >= start) && (*end == '.' || *end == ',' || *end == ' '))
    end--;

  if (end <= start)
    return true;

  if (*end != *start)
    return false;

  return isPalindromicSentenceHelper(start + 1, end - 1);
}

bool isPalindromicSentence(char *str)
{
  return isPalindromicSentenceHelper(str, str + strlen(str) - 1);
}
```

Question 15 [8 Marks]

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

```
typedef struct node
{
  int info;
  struct node *link;
} Node;
```

Write a C function called `deleteFirstLast`, the prototype of which is given below, that deletes the first and the last node in a linked list pointed to by `head`, and returns a pointer to the head of the linked list after deletion. If the linked list is empty, the function returns `NULL`. If the linked list has only one node, the function deletes that node and returns `NULL`.

Solution:

```
Node * deleteFirstLast(Node *head)
{
    if (head == NULL)
      return NULL;

    if (head->link == NULL)
    {
      free(head);
      return NULL;
    }

    if (head->link->link == NULL)
    {
      free(head->link);
      free(head);
      return NULL;
    }

    // Initialize current and newHead to point to the 2nd node
    Node *current = head->link;
    Node *newHead = head->link;

    // Delete the first node
    free(head);

    while (current->link->link != NULL)
      current = current->link;

    free(current->link);
    current->link = NULL;
    return newHead;
}
```

Question 16 [8 Marks]

Write a recursive C function called `getTwoHighest`, the prototype of which is given below, that finds the two largest elements in an array (passed in as a parameter) and stores them in two integer variables, pointers to which are passed in as parameters. An example illustrating the use of this function is given below. In this example, the output `13 12` is printed on a line (since these are the largest two elements in the array `list1`), followed by the output `7 7` (since these are the largest two elements in the array `list2`). You may assume the function is called for an array with at least 2 elements. Solutions that do not use recursion will receive 0 marks. You are allowed to use a helper function if you wish.

```
int main(void)
{
  int a, b;

  int list1[] = {5, 7, 13, 10, 12};
  getTwoHighest(list1, 5, &a, &b);
  printf("%d %d\n", a, b);

  int list2[] = {7, 6, 7, 2};
  getTwoHighest(list2, 4, &a, &b);
  printf("%d %d\n", a, b);

  return 0;
}
```

Solution:

```
void getTwoHighest(int list[], int n, int *first, int *second)
{
  if (n > 2)
  {
    getTwoHighest(list + 1, n - 1, first, second);

    if (list[0] >= *first)
    {
      *second = *first;
      *first = list[0];
    }
    else if (list[0] >= *second)
      *second = list[0];
  }
  else // n == 2
  {
    if (list[0] > list[1])
    {
      *first = list[0];
```

```
      *second = list[1];
    }
    else
    {
      *first = list[1];
      *second = list[0];
    }
  }
}
```

This page has been left blank intentionally. You may use it to answer Question 16.

Question 17 [8 Marks]

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

```
typedef struct node
{
  int info;
  struct node *link;
} Node;
```

Write a C function called `splitSwap`, the prototype of which is given below, that finds the node with `info` equal to the parameter `item`. This node is used for splitting the linked list into a left and a right list. The left list contains all the nodes to the left of the splitting node, and the right list contains all the nodes to the right of the splitting node. The splitting node itself is included in the left list. The `splitSwap` function returns the modified linked list by swapping the left and right lists.

If the input linked list is

```
1->9->13->4->5->6->17->8->NULL
```

then after calling the `splitSwap` function with `item = 5`, the modified linked list returned by the function is

```
6->17->8->1->9->13->4->5->NULL
```

Assume all `info` values are unique in the linked list. Also, if the `item` is not found, then the original linked list is simply returned without any modification.

Solution:

```
Node * splitSwap(Node *head, int item)
{
    Node *current = head, *split;

    // Handle the empty list
    if (!current)
        return head;

// Find the item for splitting the list
    while (current->link && current->info != item)
        current = current->link;

    if (current->link)
    {
     //Save the address of splitting node
        split = current;

        //Find the last element in the list
        while (current->link)
             current = current->link;

        //Swap the list at position split
        current->link = head;
        head = split->link;
        split->link = NULL;
    }

    return head;
}
```

This page has been left blank intentionally. You may use it for answers to any questions.