UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

APS 105 — Computer Fundamentals
Final Examination
December 14, 2009
6:30 p.m. – 9:00 p.m.

Examiners: J. Anderson, T. Fairgrieve, H. Ghaderi, B. Li

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 or the back of the question’s page to complete your answer. Indicate clearly which question(s) you are answering.

You must use the C programming language to answer programming questions.

The examination has 16 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**
Fairgrieve		Ghaderi		Ghaderi		Anderson		Li
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] Describe one disadvantage of using *linked lists* versus *arrays*.

Solution: To find the ith element in a linked list, we need to traverse the list, while with arrays we have fast (random) access to each element (i.e. `arname[i]`).

1.2 [3 Marks] What output does the following program fragment produce?

```
int i = 9384;
do
{
  printf("%d ", i);
  i /= 10;
} while (i > 0);
```

Solution:

```
9384 938 93 9
```

1.3 [3 Marks] Write a single C statement that declares a type `int` array named `list` with 100 elements, such that the first 5 array elements are initialized to contain the values `1, 2, 3, 4, 5` and the remaining elements are initialized to `0`.

Solution:

```
int list[100] = {1, 2, 3, 4, 5};
```

1.4 [3 marks] A student in APS 105 has written the following function to rotate the values of 3 `double` variables. For example, if `x` is 1.0, `y` is 5.0, and `z` is 3.5, after calling `rotate3Nums(x, y, z)` he wants the value of `x` to become 5.0 (i.e., `y`), `y` to become 3.5 (i.e., `z`), and `z` to become 1.0 (i.e., `x`). As written, the code is incorrect. Fix all the problems in the code.

```
void rotate3Nums(double num1, double num2, double num3)
{
  num1 = num2;
  num2 = num3;
  num3 = num1;
}
```

Solution:

```
void rotate3Nums(double *num1, double *num2, double *num3)
{
  double tmp = *num1;
  *num1 = *num2;
  *num2 = *num3;
  *num3 = tmp;
}
```

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

```
struct record
{
 int x, y;
};

struct record a;
struct record *p = &a;
```

State which of the following assignment statements are legal and which are illegal after the fragment above has been executed?

Solution:

Statements	Answer
`p.x = 3;`	illegal
`(*p).x = 3;`	legal
`a->y = 4;`	illegal
`p->x = (&a)->y;`	legal

1.6 [4 Marks] Suppose that an array initially contains the values `{8, 4, 2, 7, 3}`. If the array is to be sorted into ascending order using *insertion sort*, show the contents of the array as it would appear after each of the required passes in insertion sort.

Solution:

```
4, 8, 2, 7, 3
2, 4, 8, 7, 3
2, 4, 7, 8, 3
2, 3, 4, 7, 8
```

1.7 [4 Marks] Will the following complete C program run successfully? If so, show the output after running this program. Otherwise, if the program does not run successfully, explain the reason.

```
#include <stdio.h>

int main(void)
{
  int *x;
  int result;

  *x = 0;
  result = *x;
  printf("%d", result);
}
```

Solution: No, it will not run successfully, since the memory that the integer pointer `a` points to has not been allocated.

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

```
void f(int *i, int j)
{
  int n = 10;
  printf("Entering f: i=%d, j=%d\n", *i, j);
  *i = 20;
  j = 30;
  i = &j;
  printf("Exiting f: i=%d, j=%d, n=%d\n", *i, j, n);
}

int main(void)
{
  int i=0, j=1, n=2;
  f(&n, j);
  printf("In main: i=%d, j=%d, n=%d \n", i, j, n);
  return 0;
}
```

Solution:

```
Entering f: i=2, j=1
Exiting f: i=30, j=30, n=10
In main: i=0, j=1, n=20
```

Question 2 [6 Marks]

The *ceiling* of a `double` value is the smallest integer greater than or equal to it. For example, the ceiling of `3.12` is 4. Write a complete C function `ceiling`, the prototype of which is given below, that receives a `double` parameter and returns its ceiling. You are not allowed to use any math functions.

Solution:

```
int ceiling(double num)
{
 int result;
 if (num > 0)
    result = num + 1;
 else
    result = num;
 return result;
}
```

Question 3 [6 Marks]

Write a few C statements that take a phone number that is stored in alphabetic form in the string `alpha` and prints the phone number in numeric form. For example, if `alpha` contains the string `"CALLATT"`, your statements should print `2255288`. You may assume that the string `ALPHA` only contains uppercase letters between `’A’` and `’Y’`, inclusive.

Solution:

```
int i;
for ( i = 0; i < strlen(alpha); i++)
{
    char letter = alpha[i];
    int digit = (letter-’A’)/3 + 2;
    printf("%d", digit);
}
printf("\n");
```

Question 4 [6 Marks]

Consider the following C program:

```
#include <stdio.h>

int main(int argc, char* argv[])
{
  int i = 0;
  for (i = 0; i < argc; i++)
  {
    printf("%s %s\n", argv[i], argv[i]+i);
  }
  return 0;
}
```

Given that the program is contained in a file named `prog.c` and is compiled using the `gcc` compiler with the following command line:

```
gcc prog.c -o my_program
```

What output is produced when the program is executed with the following command line:

```
my_program hello world
```

Solution:

```
my_program my_program
hello ello
world rld
```

Question 5 [6 Marks]

Assume that the `date` structure contains three members: `month`, `day`, and `year` (all of type `int`). Write a complete C function `compareDates`, the prototype of which is given below, that returns `-1` if `d1` is an earlier date than `d2`, `1` if `d1` is a later date than `d2`, and `0` if `d1` and `d2` are the same.

Solution:

```
int compareDates(struct date d1, struct date d2)
{
    int result;
    if (d1.year < d2.year)
        result = -1;
    else if (d1.year > d2.year)
        result = 1;
    else
    {
        if (d1.month < d2.month)
            result = -1;
        else if (d1.month > d2.month)
            result = 1;
        else
        {
            if (d1.day < d2.day)
                result = -1;
            else if (d1.day > d2.day)
                result = 1;
            else
                result = 0;
        }
    }
    return result;
}
```

Question 6 [8 Marks]

The first 7 rows of Pascal's triangle are:

```
     1
     1     1
     1     2     1
     1     3     3     1
     1     4     6     4     1
     1     5    10    10     5     1
     1     6    15    20    15     6     1
```

Each interior number in the triangle is the sum of the number directly above it and the number above and to the left of it. For example, the first number 15 in the last row above is computed by adding 10 and 5.

Write a complete C program that computes the first 10 rows of Pascal's triangle and prints it in the format shown above. Store the Pascal's triangle numbers in an array having 10 rows and 10 columns. To get the numbers to appear properly, use the print format `%6d`. The `%6d` format prints an integer in decimal, using a width of at least 6 character positions.

Solution:

```
int main(void)
{
    const int SIZE = 10;
    int matrix[SIZE][SIZE];
    int i, j;

    for (i = 0; i < SIZE; i++)
    {
        matrix[i][0] = 1;
        matrix[i][i] = 1;
        for (j = 1; j < i; j++)
        {
            matrix[i][j] = matrix[i-1][j-1] + matrix[i-1][j];
        }
    }

    for (i = 0; i < SIZE; i++)
    {
        for (j = 0; j <= i; j++)
            printf("%6d",matrix[i][j]);

        printf("\n");
    }
    return 0;
}
```

Question 7 [8 Marks]

Write a recursive C function named `count`, the prototype of which is given below, that returns the number of occurrences of a character `c` in a string `s`. For example,

> `count("Toronto", ’t’)` should return `1`,
> `count("Toronto", ’o’)` should return `3`.

Note: You must make use of recursive functions. No credit will be given for a solution that uses `while-`, `for-` or `do-` loops.

Solution:

```
int count (char s[], const char c)
{
    if (s[0] == ’\0’)
    {
        if (c == ’\0’)
            return 1;
        else
            return 0;
    }
    else
    {
        if (s[0] == c)
            return (1 + count(&s[1], c));  // or replace &s[1] by s+1
        else
            return count(&s[1], c);        // or replace &s[1] by s+1
    }
}
```

Question 8 [8 Marks]

Write a complete C function named `mingle`, the prototype of which is given below, that receives two equal sized character strings (`str1` and `str2`) as parameters. The function must create and return a new string that contains the characters from `str1` and `str2` in an interleaved fashion, beginning with the first character from `str1`. For example, if `str1` were `"ABC"` and `str2` were `"XYZ"` the function must create and return a new string containing `"AXBYCZ"`.

You may use library functions from `string.h` in your solution. You may assume that the arguments are equal sized strings.

Solution:

```
char *mingle(char str1[], char str2[])
{
  char *newString = (char *)malloc(
       sizeof(char)*(strlen(str1) + strlen(str2) + 1));
  int i = 0;

  while (*str1 != '\0')
  {
    newString[i] = *str1;
    i++;
    newString[i] = *str2;
    i++;
    str1++;
    str2++;
  }
  return newString;
}
```

Question 9 [8 Marks]

Write a complete C function named `cocktailSort`, the prototype of which is given below, that receives an integer array `list` and its size `listLength` as its parameters. The function sorts the given array in ascending (nondecreasing) order, following a slight variation of the bubble sort algorithm, called the *Cocktail sort*. Instead of repeatedly passing through the list from bottom to top in each iteration of bubble sort, Cocktail sort passes alternately from bottom to top and then from top to bottom. Similar to bubble sort, the sorting algorithm stops as soon as the array is sorted.

Solution:

```
void cocktailSort(int list[], int listLength)
{
  bool goingUp = true;
  bool sorted = false;
  int bottom = 0, top = listLength - 1, i;

  while (bottom < top && !sorted)
  {
    sorted = true;
    if (goingUp)
    {
      for (i = bottom; i < top; i++)
        if (list[i] > list[i+1])
        {
          int temp = list[i];
          list[i] = list[i+1];
          list[i+1] = temp;
          sorted = false;
        }
      top--;
      goingUp = false;
    }
    else
    {
      for (i = top; i > bottom; i--)
        if (list[i-1] > list[i])
        {
          int temp = list[i-1];
          list[i-1] = list[i];
          list[i] = temp;
          sorted = false;
        }
      bottom++;
      goingUp = true;
    }
    }
}
```

Question 10 [8 Marks]

Write a complete C function named `splitList`, the prototype of which is given below. The function receives two parameters: a pointer to a pointer to the head of a linked list (called `headPtr`), and an integer `s`. The function must split the linked list parameter's argument into two linked lists, and return a pointer to the head of the second linked list. The linked list must be split such that the head node of the second linked list contains the value `s`. For example, if the original linked list contained nodes with values `1, 5, 7, 9`, and `s` were `7`, then the original linked list would be modified by `splitList` to contain nodes with values `1, 5`. A pointer to a second linked list containing nodes with values `7, 9` would be returned by `splitList`.

If the original linked list does not contain `s`, the function should not alter the original list and should return `NULL`. You may assume that items in the original list are unique (no duplicates).

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

Solution:

```
Node *splitList(NodePointer *headPtr, int s)
{
  Node* head = *headPtr;
  if (head == NULL)
    return NULL;
  if (head -> info == s)
  {
    *headPtr = NULL;
    return head;
  }

  Node * prev = head;
  Node * curr = head -> link;

  while ((curr != NULL) && (curr -> info != s))
  {
    prev = curr;
    curr = curr -> link;
  }

  prev -> link = NULL;
  return curr;
}
```

Question 11 [8 Marks]

Write a complete C function named `deleteRange`, the prototype of which is given below. The function receives three parameters: a pointer to the head of a linked list (called `head`), and two integers `low` and `high`. The function alters the linked list that is passed in as an argument by deleting all of the nodes having values in between low and high (inclusive). The function returns a pointer to the head of the altered list. For example, if the linked list `head` contains nodes with values `1, 5, 2, 13, 5, 19, 8`, `low = 5` and `high = 13`, then the returned linked list would be modified by `deleteRange` to contain nodes with values `1, 2, 19`.

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

Solution:

```
Node *deleteRange(Node *head, int low, int high)
{
  while (head != NULL && head -> info >= low && head -> info <= high)
  {
    Node *temp = head;
    head = head -> link;
    free(temp);
  }
  if (head != NULL)
  {
    Node *current = head -> link;
    Node *previous = head;
    while (current != NULL)
    {
      if (current -> info >= low && current -> info <= high)
      {
        previous -> link = current -> link;
        free(current);
        current = previous -> link;
      }
      else
        previous = current;
        current = current -> link;
    }
  }
  return head;
}
```

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