# 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 17 pages, including this one. There is no need to fill all the available space on each page, as some solutions may be shorter.

**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 a *linked list* instead of an *array* to store a list of values in a computer program.

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

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

**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,` respectively, and the remaining elements are initialized to `0`.

**1.4 [3 marks]** A student in APS 105 has written the following function to rotate the values of 3 `double` variables. When `x` is 1.0, `y` is 5.0 and `z` is 3.5, the student wants the function call `rotate3Nums(x,y,z)` to change the values of `x`, `y` and `z` to be 5.0, 3.5 and 1.0, respectively. As written, the function call does not produce the correct result. Fix the errors in the function. After fixing the errors, how would you call `rotate3Nums` to rotate `x`, `y`, and `z`?

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

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

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

Record a;
Record *p = &a;
```

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

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

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

**1.7 [4 Marks]** Will the following complete C program run successfully? If your answer is yes, show the output from running this program. If your answer is no, explain the reason.

```
#include <stdio.h>

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

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

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

```
#include <stdio.h>

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;
}
```

**Question 2** [6 Marks]

Consider the following C program:

```
#include <stdio.h>

int main (int argc, char* argv[])
{
  int i;
  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
```

**Question 3** [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 C function named `ceiling`, the prototype of which is given below, that has a single type `double` parameter named `num` and returns its ceiling. You are not allowed to use any math functions.

```
int ceiling (double num)
{




}
```

**Question 4** [6 Marks]

Assume that the `date` structure contains three members: `month`, `day`, and `year` (all of type `int`). Write a C function named `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. Assume `d1` and `d2` contain valid dates

```
int compareDates (struct date d1, struct date d2)
{









}
```

**Question 5** [6 Marks]

Write a C function named `countDistinct`, the prototype of which is given below, that has two parameters. The parameter `list` is an integer array already sorted in ascending (nondecreasing) order, and the parameter `listLength` gives the number of elements in `list`. The function counts and returns the number of distinct elements in `list`.

For example, if `list` is `{1, 1, 3, 3, 3, 3, 6, 8, 8, 9, 9, 9, 9, 9}`, the function returns `5`, as `list` has 5 distinct elements.

```
int countDistinct (int list[], int listLength)
{









}
```

**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 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.

```
int main (void)
{









}
```

**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.

```
int count (char s[], char c)
{









}
```

**Question 8** [8 Marks]

Write a C function named `mingle`, the prototype of which is given below, that has two parameters named `str1` and `str2`. Assume that both strings have the same length. The function creates and returns 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 creates and returns 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.

```
char *mingle (char str1[], char str2[])
{









}
```

**Question 9** [8 Marks]

Write a C function named `cocktailSort`, the prototype of which is given below, that has two parameters. The parameter `list` is an integer array and the parameter `listLength` gives the number of elements in `list`. 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 as 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 it is detected that the array is sorted.

```
void cocktailSort (int list[], int listLength)
{









}
```

**Question 10** [8 Marks]

Write a 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 given linked list into two linked lists, and return a pointer to the head of the second linked list. The first linked list contains the nodes from the original linked list, in their original order, up to but not including the node containing the value `s`. The second list contains the node containing the value `s` and the nodes from the original list that follow this node, in their original order. The first linked list is pointed to by `*headPtr`. A pointer to the head node of the second linked list is returned by the `splitList` function.

For example, if the original linked list contained nodes with values `5, 1, 9, 7`, and `s` were `9`, then the original linked list would be modified by `splitList` to contain nodes with values `5, 1`. A pointer to a second linked list containing nodes with values `9, 7` would be returned by `splitList`.

No new nodes should be created in your solution. 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;

NodePointer splitList (NodePointer *headPtr, int s)
{
```

*This page has been left blank intentionally. You may use it for your answer to Question 10.*

```
}
```

**Question 11** [8 Marks]

Write a 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 `deleteRange` function returns a linked list containing nodes with values `1, 2, 19`. No new nodes should be created in your solution.

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

Node *deleteRange (Node *head, int low, int high)
{









}
```

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