UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

APS 105 — Computer Fundamentals
Final Examination
December 15, 2008
9:30 a.m. – 12:00 p.m.

Examiners: Jason Anderson, Tom Fairgrieve, Baochun 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. Please indicate clearly which question(s) you are answering.

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

The examination has 19 pages, including this one.

Circle your lecture section:

L0101	or	**L0102**	or	**L0103**	or	**L0104**
Fairgrieve (Monday 2pm)		Anderson		Fairgrieve (Monday 11am)		Li

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 declares a 100 element type `int` array named `list` and initializes all of the elements to the value 0.

Solution: `int list[100] = {0};`

1.2 [3 Marks] Write a single C statement that converts the character pointed to by a character pointer `p` (declared as `char *p`) from lower case to upper case, assuming that it is a lower case letter in the alphabet before the statement is executed. C library functions are not allowed in your solution to this question.

Solution:

```
*p = *p - 'a' + 'A';
```

1.3 [3 Marks] The following C code uses the `&&` (and) logical operator. Using *de Morgan's law*, re-write the code so that it uses the `||` (or) and `!` (not) logical operators instead of the `&&` operator.

```
if ((j > 3) && (k > 2))
  q++;
```

Solution:

```
if (!((j <= 3) || (k <= 2)))
  q++;
```

1.4 [3 marks] The following function is supposed to create an identical copy of the string in the array `p`:

```
char *duplicate (const char p[])
{
    char *q;

    strcpy(q,p);
    return q;
}
```

The function will not work correctly as given. Modify one statement in the function to make it work correctly.

Solution: `char *q = (char *) malloc(sizeof(char) * (strlen(p) + 1));`

1.5 [4 Marks] State whether the following relational expressions are true or false:

Solution:

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

1.6 [4 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.

Solution:

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

1.7 [4 Marks] Write a single C statement that determines an *odd* random integer between -99 and 49 inclusive, and then assigns it to an `int` variable `r` that has already been declared.

Solution:

```
r = rand() % 75 * 2 - 99;
```

1.8 [4 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.

Solution: 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.

Question 2 [6 Marks]

Write the output from an execution of the following C program in the box below.

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

void printPattern (char *s)
{
  int j = strlen(s);
  int k, i;

  for (k = 0; k < j; k++)
  {
    printf("%s", s+k);
    for (i = j - 1; i >= k; i--)
      printf("%c",*(s+i));
    printf("\n");
  }
}

int main (void)
{
  char *t = "aps105";
  printPattern(t);
}
```

Output:

Solution:

```
aps105501spa
ps105501sp
s105501s
105501
0550
55
```

Question 3 [6 Marks]

Write the output from an execution of the following C program in the box below.

```
#include <stdio.h>

void enigma (int n)
{
  if (n != 0)
  {
    enigma(n/2);
    printf("%d", n%2);
  }
}

int main (void)
{
  enigma(13);
  printf("\n");
  return 0;
}
```

Output:

Solution: `1101`
(the program prints the binary representation of the argument)

Question 4 [6 Marks]

In the English alphabet, each letter is classified as being either a vowel or a consonant. In English, the vowels are the letters `a`, `e`, `i`, `o`, `u` and `y`.

Write a C function named `isVowel (char ch)` that returns `true` if `ch` stores a lower-case English vowel and `false` otherwise. Then write a C function named `isConsonant (char ch)` that returns `true` if `ch` stores a lower-case English consonant and `false` otherwise.

In both functions, assume that `ch` holds a lower-case letter from the English alphabet.

Write the functions in the appropriate spaces below.

```
bool isVowel (char ch)
{


Solution:
        return ( ( ch == 'a' ) || ( ch == 'e' ) || ( ch == 'i' ) ||
                 ( ch == 'o' ) || ( ch == 'u' ) || ( ch == 'y' ) );

// 3 marks for a correct working solution

}

bool isConsonant (char ch)
{

Solution:

  return ( !isVowel( ch ) );
  // or
  // return ( ( ch != 'a' ) && ( ch != 'e' ) && ( ch != 'i' ) &&
  //          ( ch != 'o' ) && ( ch != 'u' ) && ( ch != 'y' ) )
  //
  // or
  //  return !( ( ch == 'a' ) || ( ch == 'e' ) || ( ch == 'i' ) ||
  //            ( ch == 'o' ) || ( ch == 'u' ) || ( ch == 'y' ) )
  //
// 2 marks for doing it one of these smart ways
// 1 mark for testing each of the 20 consonants

}
```

Question 5 [6 Marks]

Complete the definition of the C function `maxCount` whose prototype is shown below. The function should return the number of times that the global maximum appears in the array `list`. The global maximum is defined to be the number in the array with the largest value. For example, in the list $3, 9, 7, 5, 9, 8, 2, 4, 9,$ the global maximum is 9 and it appears 3 times. The `maxCount` function returns 3 in this case. Assume that `list` has at least one element. The number of elements in the array is given in the variable `length`.

Solution:

```
int maxCount (int list[], int length)
{
  int maximum = list[0], count = 0;

  int i;
  for (i = 1; i < length; i ++)
    if (maximum < list[i])
      maximum = list[i];
  for (i = 0; i < length; i ++)
    if (maximum == list[i])
      count ++;

  return count;
}
```

Question 6 [8 Marks]

A Mersenne number is a positive integer that is one less than a power of two. For example, the n^{th} Mersenne number, M_n, is:

$$M_n = 2^n - 1$$

A prime number is a positive integer that has exactly two distinct positive integer divisors, namely 1 and itself. The number 1 is, by definition, not a prime number.

A Mersenne prime is a Mersenne number that is prime. The number 3 is the smallest Mersenne prime, because 3 is a prime number, and $2^2 - 1 = 3$. As of October 2008, less than 50 Mersenne primes are known. Write a complete C program that finds and prints the 5 smallest Mersenne primes, each by itself on a line. You may use any of the C math library functions in your solution.

Solution:

```
int main (void)
{
  int n = 2;
  int count = 0;

  while (count < 5)
  {
    int mersenneNum = pow(2, n);
    mersenneNum--;

    bool isPrime = true;
    int j;
    for (j = 2; (j < mersenneNum) && isPrime; j++)
    {
      if (mersenneNum % j == 0)
        isPrime = false;
    }
    if (isPrime)
    {
      count++;
      printf("%d\n",mersenneNum);
    }
    n++;
  }
}
```

Question 7 [8 Marks]

The following C function sorts an array named `list` of `int` values into ascending (nondecreasing) order. The function parameter `listLength` gives the number of elements in the array to be sorted. Observe that the function continues to execute even if the initial array is in sorted order or if the array becomes sorted part way through the function's execution. Re-write the `sort` function, modifying it so that the function finishes the sorting process as soon as it is discovered that the array is in sorted order.

Note: You must modify the sorting algorithm below and not use a different sorting algorithm.

```
void sort (int list[], int listLength)
{
  int bottom;
  for (bottom = 0; bottom < listLength - 1; bottom++)
  {
    int smallestLoc = bottom;
    int i;

    for (i = bottom + 1; i < listLength; i++)
    {
      if (list[i] < list[smallestLoc])
        smallestLoc = i;
    }

    int temp = list[bottom];
    list[bottom] = list[smallestLoc];
    list[smallestLoc] = temp;
  }
}
```

Solution:

```
void sort (int list[], int listLength)
{
  bool sorted = false;
  int bottom;
  for (bottom = 0; bottom < listLength - 1 && !sorted; bottom++)
  {
    int smallestLoc = bottom;
    int i;
    sorted = true;

    for (i = bottom + 1; i < listLength; i++)
    {
      if (list[i] < list[smallestLoc])
```

```
        smallestLoc = i;
      if (list[i] < list[i-1])
        sorted = false;
    }

    int temp = list[bottom];
    list[bottom] = list[smallestLoc];
    list[smallestLoc] = temp;
  }
}
```

Question 8 [8 Marks]

A degree n-polynomial of the form:

$$p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_ix^i + \cdots + a_nx^n$$

can be evaluated efficiently using a technique known as *Horner's rule*. Horner's rule avoids computing the numerous x^i values explicitly, by rewriting and evaluating $p(x)$ in the following form:

$$p(x) = a_0 + x \times (a_1 + x \times (a_2 + x \times (\cdots (a_i + x \times (\cdots (a_{n-1} + x \times (a_n)) \cdots)) \cdots)))$$

Complete the definition of the function `horner` with the following prototype:

```
double horner (double a[], int n, double x)
```

so that it uses Horner's rule to evaluate a polynomial. The first parameter is an array of coefficients of the polynomial, the second is the degree of the polynomial, and the third is the value of x at which the polynomial is to be evaluated. The array `a` has $n + 1$ elements, corresponding to $a_0, a_1, \ldots, a_n$.

Your solution must be **recursive**. A non-recursive solution will earn no marks. You may use a helper function in your solution.

Solution:

```
double horner (double a[], int n, double x)
{
  return hornerHelper (a, n, x, 0);
}

double hornerHelper (double a[], int n, double x, int i)
{
  if (i == n)
    return a[i];
  else
    return a[i] + x * hornerHelper(a, n, x, i + 1);
}
```

Question 9 [8 Marks]

For data compression and other purposes, we often need to look for runs (or sequences) of repeated characters in data. For example, in the string

`abbbbccdeeeeef`

there is a run of 4 `b`'s, a run of 2 `c`'s and a run of 5 `e`'s. We also consider that the `a`, the `d`, and the `f` are all runs of length 1.

Write a C function named `compressString` that, given an input character string, prints a possibly compressed representation of the input character string. The new representation is determined by counting the lengths of runs of a single character, as demonstrated below:

- Given the string `"abbbbccdeeeeef"`, `compressString` prints `1a4b2c1d5e1f` followed by `'\n'`.
- Given the string `"a"`, `compressString` prints `1a` followed by `'\n'`.
- Given the string `"zzzzzzzzzzzzzzzzzzzzzzzzzz"`, `compressString` prints `26z` followed by `'\n'`.
- Given the string `"xxbbbxx"`, `compressString` prints `2x3b2x` followed by `'\n'`.
- Given the string `"aaAa"`, `compressString` prints `2a1A1a` followed by `'\n'`. (Upper-case and lower-case letters are different.)
- Given the string `""`, `compressString` only prints `'\n'`.

Write your `compressString` function below. State any assumptions that you make.

Solution:

```
void compressString (const char strIn[])
{
  // initialize processing of input string
  int inLength = strlen(strIn);
  int i = 0;

  // look at each character in input string
  while (i < inLength)
  {
    // determine run length of current character
    char letter = strIn[i];
    int r = 1;
    while ((i+r) < inLength && letter == strIn[i+r])
      r++;

    // print information for current run
```

```
    printf("%d%c", r, letter);

    i = i + r;
  }
  printf("\n");
}
```

Question 10 [8 Marks]

Complete the definition of a non-recursive function called `copy`, so that it returns an exact *copy* of the linked list that is pointed to by the pointer variable `head`, which is given as a parameter. Assume that the `Node` structure and the `newNode` function have already been defined as follows.

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

NodePointer newNode (int item, NodePointer next)
{
  NodePointer p = (NodePointer) malloc(sizeof(Node));
  if (p != NULL)
  {
    p->info = item;
    p->link = next;
  }
  return p;
}
```

Solution:

```
// First alternative
NodePointer copy (NodePointer head)
{
  NodePointer newList = NULL;

  if (head != NULL)
  {
    NodePointer source = head;
    NodePointer *dest = &newList;

    while (source != NULL)
    {
      *dest = newNode(source -> info, NULL);
      source = source -> link;
      dest = &((*dest) -> link);
    }
  }
  return newList;
}

// Second alternative
NodePointer copy (NodePointer head)
{
  NodePointer newList = NULL;

  if (head != NULL)
  {
    NodePointer source = head;

    newList = newNode(source -> info, NULL);
    NodePointer dest = newList;

    while (source -> link != NULL)
    {
      source = source -> link;
      dest -> link = newNode(source -> info, NULL);
      dest = dest -> link;
    }
  }
  return newList;
}
```

Question 11 [8 Marks]

In this question, you are to write a C function that joins together two sorted linked lists of type `Node` structures, to produce a sorted linked list that contains the union of the two lists.

Assume that both of the given linked lists are sorted in increasing order and that both of the linked lists do not contain duplicate values. However the same value might appear in nodes in both lists, in which case only one of the nodes should be placed in the combined list and the other node should be deallocated using the `free` function.

The parameters to the `join` function are two `NodePointer` variables. The first parameter points to the first node in the first list, while the second parameter points to the first node in the second list. The `join` function returns a pointer to the combined list.

For example, if the first list contained nodes storing the numbers $0, 1, 3, 4, 5, 6$ and the second list contained nodes storing the numbers $2, 4, 6, 8, 10, 12, 13$, the list returned by the `join` function would contain nodes storing the numbers $0, 1, 2, 3, 4, 5, 6, 8, 10, 12, 13$.

No new nodes should be created in your solution. If both lists are empty, the `join` function should return an empty list.

Solution:

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

NodePointer join (NodePointer head1, NodePointer head2)
{

    NodePointer newHead = NULL;    // head pointer to the joined list
    NodePointer newTail = NULL;    // pointer to the last node in joined list

    while ( head1 != NULL && head2 != NULL )
    {
        if ( (head1)->info < (head2)->info )
        {   // place the front node from list 1 in to the joined list
            if (newHead == NULL)
                newHead = head1;
            else
                newTail->link = head1;
            newTail = head1;
            head1 = head1->link;
        }
        else if ( (head2)->info < (head1)->info )
```

```
        {   // place the front node from list 2 in to the joined list
            if (newHead == NULL)
                newHead = head2;
            else
                newTail->link = head2;
            newTail = head2;
            head2 = head2->link;
        }
        else
        {   // add the front node from list 1 to the joined list
            // delete the front node from list 2
            if (newHead == NULL)
                newHead = head1;
            else
                newTail->link = head1;
            newTail = head1;
            head1 = head1->link;
            // dealocate the duplicate
            NodePointer temp = head2;
            head2 = head2->link;
            free(temp);
        }
    }

    // insert the single list that remains from either
    // list 1 or list 2 into the joined list
    if (head1 != NULL)
    {
        newTail->link = head1;
    }
    else
    {
        newTail->link = head2;
    }

    return newHead;
}
```

This page has been left blank intentionally. You may use it for your answer to Question 11.

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