UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

APS 105S — Computer Fundamentals
Final Examination
April 24, 2008
9:30 a.m. – 12:00 p.m.

Examiner: Baochun Li

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

Calculator Type 4: No calculators are allowed.

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.

The test has 14 pages, including this one.

The marks allocated to the questions, out of a total of 100, are shown in the question headings.

Name: _____

Student Number: _____

MARKS

1	2	3	4	5	6	7	8	9	10	Total
/20	/5	/5	/10	/10	/10	/10	/10	/10	/10	/100

Question 1 [20 Marks]

Answer each of the following questions with a short and concise solution.

[2 Marks] Write $\sqrt{\ln(x^y) + 4.5}$ as a C expression.

[2 Marks] How many asterisks will be printed by the following code?

```
for(int i = 3; i > -4; i-=2);
  printf("*");
```

[2 Marks] What does *lazy evaluation* mean? Please explain the concept with one example.

[3 Marks] Write a C statement that will assign to the `int` variable `choice` a random number from the set $\{25, 50, 75, 100\}$.

[3 Marks] Write **two lines** of C program to implement the following `stringCopy` function, which copies from the `source` string to the `destination` string, both in the parameters. You will *not* receive partial marks if you have more than two lines in your solution.

```
void stringCopy(char * destination, const char * source)
{




}
```

[4 Marks] What is the output of the following C program?

```
char *p = "Sample";
printf("%s\n",p+2);
printf("%c\n",*p);
printf("%c\n",*(p+2));
printf("%c\n",*p+2));
```

[4 Marks] What is the output of the following C program if the following command line is entered?

```
stuff foo bar
```

```
#include <stdio.h>
int main (int argc, char *argv[])
{
  int i;
  for (i = 0; i < argc; i ++)
    printf("%s ",*argv + i);
  printf("\n");
  for (i = 0; i < argc; i ++)
    printf("%s ",*(argv + i));
  printf("\n");
  for (i = 0; i < argc; i ++)
    printf("%s ",*(argv + i) + i);
  printf("\n");
  for (i = 0; i < argc; i ++)
    printf("%c ",*(*(argv + i) + i));
  printf("\n");
}
```

Question 2 [5 Marks]

Write a function `countBlank` that returns the number of blank spaces (`' '`) in a given string `s` as a parameter. For example, if the given string is `"Let's have a rest"`, the function returns 3, as the string contains three blank spaces.

```
int countBlank(char * s)
```

Question 3 [5 Marks]

The number 6 is said to be a *perfect number* because it is equal to the sum of all its exact *divisors* (other than itself).

$6 = 1 + 2 + 3$

In this example, 1, 2, and 3 are all exact divisors of 6 (there is no remainder when 6 is divided by these numbers), and are the only exact divisors of 6, except 6 itself.

Write a complete C program that finds and prints three smallest perfect numbers, each on its own line.

Question 4 [10 Marks]

Complete the definition of the function `nearest` so that it returns the *index* of the value in the integer array `list` that is nearest to the value of `item`. The value of the parameter `size` is the number of items in the array `list`.

For example, if `list` is `{50,40,21,13,35}` and `item` is `44` then the function should return `1`.

You can assume that there will *not* be two integers in the array that are equally near to the value of `item`.

```
int nearest(int list[], int size, int item)
```

Question 5 [10 Marks]

Complete a **recursive** implementation of the function `printPattern` so that it will print the pattern shown below, with the given number of rows passed in as a parameter. For example, if `printPattern(5)` is called, the function should produce the following output.

```
*****
****
***
**
*
**
***
****
*****
```

Your implementation should **not** use a loop. You may assume that the parameter `rows` is always positive.

Hint: The function `printPattern` will need to call another function (for example `printRow`), which prints the asterisks in each row.

```
void printPattern(int rows)
```

Question 6 [10 Marks]

Complete a **recursive** implementation of the function `search`, which takes a string `s` and a character `c` as its parameters, and returns the index of the first occurrence of the character `c` in the string `s`. For example, if `c` is `'o'` and `s` is `"Toronto"`, then the function `search` returns 1. If the character is not found or the string is an empty string, the function `search` returns `-1`.

```
int search(char * s, char c)
```

Question 7 [10 Marks]

Complete a **recursive** implementation of the function `binarySearch`, which performs *binary search* to look for an integer `item` on an array of integers, sorted in ascending order. The parameter `size` is the number of items in the array. The function returns the *index* of the item if it is found, and returns `-1` if it is not found.

```
int binarySearch(int list[], int size, int item)
```

Question 8 [10 Marks]

Write a function `bubbleSort` that sorts an array of integers in *descending* order. The array and its size are both given as parameters.

```
void bubbleSort(int list[], int size)
```

Question 9 [10 Marks]

Suppose that linked lists are maintained in the usual way introduced in the lectures, with node structures defined as follows:

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

Complete the definition of the function `deleteFromFront`. The function should take the linked list as a parameter, and delete the first node in the linked list. The function should not have any return values, *i.e.*, it has a return type of `void`. If the list is already empty, the function should do nothing.

Question 10 [10 Marks]

Suppose that linked lists are maintained in the usual way introduced in the lectures, with node structures defined as follows:

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

Complete the definition of the function `addAtEnd`. The function should take the linked list as a parameter, and add a new node to the end (or *tail*) of the linked list. The `info` field of the new node is assigned the integer `item`, given as a parameter. The function should not have any return values, *i.e.*, it has a return type of `void`.

```
void addAtEnd(Node * head, int item)
```

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