UNIVERSITY OF TORONTO

Faculty of Applied Science and Engineering

APS 105H1S

Final Examination

Instructor: J. Deber

April 25, 2012 2:00 p.m. – 4:30 p.m.

Duration — 2.5 hours

Exam Type: A (no external aids) / Calculator Type: 4 (no electronic devices)

Student Number:	
Last (Family) Name(s):	
First (Given) Name(s):	

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This final examination consists of 9 questions on 20 pages (including this one). *When you receive the signal to start, please make sure that your copy of the examination is complete.*

You must write your programs in the C programming language, and you may make use of any C features or libraries that we have covered. Standard assignment style rules (including a ban on `break`, `continue`, and certain string functions) apply, unless indicated otherwise. You may make use of C99 features.

You do not have to provide any `#include` statements.

Comments are not required, although we will be marking style on coding questions.

Unless indicated otherwise, you do not have to handle abnormal input (e.g., a user entering a letter when a number is expected).

Unless indicated otherwise, you may write helper functions. Helper functions can be reused in any part of a multi-part question.

If you use any space for rough work, indicate clearly what you want marked.

There is a function reference sheet on page 19.

MARKING GUIDE

# 1:	/ 7
# 2:	/ 5
# 3:	/ 6
# 4:	/ 7
# 5:	/ 9
# 6:	/13
# 7:	/16
# 8:	/ 7
# 9:	/ 5
TOTAL:	/75

Good Luck!

Question 0.

Write your student number in the space provided at the bottom of each odd-numbered page. Failure to do so will result in a 2 mark deduction.

Question 1. [7 MARKS]

Part (a) [4 MARKS]

Write the *pseudocode* for a descending order (largest to smallest) selection sort:

Part (b) [3 MARKS]

Given the following set of `ints`, illustrate the progression of a descending order selection sort. In other words, for each iteration of the sort, clearly indicate what the algorithm is doing, and draw the configuration of the list every time a number is moved.

`9, 49, 23, 55, 31` (initial order)

`55, 49, 31, 23, 9` (final sorted order)

Question 2. [5 MARKS]

Consider the following program:

```
bool isEven(int i)
{
    bool result = (i % 2 == 0);
    if (result)
    {
        printf("%d is even\n", i);
    }
    else
    {
        printf("%d is odd\n", i);
    }

    return result;
}

int f(double d)
{
    d++;

    printf("d: %g\n", d);

    return d / 2;
}

int main(void)
{
    bool b = true;
    int i = 0;
    double d = 3.5;

    for (int j = i; j < f(d); j++, d++)
    {
        if (b || isEven(j))
        {
            printf("j: %d\n", j);
        }
        b = !b;
        i++;
    }

    f(i);
    printf("i: %d\n", i);

    return 0;
}
```

Write the output of this program. You may use the space provided to the right of the program.

Question 3. [6 MARKS]

The following code has several mistakes in it. These mistakes include syntax errors, logic problems (i.e., bugs), and style issues.

One mistake (misspelling int) is already circled.

```
1
in main(void)
{
    int size;
    printf("Enter size: \n");
    scanf("%d", size);

    int list[] = malloc(size + sizeof(int *));

    for (i == 0; i < size; i++)
    {
        list[i] = i
    }

    int Increase;
    printf("Enter increase: );
    scanf("%d", &Increase);

    int newSize = size + Increase;

    realloc(list, newSize);

    for (i = size + 1; i < newSize; i++)
    {
        list[i] = 0;
    }

    for (i = 0; i < newSize; i++)
    {
        printf("%g\n", list[i]);
    }

    if (list != '\0')
    {
        free(list, newSize);
    }
    free(Increase, 1);
}
```

Part (a) [1 MARK]

What is this code trying to do?

Part (b) [5 MARKS]

For each mistake, circle the relevant section of code, and write a number next to the circle. Then, in the space provided below, briefly describe each mistake and provide a possible correction. One mistake (misspelling `int`) is already circled, numbered, and described to demonstrate the format you should use.

There are many mistakes in this program; to earn full marks, you will need to correctly identify (and provide the appropriate correction for) 10 mistakes in addition to the one already provided below. Note: if the same error appears in more than one location, repeated occurrences can not be counted as separate errors (e.g., if there was a second function that also misspelled its return type of `int`, both misspelled return types would be considered a single error, and not a pair of errors). However, a single line can contain more than one mistake, if the errors are distinct (e.g., if the declaration read `in main()`, the absence of `void` could be counted as a second error).

1. Return type of `main()` is misspelled `in` rather than `int`.

Question 4. [7 MARKS]

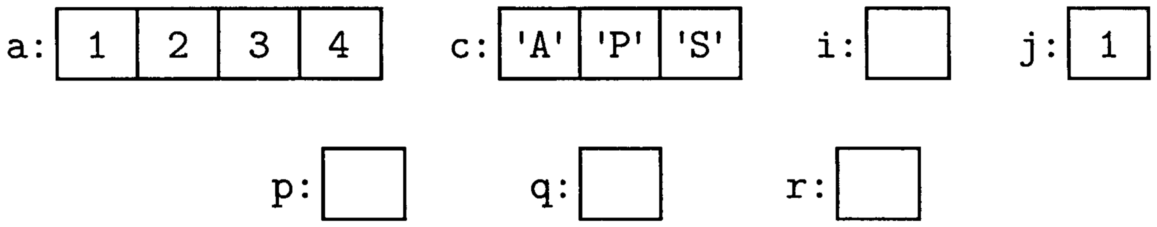
Consider the following code:

```
int i;
int j = 1;
char c[] = {'A', 'P', 'S'};
int a[] = {1, 2, 3, 4};
int *p;
int *q;
char *r;

/* Initial memory snapshot taken here. */

r = &c[2];
p = a;
q = a + 3;
*r = c[*p];
i = *(a + 1);
*q = *p;
p[j] = *(q - 1);
```

A snapshot of memory after the first seven lines have executed is pictured below:



For each of the statements listed on the next page, fill in the values that each variable would have *after* that statement has executed. Pointers should be drawn as arrows.

Note that the statements are cumulative; changes to variables that are made on one line persist to the next statement.

`r = &c[2];`	a: c: i: j: p: q: r:
`p = a;`	a: c: i: j: p: q: r:
`q = a + 3;`	a: c: i: j: p: q: r:
`*r = c[*p];`	a: c: i: j: p: q: r:
`i = *(a + 1);`	a: c: i: j: p: q: r:
`*q = *p;`	a: c: i: j: p: q: r:
`p[j] = *(q - 1);`	a: c: i: j: p: q: r:

Question 5. [9 MARKS]

Here is the documentation and prototype for a function called `generateUniqueRandoms()`:

```
/* Allocates a new array of ints of size 'n', and fills it with 'n' unique
 * pseudorandom numbers between 0 and 'max' (inclusive).
 * No number appears more than once in the array.
 * Seeds the pseudorandom number generator with 'seed' before generating any numbers.
 * 'max' must be >= 'n', and both 'max' and 'n' must be >= 0.
 * The resulting array must subsequently be destroyed by a call to free(). */
int *generateUniqueRandoms(int n, int max, int seed);
```

For example, if a call was made to `generateUniqueRandoms(5, 8, 10)`, it might return an array containing 6, 3, 1, 0, and 8 (i.e., five unique pseudorandom numbers between 0 and 8, generated using a seed of 10).

Part (a) [3 MARKS]

Assume you have `int`s named `numValues`, `maxValue`, and `seed` containing appropriate and valid values. Write a code fragment that generates `numValues` pseudorandom numbers between 0 and `maxValue`, prints them out (one per line), and performs any necessary cleanup. Your code fragment should call `generateUniqueRandoms()`.

Part (b) [6 MARKS]

Write `generateUniqueRandoms()`:

```
int *generateUniqueRandoms(int n, int max, int seed)
{
```

Question 6. [13 MARKS]

For this question, you will make use of linked lists that have been built out of the following `struct`:

```
typedef struct node
{
    double score;
    struct node *next;
} Node;
```

Part (a) [2 MARKS]

Draw the linked list containing the following scores: `4.2`, `5.8`, `8.6`, `3.1`

Part (b) [5 MARKS]

Write `findLargestIterative()`, a function that takes a linked list as a parameter, and returns the largest score in the list. For example, if given the list from Part (a), the function would return `8.6`. You can assume that the linked list always contains at least one `Node`. You must write this function iteratively (i.e., using a loop); any solution that uses recursion will receive a mark of 0.

Part (c) [6 MARKS]

Write `findLargestRecursive()`, which does the same thing as `findLargestIterative()`. `findLargestRecursive()` must be written recursively; any solution that uses a loop will receive a mark of 0.

Question 7. [16 MARKS]

Part (a) [2 MARKS]

Write a definition of a `struct` that can be used to store information about a finishing time in a race. The `struct` should be able to store a racers's first name, last name, bib number (a positive whole number between 1 and 1000), and finishing time (a whole number of seconds). You are free to use whatever data types you think are appropriate for each variable. The `struct` should be `typedef`'ed with the name `Racer`.

Part (b) [3 MARKS]

Write a function called `createRacer()` that takes four parameters (the racers's first name, last name, bib number, and finishing time), creates a new `Racer`, sets its fields to the specified values, and returns a pointer to it.

Part (c) [2 MARKS]

Write a `void` function called `destroyRacer()` that takes a pointer to a `Racer` as a parameter, and performs any tasks necessary to cleanup that `struct`.

Part (d) [5 MARKS]

Assume that a list of racers is maintained as an unsorted array of pointers to `Racer`. The array can contain empty spaces, which are denoted by `NULL` pointers.

Write the following function:

```
/* Adds 'racer' to 'racerList', which is an 'n' element array of pointers to Racers.
 * 'racerList' must contain at least one empty space; if there is more than one empty
 * space in the array, 'racer' is added to the first available space.
 * Returns the index where 'racer' was added. */
int addToRacerList(Racer *racer, Racer *racerList[], int n)
{
```

Part (e) [4 MARKS]

Assume that you have a list of racers stored in an array of `Racer *` of size `listSize` named `racerList`. Using the functions defined above, write a code fragment that creates a new racer named Homer Simpson with a bib number of 742 and a finishing time of 239 seconds, and adds him to `racerList`. Next, consider the array index immediately before the position where Homer was added. If it's a valid index and there is a racer stored in that position, print out their first and last name, and then delete them from the list.

Question 8. [7 MARKS]

Consider the following function:

```
/* Creates and returns a new string that is made up of 'n' copies of the string 's'.
 * Each copy of 's' is seperated by the character 'seperator'.  'n' must be >= 1.
 * The resulting string must subsequently be destroyed by a call to free(). */
char *duplicateString(const char s[], int n, char seperator);
```

For example, a call to `duplicateString("ABC", 2, '*')` would result in the string `"ABC*ABC"`, and a call to `duplicateString("105", 3, '!')` would result in the string `"105!105!105"`. Note that the final repetition of `'s'` is not followed by a separator character. In other words, the string in the first example is `"ABC*ABC"`, not `"ABC*ABC*"`.

Write the function:

```
char *duplicateString(const char s[], int n, char seperator)
{
```

Question 9. [5 MARKS]

Based on your experience programming in C this semester, if you were given the opportunity to change one thing about the C programming language, what would it be? This could include adding or removing features, changing how the language behaves, modifying the syntax, changing the standard library, *etc.*

Justify your answer, and provide at least one example that illustrates the existing problem with the language that you are fixing, and describe how your change would improve the situation. Be specific – a response of "it should be easier" will not earn many marks.

[You may use these pages for either rough work or as additional space for your solutions to other questions. These pages will **not** *be marked, unless you clearly indicate the part of your work that you want us to mark.]*

[You may use these pages for either rough work or as additional space for your solutions to other questions. These pages will **not** *be marked, unless you clearly indicate the part of your work that you want us to mark.]*

[You may use these pages for either rough work or as additional space for your solutions to other questions. These pages will **not** *be marked, unless you clearly indicate the part of your work that you want us to mark.]*

Function Reference Sheet

(Note: you are welcome to use functions that are not listed on this page, unless they have been expressly prohibited)

```
/* Copies at most 'n' characters from 'src' (including the terminating '\0' character) to 'dest'. */
strncpy(char *dest, const char *src, size_t n);

/* Appends at most 'n' characters from 'src' to 'dest', and then adds a terminating '\0' character. */
strncat(char *dest, const char *src, size_t n);

/* Calculates the length of 's', not including the terminating '\0' character. */
size_t strlen(const char *s);

/* Compares the two strings 's1' and 's2'.  Returns an integer less than, equal to, or greater than zero if 's1' is found
 * to be less than, equal to, or greater than 's2'. */
int strcmp(const char *s1, const char *s2);


/* Allocates 'size' bytes, and returns a pointer to the allocated memory. */
void *malloc(size_t size);

/* Frees the memory space pointed to by 'ptr'.
void free(void *ptr);

/* Changes the size of the memory block pointed to by 'ptr' to 'size' bytes, and returns a pointer to the allocated memory. */
void *realloc(void *ptr, size_t size);


/* Reads in at most 'size' - 1 characters from 'stream', and stores them in 's'.  Reading stops after a newline, which is
 * stored in 's'.  's' will be null terminated. */
fgets(char *s, int size, FILE *stream);
```

```
/* Returns a pseudo-random int between 0 and RAND_MAX. */
int rand(void);

/* Sets the seed of the PRNG to 'seed'. */
void srand(int seed);

/* Returns the cosine of 'x', which is in radians. */
double cos(double x);

/* Returns the sine of 'x', which is in radians. */
double sin(double x);

/* Returns the tangent of 'x', which is in radians. */
double tan(double x);

/* Rounds 'x' to an integer value. */
double rint(double x);

/* Rounds 'x' up to the nearest integer value. */
double ceil(double x);

/* Rounds 'x' down to the nearest integer value. */
double floor(double x);

/* Returns the absolute value of 'x'. */
int abs(int x);

/* Returns the absolute value of 'x'. */
double fabs(double x);

/* Returns the natural logarithm of 'x'. */
double log(double x);

/* Returns e raised to the power of 'x'. */
double exp(double x);

/* Returns 'x' raised to the power of 'y'. */
double pow(double x, double y);

/* Returns the square root of 'x'. */
double sqrt(double x);


/* Checks if 'c' is an alphabetic character. */
int isalpha(int c);

/* Checks if 'c' is a digit. */
int isdigit(int c);

/* Checks if 'c' is an upper case letter. */
int isupper(int c);

/* Checks if 'c' is a lower case letter. */
int islower(int c);

/* Converts 'c' to upper case, if possible. */
int toupper(int c);

/* Converts 'c' to lower case, if possible. */
int tolower(int c);
```