COMP9024 25T1 Data Structures and Algorithms

Week 3 Practical
Analysis of Algorithms

1. (Function growth)

   We would like to observe how various mathematical functions of an input size n grow as n increases.

   Fortuntately, the C standard library provides a group of functions, most of which are defined in the header file `<math.h>`.

   Unfortunately, this does not include a function to compute factorial.

   a. Write a C function `factorial` that computes the factorial of a non-negative integer, n. The function prototype should be:

   ```
   long factorial(int n);
   ```

   Note: `long` is shorthand for `long int`, and allows us to store larger integers than a generic `int`.

   b. Write a program `growth.c` that repeatedly prompts the user to input an integer n. Assuming the number is positive, the program should compute and print each of the following functions, in order:

   i. $f(n) = log_e(n)$
   ii. $f(n) = sqrt(n)$
   iii. $f(n) = n$
   iv. $f(n) = nlog_e(n)$
   v. $f(n) = n^2$
   vi. $f(n) = n^3$
   vii. $f(n) = 2^n$
   viii. $f(n) = n!$
   ix. $f(n) = n^n$

   If the user enters a number that is not strictly positive, then the program should print "Goodbye!" and terminate.

   Here is an example of the program executing, with user input highlighted in bold:

```
prompt$ ./growth
Enter an integer: 1
For n = 1:
log(n)     = 0
sqrt(n)    = 1
n          = 1
n * log(n) = 0
n^2        = 1
n^3        = 1
2^n        = 2
n!         = 1
n^n        = 1
Enter an integer: 4
For n = 4:
log(n)     = 1
```

```
sqrt(n)    = 2
n          = 4
n * log(n) = 6
n^2        = 16
n^3        = 64
2^n        = 16
n!         = 24
n^n        = 256
Enter an integer: -1
Goodbye!
prompt$
```

Hints:

- Wikipedia has an overview of functions in `<math.h>`.
- If a function returns a floating-point result, when printed it should be formatted to zero decimal places.
- Your program won't be tested on any input that would overflow a `long` integer or `double` precision floating-point when computing any of the functions.
- Make sure you `#include <math.h>`.

Note: If compiling with `gcc`, you may need to link the math library using the `-lm` option. For example:

```
prompt$ gcc -Wall -Werror -std=c11 -o growth growth.c -lm
```

c. As the very first line of `growth.c`, insert a comment indicating the smallest integer $n > 0$ such that, for every function $f(n)$ computed in the program, their output values appear in strictly increasing order as printed.

Replacing `?` with the value you find, the first line of `growth.c` should look like:

```
/* n = ? */
```

We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program that corresponds to this exercise. It expects to find a program named* `growth.c` *in the current directory. You can use dryrun as follows:*

```
prompt$ 9024 dryrun growth
```

Answer:

```
/* n = 10 */

#include <stdio.h>
#include <math.h>

long factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}

int main(void) {
```

```
    int n;

    while (1) {
        printf("Enter an integer: ");
        scanf("%d", &n);

        if (n <= 0) {
            printf("Goodbye!\n");
            break;
        }

        printf("For n = %d:\n", n);

        printf("log(n)     = %.0f\n",  log(n)       );
        printf("sqrt(n)    = %.0f\n",  sqrt(n)      );
        printf("n          = %d\n",    n            );
        printf("n * log(n) = %.0f\n",  n * log(n)   );
        printf("n^2        = %d\n",    n * n        );
        printf("n^3        = %d\n",    n * n * n    );
        printf("2^n        = %.0lf\n", pow(2, n)    );
        printf("n!         = %ld\n",   factorial(n) );
        printf("n^n        = %.0lf\n", pow(n, n)    );
    }

    return 0;
}
```

2. (Algorithms and complexity)

   Develop an algorithm to determine if a character array of length *n* encodes a *heterogram*.

   A heterogram is a word, phrase, or sentence in which no letter of the alphabet occurs more than once. For example, "until" is such a word, but "repeat" is not; "the quick fox" is such a phrase, but "the quick brown fox" is not.

   a. Write the algorithm in pseudocode.

   b. Analyse the time complexity of your algorithm.

   c. Implement your algorithm in C as a function

   ```
   bool isHeterogram(char A[])
   ```

   that returns `true` if no letter in string `A` is repeated, and `false` otherwise.

   Hint: The standard library `<stdbool.h>` defines the basic data type `bool` with the two values `true` (internally encoded as `1`) and `false` (= `0`).

   d. Use your solution to Exercise c. to write a program `heterogram.c` that prompts the user to input a word or phrase and checks whether it is a heterogram. Examples of the program executing are:

   ```
   prompt$ ./heterogram
   Enter a word or phrase: until
   "until" is a heterogram
   prompt$ ./heterogram
   Enter a word or phrase: repeat
   "repeat" is not a heterogram
   ```

```
prompt$ ./heterogram
Enter a word or phrase: the quick fox
"the quick fox" is a heterogram
prompt$ ./heterogram
Enter a word or phrase: the quick brown fox
"the quick brown fox" is not a heterogram
```

Hints:

- You should disregard all spaces when checking if the input is a heterogram.
- You may assume that the input consists of spaces and lower case letters only.
- You may assume that the input consists of no more than 63 characters (excluding the terminating `'\0'`).
- You can use the standard I/O library function `scanf("%[^\n]%*c",str)` to read a phrase from the input. This will read every character as long as it is not a newline `'\n'`, and `"%*c"` ensures that the newline is read but discarded.
- You may use the standard library function `strlen(char[])`, defined in `<string.h>`, which computes the length of a string (without counting its terminating `'\0'`-character).

We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program that corresponds to this exercise. It expects to find a program named* `heterogram.c` *in the current directory. You can use dryrun as follows:*

```
prompt$ 9024 dryrun heterogram
```

Note: You are only required to submit your solution to Exercise d.

Answer:

```
isHeterogram(A):
|  Input  array A[1..n] of chars
|  Output true if A is a heterogram, false otherwise
|
|  for i=1 to n-1 do
|  |  for j=i+1 to n do
|  |  |  if A[i]≠' ' and A[i]=A[j] then
|  |  |     return false
|  |  |  end if
|  |  end for
|  end for
|  return true
```

Time complexity analysis: In the worst case, the outer loop is executed $n-1$ times, the inner loop is executed $(n-1)+(n-2)+\ldots+1 = n(n-1)/2$ times, and the operations inside the loop are $O(1)$. Hence, this algorithm takes $O(n^2)$ time. Note, the algorithm could be improved to run in $O(n)$ time, but this is left as an exercise for the reader.

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

#define MAXLEN 64

bool isHeterogram(char A[]) {
    int i;
    int j;
    int len = strlen(A);
```

```
    for (i = 0; i < len-1; i++) {
        for (j = i+1; j < len; j++) {
            if (A[i] != ' ' && A[i] == A[j]) {
                return false;
            }
        }
    }
    return true;
}

int main(void) {
    char str[MAXLEN];

    printf("Enter a word or phrase: ");
    scanf("%[^\n]%*c", str);

    if (isHeterogram(str)) {
        printf("\"%s\" is a heterogram\n", str);
    } else {
        printf("\"%s\" is not a heterogram\n", str);
    }

    return 0;
}
```

Due Date

Tuesday, 11 March, 11:59:59. Late submissions will not be accepted.

Submission

You should submit your files using the following `give` command:

```
prompt$ give cs9024 week3 growth.c heterogram.c
```

Alternatively, you can select the option to "Make Submission" at the top of this page to submit directly through WebCMS3.

Important notes:

- Make sure you spell all filenames correctly.
- You can run `give` multiple times. Only your last submission will be marked.
- Where you're expected to submit multiple files, ensure they form a single submission. If you separate them across multiple submissions, each submission will replace the previous one.
- Whether you submit through the command line or WebCMS3, it is your responsibility to ensure it reports a successful submission. Failure to submit correctly will not be considered as an excuse.
- You cannot obtain marks by e-mailing your code to tutors or lecturers.

Assessment

- Each question is worth 1 mark, for a total of 2 marks.
- Submissions will be auto-marked shortly after the deadline.
- It is important that the output of your program follows exactly the format of the sample output, otherwise auto-marking will result in 0 marks.
- Ensure that your program compiles on a CSE machine with the standard options, i.e. `-Wall -Werror -std=c11`. Programs that do not compile will receive 0 marks.
- Auto-marking will use test cases different to `dryrun`. (Hint: do your own testing in addition to running `dryrun`).

Collection

Once marking is complete you can collect your marked submission using the following command:

```
prompt$ 9024 classrun -collect week3
```

You can also view your marks using the following command:

```
prompt$ 9024 classrun -sturec
```

You can also collect your marked submission directly through WebCMS3 from the "Collect Submission" tab at the top of this page.

Plagiarism

Group submissions will not be allowed. Your programs must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (including submissions for similar assessments in previous years, if applicable) and serious penalties will be applied, including an entry on UNSW's plagiarism register.

- ***Do not copy ideas or code from others***
- ***Do not use a publicly accessible repository or allow anyone to see your code***

Please refer to the on-line sources to help you understand what plagiarism is and how it is dealt with at UNSW:

- Plagiarism and Academic Integrity
- UNSW Plagiarism Policy Statement
- UNSW Plagiarism Procedure