COMP9024 25T1 Data Structures and Algorithms

Week 10 Practical
String Algorithms, Randomised Algorithms

1. (Boyer-Moore algorithm)

   a. Implement a C-function

   ```
   int *lastOccurrence(char *pattern, char *alphabet) { ... }
   ```

   that computes the last-occurrence function for the Boyer-Moore algorithm. The function should return a newly created dynamic array indexed by the numeric codes of the characters in the given alphabet (a non-empty string of ASCII-characters).

   Ensure that your function runs in *O(m+s)* time, where *m* is the size of the pattern and *s* the size of the alphabet.

   Hint: You can obtain the numeric code of a `char c` through type conversion: `(int)c`.

   b. Use your answer to Exercise a. to write a C-program that:

   - prompts the user to input
     - an alphabet (a string),
     - a text (a string),
     - a pattern (a string);
   - computes and outputs the last-occurrence function for the pattern and alphabet, ordered by the numeric codes of the characters;
   - uses the Boyer-Moore algorithm to match the pattern against the text.

   An example of the program executing could be

```
prompt$ ./boyer-moore
Enter alphabet: abcd
Enter text: abacaabadcabacabaabb
Enter pattern: abacab

L[a] = 4
L[b] = 5
L[c] = 3
L[d] = -1

Match found at position 10.
prompt$ ./boyer-moore
Enter alphabet: abcd
Enter text: abacaabadcabacabaabb
Enter pattern: abcd

L[a] = 0
L[b] = 1
L[c] = 2
L[d] = 3

No match.
prompt$
```

Hints:

- You may assume that
  - the pattern and the alphabet have no more than 127 characters;
  - the text has no more than 1023 characters.
- To scan `stdin` for a string with whitespace, such as `"a pattern matching algorithm"`, you can use:

```
#define MAX_TEXT_LENGTH 1024
#define TEXT_FORMAT_STRING "%[^\n]%*c"

char T[MAX_TEXT_LENGTH];
scanf(TEXT_FORMAT_STRING, T);
```

  This will read every character as long as it is not a newline `'\n'`, and `"%*c"` ensures that the newline is read but discarded.

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* `boyer-moore.c` *in the current directory.*

You can use dryrun as follows:

```
prompt$ 9024 dryrun boyer-moore
```

2. (Random numbers)

   Write a C program that generates a random word (consisting of just lower-case letters a–z). When executed, the first command-line argument is the length of the word, and the second argument is the seed used by the random number generator, i.e. by `rand()`.

   An example of the program executing could be

```
prompt$ ./randword 6 2
ebgnha
prompt$
```

   which seeds the random-number generator with 2 and generates a random word of length 6.

   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 file named* `randword.c` *in the current directory.*

   You can use dryrun as follows:

```
prompt$ 9024 dryrun randword
```

   Note: It is important in this exercise that you call the random number generator as specified in the lecture, otherwise the outputs will not match. If you use a different method to generate random numbers, the testcases will 'fail', but your program could still be perfectly correct. The output may also be different on your own computer.

Due Date

Tuesday, 29 April, 11:59:59. Late submissions will not be accepted.

Submission

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

```
prompt$ give cs9024 week10 boyer-moore.c randword.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 has passed.
- 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 week10
```

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