Lab 8

COMP9021, Session 1, 2014

The aim of this lab is to practice further command line arguments and introduce Polish notation for arithmetical notation, as well as practice further the kind of recursion techniques and processing of command-line arguments that are likely to be tested in the practical exam.

1 Polish notation

An arithmetic expression built from a binary operator and two arguments is written in reverse Polish notation if the operator follows the arguments. Hence `2 + 3` is written `2 3 +`, and `(2 + 4) * 3` is written `2 4 + 3 *`. Write a C program that

- takes as command line argument a string built from nonnegative integer constants and the binary operators `+`, `-`, `*` and `/`;
- checks whether that string represents a valid expression in reverse Polish notation;
- in case the previous check is positive, outputs the evaluation of that expression.

Make sure that spaces before of after an operator can be omitted in the expression. For instance:

```
$ a.out '1 4 + 3 2 -'
Syntax error
$ a.out '2 4 + 3 *'
18
$ a.out '2 4+3 *'
18
```

For the first part, use the following method: scan the input with a counter set to 0. When a number is encountered the counter is increased by 1. When an operator is encountered the counter is decreased by 1. The input is correct if the counter is never going down from 1 to 0 and its final value is 1.

For the second part, use the following function prototypes:

- `int evaluate(const char *const expr, int *const m, int *const n);`

  to evaluate expression pointed to by `expr` from position after position pointed to by `m` to position pointed to by `n`.
- `int number(const char *const expr, int *const m, const int *const n);`

  to determines the value of the number in `expr` located from position after position pointed to by `m` to location pointed to by `n`.

2 ☞ Merging two strings into a third one

Say that two strings s_1 and s_2 can be merged into a third string s_3 if s_3 is obtained from s_1 by inserting arbitrarily in s_1 the characters in s_2, respecting their order. For instance, the two strings *ab* and *cd* can be merged into *abcd*, or *cabd*, or *cdab*, or *acbd*, or *acdb*, ..., but not into *adbc* nor into *cbda*.

Write a C program that takes three command line arguments and displays the output as follows.

- If fewer or more than 3 command-line arguments have been provided then the program outputs an error message.
- Otherwise, if no command line argument can be obtained from the other two by merging, then the program outputs that there is no solution.
- Otherwise, the program outputs which of the command line arguments can be obtained from the other two by merging.

Your program should behave as follows:

```
$ a.out ab cd
Incorrect number of command line arguments!
$ a.out ab cd abcd ef
Incorrect number of command line arguments!
$ a.out ab cd abcd
The third string can be obtained by merging the other two.
$ a.out ab cdab cd
The second string can be obtained by merging the other two.
$ a.out acbd cd ab
The first string can be obtained by merging the other two.
$ a.out ab cd adcb
No solution
$ a.out aaaaa a aaaa
The first string can be obtained by merging the other two.
$ a.out aaab abcab aaabcaabb
The third string can be obtained by merging the other two.
$ a.out ??got ?it?go#t## it###
The second string can be obtained by merging the other two.
```

3 ☞ Switching between command-line arguments

Write a program that expects some command line arguments and behaves as follows.

- If no command line is provided then the program prints out

`Provide at least one command line argument!`

and stops.

- If some command line argument does not consist of nonzero digits only, then the program prints out

  `Invalid command line arguments!`

  and stops.

- Otherwise, the program proceeds as follows:
  - It prints the first digit d_1 of the first command line argument, which is considered to be "eaten up."
  - If there are at least d_1 command line arguments and not all digits of the d_1^{st} command line argument have been eaten up, then it prints the first digit d_2 of the d_1^{st} command line argument that has not been eaten up.
  - If there are at least d_2 command line arguments and not all digits of the d_2^{nd} command line argument have been eaten up, then it prints the first first digit d_3 of the d_2^{nd} command line argument that has not been eaten up.
  - Etc., until no digit can be printed any more.

For instance, if the command line arguments are `312 3 215` then the program prints out and eats up 3 at the beginning of 312, looks at the 3^{rd} command line argument, prints out and eats up 2 at the beginning of 215, looks at the 2^{nd} command line argument, prints out and eats up 3 (so there is nothing left from the 2^{nd} command line argument), looks at the 3^{rd} command line argument, prints out and eats up 1 in the middle of 215, looks at the 1^{st} command line argument, prints out and eats up 1 in the middle of 312, looks at the 1^{st} command line argument again and prints out and eats up 2 at the end of 312 (so there is nothing left from the 1^{st} command line argument), and as there is nothing left from the 2^{nd} command line argument, stops.

Here are sample outputs of how your program should behave:

```
$ a.out
Provide at least one command line argument!
$ a.out 12 32h45 89
Invalid command line arguments!
$ a.out 12 3245 104
Invalid command line arguments!
$ a.out 312 3 215
323112
$ a.out 11111
11111
$ a.out 22222 1111
212121212
$ a.out 4123 231 312
4
$ a.out 234 235 634 235 36357 124 458753 352 3563
22361334253
```

4 ☞ Computing the number of increasing subsequences

Write a program that prompts the user to enter some sequence of distinct digits ending in a new line character, with no other character, not even spaces, before the final new line character, and behaves as follows.

- If more than 10 characters (including spaces) are input before the final new line character then the program prints out

  `Input is too long!`

  and stops.

- Otherwise, if some character is not a digit then the program prints out

  `Input is invalid!`

  and stops.

- Otherwise, if some digit occurs twice in the input then the program prints out

  `Duplicate digits in input!`

  and stops.

- Then the program prompts the user for an integer l, assumed to be correctly entered and readable by `scanf()`, and outputs the number of increasing subsequences of the first input of length l at least.

For instance, the sequence is 34125 has one increasing subsequence of length 0, five increasing subsequences of length 1, namely, 3, 4, 1, 2 and 5, six increasing subsequences of length 2, namely, 34, 35, 45, 12, 15 and 25, two increasing sequences of length 3, namely, 345 and 125, and no increasing sequence of length greater than 3. Hence the sequence is 34125 has no increasing sequence of length 4 at least, two increasing sequences of length 3 at least, eight increasing sequences of length 2 at least, thirteen increasing sequences of length 1 at least, and fourteen increasing sequences of length 0 at least.

Here are sample outputs of how your program should behave:

```
$ a.out
Enter sequence of digits of length 10 at most: 01234567890
Input is too long!
$ a.out
Enter sequence of digits of length 10 at most: 01345 6789
Input is invalid!
$ a.out
Enter sequence of digits of length 10 at most: 012342
Duplicate digits in input!
$ a.out
Enter sequence of digits of length 10 at most: 12
Enter desired length: 1
Number of increasing sequences of length 1 at least: 3
$ a.out
Enter sequence of digits of length 10 at most: 0123456789
Enter desired length: 0
Number of increasing sequences of length 0 at least: 1024
$ a.out
Enter sequence of digits of length 10 at most: 34125
Enter desired length: 4
Number of increasing sequences of length 4 at least: 0
$ a.out
Enter sequence of digits of length 10 at most: 34125
Enter desired length: 2
Number of increasing sequences of length 2 at least: 8
$ a.out
Enter sequence of digits of length 10 at most: 34125
Enter desired length: 1
Number of increasing sequences of length 1 at least: 13
$ a.out
Enter sequence of digits of length 10 at most: 34125
Enter desired length: 0
Number of increasing sequences of length 0 at least: 14
```