Source File: `~/4301/12/lab12.asm`
Input: Under control of `main` function
Output: Under control of `main` function
Value: 2

For $\Sigma = \{1, 2, 3\}$, construct a finite state automata that accepts the set consisting of

$$\{x \mid x \in \{1, 2, 3\}^+ \text{ and } x \text{ is divisible by nine}\}$$

A number is divisible by nine (9) if the sum of its digits is divisible by nine (9). Valid strings include `333`, `1332`, `2133`, `3213`, `333333`, `1233333`, `2323233`, `33333333111`, and `121212121212`, but not ϵ, `0`, `1`, `2`, `3`, `10`, `3330`, `1331`, `23223`, `31233`, `111333`, or `232323232323`.

A sample `main` function for testing your function is shown in Figure 1, a template for starting the assembly function is shown in Figure 2, and a sample execution sequence is shown in Figure 3.

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int isValid(char *s);
5
6  int main()
7  {
8    char s[81];
9
10   while (fgets(s, sizeof(s), stdin))
11   {
12     // remove trailing newline from s
13     s[strcspn(s, "\n")] = 0;
14     printf("%s is %s\n", s, (isValid(s) ? "valid" : "invalid"));
15   }
16
17   return 0;
18 }
```

Figure 1. `/usr/local/4301/src/lab12main.c`

```
1  ; Your Name
2  ; CS 3304
3  ; Lab 12
4
5  ;-----------------------------------------------------------------------
6  global  isValid:function
7  ; Returns true if its argument is valid and false otherwise.
8  ; HLL prototype: int isValid(char *s);
9  ; Implements the following HLL function:
10 ;      int isValid(char *s)
11 ;      {
12 ;        int sum = 0;
13 ;        for ( ; *s != '\0'; ++s)
14 ;        {
15 ;          if (*s >= '1' && *s <= '3')
16 ;            sum += (*s - '0')
17 ;          else
18 ;            break
19 ;        }
20 ;        return *s != '\0' ? 0 : sum % 9 == 0
21 ;      }
22 ; Receives: one 32-bit pointer containing the address of a string
23 ;           via the system stack
24 ; Returns:  a 32-bit integer in EAX
25 ;-----------------------------------------------------------------------
26 isValid:
27 SECTION .text
28         push    ebp
29         mov     ebp,esp
30         sub     esp,4           ; create space on stack for one 32-bit int
31         pushad                  ; push eax, ecx, edx, ebx, esp, ebp, esi, edi
32 ; Stack looks like
33 ;
34 ;       | param *s    |   [ebp + 8]
35 ;       | return addr |   [ebp + 4]
36 ;       | ebp         |   ebp
37 ;       | local var   |   [ebp -  4]
38 ;       | eax         |   [ebp -  8] from pushad
39 ;       | ecx         |   [ebp - 12] from pushad
40 ;       | edx         |   [ebp - 16] from pushad
41 ;       | ebx         |   [ebp - 20] from pushad
42 ;       | esp         |   [ebp - 24] from pushad
43 ;       | ebp         |   [ebp - 28] from pushad
44 ;       | esi         |   [ebp - 32] from pushad
45 ;       | edi         |   [ebp - 36] from pushad
46 ;
47         mov     dword [ebp-4],0 ; clear accumulator
48         mov     esi,[ebp+8]     ; place addr of param in esi
49
```

Figure 2. `/usr/local/4301/src/lab12tmpl.asm` (Part 1 of 2)

```
50 ;
51 ; Put your code here
52 ;
53
54         popad                   ; pop edi, esi, ebp, esp, ebx, edx, ecx, eax
55         mov     eax,[ebp-4]     ; move local accumulator to eax
56         add     esp,4           ; clean local var from stack
57         pop     ebp
58         ret
```

Figure 2. `/usr/local/4301/src/lab12tmpl.asm` (Part 2 of 2)

```
1  newuser@csunix ~> cd 4301
2  newuser@csunix ~/4301> mkdir 12
3  newuser@csunix ~/4301> cd 12
4  newuser@csunix ~/4301/12> # Copy the sample input and output files
5  newuser@csunix ~/4301/12> cp /usr/local/4301/data/12/* .
6  newuser@csunix ~/4301/12> ls
7  00.dat  00.out
8  newuser@csunix ~/4301/12> # Copy the main function
9  newuser@csunix ~/4301/12> cp /usr/local/4301/src/lab12main.c .
10 newuser@csunix ~/4301/12> ls
11 00.dat  00.out  lab12main.c
12 newuser@csunix ~/4301/12> # Copy the template for the assembly code
13 newuser@csunix ~/4301/12> # and save it as lab12.asm
14 newuser@csunix ~/4301/12> cp /usr/local/4301/src/lab12tmpl.asm lab12.asm
15 newuser@csunix ~/4301/12> # Edit lab12.asm to complete the code
16 newuser@csunix ~/4301/12> # Compile the C main function
17 newuser@csunix ~/4301/12> gcc -m32 -c lab12main.c
18 newuser@csunix ~/4301/12> # Assemble the assembly function
19 newuser@csunix ~/4301/12> nasm -f elf32 -o lab12.o lab12.asm
20 newuser@csunix ~/4301/12> # Invoke the linker to create the executable
21 newuser@csunix ~/4301/12> gcc -m32 -o lab12 lab12main.o lab12.o
22 newuser@csunix ~/4301/12> ls
23 00.dat  00.out  lab12  lab12.asm  lab12.o  lab12main.c  lab12main.o
24 newuser@csunix ~/4301/12> # Execute lab12 on dataset 00
25 newuser@csunix ~/4301/12> ./lab12 < 00.dat
26  is invalid
27 0 is invalid
28 1 is invalid
29 2 is invalid
30 3 is invalid
31 10 is invalid
32 11 is invalid
33 12 is invalid
34 13 is invalid
```

Figure 3. Commands to Compile, Assemble, Link, & Run Lab 12 (Part 1 of 2)

```
21 is invalid
22 is invalid
23 is invalid
31 is invalid
32 is invalid
33 is invalid
123 is invalid
132 is invalid
213 is invalid
231 is invalid
312 is invalid
321 is invalid
111 is invalid
222 is invalid
232 is invalid
333 is valid
3330 is invalid
1331 is invalid
1332 is valid
2133 is valid
3213 is valid
23223 is invalid
31233 is invalid
111333 is invalid
333333 is valid
1233333 is valid
2323233 is valid
33333333111 is valid
121212121212 is valid
232323232323 is invalid
11122233311122233311122233311122233311122233311122233311122233311122333 is valid
newuser@csunix ~/4301/12> ./lab12 < 00.dat > my.out
newuser@csunix ~/4301/12> diff 00.out my.out
newuser@csunix ~/4301/12>
```

Figure 3. Commands to Compile, Assemble, Link, & Run Lab 12 (Part 2 of 2)