FORMAN CHRISTIAN COLLEGE
(A CHARTERED UNIVERSITY)

Compiler Construction - COMP 451 A

Spring 22

Lab 06

Muhammad Sulaiman Sultan - 231453415

Muhammad Sameed Gilani - 231488347

INTRODUCTION:

- **Command line arguments:**

  ➔ int argc:

  This stores the number of arguments passed by the user when the run the program. This also includes the program name.

  ➔ Char* argv[]:

  This is an array of character pointers, which point to the address of the command line arguments that were passed. argv[0] always stores the program name.

- **fopen() → <stdio.h>**

  - This is a C library function that uses a file pointer to open a file with in the chosen mode. In our case, “r” (read) was used.
  - It returns a file pointer if file opens successfully or Null when it fails.
  - We use this function to open and read the text file.

  Function Signature:

```
FILE *fopen(const char *file_name, const char
*mode_of_operation);
```

  Sample from program:

```
FILE *fp;

fp = fopen(argv[1],"r");


if (fp == NULL){

      printf("Error opening file");
```

```
        exit(0);
}
```

- **malloc() → <stdlib.h>**

    - This function is used to dynamically allocate a single block of memory in the heap, with a size of our choice.
    - It returns a pointer to the allocated memory.
    - It does not initialize memory on execution.
    - We use this to allocate a block of memory to store the each character read from the file

    Function Signature:

```
ptr = (cast-type*) malloc(byte-size)
```

    Sample from program:

```
char* buff = (char*) malloc(sizeof(char)*1000);
```

    - sizeof() returns the size of the data type in bytes. (2 for char).
      So this allocates a memory of 2000 bytes. With buff as the pointer.

- **fgetc() → <stdio.h>**
    - It takes a file pointer as input and returns a single char read from the file.
    - It returns type int however, as character literal has type int.
    - We use this function to read a file, character by character

    Function Signature:

```
int fgetc(FILE *pointer)
```

Sample From Prorgram:

currC = fgetc(fp);

c is then stored in the memory, previously allocated.

- **rewind() → <stdio.h>**
  - rewind is used to set the file pointer back to the beginning of the file.
  - We use rewind in the program to read the file twice, after reading it to the end once.

Function Signature:

```
void rewind(FILE *stream)
```

Sample From Program:

```
rewind(fp);
```

- **goto statement**
  - This is a jump statement
  - Allows us to switch control to a predefined label

    In the code we use goto statements to jump to labels(states) that correspond to the current input.

LOGIC:

- We first open the file given through command line argument by the user, using fopen(). Along with necessary check to see the existence of the file. Then initialize all the necessary variables.
- We store the input from the file into a buffer and then print the input on the terminal. Then the file pointer is reset using rewind().
- We define the labels according to the states of the DFA. Four labels are made for each state and two states to validate or terminate, given the input.
- Iterates through the input character by character, and according to the current character, we use goto to jump to its corresponding state(label).
- Each state has checks for the current input, ‘a’ or ‘b’. If a ‘$’ is the input we goto validate, which checks the variable int valid; to decide whether the input is valid. Valid = 1 if correct, 0 otherwise.
  If a character other then ‘a’ or ‘b’ is seen, we immediately goto terminate and end the program, as the input char is illegal.
- The states were made according to the state table and dfa.

CODE

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){
    FILE *fp;

     if(argc != 2)
     {
           printf("Need to have a file name in the arguments of this
program\n");
           exit(0);
     }

    fp = fopen(argv[1],"r");

    if (fp == NULL){ // Prints an error and exits if file doesnt open
           printf("Error opening file");
           exit(0);
     }

    char currC = ' ';
    int source = 0;
    int nextState = 0;
    int valid = 0;

    char* buff = (char*) malloc(sizeof(char)*100); // Allocates memory
of 2000 bytes
    int i = 0;
    while(currC != '$'){
```

```
        currC=fgetc(fp);
        buff[i] = currC;
        i++;
    }
    printf("Input String is: %s\n",buff);
    printf("State Transitions are shown below: \n\n");

    currC = ' ';
    rewind(fp);
    currC = fgetc(fp);

    if(currC == 'a'){
        nextState = 2;
        goto Q2;
    }
    else if(currC == 'b'){
        nextState = 1;
        goto Q1;
    }
    else{
        goto terminate;
    }

    terminate:
        printf("Invalid character %c at Q%d\n terminating process\
n",currC,source);
        return 0;

    validate:
```

```
        if(valid == 1){
            printf("String %s is valid\n",buff);
            return 0;
        }
        if(valid == 0){
            printf("String %s is invalid\n",buff);
            return 0;

        }
```

```
    Q1:
    printf("Recvied %c at state Q%d ---- Moving to state Q%d\
n",currC,source,nextState);
    source = 1;
    currC = fgetc(fp);

    if(currC == '$'){
        goto validate;
    }
    if(currC != 'a' && currC != 'b'){
        goto terminate;
    }
    else{
        if(currC == 'a'){
            nextState = 2;
            goto Q2;
```

```
        }
        if(currC == 'b'){
            nextState = 3;
            goto Q3;
        }
    }


    Q2:
    printf("Recvied %c at state Q%d ---- Moving to state Q%d\
n",currC,source,nextState);
    source = 2;

    currC = fgetc(fp);

    if(currC == '$'){
        goto validate;
    }
    if(currC != 'a' && currC != 'b'){
        goto terminate;
    }

    else{
        if(currC == 'a'){
            nextState = 4;
            goto Q4;
        }
        if(currC == 'b'){
```

```
            nextState = 1;
            goto Q1;
        }
    }


    Q3:
    printf("Recvied %c at state Q%d ---- Moving to state Q%d\
n",currC,source,nextState);
    source = 3;


    currC = fgetc(fp);


    if(currC == '$'){
        valid = 1;
        goto validate;
    }
    if(currC != 'a' && currC != 'b'){
        goto terminate;
    }


    else{
        if(currC == 'a'){
            nextState = 2;
            goto Q2;
        }
        if(currC == 'b'){
            nextState = 3;
            goto Q3;
```

```
        }
    }
    Q4:
    printf("Recvied %c at state Q%d ---- Moving to state Q%d\
n",currC,source,nextState);
    source = 4;


    currC = fgetc(fp);
    if(currC == '$'){
        valid = 1;
        goto validate;
    }
    if(currC != 'a' && currC != 'b'){
        goto terminate;
    }


    else{
        if(currC == 'a'){
            nextState = 4;
            goto Q4;
        }
        if(currC == 'b'){
            nextState = 1;
            goto Q1;
        }
    }



     return 0;}
```

Sample Outputs:

```
on/LABS/Lab6$ ./Lab6 infile
Input String is: bbbbbaababababaaabbbbbbb$
State Transitions are shown below:

Recvied b at state Q0 ---- Moving to state Q1
Recvied b at state Q1 ---- Moving to state Q3
Recvied b at state Q3 ---- Moving to state Q3
Recvied b at state Q3 ---- Moving to state Q3
Recvied b at state Q3 ---- Moving to state Q3
Recvied a at state Q3 ---- Moving to state Q2
Recvied a at state Q2 ---- Moving to state Q4
Recvied b at state Q4 ---- Moving to state Q1
Recvied a at state Q1 ---- Moving to state Q2
Recvied b at state Q2 ---- Moving to state Q1
Recvied a at state Q1 ---- Moving to state Q2
Recvied b at state Q2 ---- Moving to state Q1
Recvied a at state Q1 ---- Moving to state Q2
Recvied a at state Q2 ---- Moving to state Q4
Recvied a at state Q4 ---- Moving to state Q4
Recvied b at state Q4 ---- Moving to state Q1
Recvied b at state Q1 ---- Moving to state Q3
Recvied b at state Q3 ---- Moving to state Q3
Recvied b at state Q3 ---- Moving to state Q3
Recvied b at state Q3 ---- Moving to state Q3
Recvied b at state Q3 ---- Moving to state Q3
Recvied b at state Q3 ---- Moving to state Q3
String bbbbbaababababaaabbbbbbb$ is valid
sameed@SameedHpLappy:~/Desktop/6th_semester_Spring22/COMP451A_CompilerConstructi
```

```
on/LABS/Lab6$ ./Lab6 infile
Input String is: aab$
State Transitions are shown below:

Recvied a at state Q0 ---- Moving to state Q2
Recvied a at state Q2 ---- Moving to state Q4
Recvied b at state Q4 ---- Moving to state Q1
String aab$ is invalid
sameed@SameedHpLappy:~/Desktop/6th_semester_Spring22/COMP451A_CompilerConstructi
on/LABS/Lab6$ ./Lab6 infile
Input String is: aaaaa$
State Transitions are shown below:

Recvied a at state Q0 ---- Moving to state Q2
Recvied a at state Q2 ---- Moving to state Q4
Recvied a at state Q4 ---- Moving to state Q4
Recvied a at state Q4 ---- Moving to state Q4
Recvied a at state Q4 ---- Moving to state Q4
String aaaaa$ is valid
sameed@SameedHpLappy:~/Desktop/6th_semester_Spring22/COMP451A_CompilerConstructi
```

```
on/LABS/Lab6$ ./Lab6 infile
Input String is: aaabbbaaaa$
State Transitions are shown below:

Recvied a at state Q0 ---- Moving to state Q2
Recvied a at state Q2 ---- Moving to state Q4
Recvied a at state Q4 ---- Moving to state Q4
Recvied b at state Q4 ---- Moving to state Q1
Recvied b at state Q1 ---- Moving to state Q3
Recvied b at state Q3 ---- Moving to state Q3
Recvied a at state Q3 ---- Moving to state Q2
Recvied a at state Q2 ---- Moving to state Q4
Recvied a at state Q4 ---- Moving to state Q4
Recvied a at state Q4 ---- Moving to state Q4
String aaabbbaaaa$ is valid
sameed@SameedHpLappy:~/Desktop/6th_semester_Spring22/COMP451A_CompilerConstructi
on/LABS/Lab6$ ./Lab6 infile
Input String is: aaabxbbaaaa$
State Transitions are shown below:

Recvied a at state Q0 ---- Moving to state Q2
Recvied a at state Q2 ---- Moving to state Q4
Recvied a at state Q4 ---- Moving to state Q4
Recvied b at state Q4 ---- Moving to state Q1
Invalid character x at Q1
 terminating process
sameed@SameedHpLappy:~/Desktop/6th_semester_Spring22/COMP451A_CompilerConstructi
```