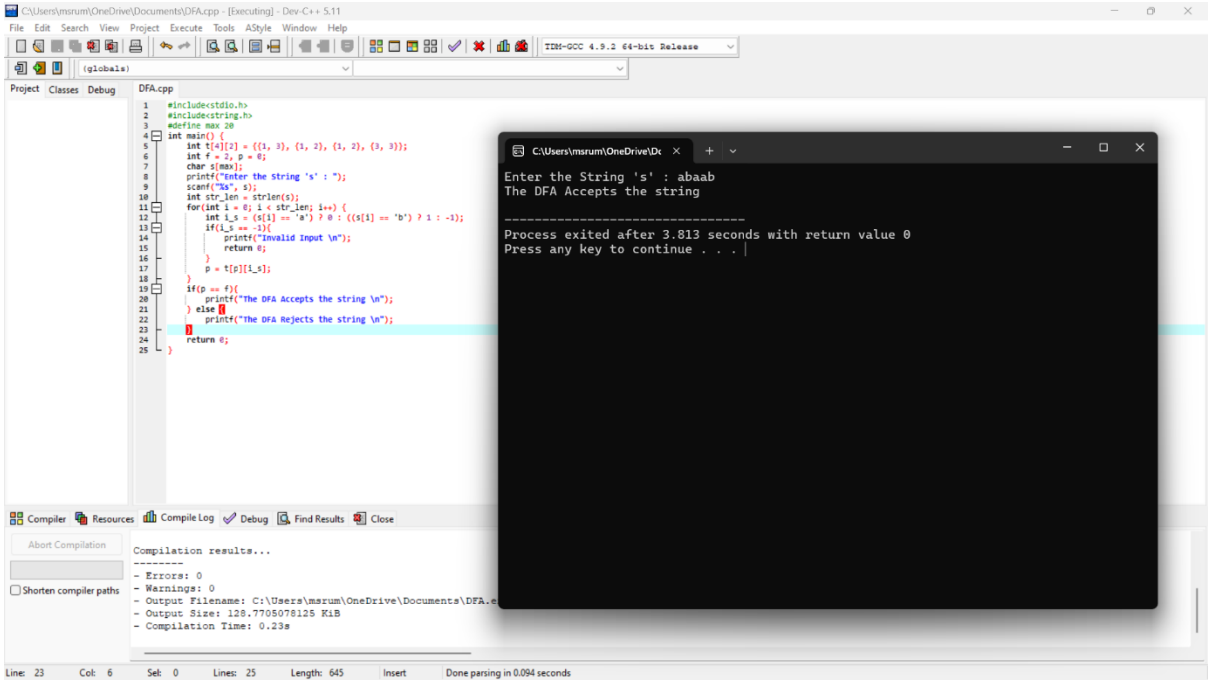
16) DFA Construction and Checking the string acceptance

Code:

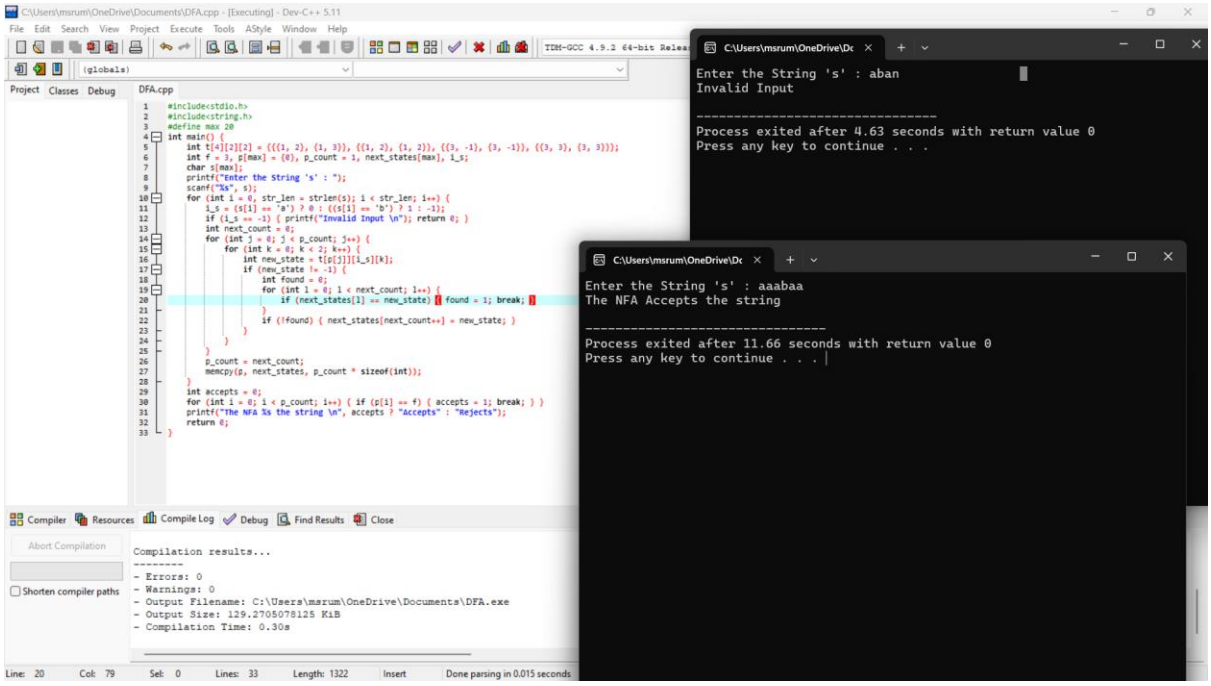
```
#include<stdio.h>
#include<string.h>
#define max 20
int main() {
   int t[4][2] = {{1, 3}, {1, 2}, {1, 2}, {3, 3}};
   int f = 2, p = 0;
   char s[max];
   printf("Enter the String 's' : ");
   scanf("%s", s);
   int str_len = strlen(s);
   for(int i = 0; i < str_len; i++) {
      int i_s = (s[i] == 'a') ? 0 : ((s[i] == 'b') ? 1 : -1);
      if(i_s == -1){
         printf("Invalid Input \n");
         return 0;
      }
      p = t[p][i_s];
   }
   if(p == f){
      printf("The DFA Accepts the string \n");
   } else {
      printf("The DFA Rejects the string \n");
   }
   return 0;
}
```



17) NFA Construction and Checking the string acceptance

```
#include<stdio.h>

#include<string.h>

#define max 20

int main() {

   int t[4][2][2] = {{{1, 2}, {1, 3}}, {{1, 2}, {1, 2}}, {{3, -1}, {3, -1}}, {{3, 3}, {3, 3}}};

   int f = 3, p[max] = {0}, p_count = 1, next_states[max], i_s;

   char s[max];

   printf("Enter the String 's' : ");

   scanf("%s", s);

   for (int i = 0, str_len = strlen(s); i < str_len; i++) {

      i_s = (s[i] == 'a') ? 0 : ((s[i] == 'b') ? 1 : -1);

      if (i_s == -1) { printf("Invalid Input \n"); return 0; }

      int next_count = 0;

      for (int j = 0; j < p_count; j++) {

         for (int k = 0; k < 2; k++) {

            int new_state = t[p[j]][i_s][k];

            if (new_state != -1) {
```

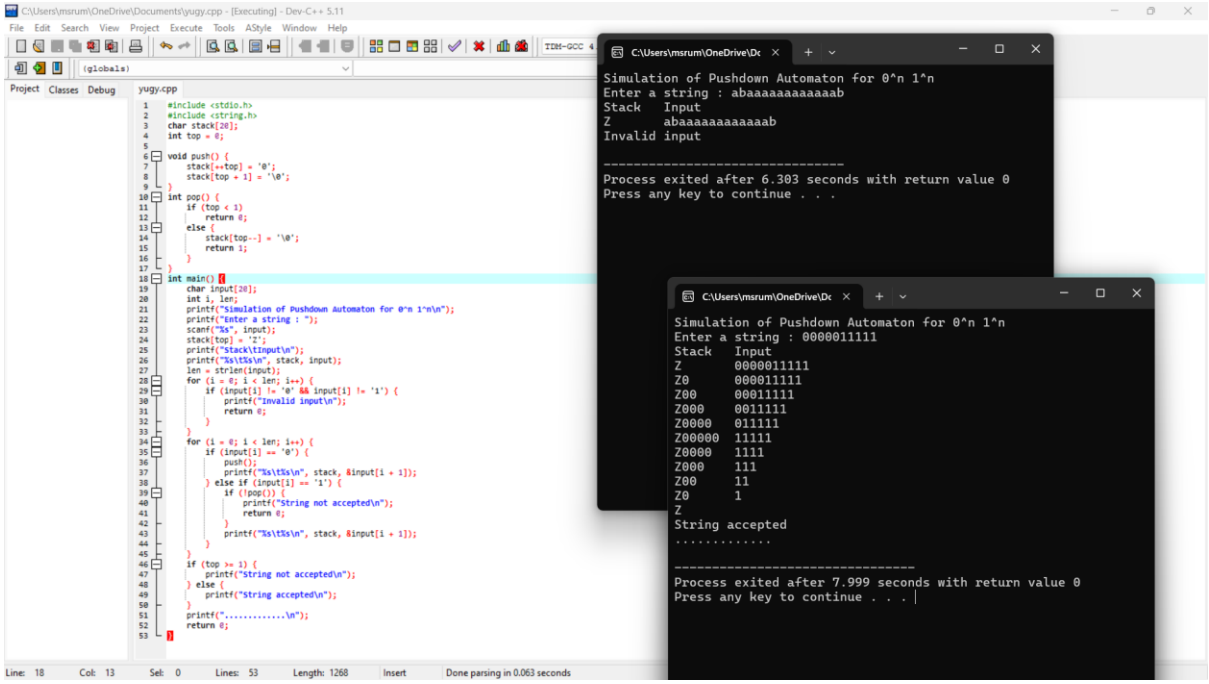
```
                int found = 0;
                for (int l = 0; l < next_count; l++) {
                    if (next_states[l] == new_state) { found = 1; break; }
                }
                if (!found) { next_states[next_count++] = new_state; }
            }
        }
    }
    p_count = next_count;
    memcpy(p, next_states, p_count * sizeof(int));
  }
  int accepts = 0;
  for (int i = 0; i < p_count; i++) { if (p[i] == f) { accepts = 1; break; } }
  printf("The NFA %s the string \n", accepts ? "Accepts" : "Rejects");
  return 0;
}
```



18) PDA 0^n 1^n

```
#include <stdio.h>
#include <string.h>
char stack[20];
int top = 0;

void push() {
   stack[++top] = '0';
   stack[top + 1] = '\0';
}
int pop() {
   if (top < 1)
      return 0;
   else {
      stack[top--] = '\0';
      return 1;
   }
}
int main() {
   char input[20];
   int i, len;
   printf("Simulation of Pushdown Automaton for 0^n 1^n\n");
   printf("Enter a string : ");
   scanf("%s", input);
   stack[top] = 'Z';
   printf("Stack\tInput\n");
   printf("%s\t%s\n", stack, input);
   len = strlen(input);
   for (i = 0; i < len; i++) {
      if (input[i] != '0' && input[i] != '1') {
         printf("Invalid input\n");
```

```
            return 0;
        }
    }
    for (i = 0; i < len; i++) {
        if (input[i] == '0') {
            push();
            printf("%s\t%s\n", stack, &input[i + 1]);
        } else if (input[i] == '1') {
            if (!pop()) {
                printf("String not accepted\n");
                return 0;
            }
            printf("%s\t%s\n", stack, &input[i + 1]);
        }
    }
    if (top >= 1) {
        printf("String not accepted\n");
    } else {
        printf("String accepted\n");
    }
    printf(".............\n");
    return 0;
}
```



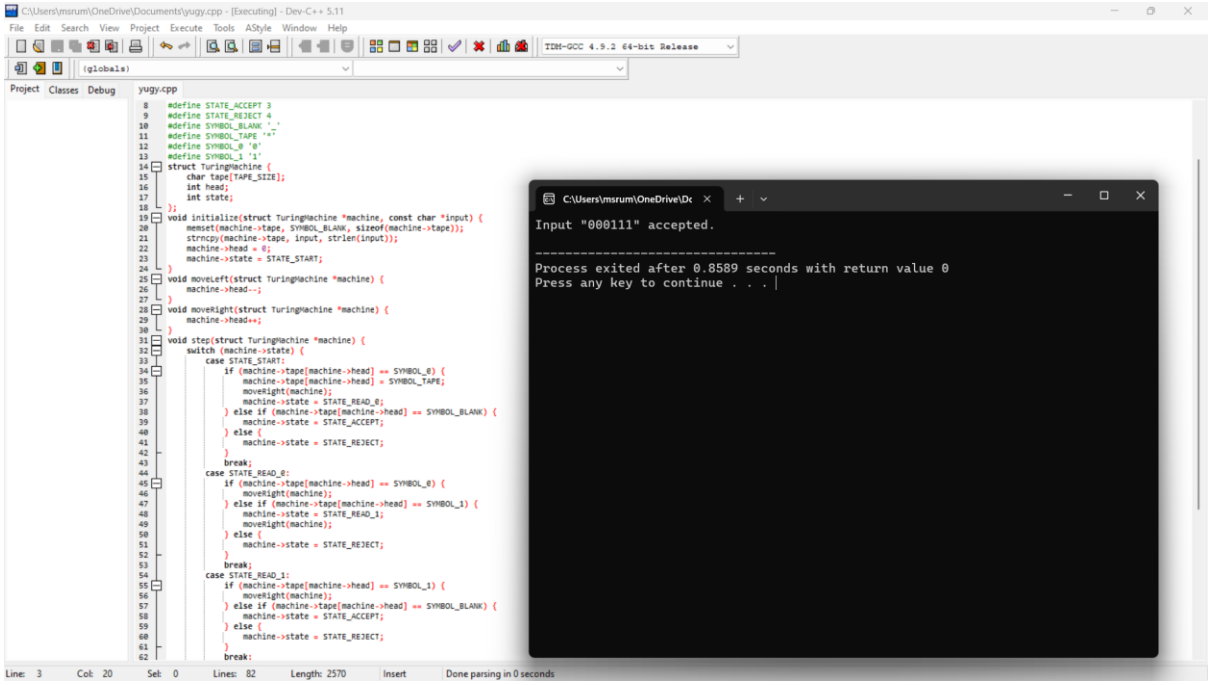
19) Turing Machine for 0^n 1^2

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TAPE_SIZE 1000

#define STATE_START 0

#define STATE_READ_0 1

#define STATE_READ_1 2

#define STATE_ACCEPT 3

#define STATE_REJECT 4

#define SYMBOL_BLANK '_'

#define SYMBOL_TAPE '*'

#define SYMBOL_0 '0'

#define SYMBOL_1 '1'

struct TuringMachine {

    char tape[TAPE_SIZE];

    int head;

    int state;
```

```
};
void initialize(struct TuringMachine *machine, const char *input) {
    memset(machine->tape, SYMBOL_BLANK, sizeof(machine->tape));
    strncpy(machine->tape, input, strlen(input));
    machine->head = 0;
    machine->state = STATE_START;
}
void moveLeft(struct TuringMachine *machine) {
    machine->head--;
}
void moveRight(struct TuringMachine *machine) {
    machine->head++;
}
void step(struct TuringMachine *machine) {
    switch (machine->state) {
        case STATE_START:
            if (machine->tape[machine->head] == SYMBOL_0) {
                machine->tape[machine->head] = SYMBOL_TAPE;
                moveRight(machine);
                machine->state = STATE_READ_0;
            } else if (machine->tape[machine->head] == SYMBOL_BLANK) {
                machine->state = STATE_ACCEPT;
            } else {
                machine->state = STATE_REJECT;
            }
            break;
        case STATE_READ_0:
            if (machine->tape[machine->head] == SYMBOL_0) {
                moveRight(machine);
            } else if (machine->tape[machine->head] == SYMBOL_1) {
                machine->state = STATE_READ_1;
                moveRight(machine);
```

```
        } else {
            machine->state = STATE_REJECT;
        }
        break;
    case STATE_READ_1:
        if (machine->tape[machine->head] == SYMBOL_1) {
            moveRight(machine);
        } else if (machine->tape[machine->head] == SYMBOL_BLANK) {
            machine->state = STATE_ACCEPT;
        } else {
            machine->state = STATE_REJECT;
        }
        break;
    case STATE_ACCEPT:
        break;
    case STATE_REJECT:
        break;
  }
}
int main() {
  struct TuringMachine machine;
  const char *input = "000111";
  initialize(&machine, input);
  while (machine.state != STATE_ACCEPT && machine.state != STATE_REJECT) {
    step(&machine);
  }
  if (machine.state == STATE_ACCEPT) {
    printf("Input \"%s\" accepted.\n", input);
  } else {
    printf("Input \"%s\" rejected.\n", input);
  }
  return 0;
```

}



20) Turing Machine for 0^n 1^2n

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TAPE_SIZE 1000

#define STATE_START 0

#define STATE_READ_0 1

#define STATE_READ_1 2

#define STATE_READ_1_SECOND 3

#define STATE_ACCEPT 4

#define STATE_REJECT 5

#define SYMBOL_BLANK '_'

#define SYMBOL_TAPE '*'

#define SYMBOL_0 '0'

#define SYMBOL_1 '1'

struct TuringMachine {

    char tape[TAPE_SIZE];
```

```
    int head;
    int state;
};
void initialize(struct TuringMachine *machine, const char *input) {
    memset(machine->tape, SYMBOL_BLANK, sizeof(machine->tape));
    strncpy(machine->tape, input, strlen(input));
    machine->head = 0;
    machine->state = STATE_START;
}
void moveLeft(struct TuringMachine *machine) {
    machine->head--;
}
void moveRight(struct TuringMachine *machine) {
    machine->head++;
}
void step(struct TuringMachine *machine) {
    switch (machine->state) {
        case STATE_START:
            if (machine->tape[machine->head] == SYMBOL_0) {
                machine->tape[machine->head] = SYMBOL_TAPE;
                moveRight(machine);
                machine->state = STATE_READ_0;
            } else if (machine->tape[machine->head] == SYMBOL_BLANK) {
                machine->state = STATE_ACCEPT;
            } else {
                machine->state = STATE_REJECT;
            }
            break;
        case STATE_READ_0:
            if (machine->tape[machine->head] == SYMBOL_0) {
```

```
        moveRight(machine);
    } else if (machine->tape[machine->head] == SYMBOL_1) {
        machine->state = STATE_READ_1;
        moveRight(machine);
    } else {
        machine->state = STATE_REJECT;
    }
    break;
case STATE_READ_1:
    if (machine->tape[machine->head] == SYMBOL_1) {
        moveRight(machine);
    } else if (machine->tape[machine->head] == SYMBOL_BLANK) {
        machine->state = STATE_REJECT;
    } else {
        machine->state = STATE_READ_1_SECOND;
        moveRight(machine);
    }
    break;
case STATE_READ_1_SECOND:
    if (machine->tape[machine->head] == SYMBOL_1) {
        moveRight(machine);
    } else if (machine->tape[machine->head] == SYMBOL_BLANK) {
        machine->state = STATE_ACCEPT;
    } else {
        machine->state = STATE_REJECT;
    }
    break;
case STATE_ACCEPT:
    break;
case STATE_REJECT:
```

```
            break;
        }
    }
int main() {
    struct TuringMachine machine;
    const char *input = "000111";
    initialize(&machine, input);
    while (machine.state != STATE_ACCEPT && machine.state != STATE_REJECT) {
        step(&machine);
    }
    if (machine.state == STATE_ACCEPT) {
        printf("Input \"%s\" accepted.\n", input);
    } else {
        printf("Input \"%s\" rejected.\n", input);
    }
    return 0;
}
```