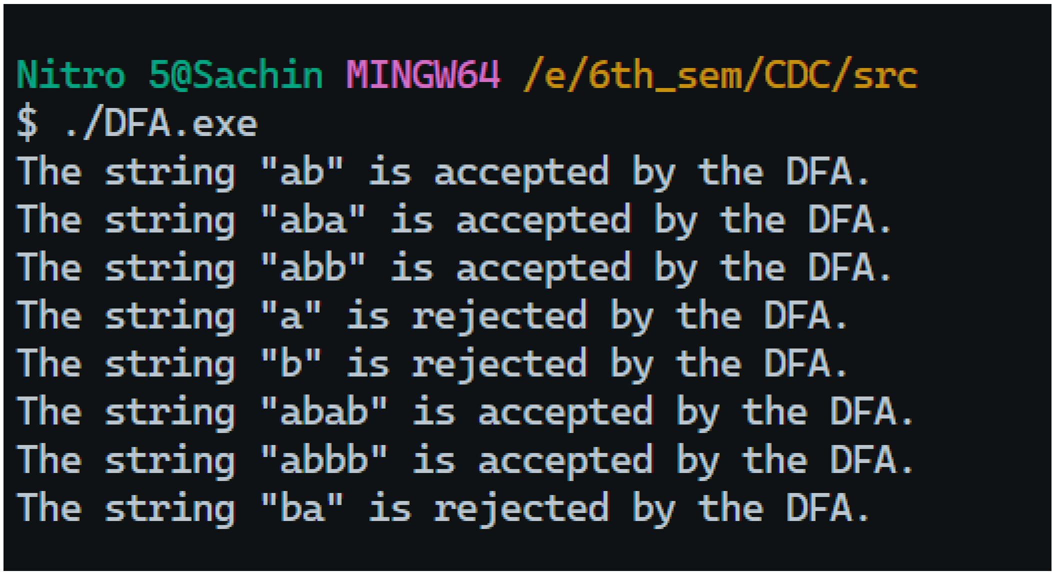
Contents

Write a program that constructs the DFA that accepts the language `L = {w | w is a string of a's and b's such that w always starts with 'ab'}`.

```
#include <stdbool.h>
#include <stdio.h>
typedef enum
{
    START,
    STATE_A,
    STATE_AB,
    REJECT
} State;

int transition(int state, char input)
{
    switch (state)
    {
    case START:
        if (input == 'a')
            return STATE_A;
        break;
    case STATE_A:
        if (input == 'b')
            return STATE_AB;
        else
            return REJECT;
    case STATE_AB:
        if (input == 'a' || input == 'b')
            return STATE_AB;
        break;
    default:
        break;
    }
    return REJECT;
}

bool accepts(const char *input)
{
    int state = START;
    for (int i = 0; input[i] != '\0'; i++)
    {
        state = transition(state, input[i]);
        if (state == REJECT)
            return false;
    }
    return state == STATE_AB;
}

int main()
{
    const char *testStrings[] = {"ab", "aba", "abb", "a", "b", "abab", "abbb", "ba"};
    int numTests = sizeof(testStrings) / sizeof(testStrings[0]);
    for (int i = 0; i < numTests; i++)
    {
        if (accepts(testStrings[i]))
        {
            printf("The string \"%s\" is accepted by the DFA.\n", testStrings[i]);
        }
        else
        {
            printf("The string \"%s\" is rejected by the DFA.\n", testStrings[i]);
        }
    }

    return 0;
}
```



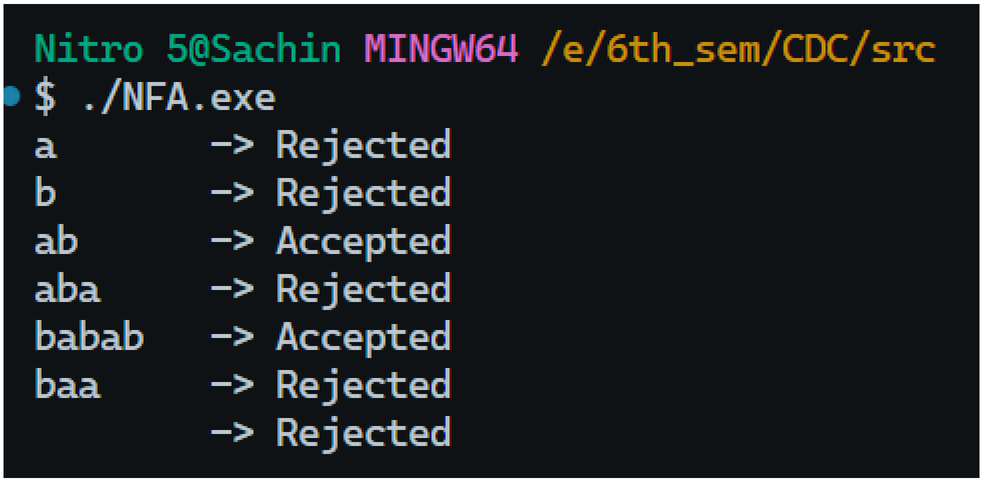
```
Nitro 5@Sachin MINGW64 /e/6th_sem/CDC/src
$ ./DFA.exe
The string "ab" is accepted by the DFA.
The string "aba" is accepted by the DFA.
The string "abb" is accepted by the DFA.
The string "a" is rejected by the DFA.
The string "b" is rejected by the DFA.
The string "abab" is accepted by the DFA.
The string "abbb" is accepted by the DFA.
The string "ba" is rejected by the DFA.
```

Write a program that constructs the NFA that ends with 'ab'`.

```
#include <stdbool.h>

#include <stdio.h>

#include <string.h>


#define MAX_STATES 3


enum State

{

    Q0,

    Q1,

    Q2

};


// Function to simulate the NFA

bool accepts(const char *input)

{

    int len = strlen(input);

    bool current_st[MAX_STATES] = {false};

    bool nxt_st[MAX_STATES] = {false};


    current_st[Q0] = true;


    for (int i = 0; i < len; i++)

    {
```

```
char symbol = input[i];

// Reset nxt_st
for (int s = 0; s < MAX_STATES; s++)
    nxt_st[s] = false;

for (int s = 0; s < MAX_STATES; s++)
{
    if (!current_st[s])
        continue;

    switch (s)
    {
    case Q0:
        if (symbol == 'a')
        {
            nxt_st[Q0] = true;
            nxt_st[Q1] = true;
        }
        else if (symbol == 'b')
        {
            nxt_st[Q0] = true;
        }
        break;
    case Q1:
        if (symbol == 'b')
        {
```

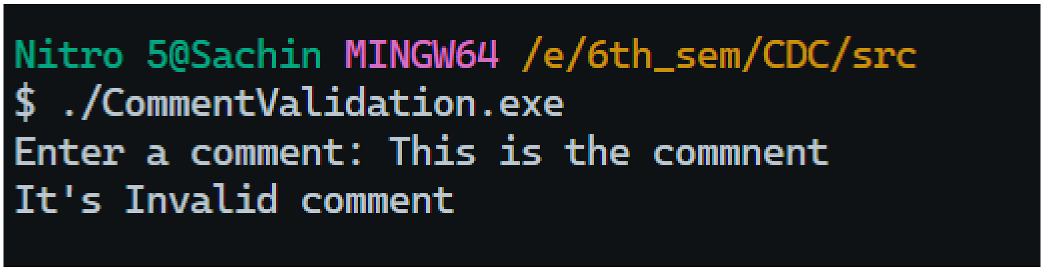
```
                    nxt_st[Q2] = true;
                }
                break;
            case Q2:
                // No transitions from Q2
                break;
            }
        }

        // Update current_st
        for (int s = 0; s < MAX_STATES; s++)
        {
            current_st[s] = nxt_st[s];
        }
    }

    return current_st[Q2];
}

int main()
{
    const char *test_strings[] = {
        "a", "b", "ab", "aba", "babab", "baa", ""};
    int num_tests = sizeof(test_strings) / sizeof(test_strings[0]);

    for (int i = 0; i < num_tests; i++)
    {
        printf("%-7s -> %s\n", test_strings[i],
                accepts(test_strings[i]) ? "Accepted" : "Rejected");
    }

    return 0;
}
```



```
Nitro 5@Sachin MINGW64 /e/6th_sem/CDC/src
$ ./NFA.exe
a       -> Rejected
b       -> Rejected
ab      -> Accepted
aba     -> Rejected
babab   -> Accepted
baa     -> Rejected
        -> Rejected
```

Write a program for `comment validation`.

```
#include <ctype.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

bool isValidComment(const char *comment)
{
    int len = strlen(comment);
    char mdcstr[len + 1];
    strcpy(mdcstr, comment);

    // trim the comment
    char *start = mdcstr;
    while (isspace((unsigned char)*start))
    {
        start++;
    }

    char *end = start + strlen(start) - 1;
    while (end > start && isspace((unsigned char)*end))
    {
        *end = '\0';
        end--;
    }
    if (start[0] == '/' && start[1] == '/')
    {
        return true;
    }
    else if (len >= 4 && start[0] == '/' && start[1] == '*' && end[0] == '/' && end[-1] == '*')  {
        return true;
    }
    return false;
}
int main()
{
    char comment[100];
    printf("Enter a comment: ");
    fgets(comment, sizeof(comment), stdin);
    comment[strcspn(comment, "\n")] = '\0';
    if (isValidComment(comment))
    {
        printf("It's Valid comment\n");
    } else
    {
        printf("It's Invalid comment\n");
    }

    return 0;
}
```



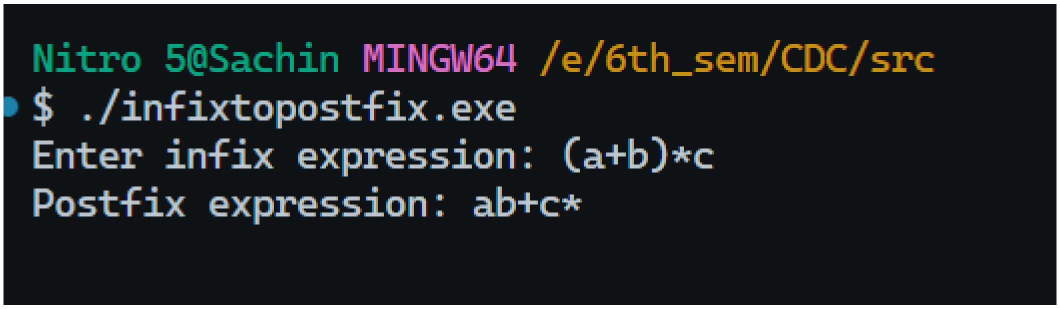
```
Nitro 5@Sachin MINGW64 /e/6th_sem/CDC/src
$ ./CommentValidation.exe
Enter a comment: This is the commnent
It's Invalid comment
```

Write a program that converts an infix expression to a postfix expression.

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

#define MAX_SIZE 100

// Stack implementation
struct Stack
{
    int top;
    char items[MAX_SIZE];
};

void initialize(struct Stack *s)
{
    s->top = -1;
}

int isEmpty(struct Stack *s)
{
    return s->top == -1;
}

void push(struct Stack *s, char value)
```

```
{
    if (s->top == MAX_SIZE - 1)
    {
        printf("Stack Overflow\n");
        return;
    }
    s->items[++(s->top)] = value;
}

char pop(struct Stack *s)
{
    if (isEmpty(s))
    {
        printf("Stack Underflow\n");
        return '\0';
    }
    return s->items[(s->top)--];
}

char peek(struct Stack *s)
{
    if (isEmpty(s))
        return '\0';
    return s->items[s->top];
}

int precedence(char op)
```

```
{
    switch (op)
    {
    case '+':
    case '-':
        return 1;
    case '*':
    case '/':
        return 2;
    case '^':
        return 3;
    default:
        return -1;
    }
}

void infixToPostfix(char *infix, char *postfix)
{
    struct Stack s;
    initialize(&s);
    int i, j = 0;

    for (i = 0; infix[i]; i++)
    {
        if (isalnum(infix[i]))
        {
            postfix[j++] = infix[i];
        }
        else if (infix[i] == '(')
        {
            push(&s, infix[i]);
        }
        else if (infix[i] == ')')
        {
            while (!isEmpty(&s) && peek(&s) != '(')
            {
                postfix[j++] = pop(&s);
            }
            if (!isEmpty(&s) && peek(&s) == '(')
            {
                pop(&s); // Remove '('
            }
        }
        else
        {
            while (!isEmpty(&s) && precedence(infix[i]) <= precedence(peek(&s)))
            {
                postfix[j++] = pop(&s);
            }
            push(&s, infix[i]);
        }
    }

    while (!isEmpty(&s))
    {
        postfix[j++] = pop(&s);
    }

    postfix[j] = '\0';
}

int main()
{
    char infix[MAX_SIZE], postfix[MAX_SIZE];

    printf("Enter infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);

    return 0;
}
```



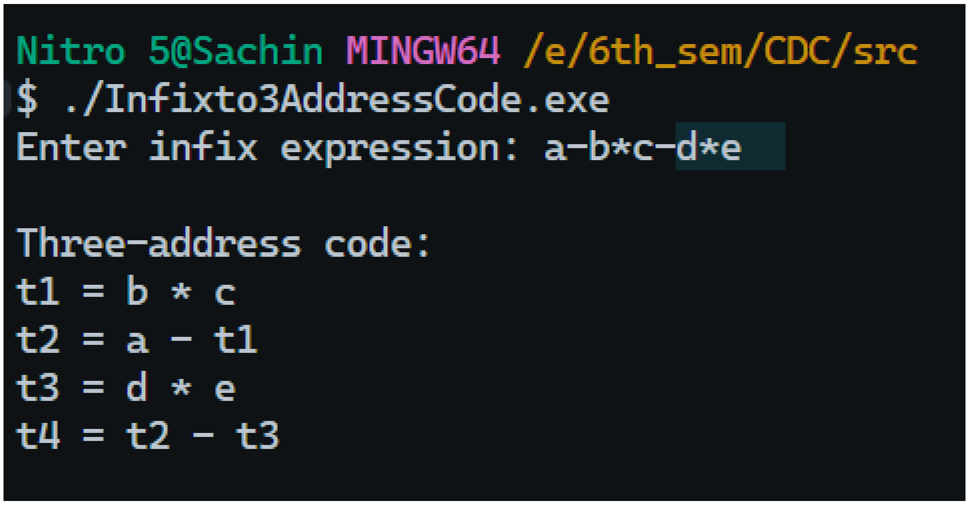
```
Nitro 5@Sachin MINGW64 /e/6th_sem/CDC/src
$ ./infixtopostfix.exe
Enter infix expression: (a+b)*c
Postfix expression: ab+c*
```

Write a program that converts an infix expression to three-address code

```
#include <ctype.h>

#include <stdio.h>

#include <string.h>


#define MAX 100


char stack[MAX];

int top = -1;

int tempVar = 1;


void push(char ch) { stack[++top] = ch; }

char pop() { return stack[top--]; }

char peek() { return stack[top]; }


int precedence(char op)

{

    if (op == '+' || op == '-')

        return 1;

    if (op == '*' || op == '/')

        return 2;

    return 0;

}


// Convert infix to postfix
```

```
void infixToPostfix(const char *infix, char *postfix)
{
   int i = 0, k = 0;
   char ch;
   while ((ch = infix[i++]) != '\0')
   {
      if (isalnum(ch))
      {
         postfix[k++] = ch;
      }
      else if (ch == '(')
      {
         push(ch);
      }
      else if (ch == ')')
      {
         while (top != -1 && peek() != '(')
            postfix[k++] = pop();
         pop(); // remove '('
      }
      else
      {
         while (top != -1 && precedence(peek()) >= precedence(ch))
            postfix[k++] = pop();
         push(ch);
      }
   }
```

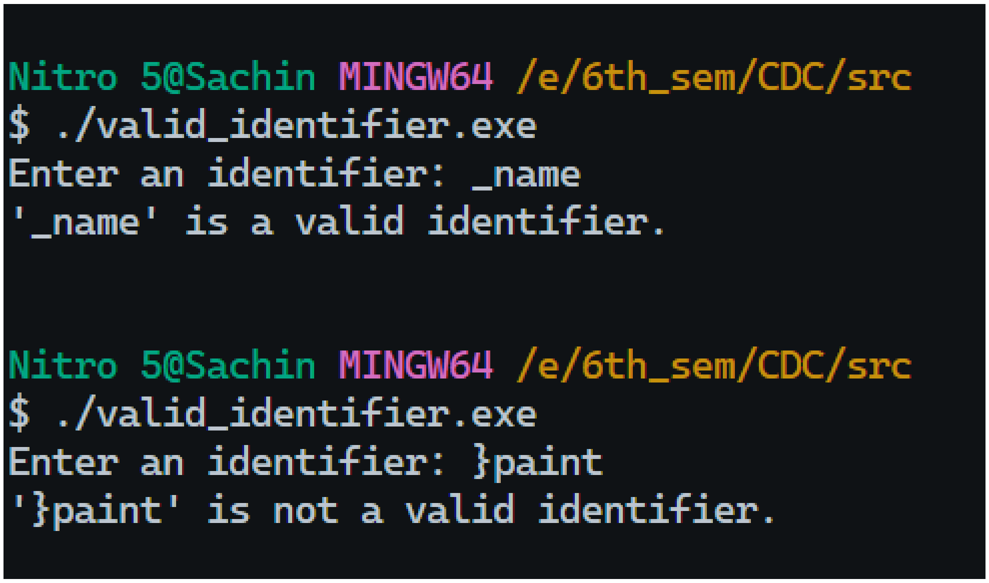
```
    while (top != -1)
        postfix[k++] = pop();
    postfix[k] = '\0';
}

// Generate three-address code from postfix
void generate3AC(const char *postfix)
{
    char stack2[MAX][MAX];
    int top2 = -1;
    char temp[MAX];

    for (int i = 0; postfix[i]; i++)
    {
        if (isalnum(postfix[i]))
        {
            char str[2] = {postfix[i], '\0'};
            strcpy(stack2[++top2], str);
        }
        else
        {
            char op2[MAX], op1[MAX];
            strcpy(op2, stack2[top2--]);
            strcpy(op1, stack2[top2--]);
            sprintf(temp, "t%d", tempVar++);
            printf("%s = %s %c %s\n", temp, op1, postfix[i], op2);
            strcpy(stack2[++top2], temp);
        }
    }
}

int main()
{
    char infix[MAX], postfix[MAX];
    printf("Enter infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);
    printf("\nThree-address code:\n");
    generate3AC(postfix);
    return 0;
}
```



```
Nitro 5@Sachin MINGW64 /e/6th_sem/CDC/src
$ ./Infixto3AddressCode.exe
Enter infix expression: a-b*c-d*e

Three-address code:
t1 = b * c
t2 = a - t1
t3 = d * e
t4 = t2 - t3
```

Write a program for identifier validation.

```
#include <ctype.h>
#include <stdio.h>
#include <string.h>
int isValidIdentifier(char *str)
{
    if (!isalpha(str[0]) && str[0] != '_')
    {
        return 0;
    }
    for (int i = 1; i < strlen(str); i++)
    {
        if (!isalnum(str[i]) && str[i] != '_')
        {
            return 0;
        }
    }
    return 1;
}

int main()
{
    char identifier[100];
    printf("Enter an identifier: ");
    scanf("%s", identifier);
    if (isValidIdentifier(identifier))
    {
        printf("'%s' is a valid identifier.\n", identifier);
    }
    else
    {
        printf("'%s' is not a valid identifier.\n", identifier);
    }

    return 0;
}
```



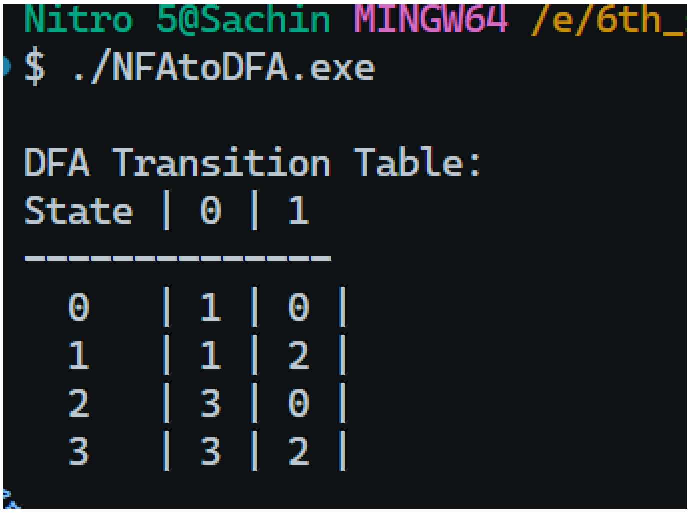
```
Nitro 5@Sachin MINGW64 /e/6th_sem/CDC/src
$ ./valid_identifier.exe
Enter an identifier: _name
'_name' is a valid identifier.


Nitro 5@Sachin MINGW64 /e/6th_sem/CDC/src
$ ./valid_identifier.exe
Enter an identifier: }paint
'}paint' is not a valid identifier.
```

Write a program that constructs the NFA to DFA.

```
#include <stdio.h>

#include <stdlib.h>


#define STATES 10

#define SYMBOLS 2


int nfa[STATES][SYMBOLS][STATES];

int dfa[1 << STATES][SYMBOLS];

int nfa_states, dfa_states = 0;


int is_state_in_set(int *set, int size, int state)

{

    for (int i = 0; i < size; i++)

        if (set[i] == state)

            return 1;

    return 0;

}


int add_state(int *set, int size, int state)

{

    if (!is_state_in_set(set, size, state))

        set[size++] = state;

    return size;

}
```

```
int compare_sets(int *a, int a_size, int *b, int b_size)
{
    if (a_size != b_size)
        return 0;
    for (int i = 0; i < a_size; i++)
        if (!is_state_in_set(b, b_size, a[i]))
            return 0;
    return 1;
}

int set_in_dfa_states(int dfa_state_sets[1 << STATES][STATES], int *set, int size)
{
    for (int i = 0; i < dfa_states; i++)
        if (compare_sets(dfa_state_sets[i], STATES, set, size))
            return i;
    return -1;
}

void convert_nfa_to_dfa()
{
    int dfa_state_sets[1 << STATES][STATES] = {0};
    int dfa_set_sizes[1 << STATES] = {0};

    dfa_state_sets[0][0] = 0; // Start with NFA state 0
    dfa_set_sizes[0] = 1;
    dfa_states = 1;
```

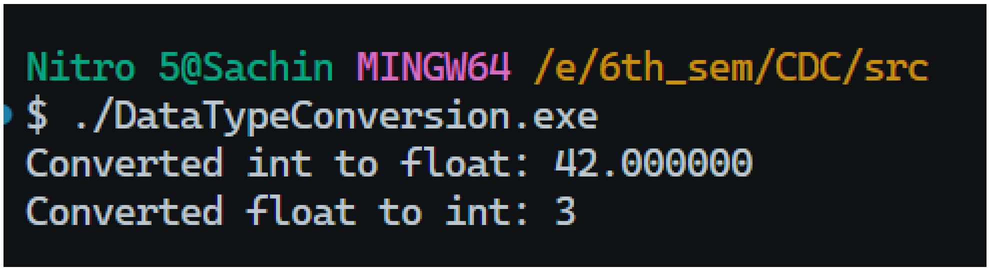
```
for (int i = 0; i < dfa_states; i++)
{
    for (int s = 0; s < SYMBOLS; s++)
    {
        int new_set[STATES] = {0}, new_size = 0;
        for (int j = 0; j < dfa_set_sizes[i]; j++)
        {
            int nfa_state = dfa_state_sets[i][j];
            for (int k = 0; k < STATES; k++)
            {
                if (nfa[nfa_state][s][k])
                    new_size = add_state(new_set, new_size, k);
            }
        }
        int existing = set_in_dfa_states(dfa_state_sets, new_set, new_size);
        if (existing == -1)
        {
            for (int j = 0; j < new_size; j++)
                dfa_state_sets[dfa_states][j] = new_set[j];
            dfa_set_sizes[dfa_states] = new_size;
            dfa[i][s] = dfa_states;
            dfa_states++;
        }
        else
        {
            dfa[i][s] = existing;
        }
        }
    }
}

void print_dfa()
{
    printf("\nDFA Transition Table:\n");
    printf("State | 0 | 1\n");
    printf("--------------\n");
    for (int i = 0; i < dfa_states; i++)
        printf("  %d   | %d | %d\n", i, dfa[i][0], dfa[i][1]);
}

int main()
{
    nfa_states = 3;

    // Example NFA
    // State 0 --0--> 0,1
    nfa[0][0][0] = 1;
    nfa[0][0][1] = 1;

    // State 0 --1--> 0
    nfa[0][1][0] = 1;

    // State 1 --1--> 2
    nfa[1][1][2] = 1;
    // State 2 --0--> 2
    nfa[2][0][2] = 1;


    convert_nfa_to_dfa();
    print_dfa();
    return 0;
}
```



```
Nitro 5@Sachin MINGW64 /e/6th_
$ ./NFAtoDFA.exe

DFA Transition Table:
State | 0 | 1
--------------
  0   | 1 | 0 |
  1   | 1 | 2 |
  2   | 3 | 0 |
  3   | 3 | 2 |
```

Write a program for data type conversion (int to float, float to int).

```
#include <stdio.h>

int main()
{
    int intVar = 42;
    float floatVar = 3.14;

    // Convert int to float
    float convertedFloat = (float)intVar;
    printf("Converted int to float: %f\n", convertedFloat);

    // Convert float to int
    int convertedInt = (int)floatVar;
    printf("Converted float to int: %d\n", convertedInt);

    return 0;
}
```



```
Nitro 5@Sachin MINGW64 /e/6th_sem/CDC/src
$ ./DataTypeConversion.exe
Converted int to float: 42.000000
Converted float to int: 3
```

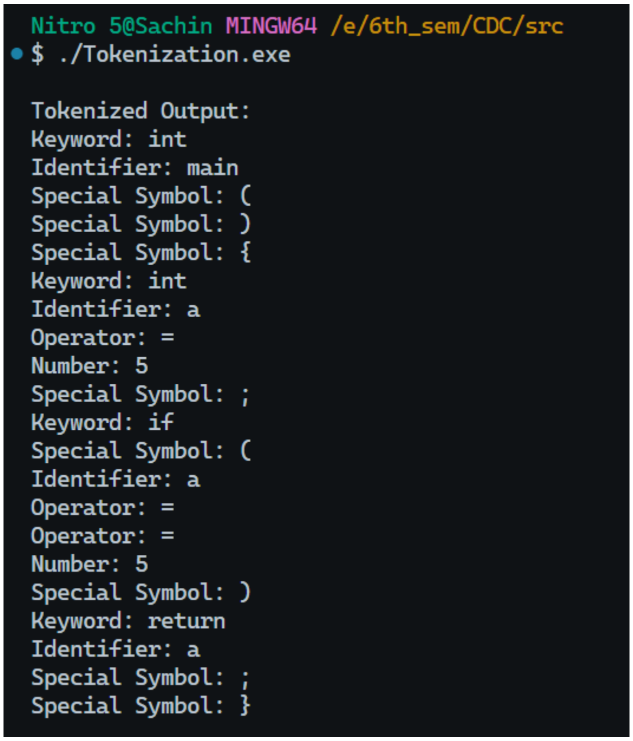
Write a program for tokenization (checking whether keyword, identifier, operator, etc.).

```
#include <ctype.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>


const char *keywords[] = {"int", "float", "if", "else", "while", "for", "return"};

const int num_keywords = 7;


const char *operators = "+-*/=";


const char *special_symbols = "{}();,";


int is_keyword(const char *str)

{

    for (int i = 0; i < num_keywords; i++)

    {

        if (strcmp(str, keywords[i]) == 0)

        {

            return 1;

        }

    }

    return 0;

}


int is_operator(char ch)
```

```
{
    for (int i = 0; operators[i] != '\0'; i++)
    {
        if (ch == operators[i])
        {
            return 1;
        }
    }
    return 0;
}

int is_special_symbol(char ch)
{
    for (int i = 0; special_symbols[i] != '\0'; i++)
    {
        if (ch == special_symbols[i])
        {
            return 1;
        }
    }
    return 0;
}

void tokenize(const char *input)
{
    int i = 0;
    char token[100];
```

```
while (input[i] != '\0')
{
    while (isspace(input[i]))
    {
        i++;
    }

    if (input[i] == '\0')
        break;

    int token_pos = 0;

    if (isalpha(input[i]) || input[i] == '_')
    {
        token[token_pos++] = input[i++];
        while (isalnum(input[i]) || input[i] == '_')
        {
            token[token_pos++] = input[i++];
        }
        token[token_pos] = '\0';

        if (is_keyword(token))
        {
            printf("Keyword: %s\n", token);
        }
        else
        {
            printf("Identifier: %s\n", token);
        }
    }
    else if (isdigit(input[i]))
    {
        token[token_pos++] = input[i++];
        while (isdigit(input[i]) || input[i] == '.')
        {
            token[token_pos++] = input[i++];
        }
        token[token_pos] = '\0';
        printf("Number: %s\n", token);
    }
    else if (is_operator(input[i]))
    {
        token[0] = input[i];
        token[1] = '\0';
        printf("Operator: %c\n", input[i]);
        i++;
    }
    else if (is_special_symbol(input[i]))
    {
        token[0] = input[i];
        token[1] = '\0';
        printf("Special Symbol: %c\n", input[i]);
        i++;
```

```
        }
        else
        {
            printf("Invalid character: %c\n", input[i]);
            i++;
        }
    }
}

int main()
{
    const char *code = "int main() { int a = 5; if (a == 5) return a; }";
    printf("\nTokenized Output:\n");
    tokenize(code);

    return 0;
}
```



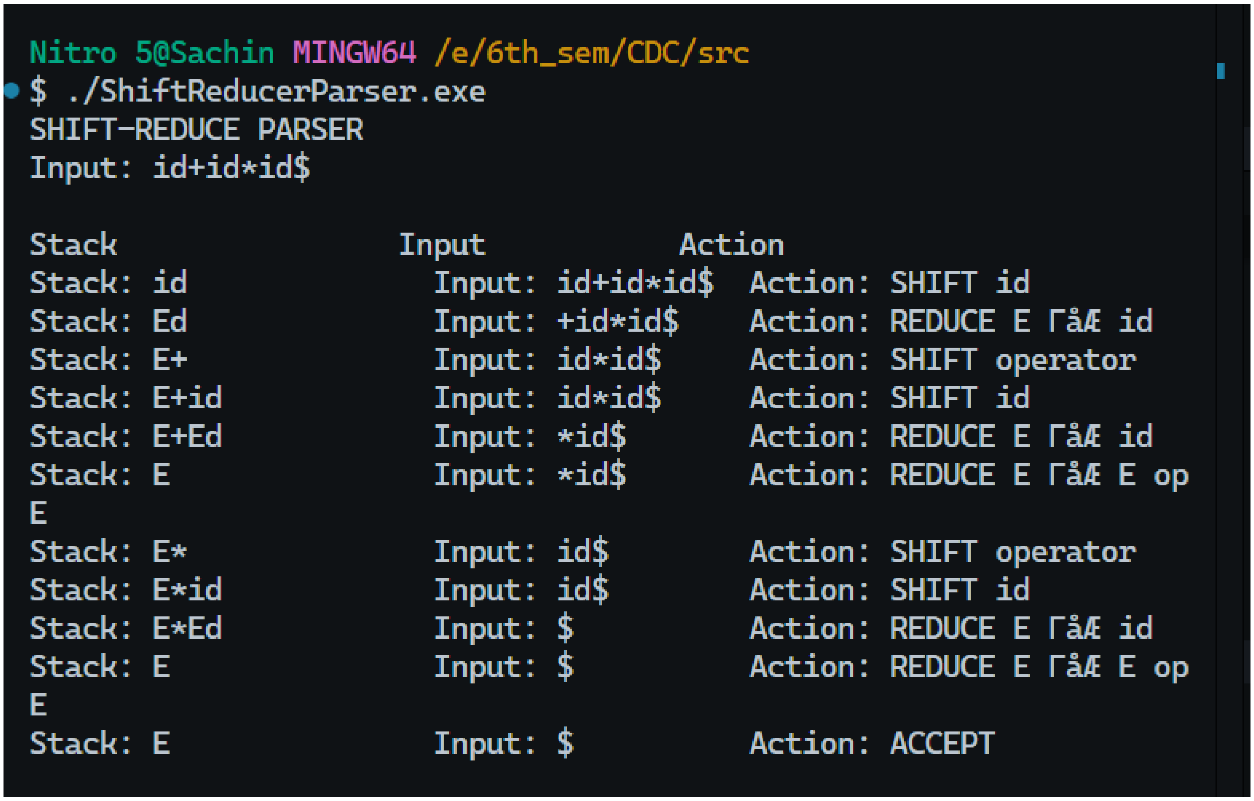
```
Nitro 5@Sachin MINGW64 /e/6th_sem/CDC/src
$ ./Tokenization.exe

Tokenized Output:
Keyword: int
Identifier: main
Special Symbol: (
Special Symbol: )
Special Symbol: {
Keyword: int
Identifier: a
Operator: =
Number: 5
Special Symbol: ;
Keyword: if
Special Symbol: (
Identifier: a
Operator: =
Operator: =
Number: 5
Special Symbol: )
Keyword: return
Identifier: a
Special Symbol: ;
Special Symbol: }
```

Write a program for the shift-reduce parser for the input string `id+id*id`:

E => E+E | E*E | (E) | id

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

char input[20] = "id+id*id$";
char stack[40];
int top = -1, i = 0;

void printstack(const char *action)
{
    printf("Stack: %-15s Input: %-10s Action: %s\n", stack, &input[i], action);
}

void replace_id_with_E()
{
    if (top > 0 && stack[top] == 'd' && stack[top - 1] == 'i')
    {
        stack[--top] = 'E'; // Replace 'd' with 'E'
        printstack("REDUCE E → id");
    }
}

void reduce_E_op_E()
{
    while (1)
    {
        if (top >= 2 && stack[top - 2] == 'E' && (stack[top - 1] == '+' || stack[top - 1] == '*') && stack[top] == 'E')
        {
            top -= 2; // remove op and E
            stack[top] = 'E';
            stack[top + 1] = '\0';
            printstack("REDUCE E → E op E");
        }
        else
            break;
    }
}

int main()
{
    printf("SHIFT-REDUCE PARSER\nInput: %s\n\n", input);
    printf("%-20s %-15s %s\n", "Stack", "Input", "Action");

    while (input[i] != '\0')
    {
        if (input[i] == 'i' && input[i + 1] == 'd')
        {
            stack[++top] = 'i';
            stack[++top] = 'd';
            stack[top + 1] = '\0';
            printstack("SHIFT id");
            i += 2;
            replace_id_with_E();
            reduce_E_op_E();
        }
        else if (input[i] == '+' || input[i] == '*')
        {
            stack[++top] = input[i++];
            stack[top + 1] = '\0';
            printstack("SHIFT operator");
        }
        else if (input[i] == '$')
        {
            reduce_E_op_E();
            if (strcmp(stack, "E") == 0)
            {
                printstack("ACCEPT");
            }
            else
            {
                printstack("ERROR");
            }
            break;
        }
        else
        {
            printstack("ERROR");
            break;
        }
```

```
    }


    return 0;
}
```



```
Nitro 5@Sachin MINGW64 /e/6th_sem/CDC/src
$ ./ShiftReducerParser.exe
SHIFT-REDUCE PARSER
Input: id+id*id$

Stack                Input           Action
Stack: id              Input: id+id*id$  Action: SHIFT id
Stack: Ed              Input: +id*id$    Action: REDUCE E ΓåÆ id
Stack: E+              Input: id*id$     Action: SHIFT operator
Stack: E+id            Input: id*id$     Action: SHIFT id
Stack: E+Ed            Input: *id$       Action: REDUCE E ΓåÆ id
Stack: E               Input: *id$       Action: REDUCE E ΓåÆ E op
E
Stack: E*              Input: id$        Action: SHIFT operator
Stack: E*id            Input: id$        Action: SHIFT id
Stack: E*Ed            Input: $          Action: REDUCE E ΓåÆ id
Stack: E               Input: $          Action: REDUCE E ΓåÆ E op
E
Stack: E               Input: $          Action: ACCEPT
```