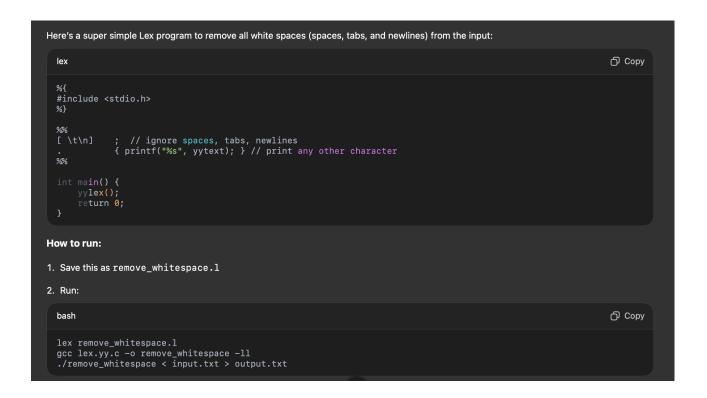
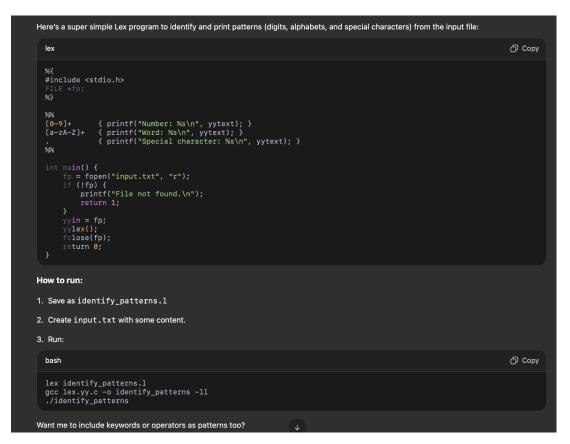
1.1) Write a lex program whose output is same as input.

```
Code:
 %{
 #include <stdio.h>
 %}
 . { printf("%s", yytext); }
 \n { printf("\n\n"); }
 int main() {
  yylex();
  return 0;
 Output:
[kazarani@MBA 1a % lex 1a.l
kazarani@MBA 1a % gcc lex.yy.c -ll
[kazarani@MBA 1a % ./a.out
hi this is Hussain
hi this is Hussain
this is a test
this is a test
```

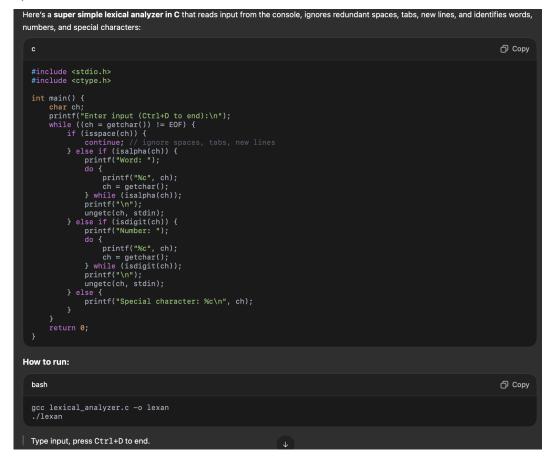
1.2) Write a lex program which removes white spaces from its input file



2.1) Write a lex program to identify the patterns in the input file.



2.2) Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines.



3.1) Simulate First and Follow of a Grammar.

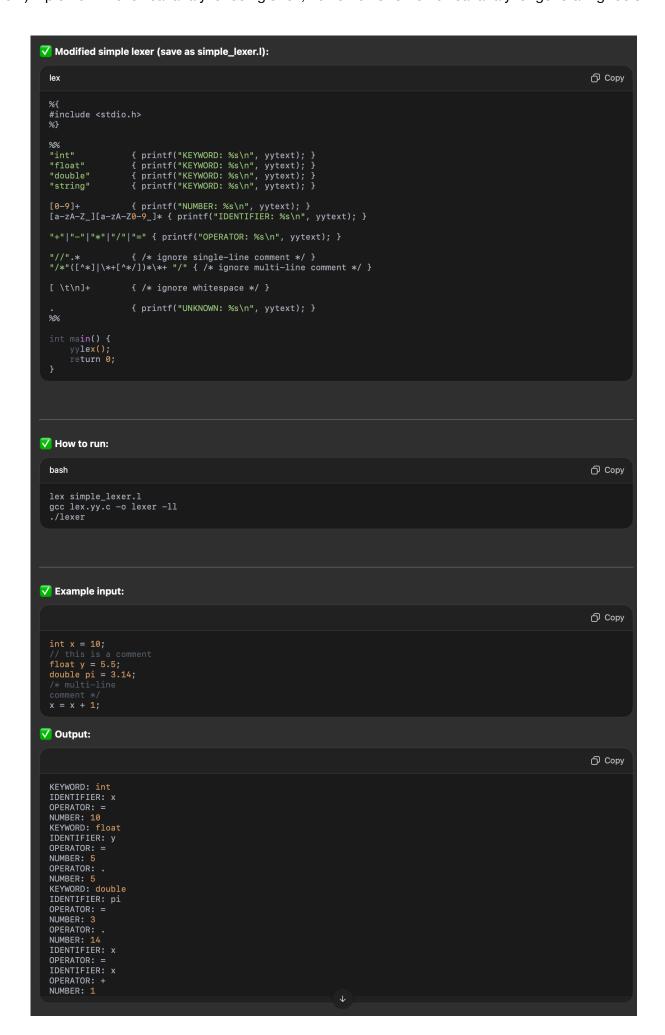
```
Perfect! Here's a super simple C program that:
• Takes grammar productions as input (like S->AB)
• Calculates FIRST sets by scanning productions and applying rules (only for single non-terminals and terminals)

    Calculates FOLLOW sets by looking for non-terminals on the right-hand side

    Works for simple grammars (no left recursion or indirect recursion)

Code (minimal logic-based):
                                                                                                                                                                                                                                Д Сору
    #include <ctype.h>
#include <string.h>
   char productions[10][10];
int count;
   void findFirst(char c) {
   if (!isupper(c)) {
      printf("%c ", c);
      return;
}
           for (int i = 0; i < count; i++) {
   if (productions[i][0] == c) {
      // If production is like A -> e or A->a or A->X...
   if (productions[i][3] == '\0') { // single char after ->
      printf("%c ", productions[i][2]);
   } else {
      findFirst(productions[i][2]);
   }
}
    void findFollow(char c) {
            // Follow of start symbol
if (c == productions[0][0]) {
   printf("$ ");
           for (int i = 0; i < count; i++) {
  for (int j = 2; j < strlen(productions[i]); j++) {
    if (productions[i][j] = c) {
        if (productions[i][j+1] != '\0') {
        findFirst(productions[i][j+1]);
    }
}</pre>
                                find rist;
} else {
   findFollow(productions[i][0]);
   int main() {
   printf("Enter number of productions: ");
   scanf("%d", &count);
   printf("Enter productions (example: S->aB):\n");
           for (int i = 0; i < count; i++) {
    scanf("%s", productions[i]);</pre>
            for (int i = 0; i < count; i++) {
   printf("FIRST(%c): ", productions[i][0]);
   findFirst(productions[i][0]);
   printf("\n");</pre>
            for (int i = 0; i < count; i++) {
   printf("FOLLOW(%c): ", productions[i][0]);
   findFollow(productions[i][0]);
   printf("\n");</pre>
            return 0;
Example Input:
                                                                                                                                                                                                                                Сору
   Enter number of productions: 3
S->AB
A->a
B->b
Output:
                                                                                                                                                                                                                                🗇 Сору
   FIRST(S): a b
FIRST(A): a
FIRST(B): b
FOLLOW(S): $
FOLLOW(A): b
FOLLOW(B):
```

3.2)Implement the lexical analyzer using JLex, flex or lex or other lexical analyzer generating tools



4.2) Construct a recursive descent parser for an expression



```
▼ Simplified LL(1) simulation:
• Only one stack (array).
· Just push expansions and pop terminals.
• No error recovery, just stops on invalid input.
· Displays stack and input step by step.
▼ Super simple code:
                                                                                                                                                                     О Сору
  #include <stdio.h>
#include <string.h>
  char input[100], stack[100];
int top = -1, i = 0;
  void push(char c) {
    stack[++top] = c;
  void pop() {
   top--;
  void display() {
    printf("\nstack: ");
    for (int k = 0; k <= top; k++)
        printf("%c", stack[k]);
    printf("\tinput: %s", &input[i]);
}</pre>
  int main() {
    printf("Enter input string ending with $ (ex: i+i*i$): ");
    scanf("%s", input);
        push('$');
push('E');
        while (stack[top] != '$' || input[i] != '$') {
    display();
              char topStack = stack[top];
char current = input[i];
              if (topStack == current) {
    // Terminal match
                     pop();
i++;
              else if (topStack == 'E') {
    pop();
    push('e'); push('T');
              else if (topStack == 'e') {
                     pop();
if (current == '+') { push('e'); push('T'); push('+'); }
              else if (topStack == 'T') {
    pop();
    push('t'); push('F');
               else if (topStack == 't') {
                     pop();
if (current == '*') { push('t'); push('F'); push('*'); }
                    pop();
if (current == 'i') push('i');
else if (current == '(') { push(')'); push('E'); push('('); }
                    printf("\nError\n");
                     return 0;
        display();
printf("\n\nSuccess: String Accepted\n");
Example output (clean and simple):
```

6.1) Write a program to perform loop unrolling.

```
Yep! Here's a much simpler version of loop unrolling that still shows the logic clearly:

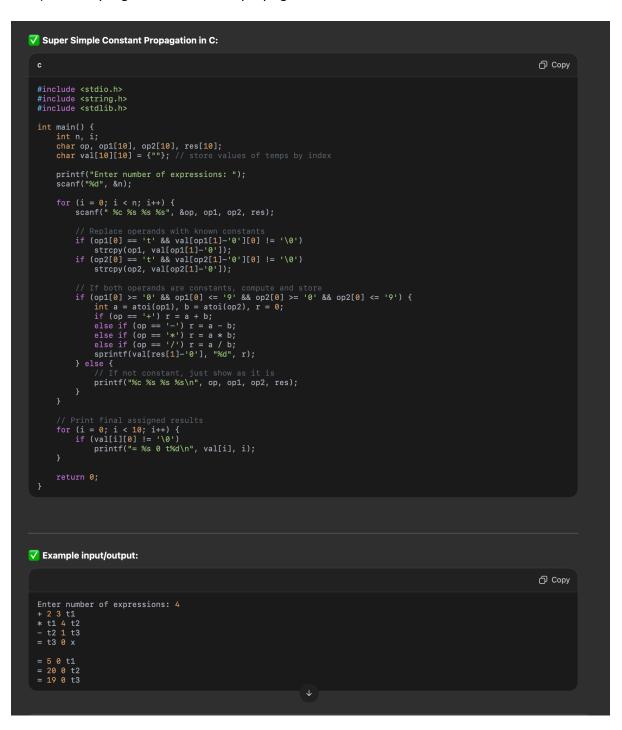
▼ Super Simple Loop Unrolling Program (Fixed Unroll by 4):

                                                                                                                                                                                                                      О Сору
   #include <stdio.h>
int main() {
   int n, i;
   printf("Enter number of iterations: ");
   scanf("%d", &n);
          int unroll = 4;
int repeat = n / unroll;
int leftover = n % unroll;
          // Unrolled part
for (i = 0; i < repeat * unroll; i += unroll) {
    printf("process(%d)\n", i);
    printf("process(%d)\n", i + 1);
    printf("process(%d)\n", i + 2);
    printf("process(%d)\n", i + 3);
}</pre>
          // Remaining iterations
for (int j = 0; j < leftover; j++) {
    printf("process(%d)\n", i + j);</pre>
Sample Output:
                                                                                                                                                                                                                      () Сору
   Enter number of iterations: 10
   process(0)
process(1)
   process(4)
process(5)
   process(6)
   process(9)
```

Explanation:

- It takes a simple loop (like for (i=0; i<n; i++) sum = sum + a[i];)</pre>
- Unrolls it by a fixed factor (e.g., 2)
- Outputs the unrolled version.

6.2) Write a program for constant propagation



13) Write a C program to identify whether a given line is a comment or not.

```
Experiment-13
  13) Write a C program to identify whether a given line is a comment or not.
     Code:
     #include <stdio.h>
     #include <conio.h>
     #include <string.h>
     int main()
       char com[30];
       int i = 2, a = 0, n;
       printf("\nEnter statement: ");
       gets(com);
       n = strlen(com);
       if (com[0] == '/')
         if (com[1] == '/')
           printf("\nIt is a comment");
         else if (com[1] == '*')
           for (i = 2; i < n; i++)
              if (com[i] = '*' \&\& com[i+1] = '/') in eering College (A))
                printf("\nIt is a comment");
                a = 1;
                break;
           if (a == 0)
             printf("\nIt is not a comment");
         else
           printf("\nIt is not a comment");
       else
         printf("\nIt is not a comment");
       getch();
       return 0;
ADITYA\ UNIVERSITY\ (\textit{Formerly Aditya Engineering College (A)})
                                                                      Roll No: 22A91A0
                             ADITYA UNIVERSITY
Exp No:
Date:
                                                                                  Page
                                   (Formerly Aditya Engineering College (A))
     Output:
   Enter statement:/Compiler Design
   It is not a comment
   Enter statement://Compiler Design
   It is a comment
   Enter statement:/*Compiler Design*/
   It is a comment
```

14) Write a C program to simulate lexical analyzer for validating operators.

```
Experiment-14
14) Write a C program to simulate lexical analyzer for validating operators.
  Code:
  #include <stdio.h>
  #include <string.h>
  const char *operators[] = {
    "+", "-", "*", "/", "%", "=", "==", "!=", "<", ">", "<=", ">=",
     "&&", "||", "!", "&", "|", "^", "<<", ">>"
  };
  int is Valid Operator (char *input) {
     int numOperators = sizeof(operators) / sizeof(operators[0]);
     for (int i = 0; i < numOperators; i++)
       if (strcmp(input, operators[i]) == 0)
         return 1;
     return 0;
  }
  int main() {
    char input[10];
printf("Enter an operator: ");
     scanf("%s", input); ormerly Aditya Engineering College (A))
     if (isValidOperator(input))
       printf("Valid Operator: %s\n", input);
     else
       printf("Invalid Operator!\n");
     return 0;
  Output:
 Enter an operator: &&
 Valid Operator: &&
 Enter an operator: 2
 Invalid Operator!
```