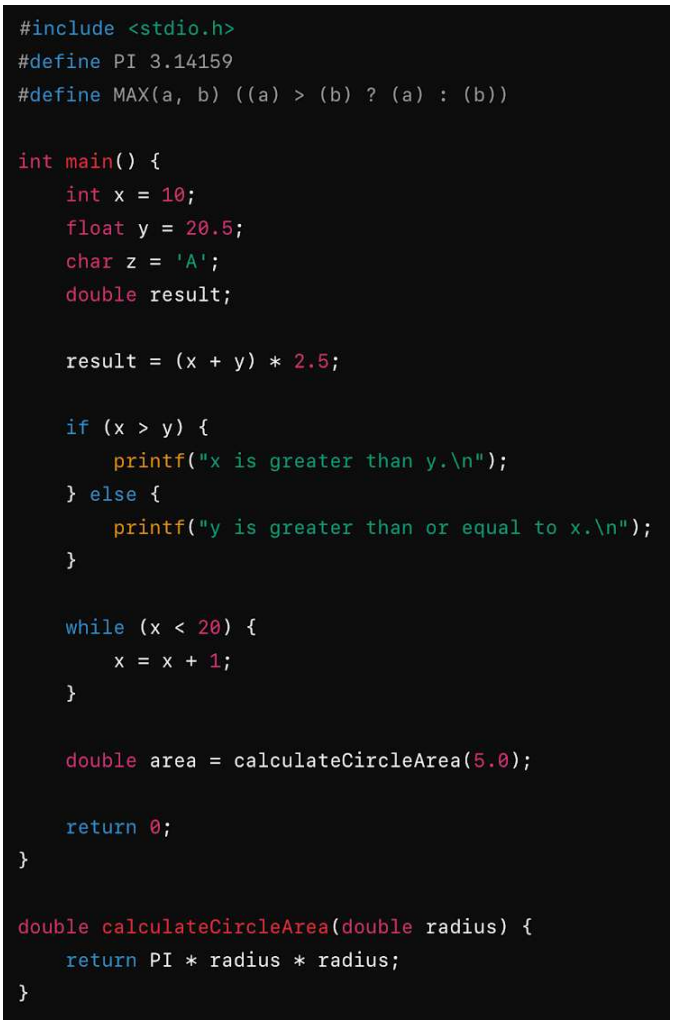
4) Dado o código-fonte abaixo, realize a análise léxica identificando os lexemas e tokens presentes. Liste os identificadores encontrados e construa a tabela de lexemas/classes/ expressões regulares, bem como a tabela de símbolos correspondente.

```
#include <stdio.h>
#define PI 3.14159
#define MAX(a, b) ((a) > (b) ? (a) : (b))

int main() {
   int x = 10;
   float y = 20.5;
   char z = 'A';
   double result;

   result = (x + y) * 2.5;

   if (x > y) {
      printf("x is greater than y.\n");
   } else {
      printf("y is greater than or equal to x.\n");
   }

   while (x < 20) {
      x = x + 1;
   }

   double area = calculateCircleArea(5.0);

   return 0;
}

double calculateCircleArea(double radius) {
   return PI * radius * radius;
}
```



A) Identifique os lexemas e classifique-os em suas respectivas classes.

Classe	Lexemas
Diretivas de pré-processador	#include, #define
Arquivo de cabeçalho	stdio.h
Palavras-chave (keywords)	int, float, char, double, return, if, else, while
Identificadores (nomes de variáveis ou funções)	PI, MAX, a, b, main, x, y, z, result, calculateCircleArea, radius, printf
Constantes numéricas (literais)	3.14159, 10, 20.5, 2.5, 5.0, 20, 1, 0
Constantes de caractere	'A'
Operadores	=, +, *, ?, :
Delimitadores / Pontuação	(,), {, }, ;, >, <, ,
Constantes de cadeias (Strings literais)	"x is greater than y.\n", "y is greater than or equal to x.\n"

B) Construa uma tabela com as classes de tokens, os lexemas correspondentes e as expressões regulares que os identificam.

Classe (Token)	Lexemas	Expressão Regular
PREP1	#include, #define,	#[a-z]$^{+}$
PREP2	stdio.h	[a-z]$^{+}$.[a-z]$^{+}$
INT	Int	[0-9]$^{+}$
REAL	float, double	[0-9]$^{+}$. [0-9]$^{+}$
CHAR	Char	[a-zA-Z]
COND	if, else	(if | else)
LOOP	while	(while)
ID	PI, MAX, a, b, main, x, y, z, result, calculateCircleArea, radius, return, printf	[a-zA-Z][a-zA-Z0-9]*
NUM_I	10, 20, 1, 0	[0-9]$^{+}$
NUM_R	3.14159, 20.5, 2.5, 5.0	[0-9]$^{+}$. [0-9]$^{+}$
CHARC	'A'	[a-zA-Z]
OPE	=, +, *, ?, >=, :	(= | + | * | ? |:)
DELIM	(,), {, }, ;, , ., >, <, ,	((|) | { | } | ; | . | > | < | ,)
STR	"x is greater than y.\n", "y is greater than or equal to x.\n”	(\".*?\")

C) Construa uma tabela de símbolos contendo os identificadores encontrados, seus tipos, valores iniciais, escopo e onde foram declarados.

Identificador	Tipo	Valor Inicial	Escopo	Local de Declaração
PI	#define (constante)	3.14159	Global	Linha 2 (diretiva de pré-processador)
MAX	Macro (função genérica)	((a) > (b) ? (a) : (b))	Global	Linha 3 (diretiva de pré-processador)
main	int (função)	—	Global	Linha 5 (assinatura da função)
x	int	10	Local (main)	Linha 6
y	float	20.5	Local (main)	Linha 7
z	char	'A'	Local (main)	Linha 8
result	double	indefinido	Local (main)	Linha 9
area	double	indefinido	Local (main)	Linha 19
calculateCircleArea	double (função)	—	Global	Linha 25 (definição da função)
radius	double (parâmetro)	recebido na chamada	Local (calculateCircleArea)	Linha 25

D) Justifique a importância das macros no contexto da análise léxica.

As macros são importantes na análise léxica porque permitem substituir identificadores simbólicos por valores ou expressões reais antes da análise. Isso reduz redundância, melhora a clareza semântica do código e garante consistência de tokens. Em essência, elas funcionam como uma camada de abstração que facilita tanto a escrita do programa quanto o trabalho do compilador.

E) Gere a Cadeia de Tokens.

<PREP1, '#include'> <DELIM, '<'> <PREP2, 'stdio.h'> <DELIM, '>'>
<PREP1, '#define'> <ID, 'PI'> < NUM_R, '3.14159'>
<PREP1, '#define'> <ID, 'MAX'> <DELIM, '('> <ID, 'a'> <DELIM, ','> <ID, 'b'> <DELIM, ')'>
<DELIM, '('> <DELIM, '('> <ID, 'a'> <DELIM, ')'><DELIM, '>'> <DELIM, '('> <ID, 'b'> <DELIM, ')'>
<DELIM, '?'> <DELIM, '('><ID, 'a'> <DELIM, ')'> <DELIM, ':'> <DELIM, '('> <ID, 'b'> <DELIM, ')'>
<DELIM, ')'>

<INT, 'int'> <ID, 'main'> <DELIM, '('> <DELIM, ')'> <DELIM, '{'>
<INT, 'int'> <ID, 'x'> <OPE, '='> < NUM_I, '10'> <DELIM, ';'>
<REAL, 'float'> <ID, 'y'> <OPE, '='> < NUM_R, '20.5'> <DELIM, ';'>
<CHAR, 'char'> <ID, 'z'> <OPE, '='> < CHARC, 'A'> <DELIM, ';'>
<REAL, 'double'> <ID, 'result'> <DELIM, ';'>

<ID, 'result'> <OPE, '='> <DELIM, '('> <ID, 'x'> <OPE, '+'> <ID, 'y'> <DELIM, ')'>
<OPE, '*'> < NUM_R, '2.5'> <DELIM, ';'>

<COND, 'if'> <DELIM, '('> <ID, 'x'> <OPE, '>'> <ID, 'y'> <DELIM, ')'> <DELIM, '{'>
<ID, 'printf'> <DELIM, '('> <STR , "x is greater than y.\n"> <DELIM, ')'> <DELIM, ';'>
<DELIM, '}'> <COND, 'else'> <DELIM, '{'>
<ID, 'printf'> <DELIM, '('> <STR , "y is greater than or equal to x.\n"> <DELIM, ')'> <DELIM, ';'>
<DELIM, '}'>

<LOOP, 'while'> <DELIM, '('> <ID, 'x'> <OPE, '<'> < NUM_I, '20'> <DELIM, ')'> <DELIM, '{'>
<ID, 'x'> <OPE, '='> <ID, 'x'> <OPE, '+'> < NUM_I, '1'> <DELIM, ';'>
<DELIM, '}'>

<REAL, 'double'> <ID, 'area'> <OPE, '='> <ID, 'calculateCircleArea'>
<DELIM, '('> < NUM_R, '5.0'> <DELIM, ')'> <DELIM, ';'>

<ID, 'return'> < NUM_I, '0'> <DELIM, ';'>

<DELIM, '}'>

<REAL, 'double'> <ID, 'calculateCircleArea'> <DELIM, '('> <REAL, 'double'> <ID, 'radius'> <DELIM, ')'> <DELIM, '{'>
<ID, 'return'> <ID, 'PI'> <OPE, '*'> <ID, 'radius'> <OPE, '*'> <ID, 'radius'> <DELIM, ';'>
<DELIM, '}'>