Compiler Construction Assignment Phase 1

Group 1		**Group 2**	
Mohit Gupta	2012A7PS021P	M Sreeharsha	2012A7PS036P
Ronil Pancholia	2012C6PS629P	G Sai Charan	2012A7PS105P

Language features

The language is meant to be used by biologist for protein sequence analysis in Bioinformatics. The language is strongly typed and uses block level static scoping. Some features are :

1. **Keywords, Identifiers and Literals**

   Keywords : functionmain, function, main, void, int, float, protein, seq, repeat, from, to, write, length, read, return, condition, otherwise, end.

   Identifiers : Identifiers must start with an alphabet followed by any alphanumeric character

   Identifiers cannot have same names as any of the keywords.

   Literals : integer, float or string literals.

2. **Data types**

   Primitive data types : integer, float

   Derived data types :

   - protein : It stores a sequence of nucleotides (characters) in a protein sequence.
   - seq : A derived data type containing an array of exactly 3 characters.

   Array declaration syntax : `type arrayName [arraySize]`

3. **Operators and operation defined on each data type**

   - Operators for primitive data types :
     Arithmetic operators : +, - , *, /, %
     Relational operators : >, <, =
     Assignment operator : is
     Logical operator : or, and
   - Operators for derived data types :
     Assignment operator : is
     Cut operator : # (used to cut the given protein by the specified sequence)
     Example : `cut is p # s`

4. **Function**

   Every program should have a main function and it should be the last function of the program.

   Syntax : `function returnType functionName ( parameters )`

   Possible return types are int and float. Return statement is compulsary. Syntax : `return value`

   0 or more parameters can be passed to a function.

   For calling functions, syntax : `functionName( arguments )`

5. **Conditional statement**

   Three types of conditional statements depending on some logical expression is allowed.

   1. Condition 2. Condition Otherwise 3. Condition Otherwise-Condition Otherwise

6. **Iterative statements**

   For iteration of a block of statements, repeat keyword is to be used with from and to keywords.

   Syntax for n iterations : `repeat` *`variable`* `from 1 to n`

7. **I/O operations.**

Reading input from user and printing output to screen is possible using read() and write() functions.

8. **Expression**

Expressions can have identifiers with operators. Evaluation for arithmetic operators will be done according to the precedence / > * > + > -. This precedence can be overridden by using parentheses.

9. **Assignment statement**

Assignment can be done using assignment operator *is*.Types of assignment statements :

For int, float, protein, seq. usage example : `a is 3`

For arrays, usage example : `a[5] is {1,2,3,4,5}`

Lexical units

Pattern	**Token**	**Purpose**
is	TK_AssignOp	Assignment operator
[1-9][0-0]*	TK_Num	Integer Number
“*”	TK_StrLit	String literals
[a-zA-z][a-zA-z0-9]*	TK_id	Identifier
[0-9]+\.[0-9]+	TK_float	Float literal
#	TK_Cut	To cut a protein by the specified sequence
(	TK_Lpar	Left parentheses
)	TK_Rpar	Right parentheses
{	TK_Lbrace	For opening a block of statements
}	TK_Rbrace	For closing a block of statements
[	TK_Lsqbrac	Used to access array elements
]	TK_Rsqbrac	Used to access array elements
,	TK_Comma	Used to seperate elements
and	TK_AND	Logical AND operator
or	TK_OR	Logical OR operator
+	TK_Plus	Addition operator
-	TK_Minus	Substraction operator
/	TK_Divide	Division operator

*	TK_Multiply	Multiplication operator
%	TK_Mod	Modulo operator
=	TK_Compare	Operator to check if two entites are equal
>	TK_LT	Operator to check if left entity is less than right
<	TK_GT	Operator to check if left entity is greater than right
!=	TK_NotEqual	Operator to check if two entites are equal
function	TK_func	Keyword for declaring a non-main function
void	TK_void	Used as return type for returning NULL
int	TK_int	Integer data type
float	TK_float	Floating point data type
protein	TK_protein	protein data type
seq	TK_seq	seq data type
main	TK_main	Keyword for declaring main function
repeat	TK_repeat	Used to run a code block multiple times
from	TK_from	Used to run a code block multiple times
to	TK_to	Used to run a code block multiple times
write	TK_write	Function to print output on screen
read	TK_read	Function to print output on screen
length	TK_length	Function to get length of specified protein
return	TK_return	Used to return value from a function
condition	TK_cond	Used to branching
otherwise	TK_other	Used as alternate branch
end	TK_end	Used to end program

LL1 Grammar

<program> ==> <functions> <main_function>
<functions> ==> <function1> TK_NEWLINE <functions> | e
<function1> ==> <function_signature> TK_NEWLINE <function_body>
<function_signature> ==> TK_FUNC <type> TK_ID TK_LPAR <params> TK_RPAR
<type> ==> TK_VOID | TK_INT | TK_FLOAT | TK_PROTEIN | TK_SEQ
<params> ==> <param1> | e
<param1> ==> <type> TK_ID <squarebracket1> <param>
<param> ==> TK_COMMA <param1> | e
<squarebracket1> ==> TK_LSQBRAC TK_RSQBRAC | e
<main_function> ==> TK_MAIN TK_LPAR TK_RPAR TK_NEWLINE <function_body>
<function_body> ==> TK_LBRACE TK_NEWLINE <stmt> TK_RETURN <Expr1> TK_NEWLINE TK_RBRACE
<squarebracket> ==> TK_LSQBRAC TK_INTEGERLITERAL TK_RSQBRAC | e
<stmt> ==> <type> TK_ID <squarebracket> TK_NEWLINE <stmt> | <loop> <stmt> | <ifelse> <stmt> | <IO> <stmt> | TK_END TK_NEWLINE <stmt> | TK_ID <X> | e
<X> ==> TK_ASSIGNOP <stmt1> TK_NEWLINE <stmt> | TK_LSQBRAC TK_ID TK_RSQBRAC TK_ASSIGNOP <stmt1> TK_NEWLINE <stmt> | <more> TK_NEWLINE <stmt>
<stmt1> ==> <Expr> | <arrayinit>
<arrayinit> ==> TK_LBRACE <array> TK_RBRACE
<array> ==> TK_ID <array1>
<array1> ==> <array> | e
<IO> ==> TK_READ TK_LPAR <L> TK_RPAR TK_NEWLINE | TK_WRITE TK_LPAR <L> TK_RPAR TK_NEWLINE
<L> ==> TK_ID <L1> | TK_STRINGLITERAL
<L1> ==> TK_LSQBRAC TK_ID TK_RSQBRAC | e
<loop> ==> TK_REPEAT TK_ID TK_FROM <Expr> TK_TO <Expr> TK_NEWLINE TK_LBRACE TK_NEWLINE <stmt> TK_RBRACE TK_NEWLINE
<ifelse> ==> TK_CONDITION TK_LPAR <logicalexpr> TK_RPAR TK_NEWLINE TK_LBRACE TK_NEWLINE <stmt> TK_RBRACE TK_NEWLINE <elseclause>
<elseclause> ==> TK_OTHERWISE <otherwise1> | e
<otherwise1> ==> <condition_otherwise> | TK_NEWLINE TK_LBRACE TK_NEWLINE <stmt> TK_RBRACE TK_NEWLINE
<condition_otherwise> ==> TK_CONDITION TK_LPAR <logicalexpr> TK_RPAR TK_NEWLINE TK_LBRACE TK_NEWLINE <stmt> TK_RBRACE TK_NEWLINE <elseclause>
<logicalexpr> ==> <Expr> <relop> <Expr> <logicop>
<relop> ==> TK_COMPARE | TK_GT | TK_LT | TK_NOTEQUAL
<logicop> ==> <lops> <logicalexpr> | e
<lops> ==> TK_AND | TK_OR
<Expr1> ==> <Expr> | e

<Expr> ==> <Expr2> <Expr3>
<Expr3> ==> <AS> <Expr> | e
<AS> ==> TK_PLUS | TK_MINUS
<Expr2> ==> <mulexp> <mulexp1>
<mulexp1> ==> TK_MULTIPLY <mulexp> | e
<mulexp> ==> <divexp> <divexp1>
<divexp1> ==> TK_DIVIDE <divexp> | e
<divexp> ==> <mod> <mod1>
<mod1> ==> TK_MOD <mod> | e
<mod> ==> <cut> <cut1>
<cut1> ==> TK_CUT <cut> | e
<cut> ==> TK_STRINGLITERAL | TK_FLOATLITERAL | TK_INTEGERLITERAL | TK_ID <ext> | TK_LPAR <Expr> TK_RPAR | TK_LENGTH <more>
<ext> ==> <more> | TK_LSQBRAC TK_ID TK_RSQBRAC | e
<more> ==> TK_LPAR <more1>
<more1> ==> TK_ID <args> TK_RPAR | TK_RPAR
<args> ==> TK_COMMA TK_ID <args> | e

Test Cases

Test Case 1

```
main()
{
    int A[3]
    int B[3]
    repeat i from 0 to 3
    {
        A[i] is 12*i
        B[i] is 2*i
    }
    int sumA
    int sumB
    repeat i from 0 to 3
    {
        sumA is sumA + A[i]
        sumB is sumB + B[i]
    }
    write("Sum of 1st Array = ")
    write(c)
    write(" Sum of 2nd Array = ")
    write(d)
    write("Total Sum = ")
    int z
```

```
    z is sumA + sumB
    write(z)
    return
}
```

Test Case 2

```
main()
{
   int i
   int j
   read(i)
   read(j)
   int c
   int d
   c is i*j + 3
   d is i*(j + 3)
   write("Precedence check : i*j+3 = ")
   write(c)
   write(" i*(j+3) = ")
   write(d)
}
```

Test Case 3

```
main()
{
    protein DNA[20]
    DNA is "AUGGCUUAGUA"
    int i
    i is length(DNA)
    int j
    repeat j from 0 to i
    {
        condition(DNA[i] = "T")
        {
            write("U")
        }
        otherwise
        {
            write(DNA[i])
        }
    }
    return
}
```

Test Case 4

```
main()
```

```
{
    protein P[20]
    P is "AAGTCCAGGATGCA"
    int i
    repeat i from 0 to length(P)
    {
        condition(P[i]="T")
        {
            write("DNA")
            end
        }
        otherwise
        {
            write("RNA")
            end
        }
    }
    write("Neither DNA nor RNA")
    return
}
```

Test Case 5

```
function void left(protein P[],int index)
{
    int i
    write("left part ")
    repeat i from 0 to index
    {
        write(P[i])
    }
    write(" ")
    return
}
function void right(protein P[],int index)
{
    int i
    write("right part ")
    repeat i from index+3 to length(P)
    {
        write(P[i])
    }
    write(" ")
    return
}
main()
{
    protein P[20]
```

```
    P is "AGGACTTAGATACCAAG"
    seq s[3]
    s is "CTT"
    int index
    index is P # S
    left(P,index)
    right(P,index)
    return
}
```

Derivations

Test Case 1

<program>
=> <functions> <main_function>
=> <main_function>
=> functionmain void main() newline <function_body>
=> functionmain void main()
{
<stmt> return <Expr1>
}
=>functionmain void main()
{
<type> id <squarebracket>
<stmt> return <Expr1>
}
=>functionmain void main()
{
int id[literal]
<stmt> return <Expr1>
}
=>functionmain void main()
{
int id[literal]
<type> id <squarebracket>
<stmt> return <Expr1>
}
=>functionmain void main()
{
int id[literal]
int id[literal]
<stmt> return <Expr1>
}
=>functionmain void main()
{
int id[literal]
int id[literal]

```
 <loop><stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   <stmt>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id<X>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is <stmt1>
   <stmt>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is <stmt1>
   <stmt>
   }
   <stmt>   return <Expr1>
}=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
```

```
 {
  id[id] is <Expr>
  <stmt>
  }
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
 {
  id[id] is <Expr2><Expr3>
  <stmt>
  }
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
 {
  id[id] is <mulexp><mulexp1>
  <stmt>
  }
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
 {
  id[id] is <divexp><mulexp1>
  <stmt>
  }
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
 {
  id[id] is <mod>*<expr2>
  <stmt>
  }
```

<stmt> return <Expr1>
}
=>functionmain void main()
{
int id[literal]
int id[literal]
repeat id from <Expr> to <Expr>
{
id[id] is <cut>*id
<stmt>
}
<stmt> return <Expr1>
}
=>functionmain void main()
{
int id[literal]
int id[literal]
repeat id from <Expr> to <Expr>
{
id[id] is id<ext>*id
<stmt>
}
<stmt> return <Expr1>
}
=>functionmain void main()
{
int id[literal]
int id[literal]
repeat id from <Expr> to <Expr>
{
id[id] is id*id
<stmt>
}
<stmt> return <Expr1>
}
=>functionmain void main()
{
int id[literal]
int id[literal]
repeat id from <Expr> to <Expr>
{
id[id] is id*id
id<X>
}
<stmt> return <Expr1>
}
=>functionmain void main()
{

```
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
  id[id] is id*id
  id[id] is <stmt1>
  <stmt>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is <stmt1>
   <stmt>
   }
   <stmt>   return <Expr1>
}=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is <Expr>
   <stmt>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is <Expr2><Expr3>
   <stmt>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
```

```
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is <mulexp><mulexp1>
   <stmt>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is <divexp><mulexp1>
   <stmt>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is <mod>*<expr2>
   <stmt>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is <cut>*id
   <stmt>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
```

```
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id<ext>*id
   <stmt>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   <stmt>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id <squarebracket>
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
```

```
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id <squarebracket>
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
  <loop><stmt>   return <Expr1>
}
=>functionmain void main()
```

```
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   <stmt>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id<x>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is <stmt1>
   id<x>
```

```
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is <stmt1>
   id is <stmt1>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is <Expr>
   id is <Expr>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
```

```
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is <Expr2>+<Expr3>
   id is <Expr2>+<Expr3>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is <mulexp><mulexp1>+<Expr3>
   id is <mulexp><mulexp1>+<Expr3>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is <divexp><divexp1>+<Expr3>
   id is <divexp><divexp1>+<Expr3>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
```

```
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is <mod><mod1>+<Expr3>
   id is <mod><mod1>+<Expr3>
   }
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is <cut><cut1>+<Expr3>
   id is <cut><cut1>+<Expr3>
   }
   <stmt>   return <Expr1>
}=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id<ext>+id<ext>
   id is id<ext>+id<ext>
   }
```

<stmt> return <Expr1>
}
=>functionmain void main()
{
int id[literal]
int id[literal]
repeat id from <Expr> to <Expr>
{
id[id] is id*id
id[id] is id*id
}
<type> id
<type> id
repeat id from <Expr> to <Expr>
{
id is id+id[id]
id is id+id[id]
}
<stmt> return <Expr1>
}
=>functionmain void main()
{
int id[literal]
int id[literal]
repeat id from <Expr> to <Expr>
{
id[id] is id*id
id[id] is id*id
}
<type> id
<type> id
repeat id from <Expr> to <Expr>
{
id is id+id[id]
id is id+id[id]
}
<IO><stmt> return <Expr1>
}
=>functionmain void main()
{
int id[literal]
int id[literal]
repeat id from <Expr> to <Expr>
{
id[id] is id*id
id[id] is id*id
}
<type> id

```
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   <IO><stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   <stmt>   return <Expr1>
}
```

=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   <IO><stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id

```
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   <IO><stmt>
   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
```

```
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   <IO><stmt>
   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
```

```
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   <type> id <squarebracket>
   <stmt>    return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
   <stmt>    return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
```

```
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id<x>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id<x> is <stmt1>
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
```

```
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id<x> is <Expr>
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id<x> is <Expr2>+<Expr3>
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
```

```
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id<x> is <mulexp><mulexp1>+<Expr>
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id<x> is <divexp><divexp1>+<Expr>
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
```

```
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id<x> is <mod><mod1>+<Expr>
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id<x> is <cut><cut1>+<Expr>
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
```

```
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id is id<ext>+id<ext>
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id is id+id
  <stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
```

```
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id is id+id
  <IO><stmt>   return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id is id+id
  write(<L>)
  <stmt>   return <Expr1>
}
=>functionmain void main()
```

```
{
 int id[literal]
 int id[literal]
 repeat id from <Expr> to <Expr>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from <Expr> to <Expr>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id is id+id
  write(<id>)
  <stmt>    return <Expr1>
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from id<ext> to id<ext>
  {
   id[id] is id*id
   id[id] is id*id
   }
  <type> id
  <type> id
   repeat id from id<ext> to id<ext>
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id is id+id
```

```
  write(<id>)
  <stmt>   return
}
=>functionmain void main()
{
 int id[literal]
 int id[literal]
 repeat id from id to id
  {
   id[id] is id*id
   id[id] is id*id
   }
   int id
   int id
   repeat id from id to id
   {
   id is id+id[id]
   id is id+id[id]
   }
   write(id)
   write(id)
   write(id)
   write(id)
   write(id)
   int id
  id is id+id
  write(<id>)
  <stmt>   return
}
```

```
=>  functionmain void main()
    {
        int A[3]
        int B[3]
        repeat i from 0 to 3
        {
              A[i] is 12*i
              B[i] is 2*i
         }
        int sumA
        int sumB
        repeat i from 0 to 3
        {
              sumA is sumA + A[i]
              sumB is sumB + B[i]
        }
        write("Sum of 1st Array = ")
        write(c)
        write(" Sum of 2nd Array = ")
```

```
        write(d)
        write("Total Sum = ")
        int e
        e is sumA + sumB
        write(e)
        return
}
```

Test Case 2

```
<program>
  => <functions> <main_function>
     =><main_function>
     =>functionmain void main ()
      <functionbody>
      =>functionmain void main ()
      {
      <stmt> return <Expr1>
      }
      =>functionmain void main ()
      {
      <stmt> return <Expr1>
      }
      =>functionmain void main ()
      {
      <type> id <squarebracket>
      <stmt> return <Expr1>
      }
      =>functionmain void main ()
      {
      int id
      <stmt> return <Expr1>
      }
      =>functionmain void main ()
      {
      int id
      <type> id <squarebracket>
      <stmt> return <Expr1>
      }
      =>functionmain void main ()
      {
      int id
      int id
      <stmt> return
      }
      =>functionmain void main ()
      {
      int id
```

int id
<IO><stmt>
return
}
=>functionmain void main ()
{
int id
int id
read(<L>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id<L1>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
<IO><stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(<L>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id<L1>)
<stmt> return
}

=>functionmain void main ()
{
int id
int id
read(id)
read(id)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
<type> id <squarebracket>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
<type> id <squarebracket>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
<stmt> return
}
=>functionmain void main ()
{

int id
int id
read(id)
read(id)
int id
int id
id<x> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is <stmt1>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is <Expr>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is <Expr2><Expr3>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id

int id
id is <Expr2>+<Expr>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is <mulexp> <mulexp1> +<Expr2><Expr3>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is <mulexp>* <mulexp> +<Expr2>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is <divexp>* <divexp> +<mulexp><mulexp1>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is <mod>* <mod1> +<divexp><divexp1>
<stmt> return
}

=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is <cut>* <cut1> +<mod><mod1>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id<ext>* id<ext> +<cut><cut1>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id<ext>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
<stmt> return
}
=>functionmain void main ()
{
int id
int id

read(id)
read(id)
int id
int id
id is id*id +id
id<x> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is <stmt1>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is <Expr>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is <Expr2><Expr3>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)

read(id)
int id
int id
id is id*id +id
id is <Expr2>+<Expr>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is <mulexp> <mulexp1> +<Expr2><Expr3>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is <mulexp>* <mulexp> +<Expr2>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is <divexp>* <divexp> +<mulexp><mulexp1>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)

read(id)
int id
int id
id is id*id +id
id is <mod>* <mod1> +<divexp><divexp1>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is <cut>* <cut1> +<mod><mod1>
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id<ext>* (<Expr>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(<Expr2><Expr3>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)

read(id)
int id
int id
id is id*id +id
id is id*(<mulexp><mulexp1>+<Expr>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(<divexp><divexp1>+<Expr2><Expr3>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(<mod><mod1>+<mulexp><mulexp1>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(<cut><cut1>+<divexp><divexp1>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)

read(id)
int id
int id
id is id*id +id
id is id*(id<ext>+<mod><mod1>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+<cut><cut1>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id<ext>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)

read(id)
int id
int id
id is id*id +id
id is id*(id+id)
<IO><stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id)
write(<L>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id)
write(id)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id)
write(id)
<IO><stmt> return
}
=>functionmain void main ()
{

int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id)
write(id)
write(<L>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id)
write(id)
write(id)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id)
write(id)
write(id)
<IO><stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id

id is id*id +id
id is id*(id+id)
write(id)
write(id)
write(<L>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id)
write(id)
write(id)
write(id)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id)
write(id)
write(id)
write(id)
<IO><stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id)
write(id)

write(id)
write(id)
write(<L>)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id)
write(id)
write(id)
write(id)
write(id)
<stmt> return
}
=>functionmain void main ()
{
int id
int id
read(id)
read(id)
int id
int id
id is id*id +id
id is id*(id+id)
write(id)
write(id)
write(id)
write(id)
return
}

=>
functionmain void main()
{
int i
int j
read(i)
read(j)
int c
int d
c is i*j + 3

```
    d is i*(j + 3)
    write("Precedence check : i*j+3 = ")
    write(c)
    write(" i*(j+3) = ")
    write(d)
}
```

Test Case 3

```
<program>
=> <functions> <main_function>
=> <main_function>
=> functionmain void main ( )
   <functionbody>
=> functionmain void main ( )
   {
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    <type> id <squarebracket>
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA <squarebracket>
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [literal]
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    id <X> return <Expr1>
   }
=> functionmain void main ( )
```

{
protein DNA [20]
DNA is <stmt1>
<stmt> return <Expr1>
}
=> functionmain void main ( )
{
protein DNA [20]
DNA is <Expr>
<stmt> return <Expr1>
}
=> functionmain void main ( )
{
protein DNA [20]
DNA is <Expr2> <Expr3>
<stmt> return <Expr1>
}
=> functionmain void main ( )
{
protein DNA [20]
DNA is <Expr2>
<stmt> return <Expr1>
}
=> functionmain void main ( )
{
protein DNA [20]
DNA is <mulexp> <mulexp1>
<stmt> return <Expr1>
}
=> functionmain void main ( )
{
protein DNA [20]
DNA is <mulexp>
<stmt> return <Expr1>
}
=> functionmain void main ( )
{
protein DNA [20]
DNA is <divexp> <divexp1>
<stmt> return <Expr1>
}
=> functionmain void main ( )
{
protein DNA [20]
DNA is <divexp>
<stmt> return <Expr1>
}
=> functionmain void main ( )

```
  {
   protein DNA [20]
   DNA is <mod> <mod1>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is <mod>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is <cut> <cut1>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is <cut>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is literal
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   <type> id <squarebracket>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int id <squarebracket>
   <stmt> return <Expr1>
```

```
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id <X> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is <stmt1>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is <Expr>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is <Expr2><Expr3>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is <Expr2>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
```

```
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is <mulexp> <mulexp1>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is <mulexp>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is <divexp> <divexp1>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is <divexp>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is <mod> <mod1>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is <mod>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
```

```
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is <cut> <cut1>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is <cut>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length <more>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (  <more1>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length ( id <args>  )
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
```

```
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   <type> id <squarebracket>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int id <squarebracket>
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int id
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   <loop> <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
```

```
    int i
    id is length (DNA)
    int j
    repeat id from <Expr> to <Expr>
    {
     <stmt>
    }
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from <Expr> to <Expr>
    {
     <stmt>
    }
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from <Expr2> <Expr3> to <Expr2> <Expr3>
    {
     <stmt>
    }
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from <Expr2> to <Expr2>
    {
     <stmt>
    }
    <stmt> return <Expr1>
   }
```

```
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from <mulexp> <mulexp1>to <mulexp> <mulexp1>
   {
    <stmt>
   }
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from <mulexp> to <mulexp>
   {
    <stmt>
   }
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from <divexp> <divexp1> to <divexp> <divexp1>
   {
    <stmt>
   }
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from <divexp> to <divexp>
   {
```

```
    <stmt>
   }
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from <mod> <mod1> to <mod> <mod1>
   {
    <stmt>
   }
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from <mod> to <mod>
   {
    <stmt>
   }
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from <cut> <cut1> to <cut> <cut1>
   {
    <stmt>
   }
   <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
```

```
    id is length (DNA)
    int j
    repeat j from <cut> to <cut>
    {
     <stmt>
    }
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from literal to literal
    {
     <stmt>
    }
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     <stmt>
    }
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     <ifelse> <stmt>
    }
    <stmt> return <Expr1>
   }
=> functionmain void main ( )
```

```
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    <ifelse> <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  (  <logicalexpr>  )
      {
       <stmt>
       }
    <elseclause> <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( <Expr> <relop> <Expr> <logicop> )
      {
       <stmt>
       }
    <elseclause> <stmt>
   }
    <stmt> return <Expr1>
  }

=> functionmain void main ( )
```

```
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( <Expr> = <Expr> <logicop> )
      {
       <stmt>
       }
    <elseclause> <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( <Expr> = <Expr> )
      {
       <stmt>
       }
    <elseclause> <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( <Expr2> <Expr3> = <Expr2> <Expr3> )
      {
       <stmt>
       }
    <elseclause> <stmt>
   }
```

```
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( <Expr2>  = <Expr2> )
      {
       <stmt>
       }
    <elseclause> <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( <mulexp> <mulexp1>  = <mulexp> <mulexp1> )
      {
       <stmt>
       }
    <elseclause> <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( <mulexp>  = <mulexp> )
      {
       <stmt>
```

```
      }
    <elseclause> <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( <divexp> <divexp1>  = <divexp> <divexp1> )
      {
       <stmt>
       }
    <elseclause> <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( <divexp> = <divexp> )
      {
       <stmt>
       }
    <elseclause> <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
```

```
     condition  ( <mod><mod1> = <mod><mod1> )
      {
       <stmt>
       }
     <elseclause> <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( <mod> = <mod> )
      {
       <stmt>
       }
     <elseclause> <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( <cut><cut1> = <cut><cut1> )
      {
       <stmt>
       }
     <elseclause> <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
```

```
    int j
    repeat j from 0 to i
    {
     condition  ( <cut> = <cut> )
       {
        <stmt>
        }
     <elseclause> <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( id <ext> = literal )
       {
        <stmt>
        }
     <elseclause> <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( id [id] = "T" )
       {
        <stmt>
        }
     <elseclause> <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
```

```
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( DNA[i] = "T" )
       {
        <stmt>
        }
     <elseclause> <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( DNA[i] = "T" )
       {
        <IO> <stmt>
        }
     <elseclause> <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( DNA[i] = "T" )
       {
        write ( <L> )
         <stmt>
        }
     <elseclause> <stmt>
    }
     <stmt> return <Expr1>
```

```
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( DNA[i] = "T" )
      {
       write ( id <L1>)
        <stmt>
       }
    <elseclause> <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( DNA[i] = "T" )
      {
       write ( id)
        <stmt>
       }
    <elseclause> <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( DNA[i] = "T" )
      {
```

```
        write ("U")
         <stmt>
        }
     <elseclause> <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( DNA[i] = "T" )
       {
        write ("U")
        }
     <elseclause> <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( DNA[i] = "T" )
       {
        write ("U")
        }
     otherwise <otherwise1> <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
```

```
    repeat j from 0 to i
    {
     condition  ( DNA[i] = "T" )
       {
        write ("U")
        }
     otherwise
       {
        <stmt>
        }
       <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( DNA[i] = "T" )
       {
        write ("U")
        }
     otherwise
       {
        <IO> <stmt>
        }
       <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( DNA[i] = "T" )
       {
        write ("U")
        }
```

```
     otherwise
       {
        <IO>
        }
       <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( DNA[i] = "T" )
       {
        write ("U")
        }
     otherwise
       {
        write ( <L> )
       }
       <stmt>
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( DNA[i] = "T" )
       {
        write ("U")
        }
     otherwise
       {
        write ( id <L1>)
       }
       <stmt>
    }
```

```
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( DNA[i] = "T" )
      {
       write ("U")
       }
    otherwise
      {
       write ( id[id])
      }
      <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
   int i
   id is length (DNA)
   int j
   repeat j from 0 to i
   {
    condition  ( DNA[i] = "T" )
      {
       write ("U")
       }
    otherwise
      {
       write ( DNA[i])
      }
      <stmt>
   }
    <stmt> return <Expr1>
  }
=> functionmain void main ( )
  {
   protein DNA [20]
   DNA is "AUGGCUUAGUA"
```

```
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( DNA[i] = "T" )
       {
        write ("U")
        }
     otherwise
       {
        write ( DNA[i])
       }
    }
     <stmt> return <Expr1>
   }
=> functionmain void main ( )
   {
    protein DNA [20]
    DNA is "AUGGCUUAGUA"
    int i
    id is length (DNA)
    int j
    repeat j from 0 to i
    {
     condition  ( DNA[i] = "T" )
       {
        write ("U")
        }
     otherwise
       {
        write ( DNA[i])
       }
    }
      return <Expr1>
   }
```

=>

```
functionmain void main()
{
    protein DNA[20]
    DNA is "AUGGCUUAGUA"
    int i
    i is length(DNA)
    int j
    repeat j from 0 to i
    {
        condition(DNA[i] = "T")
```

```
        {
            write("U")
        }
        otherwise
        {
            write(DNA[i])
        }
    }
    return
}
```

Test Case 4

```
<program>
=> <functions> <main_function>
    =><main_function>
    =>functionmain void main ( )
     <functionbody>
     =>functionmain void main ( )
     {
     <stmt> return <Expr1>
     }
     =>functionmain void main ( )
     {
     <type> id <squarebracket>
      <stmt> return
     }
     =>functionmain void main ( )
     {
     protein id [id]
      <stmt> return
     }
     =>functionmain void main ( )
     {
     protein id [id]
      id <X> return
     }
     =>functionmain void main ( )
     {
     protein id[id]
      id is <stmt1>
      <stmt> return
     }
     =>functionmain void main ( )
     {
     protein id[id]
      id is <Expr>
```

<stmt> return
}
=>functionmain void main ( )
{
protein id[id]
id is <Expr2><Expr3>>
<stmt> return
}
=>functionmain void main ( )
{
protein id[id]
id is <mulexp><mulexp1>
<stmt> return
}
=>functionmain void main ( )
{
protein id[id]
id is <divexp><divexp1>
<stmt> return
}
=>functionmain void main ( )
{
protein id[id]
id is <mod><mod1>
<stmt> return
}
=>functionmain void main ( )
{
protein id[id]
id is <cut><cut1>
<stmt> return
}
=>functionmain void main ( )
{
protein id[id]
id is id<ext>
<stmt> return
}
=>functionmain void main ( )
{
protein id[id]
id is id
<stmt> return
}
=>functionmain void main ( )
{
protein id[id]
id is id

```
 <type> id <squarebracket>
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 <loop><stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from <Expr> to <Expr>
 {
   <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from <Expr> to <Expr>
 {
   <ifelse> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from <Expr> to <Expr>
 {
   condition  (  <logicalexpr>  )
    {
```

```
     <stmt>
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from <Expr> to <Expr>
 {
   condition  (  <logicalexpr>  )
    {
     <IO> <stmt>
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from <Expr> to <Expr>
 {
   condition  (  <logicalexpr>  )
    {
     write(<L>)
     <stmt>
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from <Expr> to <Expr>
 {
   condition  (  <logicalexpr>  )
    {
     write(id)
     <stmt>
```

```
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from <Expr> to <Expr>
 {
   condition  (  <Expr> <relop> <Expr> <logicop>  )
    {
     write(id)
     end
     <stmt>
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from <Expr2><Expr3> to <Expr2><Expr3>
 {
   condition  (  <Expr2><Expr3> = <Expr2><Expr3>   )
    {
     write(id)
     end
     <stmt>
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from <mulexp><mulexp1> to <mulexp><mulexp1>
 {
   condition  (  <mulexp><mulexp1> = <mulexp><mulexp1>   )
    {
```

```
     write(id)
     end
     <stmt>
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from <divexp><divexp1> to <divexp><divexp1>
 {
   condition  (  <divexp><divexp1> = <divexp><divexp1>   )
    {
     write(id)
     end
     <stmt>
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from <mod><mod1> to <mod><mod1>
 {
   condition  (  <mod><mod1> = <mod><mod1>   )
    {
     write(id)
     end
     <stmt>
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from <cut><cut1> to <cut><cut1>
```

```
 {
   condition  (  <cut><cut1> = <cut><cut1>   )
    {
     write(id)
     end
     <stmt>
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id<ext> to length<more>
 {
   condition  (  id<ext> = id<ext>   )
    {
     write(id)
     end
     <stmt>
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(<more1>
 {
   condition  (  id[id] = id<  )
    {
     write(id)
     end
     <stmt>
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
```

```
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       condition  (  <logicalexpr>  )
        {
         <stmt>
          }
   <elseclause> <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       condition  (  <Expr> <relop> <Expr> <logicop>  )
        {
         <stmt>
```

```
        }
    <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       condition  (  <Expr2><Expr3>= <Expr2><Expr3>   )
        {
         <stmt>
          }
    <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       condition  (  <mulexp><mulexp1>= <mulexp><mulexp1>   )
        {
         <stmt>
          }
    <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
```

```
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       condition  (  <divexp><divexp1>= <divexp><divexp1>   )
        {
         <stmt>
          }
    <stmt>
 }
 <stmt> return
}
 =>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       condition  (  <mod><mod1>= <mod><mod1>   )
        {
         <stmt>
          }
    <stmt>
 }
 <stmt> return
}
 =>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
```

```
   }
      condition  (  <cut><cut1>= <cut><cut1>   )
       {
        <stmt>
         }
   <stmt>
 }
 <stmt> return
}
 =>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
      condition  (  id<ext>= id<ext>   )
       {
        <stmt>
         }
   <stmt>
 }
 <stmt> return
}
 =>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
      condition  (  id<[id]= id   )
       {
        <stmt>
         }
   <stmt>
 }
 <stmt> return
```

```
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       condition  (  id<[id]= id   )
        {
         <IO><stmt>
          }
    <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       condition  (  id<[id]= id   )
        {
         write(<L>)
         <stmt>
          }
    <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
```

```
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       condition  (  id<[id]= id   )
        {
         write(id)
         <stmt>
          }
    <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       condition  (  id<[id]= id   )
        {
         write(id)
         end
         <stmt>
          }
    <stmt>
 }
 <stmt> return
}
=>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
```

```
    }
       condition  (  id<[id]= id   )
        {
         write(id)
         end
          }
    <stmt>
 }
 <IO><stmt> return
}
 =>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       condition  (  id<[id]= id   )
        {
         write(id)
         end
          }
    <stmt>
 }
 write(<L>)
 <stmt> return
}
 =>functionmain void main ( )
{
protein id[id]
 id is id
 int id
 repeat id from id to length(id)
 {
   condition  (  id[id] = id)
    {
     write(id)
     end
    }
       condition  (  id<[id]= id   )
        {
         write(id)
         end
```

```
            }
        <stmt>
     }
     write(id)
     <stmt> return
    }
    =>functionmain void main ( )
    {
    protein id[id]
     id is id
     int id
     repeat id from id to length(id)
     {
       condition  (  id[id] = id)
        {
         write(id)
         end
        }
           condition  (  id<[id]= id   )
            {
             write(id)
             end
              }
        <stmt>
     }
     write(id)
      return
    }

 =>
functionmain void main()
{
    protein P[20]
    P is "AAGTCCAGGATGCA"
    int i
    repeat i from 0 to length(P)
    {
         condition(P[i]="T")
         {
           write("DNA")
           end
         }
         condition(P[i]="U")
        {
          write("RNA")
          end
         }
     }
```

```
    write("Neither DNA nor RNA")
    return
}
```

Test Case 5

<program>
=> <functions> <main_function>
=> <function1> <functions> <main_function>
=> <function_signature> <function_body> <functions> <main_function>
=> function void left (<params>) <function_body> <functions> <main_function>
=> function void left (<param1>) <function_body> <functions> <main_function>
=> function void left (protein p <square_bracket> <param>) <function_body> <functions> <main_function>
=> function void left (protein P[] <param>) <function_body> <functions> <main_function>
=> function void left (protein P[], <param1>) <function_body> <functions> <main_function>
=> function void left (protein P[], int index <square_bracket> <param>) <function_body> <functions>
<main_function>
=> function void left (protein P[], int index) <function_body> <functions> <main_function>
=> function void left (protein P[], int index)
  {
   <stmt> return <Expr1>
  }
  <functions> <main_function>
=> function void left (protein P[], int index)
  {
   <type> i
   <stmt> return <Expr1>
  }
  <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   <IO> <stmt> return <Expr1>
  }
  <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   write("left part ") <stmt> return <Expr1>
  }
  <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   write("left part ") <loop> <stmt> return <Expr1>
  }
  <functions> <main_function>

```
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from <Expr> to <Expr>
   {
    <stmt>
   }
   <stmt> return <Expr1>
  }
  <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from <Expr2> <Expr3> to <Expr2> <Expr3>
   {
    <stmt>
   }
   <stmt> return <Expr1>
  }
  <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from <mulexp> <mulexp1>  to <mulexp> <mulexp1>
   {
    <stmt>
   }
   <stmt> return <Expr1>
  }
  <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from <divexp> <divexp1>  to <divexp> <divexp1>
   {
    <stmt>
   }
   <stmt> return <Expr1>
  }
  <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
```

```
    repeat i from <mod> <mod1> to <mod> <mod1>

    {
     <stmt>
    }
    <stmt> return <Expr1>
   }
   <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from <cut> <cut1> to <cut> <cut1>

    {
     <stmt>
    }
    <stmt> return <Expr1>
   }
   <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index<ext>
    {
     <stmt>
    }
    <stmt> return <Expr1>
   }
   <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
     <stmt>
    }
    <stmt> return <Expr1>
   }
   <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
```

```
      <IO> <stmt>

    }
    <stmt> return <Expr1>
   }
   <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(<L>)
       <stmt>
    }
    <stmt> return <Expr1>
   }
   <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
       <stmt>
    }
    <stmt> return <Expr1>
   }
   <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     <IO> <stmt> return <Expr1>
   }
   <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
```

```
    }
     write(" ") <stmt> return <Expr1>
   }
   <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   <function1> <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   <function_signature> <function_body> <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
```

```
    write(" ")
    return
  }
  function void right (<params>) <function_body> <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index) <function_body> <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   <type> i
   <stmt> return <Expr1>
  } <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   <IO> <stmt> return <Expr1>
  } <functions> <main_function>
```

```
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write(<L>)
    <stmt> return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    <stmt> return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
```

```
    write("right part ")
    repeat i from <Expr> to <Expr>
    {
       <stmt>
    }
    return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from <Expr2><Expr3> to <Expr2><Expr3>
    {
       <stmt>
    }
    return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from <Expr2><AS><Expr>  to <mulexp> <mulexp1>
    {
       <stmt>
    }
    return <Expr1>
```

```
  } <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from <mulexp> <mulexp1>  + <Expr>  to <divexp> <divexp1>
   {
      <stmt>
   }
   return <Expr1>
  } <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from <divexp> <divexp1>  + <Expr>  to <mod><mod1>
   {
      <stmt>
   }
   return <Expr1>
  } <functions> <main_function>
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
```

```
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from <mod><mod1>  + <mod><mod1>  to <cut><cut1>
    {
       <stmt>
    }
    return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from <cut><cut1>  + <cut><cut1>  to length(<more>
    {
       <stmt>
    }
    return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
```

```
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(<args>)
    {
       <stmt>
    }
    return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       <stmt>
    }
    return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
```

```
      <IO><stmt>
    }
    return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(<L>)<stmt>
    }
    return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P<L1>)<stmt>
    }
    return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
```

```
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
       <stmt>
    }
    return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return <Expr1>
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
```

```
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   } <functions> <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   } <main_function>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
```

```
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   <function_body>
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
```

```
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      <type> P <squarebracket>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      Protein P <squarebracket>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
```

```
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     Protein P[20]
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     Protein P[20]
     id <X> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
```

```
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      Protein P[20]
      id is <stmt1>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      Protein P[20]
      id is <Expr>
      <stmt> return <Expr1>
```

```
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     Protein P[20]
     id is <Expr2> <Expr3>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
```

```
  }
  functionmain void main()
  {
     Protein P[20]
     id is <Expr2>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     Protein P[20]
     id is <mulexp> <mulexp1>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
```

```
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      Protein P[20]
      id is <mulexp>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      Protein P[20]
      id is <divexp> <divexp1>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
```

```
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      Protein P[20]
      id is <divexp>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      Protein P[20]
      id is <mod> <mod1>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
```

```
 int i
 write("left part ")
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   Protein P[20]
   id is <mod>
   <stmt> return <Expr1>
}
=> function void left (protein P[], int index)
{
 int i
 write("left part ")
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
```

```
    Protein P[20]
    id is <cut><cut1>
    <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
     write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
     write(P[i])
   }
   return
  }
  functionmain void main()
  {
    Protein P[20]
    id is <cut>
    <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
     write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
```

```
      write(P[i])
    }
    return
   }
   functionmain void main()
   {
     Protein P[20]
     P is "AGGACTTAGATACCAAG"
     <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
      write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
      write(P[i])
    }
    return
   }
   functionmain void main()
   {
     Protein P[20]
     P is "AGGACTTAGATACCAAG"
     <type> s <squarebracket>
     <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
      write(P[i])
    }
     write(" ")
     return
```

```
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s <squarebracket>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
```

```
 int i
 write("left part ")
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
   P is "AGGACTTAGATACCAAG"
   seq s
   s <X>
   <stmt> return <Expr1>
}
=> function void left (protein P[], int index)
{
 int i
 write("left part ")
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
```

```
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is <stmt1>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is <Expr>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
```

```
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is <Expr2><Expr3>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is <Expr2>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
```

```
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is <mulexp><mulexp1>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
```

```
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is <mulexp>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is <divexp><divexp1>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
```

```
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is <divexp>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is <mod><mod1>
     <stmt> return <Expr1>
  }
```

```
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is <mod>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
```

```
    return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is <cut><cut1>
     <stmt> return <Expr1>
  }

=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is <cut>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
```

```
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is "CTT"
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is "CTT"
```

```
      <type> id <squarebracket>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index <squarebracket>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
```

```
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index
      index<X> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
```

```
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index
      index is <stmt1>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
```

```
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is "CTT"
     int index
     index is <Expr>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is "CTT"
     int index
     index is <Expr2><Expr3>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
```

```
      write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
      write(P[i])
    }
    return
   }
   functionmain void main()
   {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is "CTT"
     int index
     index is <Expr2>
     <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
      write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
      write(P[i])
    }
    return
   }
   functionmain void main()
   {
```

```
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index
      index is <mulexp><mulexp1>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index
      index is <mulexp>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
```

```
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
   P is "AGGACTTAGATACCAAG"
   seq s[3]
   s is "CTT"
   int index
   index is <divexp><divexp>
   <stmt> return <Expr1>
}
=> function void left (protein P[], int index)
{
 int i
 write("left part ")
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
   P is "AGGACTTAGATACCAAG"
   seq s[3]
```

```
      s is "CTT"
      int index
      index is <divexp>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index
      index is <mod><mod1>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
```

```
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is "CTT"
     int index
     index is <mod>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is "CTT"
     int index
     index is <cut><cut1>
```

```
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index
      index is <cut> # <Expr2>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
```

```
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index
      index is P # <Expr2>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index
      index is P # <mulexp><mulexp1>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
```

```
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is "CTT"
     int index
     index is P # <mulexp>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
```

```
    }
    return
   }
   functionmain void main()
   {
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index
      index is P # <divexp><divexp1>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
       write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
       write(P[i])
    }
    return
   }
   functionmain void main()
   {
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index
      index is P # <divexp>
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
```

```
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
   P is "AGGACTTAGATACCAAG"
   seq s[3]
   s is "CTT"
   int index
   index is P # <mod><mod1>
   <stmt> return <Expr1>
}
=> function void left (protein P[], int index)
{
 int i
 write("left part ")
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
```

```
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is "CTT"
     int index
     index is P # <mod>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is "CTT"
     int index
     index is P # <cut><cut1>
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
```

```
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
   P is "AGGACTTAGATACCAAG"
   seq s[3]
   s is "CTT"
   int index
   index is P # <cut>
   <stmt> return <Expr1>
}
=> function void left (protein P[], int index)
{
 int i
 write("left part ")
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
```

```
    P is "AGGACTTAGATACCAAG"
    seq s[3]
    s is "CTT"
    int index
    index is P # s
    left<X>
    <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
    protein P[20]
    P is "AGGACTTAGATACCAAG"
    seq s[3]
    s is "CTT"
    int index
    index is P # s
    left<more>
    <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
```

```
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
   P is "AGGACTTAGATACCAAG"
   seq s[3]
   s is "CTT"
   int index
   index is P # s
   left(<more1>
   <stmt> return <Expr1>
}
=> function void left (protein P[], int index)
{
 int i
 write("left part ")
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
```

```
    P is "AGGACTTAGATACCAAG"
    seq s[3]
    s is "CTT"
    int index
    index is P # s
    left(<args>)
    <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
    protein P[20]
    P is "AGGACTTAGATACCAAG"
    seq s[3]
    s is "CTT"
    int index
    index is P # s
    left(P<args1>)
    <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
```

```
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
   P is "AGGACTTAGATACCAAG"
   seq s[3]
   s is "CTT"
   int index
   index is P # s
   left(P,<args>)
   <stmt> return <Expr1>
}
=> function void left (protein P[], int index)
{
 int i
 write("left part ")
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
```

```
    P is "AGGACTTAGATACCAAG"
    seq s[3]
    s is "CTT"
    int index
    index is P # s
    left(P,index)
    <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
    protein P[20]
    P is "AGGACTTAGATACCAAG"
    seq s[3]
    s is "CTT"
    int index
    index is P # s
    left(P,index)
    <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
```

```
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
   P is "AGGACTTAGATACCAAG"
   seq s[3]
   s is "CTT"
   int index
   index is P # s
   left(P,index)
   right<X> return <Expr1>
}
=> function void left (protein P[], int index)
{
 int i
 write("left part ")
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
```

```
    P is "AGGACTTAGATACCAAG"
    seq s[3]
    s is "CTT"
    int index
    index is P # s
    left(P,index)
    right<more>
    <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
    protein P[20]
    P is "AGGACTTAGATACCAAG"
    seq s[3]
    s is "CTT"
    int index
    index is P # s
    left(P,index)
    right(<more1>
    <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
```

```
      write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
      write(P[i])
    }
    return
   }
   functionmain void main()
   {
      protein P[20]
      P is "AGGACTTAGATACCAAG"
      seq s[3]
      s is "CTT"
      int index
      index is P # s
      left(P,index)
      right(<args>)
      <stmt> return <Expr1>
   }
=> function void left (protein P[], int index)
   {
    int i
    write("left part ")
    repeat i from 0 to index
    {
      write(P[i])
    }
     write(" ")
     return
   }
   function void right (protein P[],int index)
   {
    int i
    write("right part ")
    repeat i from index + 3  to length(P)
    {
      write(P[i])
    }
    return
   }
```

```
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is "CTT"
     int index
     index is P # s
     left(P,index)
     right(P,<args1>)
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
   write("left part ")
   repeat i from 0 to index
   {
      write(P[i])
   }
    write(" ")
    return
  }
  function void right (protein P[],int index)
  {
   int i
   write("right part ")
   repeat i from index + 3  to length(P)
   {
      write(P[i])
   }
   return
  }
  functionmain void main()
  {
     protein P[20]
     P is "AGGACTTAGATACCAAG"
     seq s[3]
     s is "CTT"
     int index
     index is P # s
     left(P,index)
     right(P,<args>)
     <stmt> return <Expr1>
  }
=> function void left (protein P[], int index)
  {
   int i
```

```
 write("left part ")
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
 }
 return
}
functionmain void main()
{
   protein P[20]
   P is "AGGACTTAGATACCAAG"
   seq s[3]
   s is "CTT"
   int index
   index is P # s
   left(P,index)
   right(P,index)
   return <Expr1>
}
```

```
=> function void left (protein P[], int index)
{
 int i
 write("left part ")
 repeat i from 0 to index
 {
    write(P[i])
 }
  write(" ")
  return
}
function void right (protein P[],int index)
{
 int i
 write("right part ")
 repeat i from index + 3  to length(P)
 {
    write(P[i])
```

```
 }
 return
}
functionmain void main()
{
   protein P[20]
   P is "AGGACTTAGATACCAAG"
   seq s[3]
   s is "CTT"
   int index
   index is P # s
   left(P,index)
   right(P,<args1>)
   return
}
```