CS 327
Group 15

BASIC TYPES:

Declaring variable type :- **#variable_type variable_name = variable_value;**

Variable_type for an integer is “I”
Variable_type for a float is “F”
Variable_type for a character is “C”
Variable_type for a string is “S”

COMPOUND TYPES:

Variable_type for an array is “A”
Variable_type for a list is “L”
Variable_type for a tuple is “T”

Comments :-
i) Single line comments: *// single line comment*
ii) Multi-line comments: *!! this is a*
multi-line
comment !!

Integer :-
Declaring Integer type: #I variable_name = varible_value;
Examples: #I x = 10;
#I int = -7;

Float :-
Declaring Float type : #I variable_name = varible_value;
Examples : #F y = 10.0;
#F pq = 12.56;

Syntax for operations :- opreand_1 operator operand_2;

i) Arithmetic Operators:

- **‘ + ‘** : performs addition between two integers
  Example : #I x = 10;
  #I int = 7;
  #F flo = 3.5;
  #I result = x + int;
  //result is 17
  #F result = x + int;

```
//result is 17.0
#I result = x + flo;
//result is 13
#F res = int + flo;
//result is 10.5
```

- **‘ - ’** : performs subtraction between integers

  Example :
```
#I x = 10;
#I int = 7;
#F flo = 3.5;
#I result = x - int;
//result is 3
#F result = x - int;
//result is 3.0
#I result = x - flo;
//result is 6
#F res = int - flo;
//result is 3.5
```

- **‘ * ‘ :** performs multiplication between integers

  Example :
```
#I x = 10;
#I int = 7;
#I result = x * int;
#F float = 10.0 * 7;
//float  is 70.0
//result is 70
```

- **‘ / ‘:** performs division between integers

  Example :
```
#I x = 10;
#I int = 7;
#I result = x / int;
//result is 1

#F x = 10.0;
#F int = 7;
#F result = x / int;
//result is 1.428571
```

- **‘%’:** gives remainder

  Example :
```
#I x = 10;
#I int = 7;
#I result = x % int;
//result is 3
```

ii)Unary Operators: {“-”,”!”,”++”,”--”}
// first two are type_1 operators and next two are type_2

Syntax for type_1 operators:
#variable_type variable_name = type_1_operatorvariable_value;
Example:
#B is_true = true;
#B result = !is_true; // result is false

syntax for type_2 operators:
#variable_type variable_name = variable_value;
variable_nametype_2_operator;

Example:
#I count = 5;
count++; // Increment count by 1
// Now, count is 6

iii)Comparison Operators: {“>”,”<”,”==”,”>=”,”<=”,”!=”}

Syntax: #B result_variable = operand_1 comparison_operator operand_2;
Examples : #I x = 10;
#I y = 5;
#B is_greater = x > y; *// is_greater is true*

#F a = 3.5;
#F b = 2.0;
#B is_equal = a == b; *// is_equal is false*

Logical Operators: {“&&”,”||”}

Syntax: #B result_variable = operand_1 logical_operator operand_2;

Examples : #B condition1 = true;
#B condition2 = false;
#B and_result = condition1 && condition2; *// and_result is false*
#B or_result = condition1 || condition2;   *// or_result is true*

Character:-
Declaring character type : #C variable_name = ‘character’;
Examples : #C char = ‘v’;
#C mn = ‘q’;

String :-

Declaring string type : #S variable_name = “string”;
Examples : #S str = “compiler”;
#S a = “qwerty”;

String operations:

i)Concatenating two strings: #S new_string = string_1.add(string_2);
//string_2 is added at the end of string_1
Examples : #S string_1 = “comp”;
#S string_2 = “iler”;
#S new_string = string_1.add(string_2);
//new_string is “compiler”

ii)Removing a part of the string:
#S new_string = string_1.cut(initial_index,final_index);
!! the new string will not contain a specific part of string_1 from mentioned initial_index to final-index.!!
Examples : #S string_1 = “Himasagar”;
#S new_string = string_1.cut(0,3);
//new_string is “sagar”

iii)Obtaining a part of string:
#S new_string = string_1.get(initial_index,final_index);
!! the new string will contain a specific part of string_1 from mentioned initial_index to final-index.!!
Example : #S string_1 = “udaykumar”;
#S new_string = string_1.get(4,8);
//new_string is “kumar”;

iv)Replacing a part of string:
#S new_string = string_1.edit(initial_index,final_index,replacing_string);
//the new-string contains string_1 with some part as replacing_string
Example : #S string_1 = “sujith”;
#S new_string = string_1.edit(2,5,”doku”);
//new_string is “sudoku”

v)Size of string: string_name.len();
// returns the size of string

Boolean :-
Declaring boolean type : #B variable_name = varible_value;
Examples : #B bool = true;

#B bool = false;

Print Statement :-
write(var1,var2,constant,"any expression");
!! variables, constants, or any expression that wants to be printed should be inserted inside the print function separated by commas !!
write(variable1\n,variable2);
// varaible1 and variable2 will be printed on two different lines

Lists :- #L list_name = [values separated by commas];

List operations :

i)Inserting to the end of the list : list_name.ladd(variable or variable value to be added);

ii)Removing from the list: list_name.lcut(initial_index, final_index);

iii)Obtaining a single element: list_name[index]; //here the list starts from 0-index

iv)size of the list: list_name.len(); //returns an integer

Tuples :- #T tuple_name = (element1, element2, …);

Example: #T my_tuple = (1, "apple", 3.14);

Tuple operations:

i)Accessing Elements: #I element_at_index = tuple_name[index];
Example: #I first_element = my_tuple[0];
#S second_element = my_tuple[1];
// first_element is 1 and second_element is apple

ii)Tuple Size: #I tuple_size = tuple_name.len();
Example: #I size_of_tuple = my_tuple.len();
//size_of_tuple is 3

ii)Tuple Slicing: #T sliced_tuple = tuple_name.slice(start_index, end_index);
Example: #T sliced_tuple = my_tuple.slice(1, 2);
//siced_tuple is (1, 'apple')

Arrays :- #A array_name[fixed_size] = [variables separated by commas];

Array operations:

i)Obtaining a single element: array_name[index]; *//here the array starts from 0-index*

ii)Modifying the elements: array_name[required_index] = new_value;

iii)Adding an element at the end: array_name.aadd(variable_value);
//this operation will be performed only if there is available memory otherwise the error will be reported that there is not enough space to perform the operation

iii)size of the array: array_name.len(); //returns an integer

Conditional Statements :-

```
if (condition1) {
        //execute this if the condition1 is true
}
elsif (condition2) {
        //execute this if the condition2 is true
}
else {
        //execute this if none of the above conditions are true
}
```

Note: There can be multiple or zero elsif conditions after and before else. We cannot use elsif and else before the if statement. We may choose to use the else statement or elsif statement at last.

Loops :-

```
loopy (condition) do {
        //execute this command until the condition doesn't fail
}
//similar to the while loop in C
```

- The syntax for the condition in loopy: 'variable_1 of any data type' 'comparison operator' 'variable_2 of the similar data type of variable_1' //float and integer type variables will be considered as similar data types
  Example :

```
repeat (expression) do {
        //execute this command until the condition doesn't fail
}
```

- The syntax for expression in repeat: declaring a variable; operation on the variable; condition the variable should be satisfying
  Example :

Functions :-

```
define function_name(variable declarations separated by commas) {
        //body of the function
        submit variable;
}
```

```
Example : define add_numbers(#I x, #I y) {
                #I sum = x + y;
                submit sum;
            }
```

!! To call the function, just specify the function's name with parameters following the above syntax !!

```
Example :  //calling the function
           #I result = add_numbers(5,7);
           //result is 12
```

Closures :-

```
define closure_name(captured_variables separated by commas) {
   // body of the closure
   submit result;  // optional: return a value
}
```

```
Example : define my_closure(#I x) {
                define func2(#I y) {
                        submit x*y;
                }
                submit func2;
            }
           #I mmm = my_closure(5);
           #I closure_result = mmm(3);
           // closure_result is 15
```

Mutable variables :-

```
#M mutable_var = initial_value;
```

```
// Update the value
mutable_var = new_value;
```

Example :
```
#M counter = 0;
// Increment the counter
counter = counter + 1;
```

Exceptions :-

```
try {
   // code that may raise an exception
   throw exception_type(message);  // throw an exception
}
catch (exception_type variable) {
   // handle the exception
   // variable contains information about the exception
}
```

Example :
```
try {
   #I divisor = 0;
   #F result = 10 / divisor;  // this may raise a division by zero exception
}
catch (#ExceptionDivideByZero e) {
   // handle the division by zero exception
   submit "Error: Division by zero";
}
```
.

Let expressions :-

```
let variable_name = expression1 in {
   // body of the let expression
   // you can use variable_name within this scope
   expression2;  // result of the let expression
}
```

Example :
```
#I x = 5;
let y = x * 2 in {
    #I z = y + 3;
    submit z;  // result is 13
}
```
!! the variable y is introduced within the let expression, and you can use it within the scope of the let expression !!

```
letter            ::= 'a' | 'b' | ... | 'z' | 'A' | 'B' | ... | 'Z'
digit             ::= '0' | '1' | ... | '9'

identifier        ::= letter (letter | digit)*
integer_constant  ::= digit+
float_constant    ::= digit+ '.' digit+
char_constant     ::= '\'' (letter | digit) '\''
string_constant   ::= '"' (letter | digit | '\s')* '"'

### Token Definitions:

variable_type     ::= 'I' | 'F' | 'C' | 'S' | 'A' | 'L' | 'T' | 'B' | 'M'
comparison_operator ::= '>' | '<' | '==' | '>=' | '<=' | '!='
logical_operator  ::= '&&' | '||'
unary_operator    ::= '-' | '!' | '++' | '--'

### Grammar Definitions:

program           ::= statement*

statement         ::= declaration_statement
                    | assignment_statement
                    | expression_statement
                    | if_statement
                    | while_loop
                    | repeat_loop
                    | print_statement
                    | function_definition
                    | closure_definition
                    | mutable_variable
                    | exception_handling
                    | let_expression

declaration_statement ::= '#' variable_type variable_name '=' expression ';'
assignment_statement ::= variable_name '=' expression ';'
expression_statement ::= expression ';'
if_statement      ::= 'if' '(' condition ')' '{' statement* '}'
                    ('elsif' '(' condition ')' '{' statement* '}')*
                    ('else' '{' statement* '}')?
while_loop        ::= 'loopy' '(' condition ')' 'do' '{' statement* '}'
repeat_loop       ::= 'repeat' '(' expression ')' 'do' '{' statement* '}'
print_statement   ::= 'write' '(' (expression (',' expression)*)? ')'
variable_type     ::= 'I' | 'F' | 'C' | 'S' | 'A' | 'L' | 'T' | 'B' | 'M'
condition         ::= expression comparison_operator expression
```

```
comparison_operator ::= '>' | '<' | '==' | '>=' | '<=' | '!='
logical_operator  ::= '&&' | '||'

expression        ::= term (('+' | '-' | '' | '/' | '%') term)
term              ::= variable
                    | constant
                    | '(' expression ')'
                    | unary_operator expression
constant          ::= integer_constant | float_constant | char_constant | string_constant
integer_constant  ::= digit+
float_constant    ::= digit+ '.' digit+
char_constant     ::= '\'' (letter | digit) '\''
string_constant   ::= '"' (letter | digit | '\s')* '"'
variable          ::= variable_name ('.' function_call)?

function_call     ::= function_name '(' (expression (',' expression)*)? ')'
function_name     ::= identifier
function_definition ::= 'define' function_name '(' parameter_list? ')' '{' statement*
'submit' expression ';}'

parameter_list    ::= variable_type variable_name (',' variable_type variable_name)*
closure_definition ::= 'define' closure_name '(' parameter_list? ')' '{' statement*
'submit' expression ';}'
closure_name      ::= identifier

mutable_variable  ::= '#M' variable_name '=' expression ';'

exception_handling ::= 'try' '{' statement* 'throw' exception_type '(' string_constant ')'
';' '}'
                    ('catch' '(' exception_type variable_name ')' '{' statement* '}')?

let_expression    ::= 'let' variable_name '=' expression 'in' '{' statement* 'submit'
expression ';}'
```