Ocean Compiler

Run:

```
make
./a.out <input_file.txt> <output_file.txt>
```

Where: The input_file.txt is the input file and the output_file.txt is the output file.

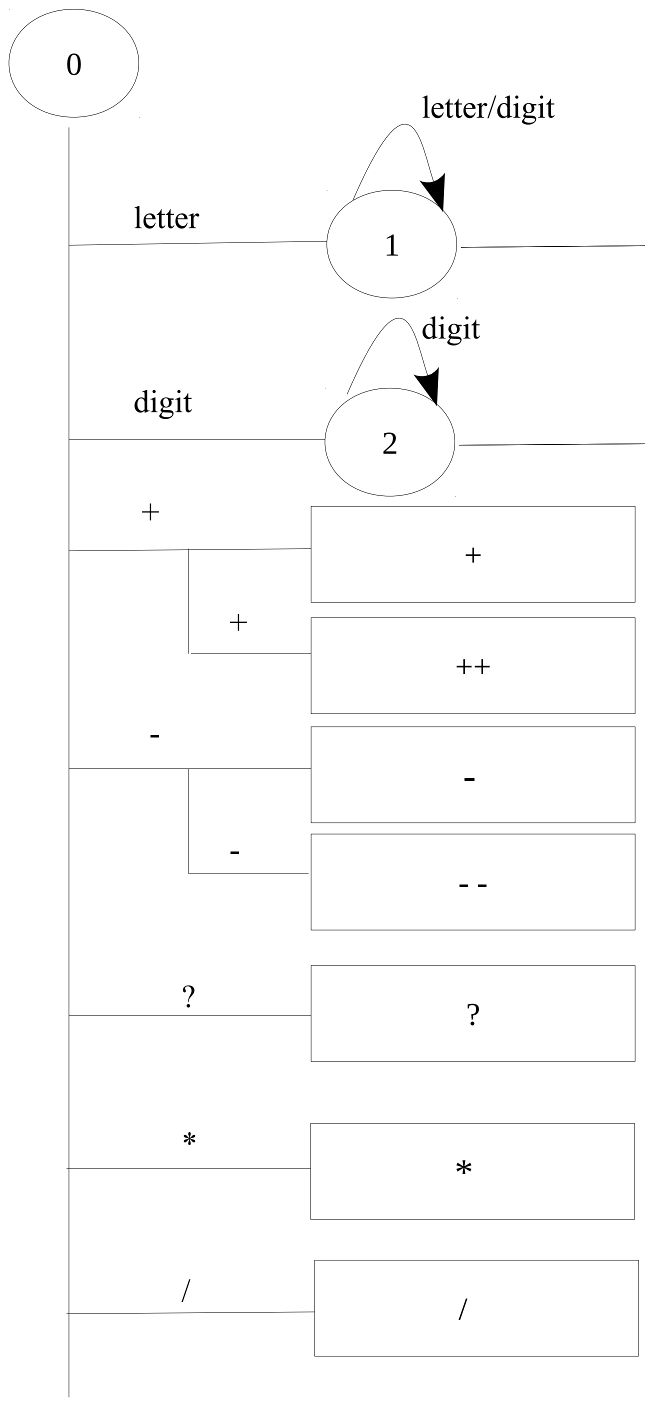
Ocean Programming Language.

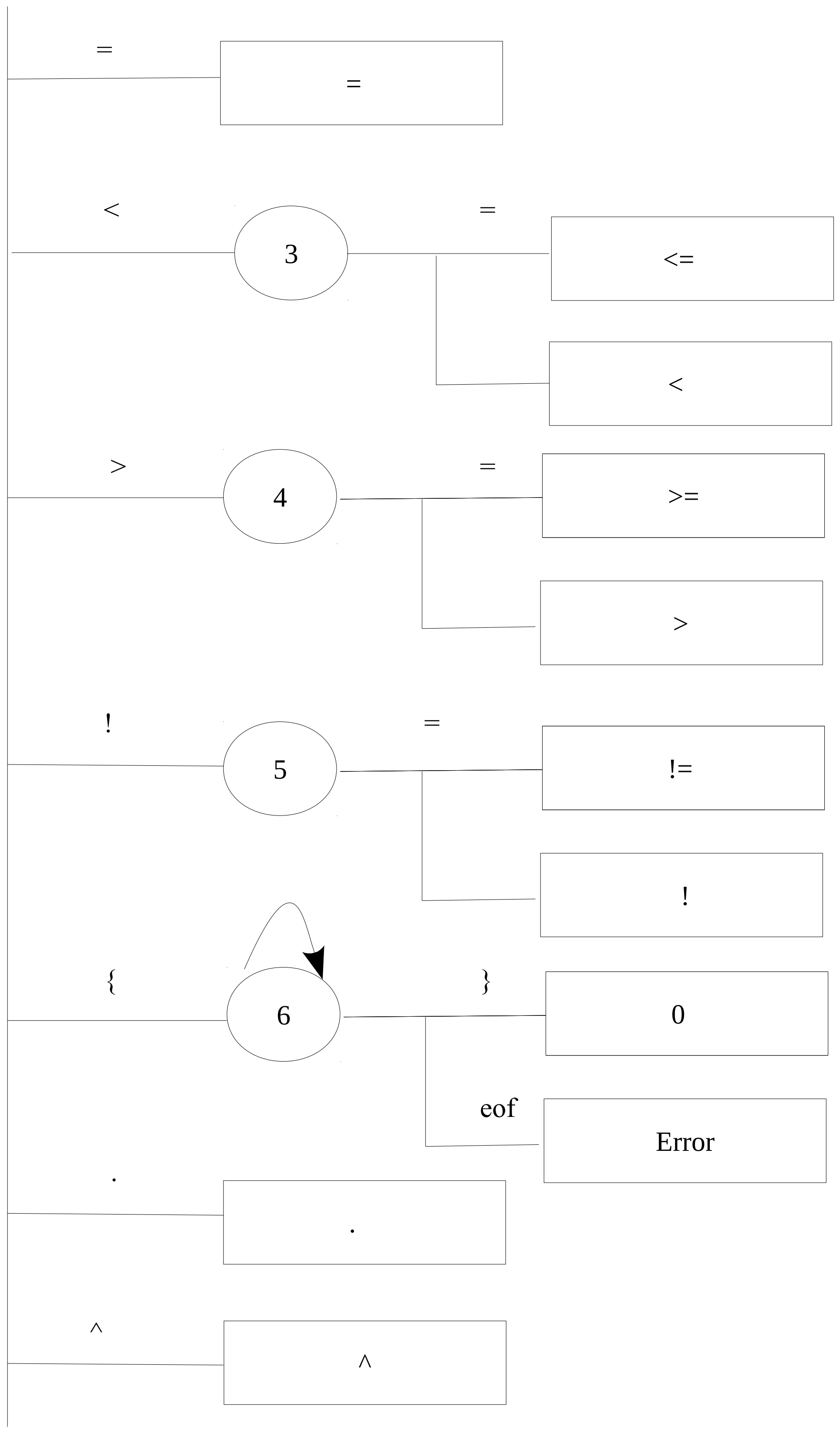
Ocean includes:

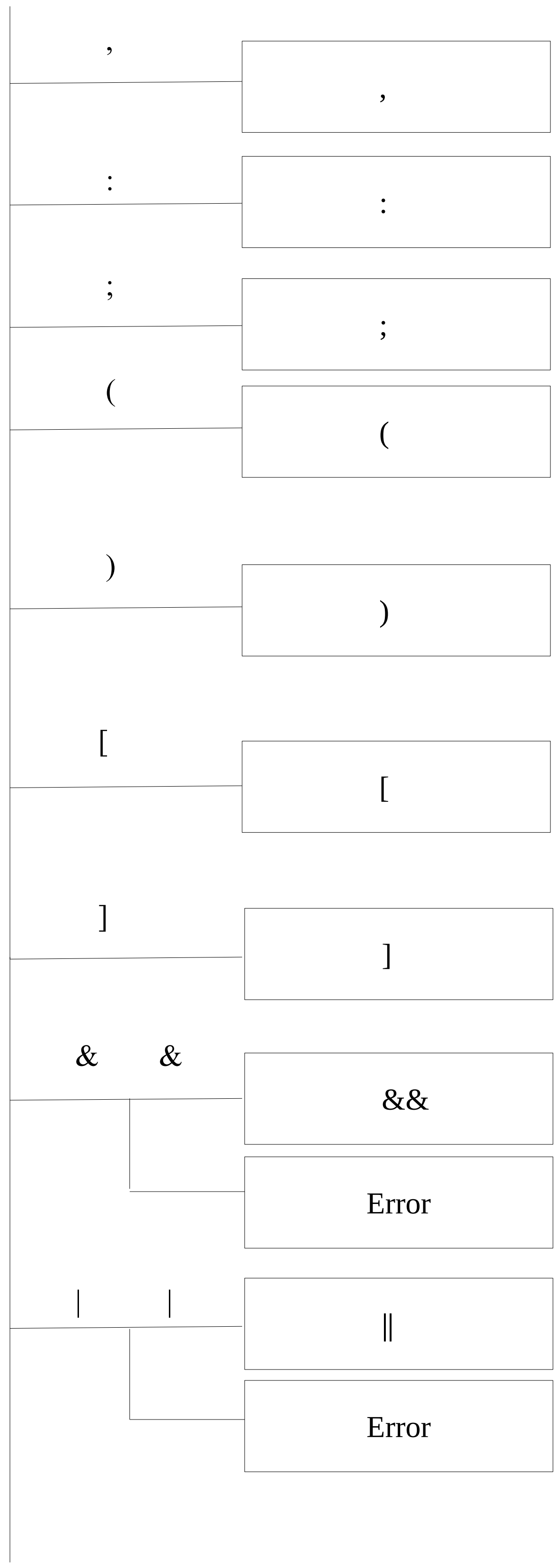
- Int, int[], char, double.
- Classes.
- Objects.
- Constructos.
- Inheritance.
- Constructor overloading.
- Method overriding.
- Polymorphism.
- Abstract method.
- Interface class.

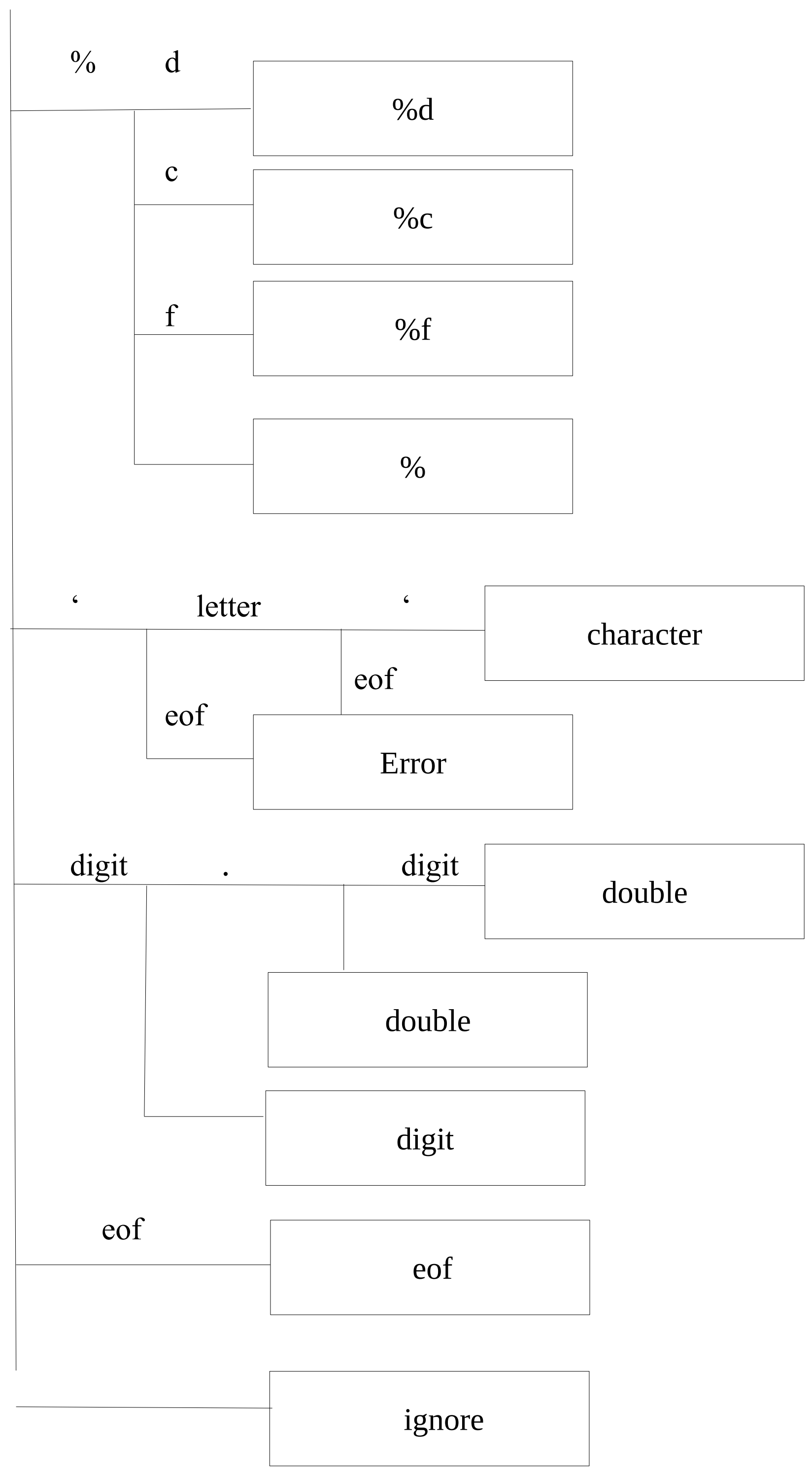
Lexical Analyzer.
Automaton.

White character.




=
=
<
3
=
<=
<
>
4
=
>=
>
!
5
=
!=
!
{
6
}
0
eof
Error
.
.
^
^


,
,
:
:
;
;
(
(
)
)
[
[
]
]
& &
&&
Error
| |
||
Error


%
d
%d
c
%c
f
%f
%
‘
letter
‘
character
eof
eof
Error
digit
.
digit
double
double
digit
eof
eof
ignore

Ocean's Grammar.

<program>	::= <class_type> <main>
<main>	::= **main** ( ) **{** <method_block> **}**
<class_type>	::= ε | (<class>)* | (<interface>)*
<class>	::= **class** id **{** <block> **}** | **class** id **extends** id { <block> }
<interface>	::= **interface** id **{** (<interface_block>)* **}**
<block>	::= <declarations> (<constructors>)* (<methods>)*
<interface_block>	::= ε | id **( ) ;**
<declarations>	::= ε | **public (**<values_list>)* **endpublic**
<values_list>	::= ε | <int_values> | <double_values> | <char_values> | <object_values>
<int_values>	::= **int** <array_or_not>
<array_or_not>	::= id <init_int_rule> | **[**digit**]** id <int_array_rule>
<int_array_rule>	::= **;** | **= {** <many_digits> **} ;**
<many_digits>	::= digit | (digit)*
<init_int_rule>	::= ε | **;** | **=** (<expression>)* ;
<double_values>	::= **double** id **;** | **double** id **=** (<expression>)* ;
<char_values>	::= **char** id **;** | **char** id **=** char **;**
<object_values>	::= **object** id id **;**
<constructors>	::= ε | id **(** <parlist> **)** { <constructor_block> }

<methods>	::= ε | **void** id **(** <parlist> **) {** (<method_block>)* **}**
	 | **int** id **(** <parlist> **) {** (<method_block>)* **}**
	 | **double** id **(** <parlist> **) {** (<method_block>)* **}**

```
                                   | char id ( <parlist> ) { (<method_block>)* }
                                   | abstract id ( ) ;


<parlist>                     ::= ε | <parlist_types> | (, <parlist_types> )*

<par_int>                     ::= ε | id | [ digit ] id

<constructor_block>           ::= (<method_block>)*

<method_block>                ::= ε |  <assignment> |
                                      <ifstat> |
                                      <whilestat> |
                                      <switchstat> |
                                      <forstat> |
                                      <callstat> |
                                      <returnstat> |
                                      <inputstat> |
                                      <printstat> |
                                      <super>

<assignment>                  ::= int <local_array_or_not>
                                      | id = (<expression>)* ;
                                      | double id = (<expression>)*;
                                      | double id ;
                                      | char id = (<expression>)*;
                                      | char id ;
                                      | object id id <init_object>

<local_array_or_not>          ::= id <local_int> | [ digit ] id <init_array_rule>

<local_int>                   ::= ; | = (<expression>)* ;

<init_object>                 ::= = id ( <actualpars_constructor> ) ;
                                      | ;

<actualspars_constructor>::= ε | id | char | digit | double | (, id)* | (, char)* | (,  digit)*
                                      | (, double)*

<ifstat>                      ::= if ( <condition> ) { (<method_block)* }
                                      | elif ( <condition> ) { (<method_block)* }
                                      | else { (<method_block)* }
```

<whilestat>	::= **while (** <condition> **)** **{** (<method_block)* **}**
<forstat>	::= **for (** <int_values_for> id <relationaloper> <number_or_id>**;** <for_step> **)** **{** (<method_block>)* **}**
<int_values_for>	::= **int** id **=** (<expression>)* **;** | id **=** (<expression>)* **;**
<number_of_id>	::= digit | id
<forstep>	::= id **=** (<expression>)* | id **= ++** | id **= –**
<returnstat>	::= **return** (<expression>)* **;**
<switchstat>	::= **switch (** id **)** **{** (**<**caserule>)***}**
<caserule>	::= ε | **case** <id_char_digit>**:** (<method_block>)* **break;** | **default**: **break;** | **default:** (<method_block>)* **break;**
<id_char_digit>	::= id | char | digit
<printstat>	::= **print (** (<inside_print>)* **);**
<inside_print>	::= ε | (**‘** <inside_apostrophe> **‘**)* | (**,** <aftercomma>)*
<aftercomma>	::= <inside_apostrophe> | <outside_apostrophe>
<inside_apostrophe>	::= ε | id | **%d** | **%f** | **%c**
<outside_apostrophe>	::= id | digit | double | char
<id_print>	::= id | digit | double | char
<inputstat>	::= **input (** id **);**
<callstat>	::= **call** id <callcase>
<callcase>	::= **.** id **(** <actualpars> **);** | **=** id **.** id **(** <actualpars> **);**
<actualpars>	::= ε | id | char | digit | double | (**,** id)* | (**,** digit)* | (**,** double)* | (**,** char)*

```
<super>                 ::= super ( <actualpars_constructor> );

<condition>             ::= <boolterm> | (|| <boolterm>)*

<boolterm>              ::= <boolfactor> | (&& <boolfactor)*

<boolfactor>            ::= ε | not <condition> | != ( <condition> ) |
                                (<expression>)* <relationarloper> (<expression>)*
                                | <condition> | true | false

<expression>            ::= ε | digit <operations> digit | double <operations> double
                                | char | <array_expr> <operations> <array_expr>
                                | <array_expr> <operations> | id <operations> id
                                | id <operations> digit | id <operations> double
                                | double <operations> id | digit <operations> id
                                | ( (<expression>)* ) | <operations> (  (<expression>)* )
                                | digit <operations> | id <operations> | (<expression>)*
                                <operations> digit | (<expression>)* <operations>
                                <array_expr> | (<expression>)* <operations> id | double

<array_expr>            ::= id [digit]

<operations>            ::= + | - | * | / | ^ | – | ++

<relationaloper>        ::= == | < | > | <= | >= | !=
```