Generate x86 assembly code (`test.s`) for the following grammar. We will test your submission in the following way:

```
unzip <student ID>.zip
cd <student ID>
./build.sh
grun Cmpt379Compiler program <path to test case> > test.s
gcc –s test.s
./a.out
```

For making library calls, we will be using the following source code format:

```
callout("printf", "hello world\n");
```

Grammar:

```
<program>
     -> class Program { <field_decl>* <method_decl>* }
<field_decl>
     -> <type> (<id> | <id> [ int_literal ] )
     ( , <id> | <id>[int_literal ] )* ;
     | <type> <id> = <literal> ;
<method_decl>
     -> ( <type> | void ) <id>
     (( (<type> <id>) ( , <type> <id>)*)? ) <block>
<block>
     -> { <var_decl>* <statement>* }

<var_decl>
     -> <type> <id> ( , <id>)* ;
<type>
     -> int
     | boolean
<statement>
     -> <location> <assign_op> <expr> ;
     | <method_call> ;
     | if ( <expr> ) <block> ( else <block> )?
     | switch <expr> {(case <literal> : <statement>*)+}
     | while ( <expr> ) <statement>
     | return ( <expr> )? ;
     | break ;
     | continue ;
     | <block>
<assign_op>
     -> =
     | +=
     | -=
```

```
<method_call>
     -> <method_name> ( (<expr> ( , <expr> )*)? )
     | callout ( <string_literal> ( , <callout_arg> )* )
<method_name>
     -> <id>
<location>
     -> <id>
     | <id> [ <expr> ]
<expr>
     -> <location>
     | <method_call>
     | <literal>
     | <expr> <bin_op> <expr>
     | - <expr>
     | ! <expr>
     | ( <expr> )
<callout_arg>
     -> <expr>
     | <string_literal>
<bin_op>
     -> <arith_op>
     | <rel_op>
     | <eq_op>
     | <cond_op>
<arith_op>
     -> +
     | -
     | *
     | /
     | %
<rel_op>
     -> <
     | >
     | <=
     | >=
<eq_op>
     -> ==
     | !=
<cond_op>
     -> &&
     | ||
<literal>
     -> <int_literal>
     | <char_literal>
     | <bool_literal>
<id>
     -> <alpha> <alpha_num>*
```

<alpha>
    -> [**a-zA-Z_**]
<alpha_num>
    -> <alpha>
    | <digit>
<digit>
    -> [**0-9**]
<hex_digit>
    -> <digit>
    | [**a-fA-F**]
<int_literal>
    -> <decimal_literal>
    | <hex_literal>
<decimal_literal>
    -> <digit>$^{+}$
<hex_literal>
    -> **0x** <hex_digit>$^{+}$
<bool_literal>
    -> **true**
    | **false**
<char_literal>
    -> ‘<char>’
<string_literal>
    -> “<char>*”