1. Generate x86 assembly code for programs written using the following grammar:

```
class Program { <field_decl>* <method decl>* }
<field decl> -> <type> (<id> | <id> [ <int literal> ]) (, <id> | <id> [ <int literal> ])*;
<field_decl> -> <type> <id> = teral>;
<method decl> -> ( <type> | void ) <id> ( ((<type> <id>) ( , <type> <id>)*)? ) <block>
<blook> -> { <var_decl>* <statement>* }
<var_decl> -> <type> <id> ( , <id>)*;
<type> -> int | boolean
<statement> -> <location> <assign op> <expr> ;
<statement> -> <method call>;
<statement> -> if ( <expr> ) <block> ( else <block> )?
<statement> -> for <id> = <expr> , <expr> <block>
<statement> -> return ( <expr> )?;
<statement> -> break;
<statement> -> continue;
<statement> -> <block>
<assign op> -> =
<assign op> -> +=
<assign_op> -> -=
<method_call> -> <method_name> ( (<expr> ( , <expr> )*)? )
<method_call> -> callout ( <string_literal> ( , <callout_arg> )* )
<method name> -> <id>
<location> -> <id>
<location> -> <id>[ <expr> ]
<expr> -> <location>
<expr> -> <method call>
<expr> -> <literal>
<expr> -> <expr> <bin op> <expr>
<expr> -> - <expr>
<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> -> && | ||
-> <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> <digit>* <hex_literal> -> 0x <hex_digit> <hex_digit>* <bool_literal> -> true | false <char_literal> -> '<char>' <string_literal> -> "<char>*"
```

(30)