1. (15 points) Give the output for the program.

```
#include <iostream>
class Token {
public:
  Token()              { std::cout << "default" << std::endl;    }
  explicit Token(const char*)   { std::cout << "convert" << std::endl;    }
  Token(const Token&){ std::cout << "copy" << std::endl;       }
  Token& operator=(const Token&) {
    std::cout << "assign" << std::endl;
    return *this;
  }
};
class TokenDB {
public:
  TokenDB(const char* name) {
    token = Token(name);
  }
private:
  Token token;
};
int main() {
  TokenDB db("IDENT");
}
```

```
default
convert
assign
```

---

2. (15 points) Give the output for the program.

```
#include <iostream>
#include <vector>

class Token {
public:
  Token()              { std::cout << "default" << std::endl;    }
  Token(const char*)   { std::cout << "convert" << std::endl;    }
  Token(const Token&){ std::cout << "copy" << std::endl;       }
  Token& operator=(const Token&) {
    std::cout << "assign" << std::endl;
    return *this;
  }
};

int main() {
  std::vector<Token> tokens;
  tokens.push_back( "IDENT" );
  tokens.push_back( "FLOAT" );
}
```

```
convert
copy
convert
copy
copy
```

3. (10 points) The previous program contains a vector whose capacity changes by multiples of two on many installations, which requires allocation and deallocation of memory as the vector grows in size. This size growth will likely generate a lot of unnecessary copies. Add a line of code, without changing any existing code, to eliminate the multiple-of-two size growth.

```
1  #include <iostream>
2  #include <vector>
3
4  class Token {
5  public:
6    Token()              { std::cout << "default" << std::endl;    }
7    Token(const char*)   { std::cout << "convert" << std::endl;    }
8    Token(const Token&){ std::cout << "copy" << std::endl;       }
9    Token& operator=(const Token&) {
10     std::cout << "assign" << std::endl;
11     return *this;
12   }
13 };
14
15 int main() {
16   std::vector<Token> tokens;
17   tokens.reserve(2);
18   tokens.push_back( "IDENT" );
19   tokens.push_back( "FLOAT" );
20 }
```

---

4. (+2 extra credit points) Give the output for the following program. Notice the use of emplace_back.

```
1  #include <iostream>
2  #include <vector>
3
4  class Token {
5  public:
6    Token()              { std::cout << "default" << std::endl;    }
7    Token(const char*)   { std::cout << "convert" << std::endl;    }
8    Token(const Token&){ std::cout << "copy" << std::endl;       }
9    Token& operator=(const Token&) {
10     std::cout << "assign" << std::endl;
11     return *this;
12   }
13 };
14
15 int main() {
16   std::vector<Token> tokens;
17   tokens.emplace_back( "IDENT" );
18   tokens.emplace_back( "FLOAT" );
19 }
```

```
convert
convert
copy
```

5. (15 points) For the following program:

   (a) Give the output for the flex program below. Assume the input is (^d is control d):

```
C
xx ab c c
d
D
^d
```

```
c
SEVEN

xx ab c c
letters
Matching ab
ONE
SEVEN

d
SEVEN

D
SEVEN
```

   (b) What rule cannot be matched? (give the line number)
   line #13

```
%{

#include <iostream>

%}
%array
letter      [a-zA-Z]

%%
(?i:c)        { std::cout << "ONE" << std::endl;          }
(?-i:d)       { std::cout << "TWO" << std::endl;          }
(?-i:E)       { std::cout << "THREE" << std::endl;        }
c             { std::cout << "FOUR" << std::endl;         }
ab            { std::cout << "Matching ab" << std::endl; }

^0            { std::cout << "FIVE" << std::endl;     }
0$            { std::cout << "SIX" << std::endl;     }
0             { std::cout << "ZERO " << std::endl; }
{letter}$     { std::cout << "SEVEN" << std::endl; }
{letter}*     { std::cout << "letters" << std::endl;}
.             { }
%%
int yywrap() {
  yylex_destroy();
  return 1;
}
```

```
#include <iostream>
int yylex();

int main() {
  yylex();
}
```

6. (15 points) Give the output for the following program, assuming the input is:

```
Nancy Astor said to Winston Churchill: "if you were my husband I'd put poison in your tea."
```

OUTPUT:

words: 6

```
%{
int words = 0;
%}

%option yylineno
yylineno = 1

word          [a-zA-Z]+

%x START

%%

\"            { BEGIN(START);         }
<START>[^"]*  {                       }
<START>["]    { BEGIN(INITIAL);       }

{word}        { ++words;              }
.             { }
"\n"          { }
%%

int yywrap() { return 1; }
```

```
#include <iostream>
int yylex();
extern int words;

int main() {
  yylex();
  std::cout << "words: " << words << std::endl;
}
```

---

7. (+1 extra credit) What was Winston Churchill's alleged response to the quote above, attributed to Nancy Astor?

8. (30 points) Give short answers to the following questions:

(a) All languages evolve over time and these evolutions are usually marked by version numbers. However, Python's evolution is different from the evolution of C^{++} in a significant way. What is this difference? Be specific about Python version numbers.

Most languages maintain backward compatibility with previous versions of the language so that a program written using version 2 of a language should still compile using version 3. However, versions of Python from 3.0 onward are not backward compatible with previous versions. Thus, a program written using Python 2.6 will not compile using Python 3.0.

(b) List two advantages attributed to using Python (these were listed in the background section of the paper).

Python is convenient for rapid prototyping and scripting.

(c) What is *PyComply*?

PyComply is a tool that permits analysis of Python language feature usage. It was constructed by inserting semantic actions into the Python parser to count uses of rules that represent back ported language features.

(d) In Python language evolution, what is a *back-ported* feature?

A back ported feature occurs when a language feature that targets a later language version is incorporated into an existing version. For example dictionary comprehension is a language feature intended for versions of Python in the 3x series; however, it was back ported to version 2.7.2 of Python.

(e) Python 3x contains some interesting new features and some fixes to existing features. According to the findings of the paper, to what extent are developers using these new features including both back-ported and non back-ported features?

Developers are using back ported features of Python but on the whole they are not using language features that are new to the Python 3x series.

(f) What is the basic conclusion of the paper about the way that program developers are adapting to Python language evolution?

That program developers are restricting themselves to a subset of Python that will compile in both 2x and 3x compilers.