1. (10 points) Give the output for the following program.

```
#include <iostream>
class Game {
public:
  Game()             { std::cout << "default" << std::endl;     }
  Game(const char*) { std::cout << "convert" << std::endl;     }
  Game(const Game&) { std::cout << "copy" << std::endl;        }
  ~Game()            { std::cout << "destructor" << std::endl; }
  Game& operator=(const Game&) {
    std::cout << "copy assign" << std::endl;
    return *this;
  }
};
Game fun(Game g) {
  return g;
}

int main() {
  Game cat("cat");
  fun(cat);
}
```

```
convert
copy
copy
destructor
destructor
destructor
```

---

2. (10 points) Give the output for the following program.

```
#include <iostream>
class Game {
public:
  Game()             { std::cout << "default" << std::endl;     }
  Game(const char*) { std::cout << "convert" << std::endl;     }
  Game(const Game&) { std::cout << "copy" << std::endl;        }
  ~Game()            { std::cout << "destructor" << std::endl; }
  Game& operator=(const Game&) {
    std::cout << "copy assign" << std::endl;
    return *this;
  }
};
Game fun() {
  return Game( Game() );
}
int main() {
  Game cat("cat");
  fun();
}
```

```
convert
default
destructor
destructor
```

3. (10 points) Give the output for the following program.

```
#include <iostream>

void f(const int& x)  { std::cout << "l-value ref: " << x << std::endl; }
void f(int&& x)       { std::cout << "r-value ref: " << x << std::endl; }
int  f()              { return 19; }

int main() {
  f(20);
  int x = 7;
  f(x);
  f(std::move(x));
  f(x+1);
  f(f());
}
```

```
r-value ref: 20
l-value ref: 7
r-value ref: 7
r-value ref: 8
r-value ref: 19
```

---

4. (10 points) Give the output for the following program.

```
#include <iostream>

class A{
public:
  A()           { std::cout << "default constructor" << std::endl; }
  A(int)        { std::cout << "conversion constructor" << std::endl; }
  A(const A&)   { std::cout << "copy constructor" << std::endl; }
  A(const A&&) { std::cout << "move constructor" << std::endl; }
  A& operator=(const A&) {
    std::cout << "copy assignment" << std::endl;
    return *this;
  }
  A& operator=(const A&&) {
    std::cout << "move assignment" << std::endl;
    return *this;
  }
};

int main() {
  A a, b = a;
  a = b;
  b = 99;
}
```

```
default constructor
copy constructor
copy assignment
conversion constructor
move assignment
```

5. (10 points) Give the output for the following program.

```
1  #include <iostream>
2  #include <vector>
3  class Game {
4  public:
5    Game()             { std::cout << "default" << std::endl;     }
6    Game(const char*) { std::cout << "convert" << std::endl;     }
7    Game(const Game&) { std::cout << "copy" << std::endl;        }
8    ~Game()            { std::cout << "destructor" << std::endl; }
9  private:
10   const char* name;
11 };
12 Game fun(Game g) {
13   return g;
14 }
15
16 int main() {
17   std::vector<Game> games;
18   games.emplace_back("Monopoly");
19   games.push_back("Magic the Gathering");
20 }
```

```
convert
convert
copy
copy
destructor
destructor
destructor
destructor
```

---

6. (5 points) Write a move constructor for class Game above.

```
Game(Game&& s) : buf(std::move(g.name)) {
  g.name = nullptr;
}
```

7. (5 points) The Python program listed below results in the following output from radon:

```
easy.py
    F 1:0 f - A (2)
```

What is the significance and meaning of each part of the output:

```
1 def f(x):
2   if x % 2 == 0:
3     print "even"
4   else:
5     print "odd"
6
7 f(2)
8 f(5)
```

```
F -- the type is Function
1:0 -- line, column
f -- name of the function
A -- McCabe Complexity category -- lowest
(2) -- actual McCabe complexity
```

---

8. (5 points) There are two (2) test scripts in the project 3 directory. What are they called and what does each test script do?

```
test.py -- runs the test cases in a directory called "cases", comparing
           the user computed cyclomatic complexity to the
           complexity computed by Radon

alltest.py -- runs the test cases in Python-2.7.2/.
           The Python-2.7.2/ directory contains all of the test cases
           from the Python developers web site for the particular
           version of Python under study. This script has nothing
           to do with cyclomatic complexity.
```

9. (15 points) Listed below is a specification for a scanner and parser that can be used to implement exponentiation. Modify the specifications so that they correctly compute exponentiation. Your program must compile, link, and execute correctly. For example, your solution should work as follows:

```
2**3**2
512
```

```
%{
#include "parse.tab.h"
%}

%%

"**"        { return EXP; }
[0-9]+      { yylval = atoi(yytext);
              return NUMBER;
            }
"\n"        { return CR; }
<<EOF>>     { yyterminate(); }

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

```
%{
#include <iostream>
extern int yylex();
extern int yylval;
void yyerror(const char * msg);
int power(int, int);
%}

%token CR NUMBER
%right EXP

%%
lines    : lines expr CR
           { std::cout << $2 << std::endl; }
         | lines CR
         |
         ;

expr     : expr EXP expr    { $$ = power($1, $3); }
         | NUMBER           { $$ = $1;       }
         ;
%%
void yyerror(const char * msg) { std::cout << msg << std::endl; }
int power(int base, int exp) {
  int result = 1;
  for (int i = 0; i < exp; ++i) {
    result *= base;
  }
  return result;
}
```

10. (20 points) Listed below is a specification for a scanner that identifies C-style comments, and a main program. However, both the generated scanner and the main program access the global variable count. Write a GoF Singleton class, Count, that maintains this global variable for the scanner and for the main program. Modify both the scanner specification and the main program to use your Singleton. Your program must compile, link, and execute correctly. A sample execution might be:

```
/*  this is a comment */
/* and so is this */
^d
There were 2 comments
```

```
%{
  #include <iostream>
  #include <string>
  #include "count.h"
%}

%x COMMENT
%%

"/*"                { BEGIN(COMMENT); }
<COMMENT>"*/"       { BEGIN(0); Count::getInstance()->incrCount(); }
<COMMENT>.          { ; }
.                   { std::cout << yytext << std::endl; }
'\n'                { }
%%
int yywrap() { return 1; }
```

```
#include <iostream>

class Count {
public:
  static Count* getInstance() {
    if ( !instance ) instance = new Count;
    return instance;
  }
  void incrCount() { ++count; }
  int getCount() const { return count; }
private:
  static Count* instance;
  int count;
  Count() : count(0) {}
};
std::ostream& operator<<(std::ostream&, const Count&);
```

```
#include "count.h"
std::ostream& operator<<(std::ostream& out, const Count& count) {
  return out << count.getCount();
}
```

```
#include <iostream>
#include "count.h"
int yylex();
```

```
4
5  Count* Count::instance;
6
7  int main() {
8    yylex();
9    std::cout << "There were " << Count::getInstance()->getCount()
10             << " comments"  << std::endl;
11   return 0;
12 }
```