1. (10 points) Give the output for the program; notice that the destructor has an output statement:

```
#include <iostream>
class Pokemon {
public:
  Pokemon()               { std::cout << "default" << std::endl;     }
  Pokemon(const char*)    { std::cout << "convert" << std::endl;     }
  Pokemon(const Pokemon&){ std::cout << "copy" << std::endl;        }
  ~Pokemon()              { std::cout << "destructor" << std::endl; }
  Pokemon& operator=(const Pokemon&) {
    std::cout << "assign" << std::endl;
    return *this;
  }
};
int main() {
  Pokemon x("Dragonite"), y = x;
  Pokemon* z = new Pokemon(x) ;
}
```

---

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

```
#include <iostream>
class Pokemon {
public:
  Pokemon()               { std::cout << "default" << std::endl;     }
  explicit Pokemon(const char*)   { std::cout << "convert" << std::endl;     }
  Pokemon(const Pokemon&){ std::cout << "copy" << std::endl;        }
  Pokemon& operator=(const Pokemon&) {
    std::cout << "assign" << std::endl;
    return *this;
  }
};
class PokemonGym {
public:
  PokemonGym(const char* name) {
    poke = Pokemon(name);
  }
private:
  Pokemon poke;
};
int main() {
  PokemonGym gym("Polywrath");
}
```

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

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

class PokemonGym {
public:
  PokemonGym(const char* name) : poke(name) { }
private:
  Pokemon poke;
};

int main() {
 try {
  PokemonGym gym("Ponyta");
 }
 catch ( ... ) {
     std::cout << "oops" << std::endl;
 }
}
```

---

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

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

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

int main() {
  std::vector<Pokemon> poke;
  poke.push_back( "Polywrath" );
  poke.push_back( "Snorlax" );
}
```

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

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

int main() {
  std::vector<int> vec(10);
  vec.push_back(23);
  std::cout << "size: " << vec.size() << std::endl;
  std::cout << "cap: " << vec.capacity() << std::endl;
}
```

---

6. (10 points) Give the output for the program.

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

int main() {
  std::vector<int> vec;
  vec.reserve(10)
  vec.push_back(23);
  std::cout << "size: " << vec.size() << std::endl;
  std::cout << "cap: " << vec.capacity() << std::endl;
}
```

---

7. (10 points) Draw the dot diagram described by the following specification:

```
digraph G {
  node [shape=box];
  A->B
  B->C
  C->D
  A->D
}
```

8. (15 points) The program below uses flex to generate a scanner that recognizes words, and then prints the number of words that were typed. Extend the code below so that it also counts and prints the number of floating point numbers. Consider a floating point number to be specified in EBNF as {*digit*}.{*digit*}, where the "curly braces" indicate zero or more times. A sample execution might be:

```
sum = 25.66 - 4. + .34 + -1
No. of words:  1
No. of floats: 3
```

```
%{
#include <iostream>
  int words = 0;
%}
letter      [a-zA-Z]

%%
{letter}+     { ++words; }
.             { }
"\n"          { }
%%
int yywrap() { return 1; }
```

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

extern int words;
extern int lines;
extern int chars;

int main() {
  yylex();
  std::cout << "words: " << words << std::endl;
  return 0;
}
```

---

9. (15 points) The program below uses flex and bison to generate a scanner and parser that recognizes balanced parens. (1) Explain the reason for including parse.tab.h on line #4 of the scanner; (2) Extend the bison specification so that it also counts and prints the number of balanced parens. A sample execution might be:

```
()(()())    ()
No. of Parens: 5
()
No. of Parens: 1
(()())
No. of Parens: 3
Gotta catch 'em all!
```

```
1  %{
2  #include <iostream>
3  #include <cstring>
4  #include "parse.tab.h"
5  %}
6  %%
7  "("         { return LPAR; }
8  ")"         { return RPAR; }
9  "\n"        { return CR; }
10 .           { ; }
11 %%
12 int yywrap() { return 1; }
```

```
1  %{
2  #include <iostream>
3  extern int yylex();
4  void yyerror(const char * msg) { std::cout << msg << std::endl; }
5  %}
6  %token CR LPAR RPAR
7  %%
8  lines    : lines expr CR
9             { std::cout << "accept" << std::endl; }
10           | { ; }
11           ;
12 expr     : LPAR expr RPAR expr
13           | { ; }
14           ;
```

```
1  #include <iostream>
2  extern int yyparse();
3  int main() {
4    if ( yyparse() == 0 ) { std::cout << "Gotta catch 'em all!" << std::endl; }
5    else { std::cout << "Oops!" << std::endl; }
6  }
```