1. (3 points) Formally, a grammar G is a four tuple (N, T, S, P). S is the start symbol. Define N, T, P.

2. (3 points) Using the following grammar, write a parse tree or derivation tree for: 7 + 3 * 5:

```
expr    : expr '+' expr
        | expr '*' expr
        | DIGIT
```

3. (4 points) Is the following grammar ambiguous? Show that it is or is not ambiguous?

$$S \rightarrow SS \mid a \tag{1}$$

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

```
#include <iostream>
class string {
public:
  string()               { std::cout << "default" << std::endl;     }
  string(const char*)    { std::cout << "convert" << std::endl;     }
  string(const string&) { std::cout << "copy" << std::endl;        }
  ~string()              { std::cout << "destructor" << std::endl; }
  string& operator=(const string&) {
    std::cout << "assign" << std::endl;
    return *this;
  }
};
int main() {
  string a("aarvark");
  string* rope = new string("hang");
}
```

---

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

```
#include <iostream>
void incrCount(int count) {
  ++count;
}

void makeSwitch(int & count) {
  switch ( count ) {
    case 2:  ++count;
    case 3:  ++count;
    case 4:  ++count;
    case 5:  ++count;
    default: ++count;
  }
}
int main() {
  int count = 10;
  incrCount(count);
  std::cout << count << std::endl;

  count = 8;
  count = (count % 5) ? 2 : 3;
  makeSwitch(count);
  std::cout << count << std::endl;
}
```

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

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

---

7. (10 points) The following program crashes with a double free error. Explain why,

```
#include <cstring>
#include <iostream>
class string {
public:
   string(const char* s) : buf(new char[strlen(s)+1]) { strcpy(buf, s); }
   ~string() { delete [] buf; }
   const char* getBuf() const { return buf; }
private:
   char * buf;
};

int main() {
  string a("cat"), b = a;
}
```

8. (5 points) Give the output for the following program.

```
#include <iostream>
class Bird {
public:
  Bird(int w) : wingSpan(w), speed(2*wingSpan) {
    std::cout << "Speed: " << speed << std::endl;
    std::cout << "Wing Span: " << wingSpan << std::endl;
  }
private:
  int speed;
  int wingSpan;
};

int main() {
  Bird robin(5);
}
```

---

9. (5 points) The following program compiles and executes but gives a warning. Give the output for the program. The warning given by g++ is:

```
main.cpp:3:21: warning: parameter x set but not used [-Wunused-but-set-parameter]
 void initialize(int x, int y) {
                     ^
main.cpp:3:28: warning: parameter y set but not used [-Wunused-but-set-parameter]
 void initialize(int x, int y) {
                           ^
```

```
#include <iostream>

void initialize(int x, int y) {
  x = 7;
  y = 8;
}

int main() {
  int a = 10, b = 11;
  initialize(a, b);
  std::cout << a << std::endl;
  std::cout << b << std::endl;
}
```

10. (40 points) The rule of three states that for a class that contains dynamically allocated data attributes (pointers), the programmer should write a destructor, a copy constructor, and an assignment operator. Write a class, Student, that contains a single data attribute: char* name. Write all three constructors and any addional functions needed to fulfill the rule of three. Also, overload an output operator for class Student, including any accessor functions needed (getters).