Problem 1

Q: *Pick a language that you are most familiar with, discuss three most important features, e.g., data model, the type system, the I/O or object-orientation.*

In Python, the three most important features in my opinion are:

1. **Block Indent**

   This is a rare feature in most programming languages. Almost all the programming languages ignore spaces (except some special language such as Whitespace), but Python DO NOT.

   We use "begin" "end" to indicate a block in Pascal, and "{" "}" in C-like languages. But in Python, we use indentation to show the boundary of a block.

   For good programming habits, we usually indent our code though languages like C don't have a forced limit. But some lazy programmers or newbies don't know how important indentation is – they produce lots of "bad-habit codes" which are always not readable. And Python solved this problem, almost all python code are "beautiful", even if the programmer never got in touch with programming before.

2. **Sequences**

   Sequences in Python (strings, lists and tuples) have several great features. It looks a little like array in some other languages such as C or C++, but in fact, there are more useful usages for sequence.

   One of the interesting features is slice. For a sequence *foo* in python, we could visit its element 5 by a simple operation like this:

```
1 print foo[5]
```

   such as the same operation on an array *foo* in C++:

```
1 cout << foo[5];
```

   So far, we couldn't find out what sequence is better than arrays. But consider this situation: we wanna copy the last 7 elements in *foo* into another sequence (or array) *bar*. Here comes the C++ code:

```
1 // foo_len is the length of foo, so bar_len
2 for (int i = 0; i != 7; ++i)
3     bar[i] = foo[foo_len - 7 + i];
```

   or we have another simpler implementation by funciton *memcpy*:

```
1 // foo_len is the length of foo, so bar_len
2 // foo_t is the type of the elements in foo
3 memcpy(bar, (foo + foo\_len - 7), sizeof(foo\_t) * 7);
```

   The second one is shorter but not so easy for reading.

   Now see how Python do this:

```
1 bar = foo[-7:]
```

   It is THE Python.

   And there are much more features with sequence, like multiplication operation. An example is shown below.

```
foo = [1, 2] * 3
print foo // result is [1, 2, 1, 2, 1, 2] here
```

3. **Language Support**

   Python has won "Programming Language of the Year" award by TIOBE twice in 2007 and 2010. It's not a accidental event – Python is welcome, because it could work well along with other languages, such as C and Java.

   It's easy to use C and C++ libraries in Python. And there are several varieties of Python, like Jython for Java and IronPython for .NET. So programmers could easily move their previous work into a Python project, without rewriting the code.

Problem 2

Q: *Give an example statement in C, C++, or Java that is particularly unreadable. Rewrite that statement in a more readable style. For instance, have you ever seen the expression A[i++] in a C/C++ program?*

A typical unreadable example of C is:

```
int a, b, c;
b = 0;
c = 1;
a = b+++c; // "a = b++ + c", or "a = b + ++c"?
cout << a << "␣" << b << "␣" << c;
```

In fact, the first explaination is the correct one. The result of this program will be "1 1 1".

Problem 3

Q: *Download Clite interpreter from http://highered.mcgraw-hill.com/sites/0072866098/, run a Clite program (pick one from the samples or write one on yourself) and get the output.*

The program

```
int main () {
   int c;
   float b;
   c = 1;
   b = -float(c);
}
```

has the output

```
[sqybi@myhost 01]$ java -jar Clite.jar cast.cpp
Begin parsing... cast.cpp

Program (abstract syntax):
  Declarations:
    Declarations = {<c, int>, <b, float>}
  Block:
    Assignment:
      Variable: c
      IntValue: 1
    Assignment:
```

```
      Variable: b
      Unary:
        Operator: neg
        Unary:
          Operator: float
          Variable: c

Begin type checking...cast.cpp

Type map:
{ <b, float>, <c, int> }

Transformed Abstract Syntax Tree

Program (abstract syntax):
  Declarations:
    Declarations = {<c, int>, <b, float>}
  Block:
    Assignment:
      Variable: c
      IntValue: 1
    Assignment:
      Variable: b
      Unary:
        Operator: FLOATNEG
        Unary:
          Operator: I2F
          Variable: c

Begin interpreting...cast.cpp

Final State
{ <b, -1.0>, <c, 1> }
```

Problem 4

Q: *Give 3 example statements in your favorite languages that are particularly unreadable. E.g., what does the C expression while (*p++ = *q++) mean? And explain what each of the statement means.*

My favourite language is Python. Although it's really difficult to write an unreadable program by this cute language, here is several examples.

Swaping Two Variables

```
1 a = 1
2 b = 2
3 a, b = b, a
```

This program simply swap the two variables a and b. The code below is doing the same thing.

```
1 a = 1
2 b = 2
3 c = a
4 a = b
```

```
5 b = c
```

Lambda Expression

```
1 x = 10
2 y = 5
3 f = lambda a, b: a + b
4 print(f(x, y))
```

Here z is a lambda expression, which looks like a function. The following code is a expansion (in fact, not exactly a expansion).

```
1 def f(a, b):
2     return a + b
3 x = 10
4 y = 5
5 print(f(x, y))
```

Multiple Inherit

```
1  class P1: #(object:)
2      def foo():
3          print("P1")
4
5  class P2: #(object:)
6      def foo():
7          print("P2")
8
9  class C(P1, P2):
10     pass
11
12 C.foo()
```

Multiple inherit also exists in C++. In Python, to determine which function to run, the compiler will search the inherit tree (DFS if no "object" in classes of BFS if has an "object" here). The program below will have a result "P1".

Here z is a lambda expression, which looks like a function. The following code is a expansion (in fact, not exactly a expansion).

Problem 5

Q: *Using the grammar Ginteger, develop a leftmost and a rightmost derivation for the integers 4520, 115511.*

The leftmost derivation for 4520:

$$\begin{aligned} \textit{Integer} &\Rightarrow \textit{Integer Digit} \\ &\Rightarrow \textit{Integer Digit Digit} \\ &\Rightarrow \textit{Integer Digit Digit Digit} \\ &\Rightarrow \textit{Digit Digit Digit Digit} \\ &\Rightarrow 4\ \textit{Digit Digit Digit} \\ &\Rightarrow 45\ \textit{Digit Digit} \\ &\Rightarrow 452\ \textit{Digit} \\ &\Rightarrow 4520 \end{aligned}$$

The rightmost derivation for 4520:

$$\begin{aligned} \textit{Integer} &\Rightarrow \textit{Integer Digit} \\ &\Rightarrow \textit{Integer}\ 0 \\ &\Rightarrow \textit{Integer Digit}\ 0 \\ &\Rightarrow \textit{Integer}\ 20 \\ &\Rightarrow \textit{Integer Digit}\ 20 \\ &\Rightarrow \textit{Integer}\ 520 \\ &\Rightarrow \textit{Digit}\ 520 \\ &\Rightarrow 4520 \end{aligned}$$

The leftmost derivation for 115511:

$$\begin{aligned} \textit{Integer} &\Rightarrow \textit{Integer Digit} \\ &\Rightarrow \textit{Integer Digit Digit} \\ &\Rightarrow \textit{Integer Digit Digit Digit} \\ &\Rightarrow \textit{Integer Digit Digit Digit Digit} \\ &\Rightarrow \textit{Integer Digit Digit Digit Digit Digit} \\ &\Rightarrow \textit{Digit Digit Digit Digit Digit Digit} \\ &\Rightarrow 1\ \textit{Digit Digit Digit Digit Digit} \\ &\Rightarrow 11\ \textit{Digit Digit Digit Digit} \\ &\Rightarrow 115\ \textit{Digit Digit Digit} \\ &\Rightarrow 1155\ \textit{Digit Digit} \\ &\Rightarrow 11551\ \textit{Digit} \\ &\Rightarrow 115511 \end{aligned}$$

The rightmost derivation for 115511:

$$\begin{aligned} \textit{Integer} &\Rightarrow \textit{Integer Digit} \\ &\Rightarrow \textit{Integer}\ 1 \\ &\Rightarrow \textit{Integer Digit}\ 1 \\ &\Rightarrow \textit{Integer}\ 11 \\ &\Rightarrow \textit{Integer Digit}\ 11 \\ &\Rightarrow \textit{Integer}\ 511 \\ &\Rightarrow \textit{Integer Digit}\ 511 \\ &\Rightarrow \textit{Integer}\ 5511 \\ &\Rightarrow \textit{Integer Digit}\ 5511 \\ &\Rightarrow \textit{Integer}\ 15511 \\ &\Rightarrow \textit{Digit}\ 15511 \\ &\Rightarrow 115511 \end{aligned}$$

Problem 6

Q: *Do Exercise 2.6 on page 55. Then write a Clite program including these two expressions. Run it to see what the output is.*

The parse tree is shown in Figure 1 and Figure 2.

The Clite program is shown below:

```
1 int main()
2 {
3     int result_1, result_2;
4     result_1 = 5 + 4 * 3;
5     result_2 = 5 * 4 + 3;
6 }
```

With the output:

```
[sqybi@myhost 01]$ java -jar Clite.jar test.cpp
Begin parsing... test.cpp

Program (abstract syntax):
  Declarations:
    Declarations = {<x, int>, <y, int>}
  Block:
    Assignment:
      Variable: x
      Binary:
        Operator: +
        IntValue: 5
        Binary:
          Operator: *
          IntValue: 4
          IntValue: 3
    Assignment:
      Variable: y
      Binary:
        Operator: +
        Binary:
          Operator: *
```

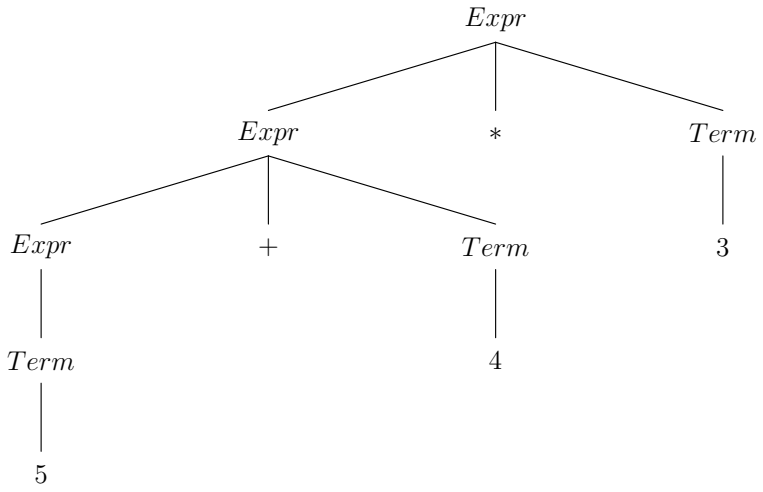

Figure 1: Abstract Syntax Tree for "5+4*3"

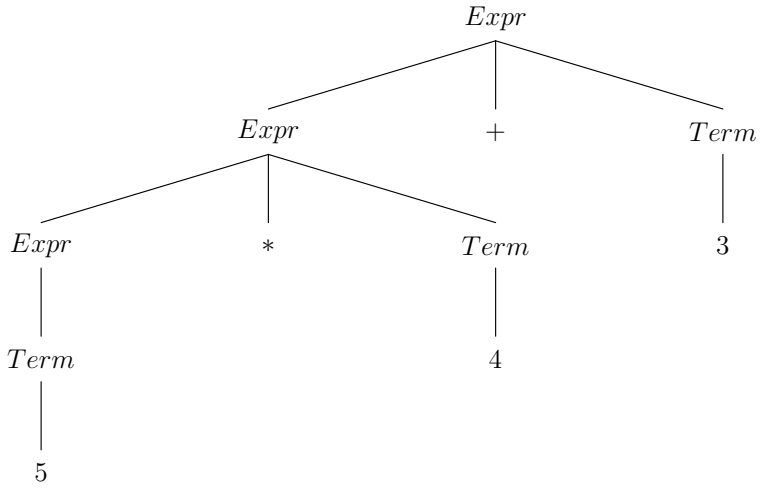

Figure 2: Abstract Syntax Tree for "5*4+3"

```
          IntValue: 5
          IntValue: 4
        IntValue: 3

Begin type checking...test.cpp

Type map:
{ <y, int>, <x, int> }

Transformed Abstract Syntax Tree

Program (abstract syntax):
  Declarations:
    Declarations = {<x, int>, <y, int>}
  Block:
    Assignment:
      Variable: x
      Binary:
        Operator: INT+
        IntValue: 5
        Binary:
          Operator: INT*
          IntValue: 4
          IntValue: 3
    Assignment:
      Variable: y
      Binary:
        Operator: INT+
        Binary:
          Operator: INT*
          IntValue: 5
          IntValue: 4
        IntValue: 3

Begin interpreting...test.cpp

Final State
{ <y, 23>, <x, 17> }
```

Problem 7

Q: *Do Exercise 2.14, page 56. Expand the IF statements in these languages into three short programs, write down these programs and get them to run in their environments.*

Perl

The grammar definition is:

$$\mathit{IfStatement} \rightarrow \text{if (} \mathit{Expression} \text{) \{ } \mathit{Statement} \text{ \} [else \{ } \mathit{Statement} \text{ \}]}$$

A sample code here:

```
1 #!/usr/bin/perl
2 $x = 100;
3 if ($x > 0)
4 {
```

```
    print "positive\n";
}
else
{
    print "non-positive\n";
}
```

And the result is:

```
[sqybi@myhost 01]$ ./perl_prog.pl
positive
```

Python

The grammar definition is:

$$\begin{aligned} \mathit{IfStatement} \rightarrow \text{if } \mathit{Expression} \ : \\ \mathit{Statement} \\ [\ \text{else} \ : \\ \mathit{Statement}\ ] \end{aligned}$$

A sample code here:

```
#!/usr/bin/python
x = 100
if x > 0:
    print("positive")
else:
    print("non-positive")
```

And the result is:

```
[sqybi@myhost 01]$ ./python_prog.py
positive
```

Ada

The grammar definition is:

$$\mathit{IfStatement} \rightarrow \text{if } \mathit{Expression} \text{ then } \mathit{Statement} \text{ else } \mathit{Statement} \text{ endif;}$$

A sample code here:

```
with Ada.Text_IO;
use Ada.Text_IO;

procedure ada_prog is
    x : integer;
begin
    x := 100;
    if x > 0 then
        Put_Line("positive");
    else
        Put_Line("non-positive");
    end if;
end ada_prog;
```

And the result is:

```
[sqybi@myhost 01]$ gnatmake ada_prog.adb
gcc -c ada_prog.adb
gnatbind -x ada_prog.ali
gnatlink ada_prog.ali
[sqybi@myhost 01]$ ./ada_prog
positive
```

Problem 8

Q: *Design another grammar that has the ability to derivate the following expressions: a) 5+4*3; b) 5*4+3.*

One grammar is like this:

$$
\begin{aligned}
Expr &\rightarrow Expr \ + \ Term \mid Term \\
Term &\rightarrow Term \ * \ Factor \mid Factor \\
Factor &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid (\ Expr\ )
\end{aligned}
$$