0. [-] 처리 구현 하였음

1. 언어들 간의 차이
   - Exception 처리
      - c: Exception 처리 기능을 제공하지 않아 error flag 변수를 생성해 처리
      - java: try-catch 및 throw keyword를 이용하여 오류를 손쉽게 처리
         - try-catch-finally
```
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            RDParser p = new RDParser();
            while (true) {
                System.out.print(">> ");
                String tempString = sc.nextLine();
                if (tempString.length() == 0)
                    break;
                p.parse(tempString);
            }
        } catch (ParsingException e) {
            System.out.println("syntax Error!!");
        } finally {
            sc.close();
        }
    }
```
         -throw
```
public static void main(String[] args) {
    if(input.charAt(idx) != '$')
        throw new ParsingException("parsing is not ended!");
```

      - python: try-excpt 및 raise keyword를 이용하여 오류를 손쉽게 처리
         -try-except:
```
while(True):
    try:
        p.parser(input(">>"))
    except ParseException:
        print('syntax error!!')
        break
```
         -raise:
```
if self.input[self.idx] != '$':
    raise ParseException()
```

   - string 문자 접근
      - c, python: 배열에 접근하듯이 index를 이용해 접근
```
c: input_buffer[iterator]
python : self.input[self.idx]
```
      - java : charAt 명령어를 이용해서 접근
```
input.charAt(idx)
```
   - 문자 숫자 변환
      - c, java : 문자를 숫자로 생각하고 연산 + 예외처리 필수
         - 예외처리
```
flag is_valid(char a) {
        switch(a)
        {
```

```
                        case '+':
                        case '-':
                        case '*':
                        case '/':
                        case '(':
                        case ')':
                        case '0':
                        case '1':
                        case '2':
                        case '3':
                        case '4':
                        case '5':
                        case '6':
                        case '7':
                        case '8':
                        case '9':
                        case ' ':
                        case '\t':
                        case '$':
                                return 1;
                        default:
                                return 0;
                }
        }
```

- 문자 변환

```
        int digit = cur_char - '0';
```

- python : 함수를 이용한 변환
  - python : int(self.input[self.idx])

2. 수행 결과

- c

```
➜  hw2 git:(main) ✗ ./a.out
warning: this program uses gets(), which is unsafe.
>>4531 + (32 * 5) + (86+5)
4782
>>(69*8)                +911
1463
>>(74           + 88)
162
>>((32*4) + 1
syntax error!!
```

- java

```
➜  hw2 git:(main) ✗ java JavaLang.java
>> 4531 + (32 * 5) + (86+5)
4782
>> (69*8)         + 911
1463
>> (74          +88)
162
>> ((32*4)      + 1
syntax Error!!
```

- python

```
➜  hw2 git:(main) ✗ python3 python.py
>> 4531 + (32 * 5) + (86+5)
4782
>>(69*8)         + 911
1463
>> (74   +88)
162
>>((32*4)
syntax error!!
```

3. 느낀점
   - Readability (가독성) 및 Writability (작성 유용성)
     - python > java >> c
     - python과 java는 exception 처리가 매우 간단하여 읽기도 쓰기도 쉬웠다.
     - 하지만 c에는 exception 기능이 주어지지 않아 특정 변수에 해당 오류를 작성해 사용하였어야 하였다.
     - python 보다 java가 불편했던 점은 scanner 등 사용해야하는 util들이 모두 class로 이뤄져있어 사용하기에 익숙하지 않았다는 점이다.
   - Reliability (신뢰성)
     - java = python >> c
     - java와 python은 Exception을 제공하여 따로 처리하지 않아도 알아서 string index 오류등을 뱉어주는 반면, c는 모든 함수에 오류가 발생했는지 여부를 체크해줘야 했었다.
     - 또한 python에서는 변환 불가능한 문자에 대한 처리를 따로 제공해주어 해당처리가 매우 쉬웠으며, java에서는 type을 강제하여 부차적으로 발생하는 오류를 막아주었다.

     어떤 것이 더 우월하다고 하기 어려워 둘의 신뢰성을 동일하다고 판단하였다.
   - 결론
     - 해당 프로그램에서는 Exception handling이 필수적으로 구현되어야 했으며, 단순한 문자를 처리하는 프로그램이었으므로, python과 java가 c에 비해서 매우 유리하였다.