

CPU 시뮬레이션과 예외

객체지향 프로그래밍

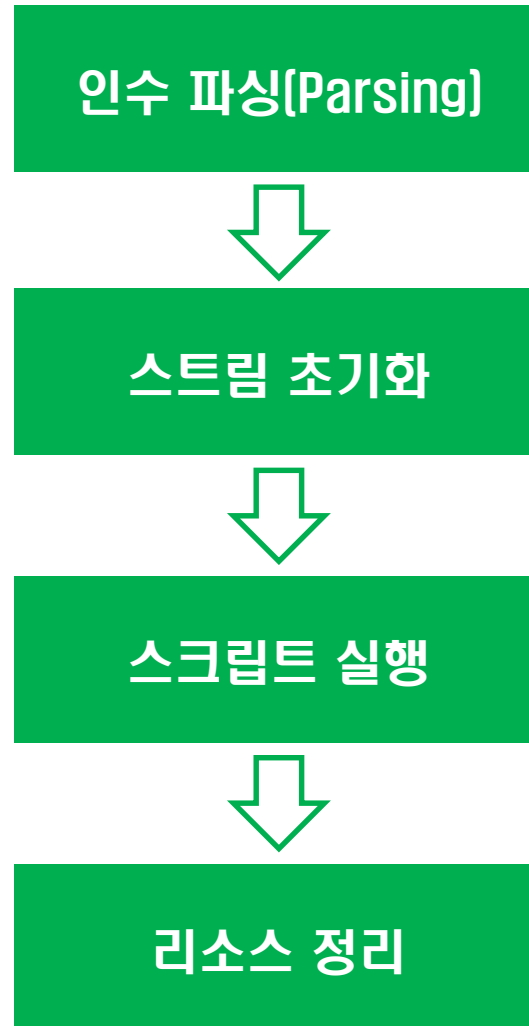
목차

- 프로그램 구조
 - <main> 동작 이론
 - <Process> 동작 이론
- 예외
- 실행결과
- 소감



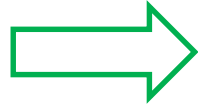
<https://github.com/Vazrupe/Stu20PPProject>

<MAIN> 동작 이론 [1]

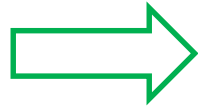


<MAIN> 동작 이론 [2]

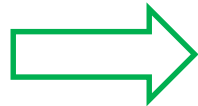
```
// Argument Parsing
for (int i = 1; i < argc; i++) {
    bool isFlag = args[i][0] == '-';
    if (isFlag) {
        if (args[i][1] == 'i') {
            inFile = args[i + 1];
            existInFile = true;
            i++;
            continue;
        }
        if (args[i][1] == 'o') {
            outFile = args[i + 1];
            existOutFile = true;
            i++;
            continue;
        }
        if (args[i][1] == 'd') {
            dump = true;
            continue;
        }
    }
    else {
        sourceFile = args[i];
    }
}
```



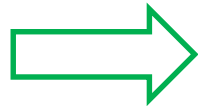
‘-’ 로 시작하는지 확인



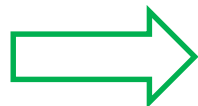
입력 파일 연결



출력 파일 연결



내부 변수 출력

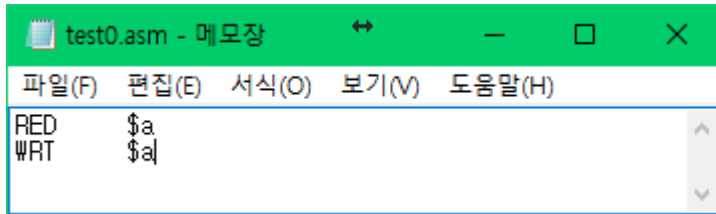


스크립트 파일 지정

Ex1) file.asm : ‘file.asm’ 스크립트를 실행

Ex2) -i input.txt -o output.txt file.asm : ‘input.txt’ 파일을 읽고, ‘output.txt’ 파일에 출력하고, ‘file.asm’ 스크립트를 실행

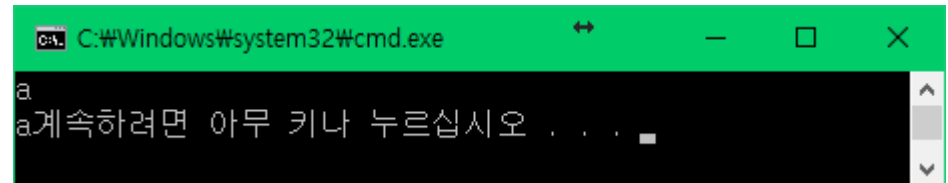
<MAIN> 동작 이론 [3]



```
test0.asm - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
RED    $a
WRT    $a
```

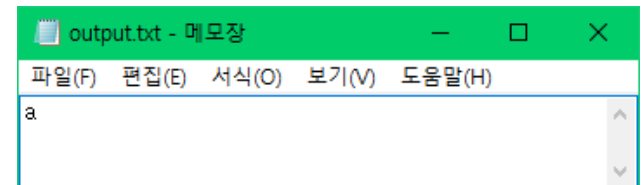
1개의 문자를 읽어 레지스터 A에 저장하고,
레지스터 A에 저장된 문자를 출력하는 스크립트

Ex1) test0.asm : 스크립트 실행



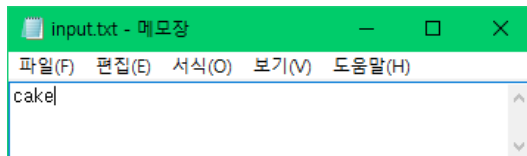
```
C:\Windows\system32\cmd.exe
a
계속하려면 아무 키나 누르십시오 . . .
```

Ex2) -o output.txt test0.asm
: 스크립트 실행하고, 'output.txt'에 결과 저장

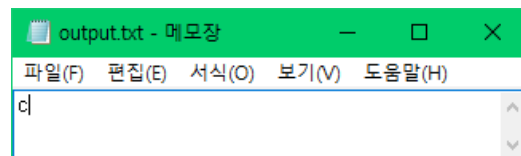


```
output.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
a
```

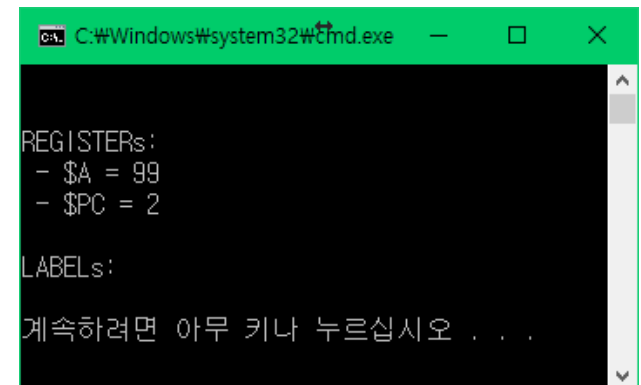
Ex3) -i input.txt -o output.txt -d test0.asm
: 'input.txt'에서 한 문자를 읽어,
'output.txt'에 저장하고 내부 변수 출력



```
input.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
cake|
```



```
output.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
d|
```

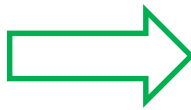


```
C:\Windows\system32\cmd.exe
REGISTERS:
- $A = 99
- $PC = 2

LABELS:
계속하려면 아무 키나 누르십시오 . . .
```

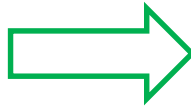
<MAIN> 동작 이론 [4]

```
if (existInFile && existOutFile && (inFile == outFile)) {  
    cout << "Compare In stream And Out stream" << endl;  
    return 0;  
}
```



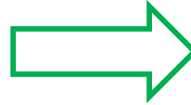
같은 파일인지 확인

```
// Stream open  
istream *in;  
if (existInFile) {  
    in = new ifstream(inFile);  
  
    if (!in) {  
        cout << "In File Open Failed" << endl;  
        return 0;  
    }  
}
```



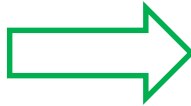
입력 파일 연결

```
}  
else {  
    in = &cin;  
}
```



없다면, Cin으로 연결

```
ostream *out;  
if (existOutFile) {  
    out = new ofstream(outFile);  
  
    if (!out) {  
        cout << "Out File Open Failed" << endl;  
        return 0;  
    }  
}  
else {  
    out = &cout;  
}
```



출력 파일 연결

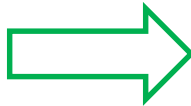
<MAIN> 동작 이론 [5]

```
// init
Process *p;
try {
    p = new Process(sourceFile, in, out);
}
catch (Exception *e) {
    cout << e->what() << endl;
    if (in != &cin) ((ifstream*)in)->close();
    if (out != &cout) ((ofstream*)out)->close();
    return 0;
}

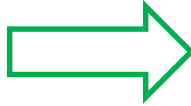
// run
try {
    p->run();
}
catch (Exception *e) {
    cout << e->what() << endl;
    cout << p->dump() << endl;

    delete p;
    if (in != &cin) ((ifstream*)in)->close();
    if (out != &cout) ((ofstream*)out)->close();
    return 0;
}

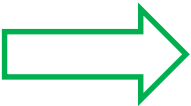
if (dump) {
    cout << endl << endl;
    cout << p->dump() << endl;
}
```



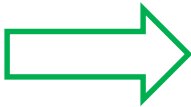
<Process> 초기화



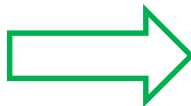
초기화 실패 시 동작



스크립트 실행



실행 실패 시 동작



실행 후 내부 출력

<MAIN> 동작 이론 [6]

```
delete p;  
if (in != &cin) ((ifstream*)in)->close();  
if (out != &cout) ((ofstream*)out)->close();
```

포인터들을 전부 초기화

<PROCESS> 동작 이론 [1]

토큰화(Fetch)



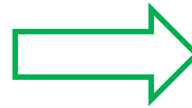
명령 목록 생성
(Decode)



스크립트 실행(Run)

<PROCESS> 동작 이론 [2]

```
Tokenizer Process::Fetch(istream &stm) {  
    Tokenizer tokens(stm, " \\t\\r\\n,");  
    return tokens;  
}
```

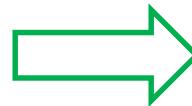


스크립트 데이터 토큰화

```
Tokenizer::Tokenizer(istream &source, string delimiters) {  
    char c;  
    string tokenString = "";  
    unsigned int fnd;  
    do {  
        c = source.get();  
        fnd = delimiters.find(c);  
        bool isDelim = (fnd != string::npos) || (c == EOF);  
        if (isDelim) {  
            if (tokenString.length() > 0) {  
                this->tokens.push_back(tokenString);  
  
                tokenString = "";  
            }  
            continue;  
        }  
  
        tokenString += c;  
    } while (c != EOF);  
}
```



구분자를 여러 개
입력 가능



한 문자씩 읽으며
토큰으로 생성

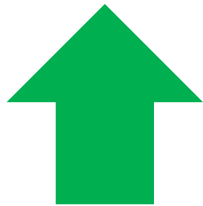


파일 끝까지 읽음

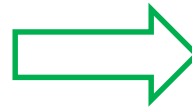
<PROCESS> 동작 이론 [3]

Factory?

; 명령어(Operation) 객체를 동적
생성해주는 모듈



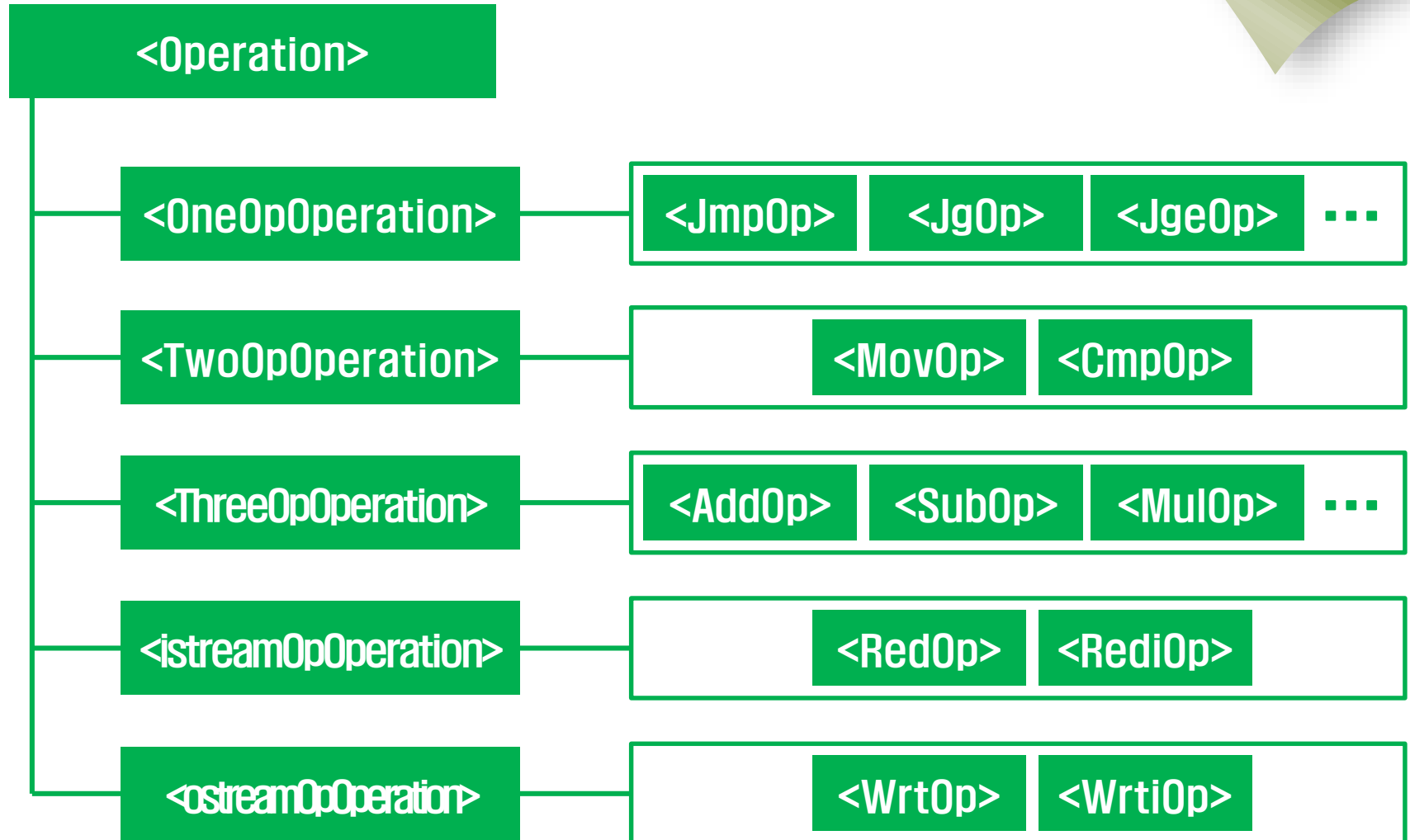
```
void Process::Decode(Tokenizer tokens, istream *in, ostream *out) {  
    OneOpOperationFactory oneFac;  
    TwoOpOperationFactory twoFac;  
    ThreeOpOperationFactory threeFac;  
    istreamOpOperationFactory istmFac;  
    ostreamOpOperationFactory ostmFac;  
  
    for (unsigned int i = 0; i < tokens.size(); i++) {  
        try {  
            string token = uppercase(tokens.getToken(i));
```



한 개의 토큰 단위,
토큰은 대문자 처리

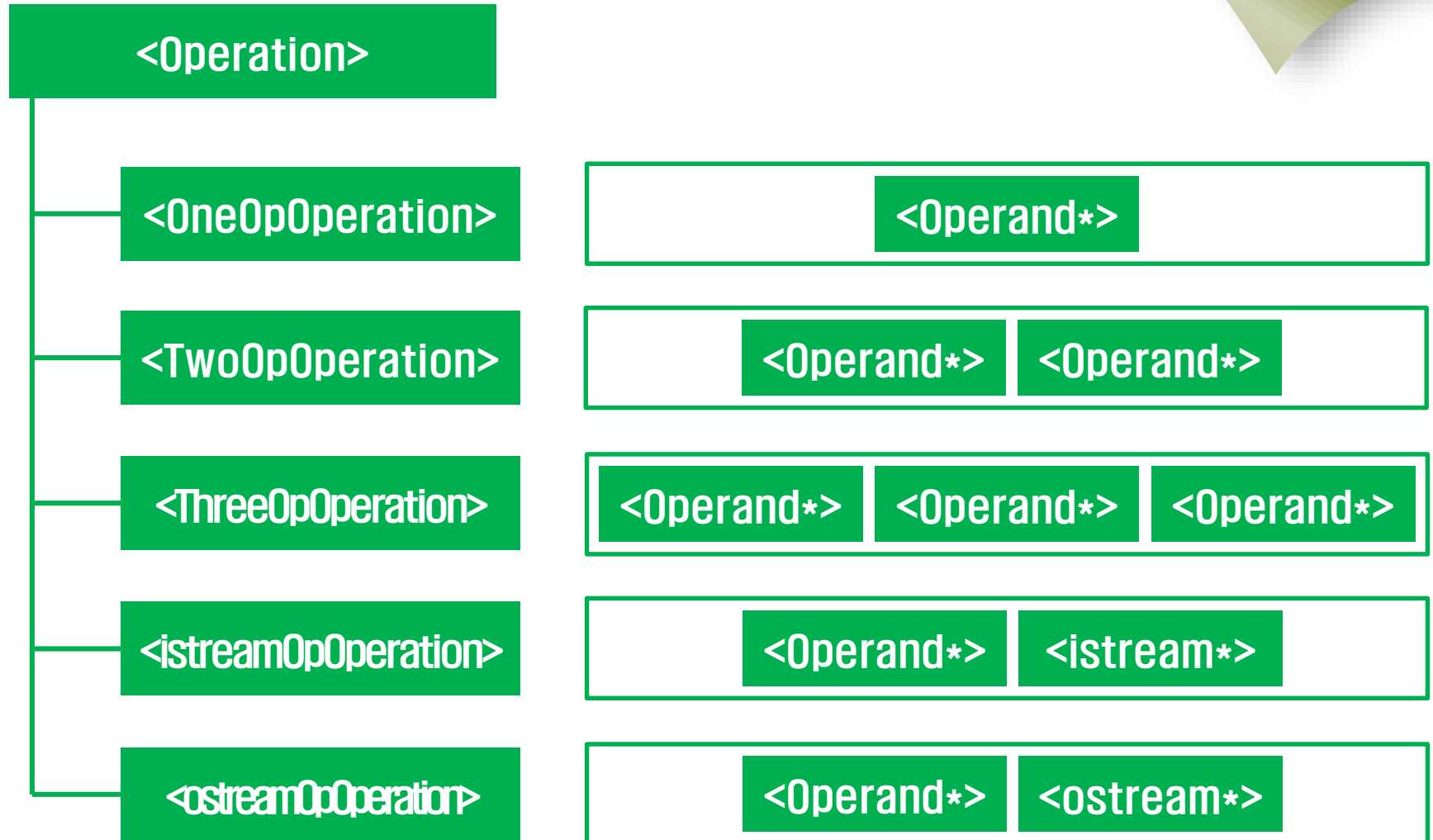
<PROCESS> 동작 이론 [4]

<Operation> 계층 구조



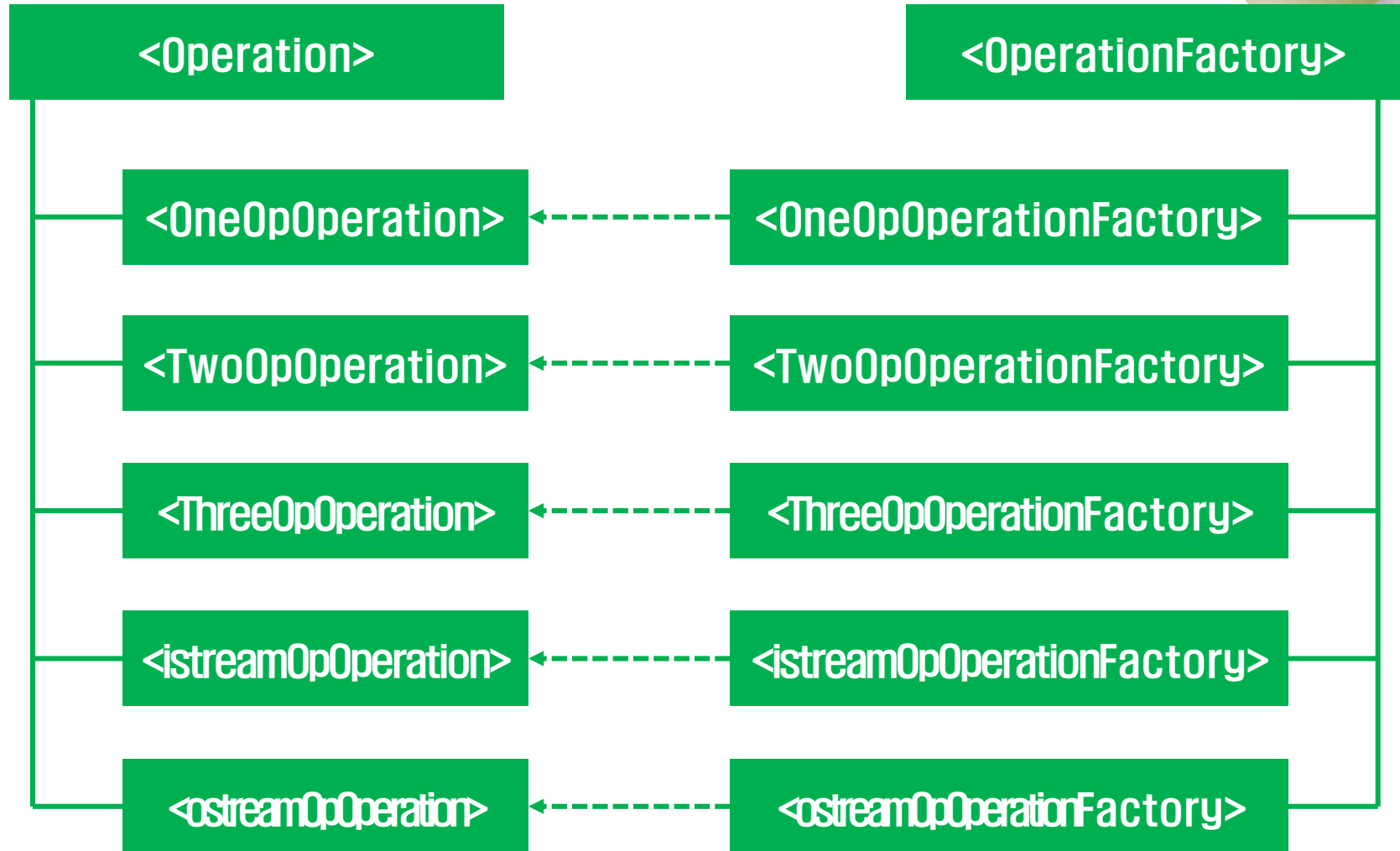
<PROCESS> 동작 이론 [5]

<Operation> 파라미터



<PROCESS> 동작 이론 [6]

<Operation> 생성자



<PROCESS> 동작 이론 [7]

<Operand>도 모양보고
자동 생성

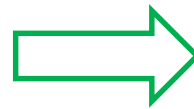
; `@label` -> 라벨
`123` -> 숫자
` 'c' ` -> 단일 문자[아스키]



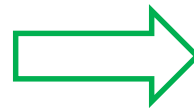
```
// Check Token Type is Operation
if (Operation::isOperation(token)) {
    if (OneOpOperation::isOperation(token)) {
        Operand *op = Operand::Build(tokens.getToken(i + 1));

        OneOpOperation* oper = oneFac.OneOpOperationBuild(token, op);
        ops.push_back((Operation*)oper);

        i += 1;
        continue;
    }
    else if (TwoOpOperation::isOperation(token)) {
```



토큰과 오퍼랜드를
넣으면 동적 생성



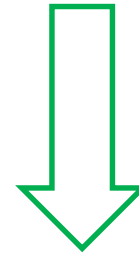
다음 토큰은 오퍼랜드,
인덱스를 하나 늘려줌

<PROCESS> 동작 이론 [8]

```
// Check Token Type is Label  
if (isLabel(token)) {  
    r.setLabel(token.substr(0, token.size() - 1), ops.size() - 1);  
    continue;  
}
```



라벨 형식 토큰 읽기
Label:이면 Label로 저장



다음에 오는
<Operation>을 가리킴

<PROCESS> 동작 이론 [8]

만약 라벨(Label)도, 명령(Operation)도 아니라면?

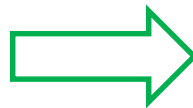
```
// Undefined Token
```

```
throw new UndefinedTokenException(token);
```

; 정의되지 않은(Undefined) 토큰이라는 예외를 출력함

=> 예외에 대해선 나중에 알아보자

```
void Process::run() {  
    try {  
        while ((unsigned int)r.getReg("PC") < ops.size())  
            ops.at(r.getReg("PC"))->run(r);  
    }  
    catch (Exception *e) {  
        throw e;  
    }  
}
```

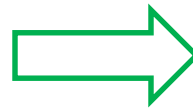


실제 소스는 딱 2줄

<PROCESS> 동작 이론 [9]

이걸 가능하게 하는게 상속과 가상함수!

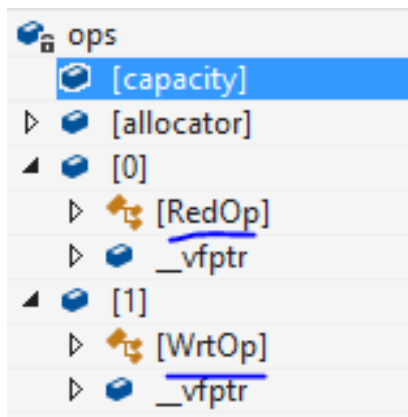
```
class Operation {  
    static vector<string> operationList;  
protected:  
    static bool isOperation(vector<string> &  
public:  
    void nextRow(Register& reg);  
    virtual void run(Register& reg) = 0;  
    static bool isOperation(const string &to  
};
```



```
// 덧셈  
void AddOp::run(Register& reg) {  
    op1->setValue(reg, op2->getVal  
  
    nextRow(reg);  
}
```

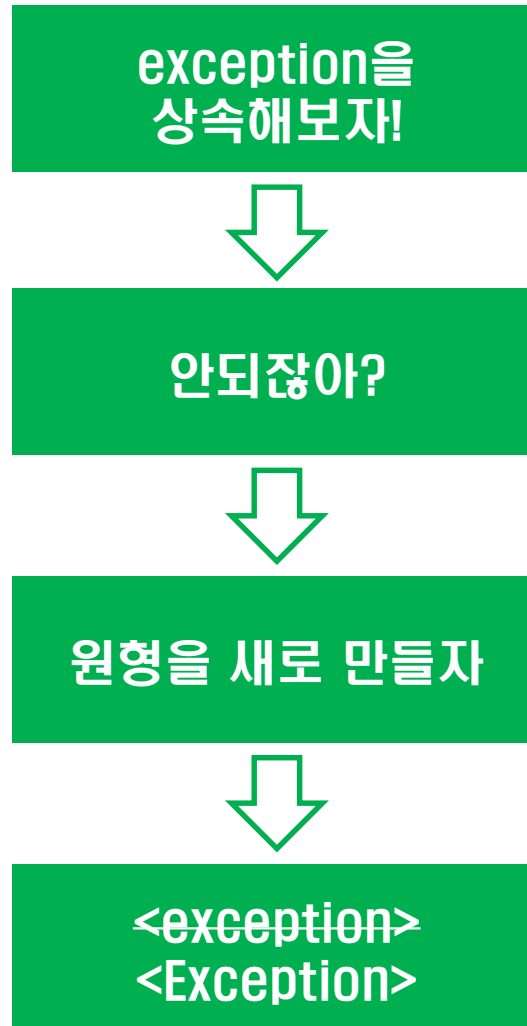
이걸 선언하고...

실제 객체에서 구현하면 끝!



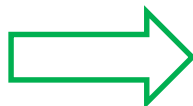
실제 구현 객체가 목록에 들어있음!

예외 [1]

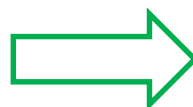


예외 [2]

```
class Exception {  
    string name;  
  
    virtual string message();  
public:  
    Exception(const string &name = "");  
    virtual string what();  
};
```

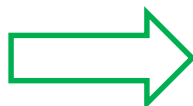


예외 이름을 지정

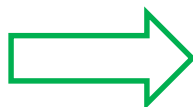


what()으로 정보를 확인

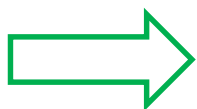
```
Exception::Exception(const string &name) : name(name) {}  
string Exception::what() {  
    return name + ", " + message();  
}  
string Exception::message() {  
    return "";  
}
```



멤버 변수(인자),
되더라...



what()은 exception에서
가져옴

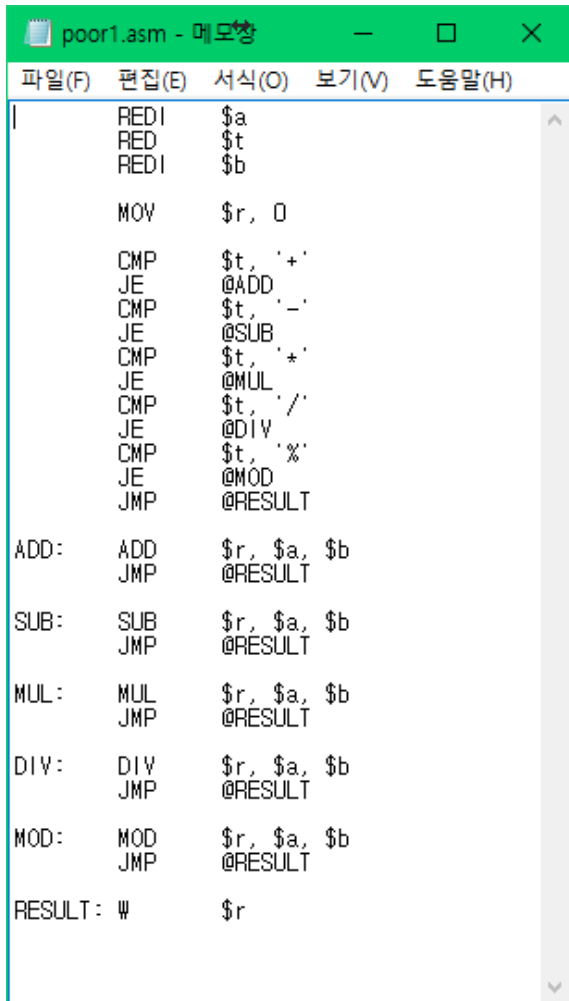


message()의
반환 문자열이 정보

결과는 “*name, message()*” 형식으로 나옴!

실행결과 : 불량 (1)

-i calc_in1.txt -o calc_out1.txt poor1.asm



```
poor1.asm - 메모장
파일(F)  편집(E)  서식(O)  보기(V)  도움말(H)
|
    REDI    $a
    RED     $t
    REDI    $b

    MOV     $r, 0

    CMP     $t, '+'
    JE      @ADD
    CMP     $t, '-'
    JE      @SUB
    CMP     $t, '*'
    JE      @MUL
    CMP     $t, '/'
    JE      @DIV
    CMP     $t, '%'
    JE      @MOD
    JMP     @RESULT

ADD:    ADD     $r, $a, $b
        JMP     @RESULT

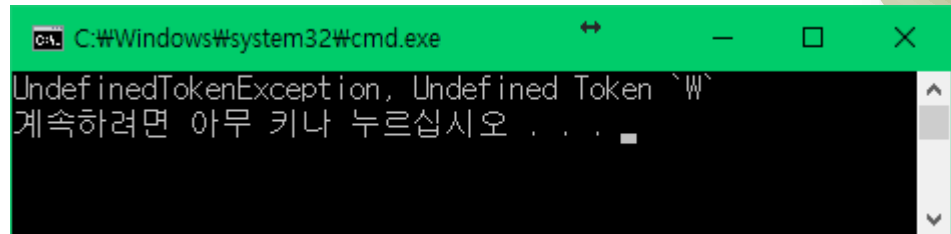
SUB:    SUB     $r, $a, $b
        JMP     @RESULT

MUL:    MUL     $r, $a, $b
        JMP     @RESULT

DIV:    DIV     $r, $a, $b
        JMP     @RESULT

MOD:    MOD     $r, $a, $b
        JMP     @RESULT

RESULT:  $r
```



```
C:\Windows\system32\cmd.exe
UndefinedTokenException, Undefined Token `W`
계속하려면 아무 키나 누르십시오 . . .
```

- 명령어가 W로 잘 못 적힘(마지막)
- 정의되지 않은 토큰에 대한 예외

실행결과 : 불량 [2]

-i calc_in1.txt -o calc_out1.txt poor2.asm

```
poor2.asm - 메모장
파일(F)  편집(E)  서식(O)  보기(V)  도움말(H)
|
    RED    $a
    RED    $t
    RED    $b

    MOV     $r, 0

    CMP     $t, '+'
    JE      @ADD
    CMP     $t, '-'
    JE      @SUB
    CMP     $t, '*'
    JE      @MUL
    CMP     $t, '/'
    JE      @DIV
    CMP     $t, '%'
    JE      @MOD
    JMP     @RESULT

ADD:  ADD     $r, $a, $b
      JMP     @RESULT

SUB:  SUB     $r, $a, $b
      JMP     @RESULT

MUL:  MUL     $r, $a, $b
      JMP     @RESULT

DIV:  DIV     $r, $a, $b
      JMP     @RESULT

MOD:  MOD     $r, $a, $b
      JMP     @RESULT

RESULT: WRTI
```

```
C:\Windows\system32\cmd.exe
OutOfRangeException, Index Out of Range `73 < 73`
계속하려면 아무 키나 누르십시오 . . .
```

- WRTI의 오퍼랜드(Operand)가 없음
- 토큰의 수가 부족해 OutOfRange 예외

실행결과 : 불량 [3]

-i calc_in1.txt -o calc_out1.txt poor3.asm

```
poor3.asm - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
REDI $a
REDI $t
REDI $b

MOV $r, 0

CMP $t, '+'
JE @AD
CMP $t, '-'
JE @SUB
CMP $t, '*'
JE @MUL
CMP $t, '/'
JE @DIV
CMP $t, '%'
JE @MOD
JMP @RESULT

ADD: ADD $r, $a, $b
JMP @RESULT

SUB: SUB $r, $a, $b
JMP @RESULT

MUL: MUL $r, $a, $b
JMP @RESULT

DIV: DIV $r, $a, $b
JMP @RESULT

MOD: MOD $r, $a, $b
JMP @RESULT

RESULT: WRTI $r
```

```
C:\Windows\system32\cmd.exe
UndefinedLabelException, Undefined Label Name `AD`
REGISTERS:
- $A = 123
- $B = 24
- $EQUAL = 1
- $OVER = 0
- $PC = 5
- $R = 0
- $T = 43
- $UNDER = 0

LABELS:
- $ADD => 0p(14)
- $DIV => 0p(20)
- $MOD => 0p(22)
- $MUL => 0p(18)
- $RESULT => 0p(24)
- $SUB => 0p(16)

계속하려면 아무 키나 누르십시오 . . .
```

- 정의되지 않은 라벨로 점프
- 정의되지 않은 라벨에 대한 예외

실행결과 : 불량 [4]

-i calc_in1.txt -o calc_out1.txt poor4.asm

```
poor4.asm - 메모장
파일(F)  편집(E)  서식(O)  보기(V)  도움말(H)

REDI    $a
REDI    $t
REDI    $b

MOV      0, 0

CMP      $t, '+'
JE        @AD
CMP      $t, '-'
JE        @SUB
CMP      $t, '*'
JE        @MUL
CMP      $t, '/'
JE        @DIV
CMP      $t, '%'
JE        @MOD
JMP      @RESULT

ADD:     ADD    $r, $a, $b
        JMP    @RESULT

SUB:     SUB    $r, $a, $b
        JMP    @RESULT

MUL:     MUL    $r, $a, $b
        JMP    @RESULT

DIV:     DIV    $r, $a, $b
        JMP    @RESULT

MOD:     MOD    $r, $a, $b
        JMP    @RESULT

RESULT:  WRTI   $r
```

```
C:\Windows\system32\cmd.exe
NotReferenceDestinationException, Destination Only Register
REGISTERS:
- $A = 123
- $B = 24
- $PC = 3
- $T = 43

LABELs:
- $ADD => 0p(14)
- $DIV => 0p(20)
- $MOD => 0p(22)
- $MUL => 0p(18)
- $RESULT => 0p(24)
- $SUB => 0p(16)

계속하려면 아무 키나 누르십시오 . . .
```

- 목적 레지스트가 지정되지 않음
- 목적지는 항상 레지스트만 가능

실행결과 : 불량 [5]

-i calc_in1.txt -o calc_out1.txt poor5.asm

```
poor5.asm - 메모장
파일(F)  편집(E)  서식(O)  보기(V)  도움말(H)

    REDI    $a
    REDI    $t
    REDI    $b

    MOV     $r, test

    CMP     $t, '+'
    JE      @AD
    CMP     $t, '-'
    JE      @SUB
    CMP     $t, '*'
    JE      @MUL
    CMP     $t, '/'
    JE      @DIV
    CMP     $t, '%'
    JE      @MOD
    JMP     @RESULT

ADD:    ADD     $r, $a, $b
        JMP     @RESULT

SUB:    SUB     $r, $a, $b
        JMP     @RESULT

MUL:    MUL     $r, $a, $b
        JMP     @RESULT

DIV:    DIV     $r, $a, $b
        JMP     @RESULT

MOD:    MOD     $r, $a, $b
        JMP     @RESULT

RESULT: WRTI    $r
```

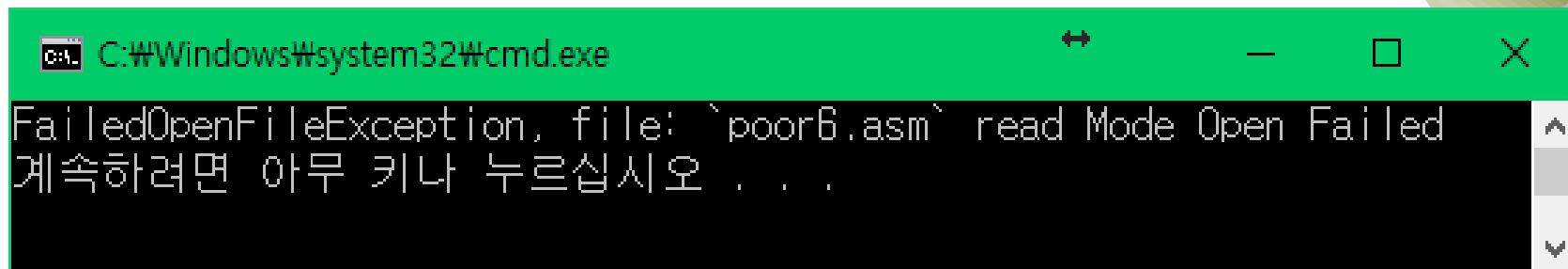
```
C:\Windows\system32\cmd.exe

InvalidOperandException, This Token Unvalid Operand `test`
계속하려면 아무 키나 누르십시오 . . .
```

- 유효하지 않은 오퍼랜드 사용
- 잘못된 오퍼랜드 예외 반환

실행결과 : 불량 [6]

`-i calc_in1.txt -o calc_out1.txt poor6.asm`



A screenshot of a Windows command prompt window. The title bar is green and contains the text "C:\Windows\system32\cmd.exe". The command prompt area is black with white text. The first line shows an error: "FailedOpenFileException, file: `poor6.asm` read Mode Open Failed". The second line shows a Korean message: "계속하려면 아무 키나 누르십시오 . . .". The window has standard Windows window controls (maximize, minimize, close) in the title bar.

```
C:\Windows\system32\cmd.exe
FailedOpenFileException, file: `poor6.asm` read Mode Open Failed
계속하려면 아무 키나 누르십시오 . . .
```

- 스크립트 파일이 없을 경우
- 파일을 여는데 실패했다는 예외 반환

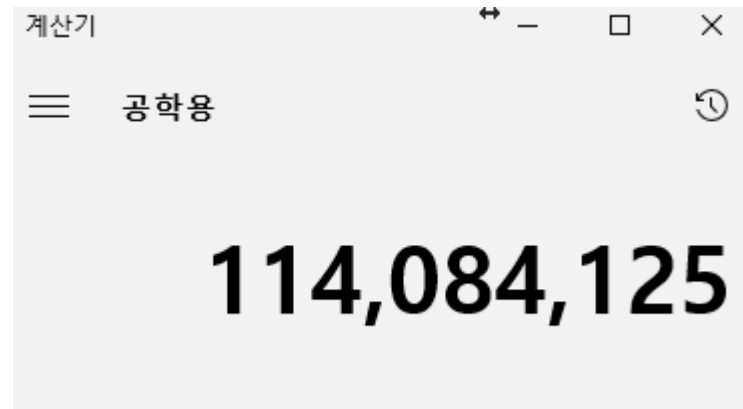
실행결과 : 지수승

power.asm

```
power.asm - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
    REDI    $a
    REDI    $n
    MOV     $i, 0
    MOV     $r, 1
LOOP:
    CMP     $i, $n
    JGE     @END
    MUL     $r, $r, $a
    ADD     $i, $i, 1
    JMP     @LOOP
END:
    WRTI    $r
    WRT     10
```

- $485^3 = 114084125$

```
C:\Windows\system32\cmd.exe
485 3
114084125
계속하려면 아무 키나 누르십시오 . . .
```



실행결과 : 피보나치

fibonacci.asm

```
fibonacci.asm - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

MOV    $a, 0
MOV    $b, 1

MOV    $i, 0
REDI    $cnt

LOOP:
CMP    $cnt, 1
JL     @END

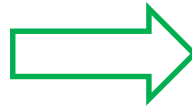
ADD    $c, $a, $b
MOV    $a, $b
MOV    $b, $c

WRTI    $b
WRT     10

ADD    $i, $i, 1

CMP    $i, $cnt
JL     @LOOP

END:|
```



```
C:\Windows\system32\cmd.exe

13
1
2
3
5
8
13
21
34
55
89
144
233
377
계속하려면 아무 키나 누르십시오 . . .
```

입력한 값 만큼 피보나치 수를 출력해주는 소스

실행결과 : 계산 [1]

```
calc.asm - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
|
REDI $a
REDI $t
REDI $b

MOV $r, 0

CMP $t, '+'
JE @ADD
CMP $t, '-'
JE @SUB
CMP $t, '*'
JE @MUL
CMP $t, '/'
JE @DIV
CMP $t, '%'
JE @MOD
JMP @RESULT

ADD: ADD $r, $a, $b
JMP @RESULT

SUB: SUB $r, $a, $b
JMP @RESULT

MUL: MUL $r, $a, $b
JMP @RESULT

DIV: DIV $r, $a, $b
JMP @RESULT

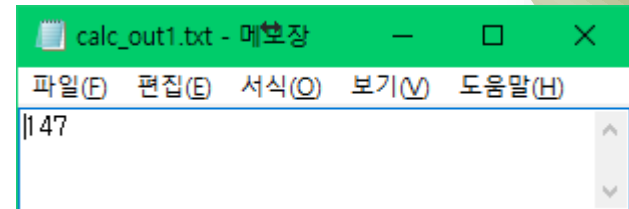
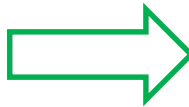
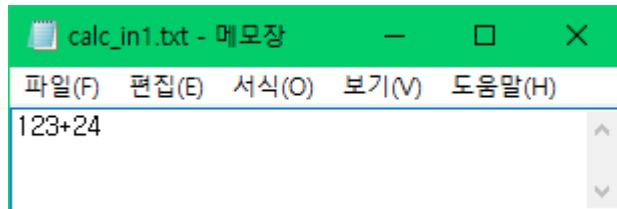
MOD: MOD $r, $a, $b
JMP @RESULT

RESULT: WRTI $r
```

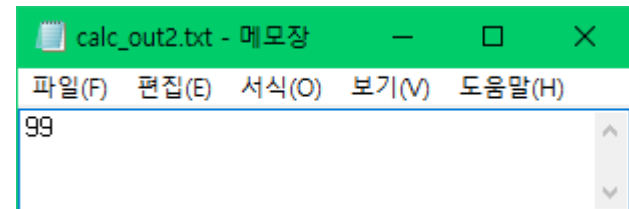
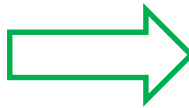
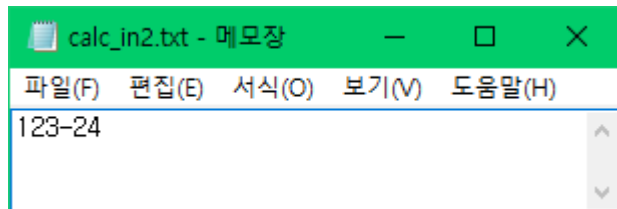
- $\$R \leq \$A \text{ op } \$B$
계산하고 출력해주는게 끝
- op는 +, -, *, /, %가 가능
- + => 더하기
- - => 빼기
- * => 곱하기
- / => 나누기
- % => 나머지 구하기

실행결과 : 계산 [2]

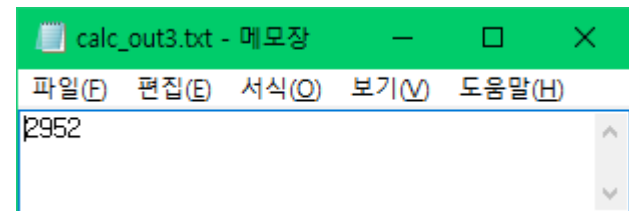
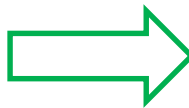
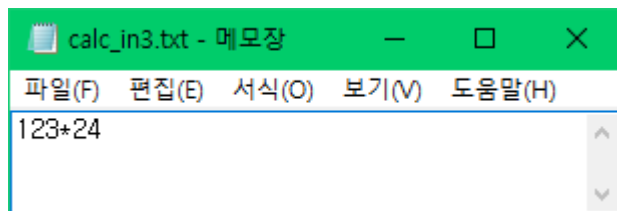
1) `-i calc_in1.txt -o calc_out1.txt calc.asm`



2) `-i calc_in2.txt -o calc_out2.txt calc.asm`

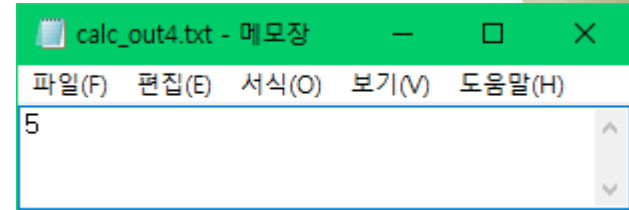
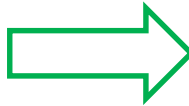
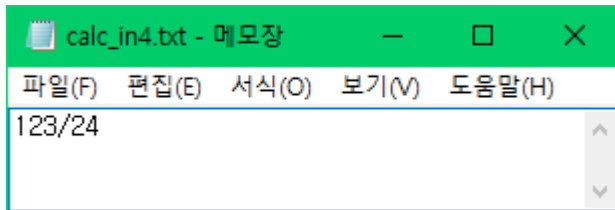


3) `-i calc_in3.txt -o calc_out3.txt calc.asm`

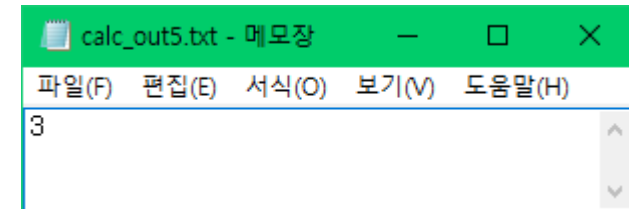
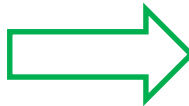
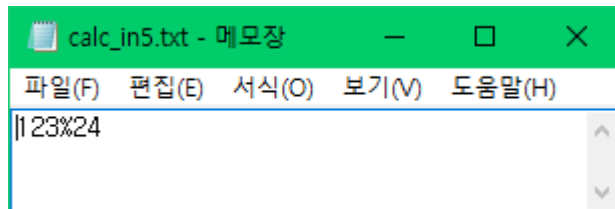


실행결과 : 계산 [3]

4) `-i calc_in4.txt -o calc_out4.txt calc.asm`

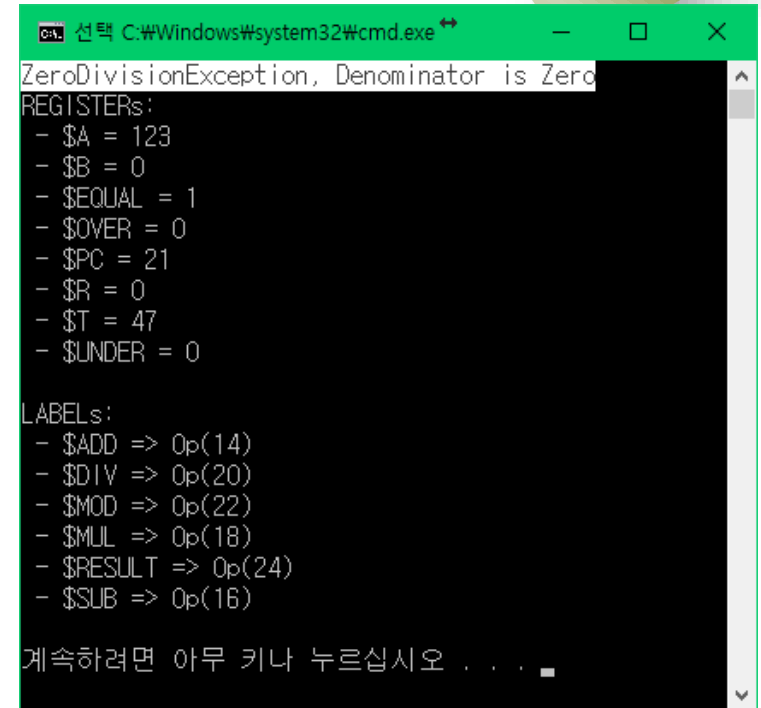
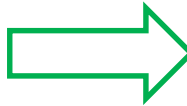
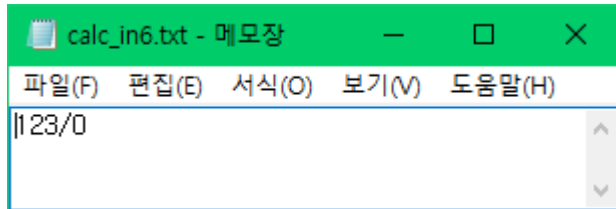


5) `-i calc_in5.txt -o calc_out5.txt calc.asm`



실행결과 : 계산 [4]

6) `-i calc_in6.txt -o calc_out6.txt calc.asm`



분모가 0이라 예외 발생

예외 발생시 우측처럼,
레지스터와 라벨 목록을 출력해줌

실행결과 : GCD와 LCM [1]

gcd_lcm.asm

```
gcd_lcm.asm - 메모장
파일(F)  편집(E)  서식(O)  보기(V)  도움말(H)

    REDI    $a
    REDI    $b

    MOV     $lcm, $a
    MOV     $tmpb, $b

    CMP     $a, $b
    JL      @REV
    JMP     @LOOP

REV:
    MOV     $tmp, $a
    MOV     $a, $b
    MOV     $b, $tmp

LOOP:
    MOD     $c, $a, $b

    CMP     $c, 0
    JE      @END_GCD

    MOV     $a, $b
    MOV     $b, $c

    JMP     @LOOP

END_GCD:
    MOV     $gcd, $b

    DIV     $lcm, $lcm, $gcd
    MUL     $lcm, $lcm, $tmpb

    WRTI    $gcd
    WRT     32
    WRTI    $lcm
    WRT     10
```

- 유클리드 호제법으로 최대공약수(GCD) 계산
- $A \times B / \text{GCD} = \text{LCM}$ 계산하여 출력

실행결과 : GCD와 LCM [1]

gcd_lcm.asm

```
gcd_lcm.asm - 메모장
파일(F)  편집(E)  서식(O)  보기(V)  도움말(H)

    REDI    $a
    REDI    $b

    MOV     $lcm, $a
    MOV     $tmpb, $b

    CMP     $a, $b
    JL      @REV
    JMP     @LOOP

REV:
    MOV     $tmp, $a
    MOV     $a, $b
    MOV     $b, $tmp

LOOP:
    MOD     $c, $a, $b

    CMP     $c, 0
    JE      @END_GCD

    MOV     $a, $b
    MOV     $b, $c

    JMP     @LOOP

END_GCD:
    MOV     $gcd, $b

    DIV     $lcm, $lcm, $gcd
    MUL     $lcm, $lcm, $tmpb

    WRTI    $gcd
    WRT     32
    WRTI    $lcm
    WRT     10
```

- 유클리드 호제법으로 최대공약수(GCD) 계산
- $A \times B / \text{GCD} = \text{LCM}$ 계산하여 출력

실행결과 : GCD와 LCM [2]

gcd_lcm.asm

- $24 = 2^2 \times 3$, $42 = 2 \times 3 \times 7$
- $\text{gcd} = 2 \times 3 = 6$
- $\text{lcm} = 24 \times 42 / 6 = 168$

- $54 = 2 \times 3^2$, $72 = 2^3 \times 3^2$
- $\text{gcd} = 2 \times 3^2 = 18$
- $\text{lcm} = 54 \times 72 / 18 = 216$

```
C:\Windows\system32\cmd.exe
24 42
6 168
계속하려면 아무 키나 누르십시오 . . .
```

```
C:\Windows\system32\cmd.exe
54 72
18 216
계속하려면 아무 키나 누르십시오 . . .
```

실행결과 : 바벨로니아 법 [1]

sqrt.asm

```
sqrt.asm - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

REDI    $a
MOV     $x, $a

LOOP:   CMP     $a, 25
        JL      @END

        CMP     $x, 0
        JE      @END

        DIV     $sub, $a, $x
        ADD     $y, $x, $sub
        DIV     $y, $y, 2

        SUB     $chk, $x, $y
        CMP     $chk, 0
        JL      @ABS
        JMP     @CHK

ABS:    SUB     $chk, 0, $chk

CHK:    CMP     $chk, 1
        JLE     @END
        MOV     $x, $y

        JMP     @LOOP

END:    WRTI    $x
        WRT     10
```

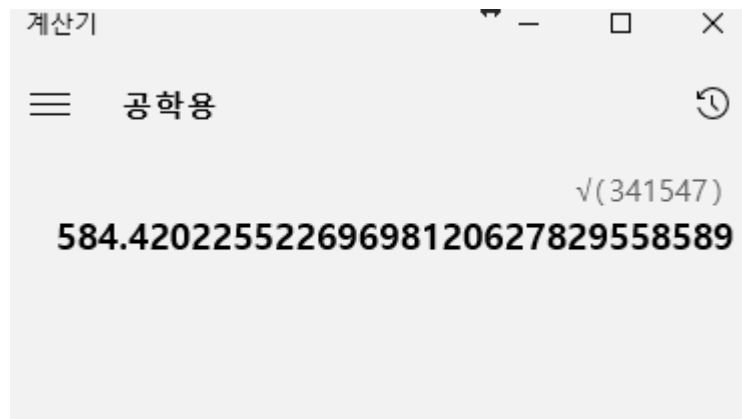
- 바벨로니아 법으로 루트의 근사값을 구함
- 전단계와의 비교 값을 1로 잡음
=> 비교적 정확도가 떨어짐
- 정확도 향상을 위해 25이상의 값으로만 계산

실행결과 : 바벨로니아 법 [2]

sqrt.asm

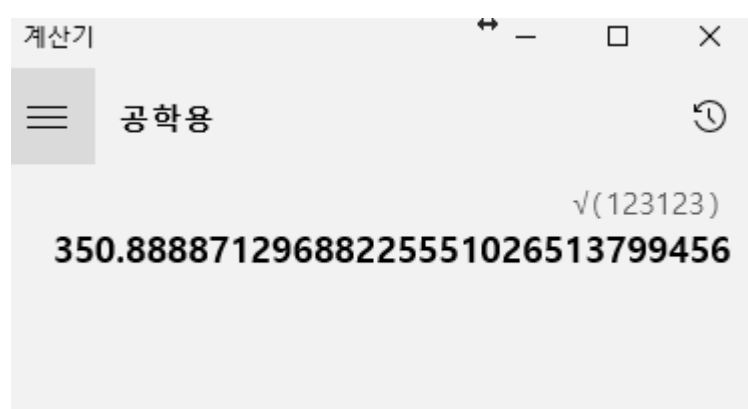
- $\sqrt{341547} \doteq 585$

```
C:\Windows\system32\cmd.exe
341547
585
계속하려면 아무 키나 누르십시오 . . .
```



- $\sqrt{123123} \doteq 350$

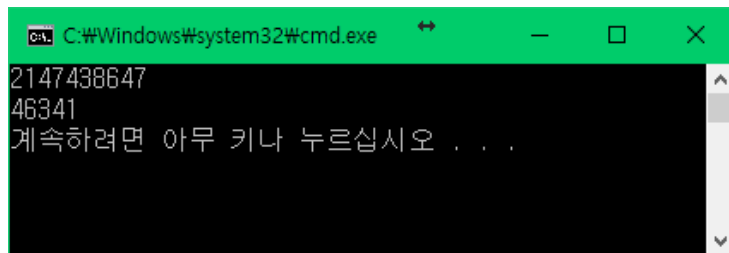
```
C:\Windows\system32\cmd.exe
123123
350
계속하려면 아무 키나 누르십시오 . . .
```



실행결과 : 바벨로니아 법 [3]

sqrt.asm

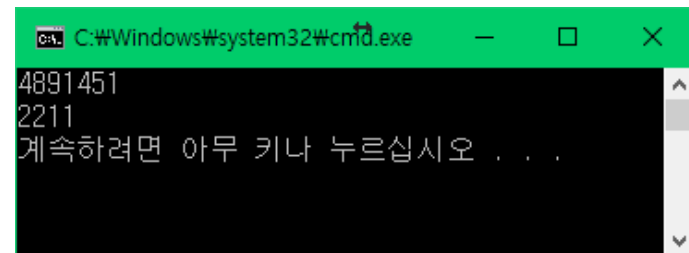
- $\sqrt{2147438647} \doteq 46341$



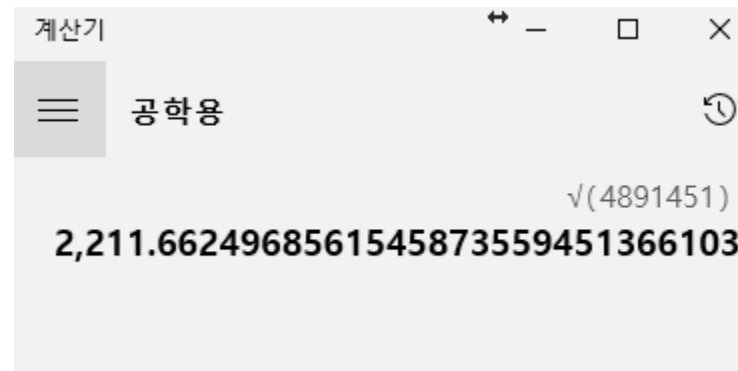
```
C:\Windows\system32\cmd.exe
2147438647
46341
계속하려면 아무 키나 누르십시오 . . .
```



- $\sqrt{4891451} \doteq 2211$



```
C:\Windows\system32\cmd.exe
4891451
2211
계속하려면 아무 키나 누르십시오 . . .
```



실행결과 : CAESAR 암호

-i plain.txt -o cipher.txt caesar.asm

```
caesar.asm - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

| REDI $key
  SUB $revkey, 0, $key
  WRTI $revkey

  RED $dummy
  WRT 10

  MOD $key, $key, 26
  CMP $key, 0
  JL @REV
  JMP @LOOP

REV:
  ADD $key, $key, 26

LOOP:
  RED $c

  CMP $c, 10
  JE @END

  CMP $c, 'A'
  JL @WRITE
  CMP $c, 'Z'
  JLE @UPPER

  CMP $c, 'a'
  JL @WRITE
  CMP $c, 'z'
  JLE @LOWER

  JMP @WRITE
```

```
caesar.asm - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

UPPER:
  SUB $c, $c, 'A'
  ADD $c, $c, $key

  CMP $c, 26
  JGE @U_MOD
  JMP @U_GO

U_MOD: SUB $c, $c, 26
U_GO:  ADD $c, $c, 'A'
      JMP @WRITE

LOWER:
  SUB $c, $c, 'a'
  ADD $c, $c, $key

  CMP $c, 26
  JGE @L_MOD
  JMP @L_GO

L_MOD: SUB $c, $c, 26
L_GO:  ADD $c, $c, 'a'

WRITE: WRT $c
      JMP @LOOP

END:   WRT 10
```

실행결과 : CAESAR 암호 [1]

-i plain.txt -o cipher.txt caesar.asm

```
caesar.asm - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

|      RED    $key
      SUB     $revkey, 0, $key
      WRT    $revkey

      RED     $dummy
      WRT     10

      MOD     $key, $key, 26
      CMP     $key, 0
      JL      @REV
      JMP     @LOOP
REV:
      ADD     $key, $key, 26

LOOP:
      RED     $c
      CMP     $c, 10
      JE      @END

      CMP     $c, 'A'
      JL      @WRITE
      CMP     $c, 'Z'
      JLE     @UPPER

      CMP     $c, 'a'
      JL      @WRITE
      CMP     $c, 'z'
      JLE     @LOWER

      JMP     @WRITE
```

```
caesar.asm - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

UPPER:
      SUB     $c, $c, 'A'
      ADD     $c, $c, $key

      CMP     $c, 26
      JGE     @U_MOD
      JMP     @U_GO

U_MOD: SUB     $c, $c, 26
U_GO:  ADD     $c, $c, 'A'
      JMP     @WRITE

LOWER:
      SUB     $c, $c, 'a'
      ADD     $c, $c, $key

      CMP     $c, 26
      JGE     @L_MOD
      JMP     @L_GO

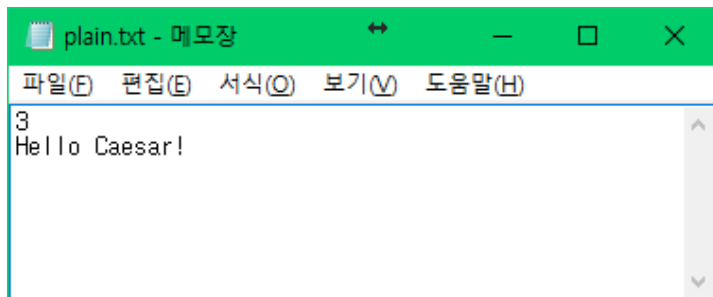
L_MOD: SUB     $c, $c, 26
L_GO:  ADD     $c, $c, 'a'

WRITE: WRT     $c
      JMP     @LOOP

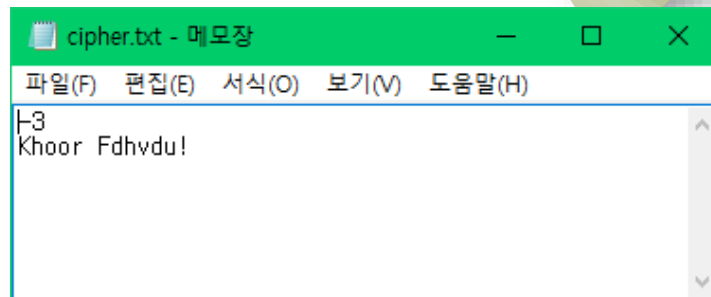
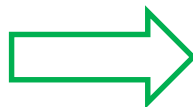
END:   WRT     10
```


실행결과 : CAESAR 암호 [2]

`-i plain.txt -o cipher.txt caesar.asm`



```
plain.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
3
Hello Caesar!
```

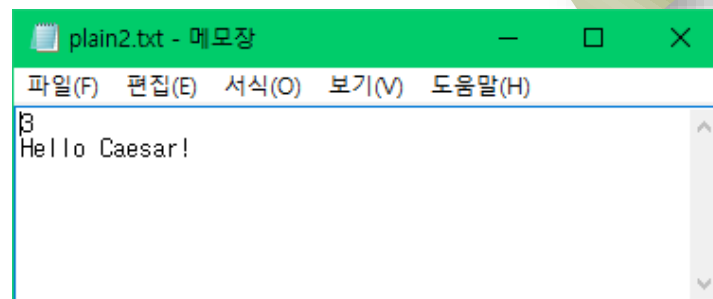
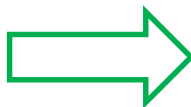
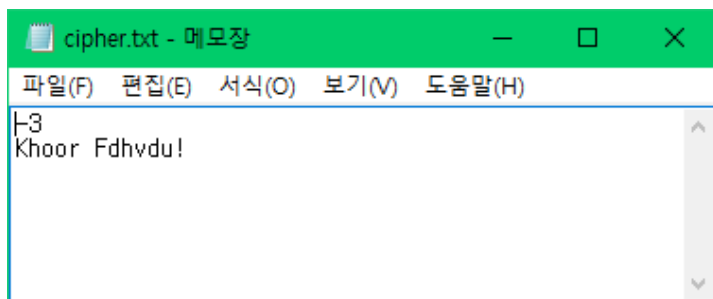


```
cipher.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
3
Khoor Fdhvdu!
```

- 첫 줄에는 암호화키, 둘째 줄에는 문장을 기록함
- 암호화와 동시에 복호화 키를 같이 저장함
- 암호복호화가 모두 가능한 코드

실행결과 : CAESAR 암호 [3]

-i cipher.txt -o plain2.txt caesar.asm



- 암호화된 파일을 읽어 정상적으로 복호화

소감

- 지금까지 한 프로젝트 중에 가장 추상화가 많이 됨
- 추후 DSL을 개발해볼 때 도움이 될 것이라 생각됨
- 다음에는 꼭 파서 제너레이터를 익혀보고 싶음
- 디버깅 시간이 의외로 오래 걸리지 않음
- 다른 프로젝트는 좀 더 일찍일찍 하기로 마음먹음



THANK YOU