

컴파일러 Project#2

컴파일(make)과 실행(chk_examples) 과정을 보여주는 화면 캡처

```
● (base) minsung@minseong-ui-MacBookAir ~/Documents/3학년 2학기/컴파일러/dev/project#2 (main*) $ make
bison -d cool.y
flex cool.l
gcc -Wall -O3 -c lex.yy.c
gcc -Wall -O3 -c cool.tab.c
gcc -Wall -O3 -c node.c
gcc -o cool_parser lex.yy.o cool.tab.o node.o -ll -mmacosx-version-min=13.3
```

```
(base) minsung@minseong-ui-MacBookAir ~/Documents/3학년 2학기/컴파일러/dev/project#2 (main*) $ ./chk_examples
examples/arith.cl → FAILED
1,1c1,283
< [A]([varInt]=0){value{[int]var}{set_var{numInt}}[SELF_TYPE]{var=num self}}{method1{numInt}[SELF_TYPE]{self}{method2{numInt}[numInt][B]}(let (x[Int]) (x<= (* num1 num2) (new B).set_var(x)))){method3{numInt}[C]}(let (x[Int]) (x<=
- num) (new C).set_var(x)))){method4{numInt}[numInt][D]}(if (< num2 num1) (let (x[Int]) (x<= num1 num2) (new D).set_var(x))) (let (x[Int]) (x<= num2 num1) (new D).set_var(x)))){method5{numInt}[E]}(let (x[Int]←1) ((let (y[Int]←1
) (while (<= y num) (x←(* x y) y←(+ y 1))) (new E).set_var(x))))
< [B][A]{method5{numInt}[E]}(let (x[Int]) (x←(* num num) (new E).set_var(x))))
< [C][B]{method6{numInt}[A]}(let (x[Int]) (x←(- num) (new A).set_var(x)))){method5{numInt}[E]}(let (x[Int]) (x←(* num num) (new E).set_var(x))))
< [D][B]{method7{numInt}[B]}(let (x[Int]←num) (if (< x 0) method1((- x)) (if (= 0 x) true (if (< 2 x) false (method2((- x 3)))))))
< [E][D]{method8{numInt}[E]}(let (x[Int]) (x←(- num B) (new A).set_var(x))))
< [A21]{c2(charString)[Int]}(if (= char "1") 0 (if (= char "2") 1 (if (= char "3") 2 (if (= char "4") 3 (if (= char "5") 4 (if (= char "6") 5 (if (= char "7") 6 (if (= char "8") 7 (if (= char "9") 8 (if (= char "0") 9 (abort() 0))))))))
)))){let(c[Int])String(if (= 0) "0" (if (= 1) "1" (if (= 1 2) "2" (if (= 1 3) "3" (if (= 1 4) "4" (if (= 1 5) "5" (if (= 1 6) "6" (if (= 1 7) "7" (if (= 1 8) "8" (if (= 1 9) "9" (abort() ")))))))){a2(s(String))Int(if (= s.le
ngth() 0) 0 (if (= s.substr(0 1) <->) (- a2_num(s.substr(1 (- s.length() 1))) (if (= s.substr(0 1) <->) a2_num(s.substr(1 (- s.length() 1))) a2_aux(s)))){a2_aux(s(String))Int(let (int[Int]←0) (let (j[Int]←s.length()) (let (lin
t←1) (while (< j) (int←(+ int 10) C2(s.substr(1)) i←(+ i 1)))) int))){a2(a1[Int])String(if (= i 0) "0" (if (= 0 i) i2a_aux(i) <->.concat(i2a_aux(+ i (- i 1))))){i2a_aux(i[Int])String(if (= i 0) "" (let (next[Int]←(- i
1 10)) i2a_aux(next).concat(i2c(- i (- next 10))))))}
< [Main][D](charString)[Avar[A]](A).var[A]([flag{bool}=true][num0][String]out_string("vto add a number to ") print(avar) out_string("... enter a\n") out_string("vto negate ") print(avar) out_string("... enter b\n") out_string("v
to find the difference between ") print(avar) out_string("and another number... enter c\n") out_string("vto find the factorial of ") print(avar) out_string("... enter d\n") out_string("vto square ") print(avar) out_string("... enter e\n
") out_string("vto cube ") print(avar) out_string("... enter f\n") out_string("vto find out if ") print(avar) out_string("is a multiple of 3... enter g\n") out_string("vto divide ") print(avar) out_string("by 8... enter h\n") out_strin
g("vto get a new number... enter i\n") out_string("vto quit... enter q\n\n") in_string())){if(prompt()String)out_string("\n") out_string("Please enter a number... ") in_string())}{get_int()Int}{let (r[A21]←(new A21)) (let (s[String]←
prompt()String){if (s.isEven(numInt)[bool]) (let (a[Int]←num) (if (< x 0) is_even((- x)) (if (= 0 x) true (if (< 1 x) false is_even((- x 2))))){class_type(var[A])[[SELF_TYPE]{case var(c[A])→out_string("class type is now A\n") h[B]→o
ut_string("class type is now B\n") c[C]→out_string("class type is now C\n") d[D]→out_string("class type is now D\n") e[E]→out_string("class type is now E\n") o[Object]→out_string("ooops\n"))}{print(var[A])[[SELF_TYPE]{let (r[A21]←(n
ew A21)) (out_string(r.i2a(var.value())) out_string(" "))}{main()Object}{avar←(new A) (while flag (out_string("number ") print(avar) (if is_even(avar.value()) (out_string("is even\n") out_string("is odd\n") class_type(avar) char←men
u() (if (= char "a") (avar←(new A).set_var(int()) avar←(new B).method2(avar.value()) a_var.value()) (if (= char "b") (case avar(c[C])→avar←c.method(c.value()) a[A]→avar←a.method(a.value()) o[Object]→(out_string("ooops\n") ab
ort() 0)) (if (= char "c") (avar←(new A).set_var(get_int()) avar←(new D).method4(avar.value()) a_var.value()) (if (= char "d") avar←(new C).method3(avar.value()) (if (= char "e") avar←(new C).method5(avar.value()) (if (= char
"f") avar←(new C).method5(avar.value()) (if (= char "g") (if (new D).method7(avar.value()) (out_string("number ") print(avar) out_string("is divisible by 3.\n")) (out_string("number ") print(avar) out_string("is not divisible by 3.\n
")) (if (= char "h") (let (x[A]) (let (x←(new E).method8(avar.value()) (let (r[Int]←(- avar.value() (+ x.value() B))) (out_string("number ") print(avar) out_string("is equal to ") print(x) out_string("times 8 with a remainder of ") (let (a[A
21]←(new A21)) (out_string(r.i2a(r)) out_string("\n")))) avar←x) (if (= char "j") avar←(new A) (if (= char "q") flag←false avar←(new A).method1(avar.value()))))))))))))}
> syntax error in line 9 at ";
> Error in class definition, skipping. in line 9 (unexpected EOF)
> syntax error in line 11 at "value"
> Error in class definition, skipping. in line 11 (unexpected EOF)
> Error in class definition, skipping. in line 11 (unexpected EOF)
> syntax error in line 13 at "set_var"
> Error in class definition, skipping. in line 15 (unexpected EOF)
> syntax error in line 16 at "self"
> Error in class definition, skipping. in line 16 (unexpected EOF)
> syntax error in line 17 at "i"
> Error in class definition, skipping. in line 18 (unexpected EOF)
> syntax error in line 20 at "method1"
> Error in class definition, skipping. in line 22 (unexpected EOF)
> syntax error in line 24 at "method2"
> Error in class definition, skipping. in line 27 (unexpected EOF)
> syntax error in line 28 at "c"
> Error in class definition, skipping. in line 28 (unexpected EOF)
> syntax error in line 29 at "i"
> Error in class definition, skipping. in line 31 (unexpected EOF)
> syntax error in line 33 at "method3"
> Error in class definition, skipping. in line 36 (unexpected EOF)
> syntax error in line 37 at "c"
> Error in class definition, skipping. in line 37 (unexpected EOF)
> syntax error in line 38 at "j"
> Error in class definition, skipping. in line 40 (unexpected EOF)
> syntax error in line 42 at "method4"
> Error in class definition, skipping. in line 46 (unexpected EOF)
> syntax error in line 47 at "c"
> Error in class definition, skipping. in line 47 (unexpected EOF)
> syntax error in line 48 at "j"
> Error in class definition, skipping. in line 53 (unexpected EOF)
> syntax error in line 54 at "c"
> Error in class definition, skipping. in line 54 (unexpected EOF)
> syntax error in line 55 at "j"
> Error in class definition, skipping. in line 58 (unexpected EOF)
> syntax error in line 60 at "method5"
> Error in class definition, skipping. in line 66 (unexpected EOF)
```

실행 결과에 대한 설명, 소감, 문제점

● 실행 결과 설명

1. bad.ci의 결과

- 구문 오류가 다수 발생하였으며, 대부분 **unexpected EOF** 또는 특정 토큰(**;**, **test1**, **C** 등)에서 발생한 오류입니다.
- 오류 메시지는 **syntax error in line X at <토큰>** 형식으로 표시되며, 이어지는 메시지로 "Error in class definition, skipping."이 출력됩니다.
- 최종적으로 **36개의 오류**가 발견되었다고 출력되었습니다.

2. good.ci의 결과

- 오류 메시지 패턴은 **bad.ci** 과 유사하며, 특정 연산자(**~**, **+**, **@**) 또는 변수 이름(**y**, **z**, **foo**)에서 구문 오류가 발생하였습니다.
- 결과적으로 추상구문트리(AST)를 출력하지 못했으며, **54개의 오류**가 발견되었다고 출력되었습니다.

- 소감
 - 이번 과제는 COOL 언어의 문법을 BNF로 정의하고 구문 검사 및 AST 생성을 구현하는 과제였습니다.
 - 엄청난 노력과 시간을 쏟아부었음에도 실행 결과가 기대에 미치지 못하여 아쉬움이 큼니다.
 - Bison과 Flex를 사용한 파서 작성은 처음이었고, 이를 통해 구문 분석기의 내부 작동 방식을 배우는 좋은 기회였습니다.
 - 하지만 많은 오류가 발생하면서 구현의 복잡함과 디버깅의 중요성을 다시 한번 느꼈습니다.
- 문제점 분석
 1. BNF 정의의 불완전성
 - 구문 오류가 지나치게 많이 발생한 이유는 BNF 정의가 COOL 언어의 문법을 정확히 반영하지 못했기 때문일 가능성이 큼니다.
 - 예를 들어, 클래스 정의, 조건문, 연산자 우선순위와 같은 문법 규칙이 누락되었거나 잘못 작성되었을 수 있습니다.
 2. 에러 핸들링 부족
 - 오류가 발생했을 때 오류 토큰만 출력하고 이후 처리가 제대로 이루어지지 않는 문제가 있습니다.
 - `unexpected EOF` 와 같은 메시지가 반복적으로 출력된 이유는 구문 오류 후 복구 과정이 미흡했기 때문입니다.
 3. 추상구문트리(AST) 생성 실패
 - `good.c1` 파일의 AST가 출력되지 않은 이유는 의미 행위(semantic actions)가 제대로 정의되지 않았기 때문일 수 있습니다.
 - `node.h` 와 `node.c` 에서 정의된 노드 생성 관련 코드에 문제가 있거나, Bison의 의미 행위 연동이 실패했을 가능성이 있습니다.
 4. 디버깅 도구 부족
 - Bison에서 제공하는 디버깅 옵션(`-debug` 등)을 활용하지 않아 구문 분석 과정을 시각적으로 확인하지 못했을 수 있습니다.
- 개선 방안
 1. 문법 정의 재검토
 - COOL 언어의 문법 정의를 다시 확인하고, BNF 정의가 정확히 반영되었는지 검토해야 합니다.
 - 공식 COOL 문법 또는 수업 자료와 비교하여 누락된 규칙이나 잘못된 정의를 수정해야 합니다.
 2. 디버깅 도구 활용
 - Bison의 디버깅 옵션을 활성화하여 토큰 처리 및 구문 분석 과정에서 문제가 발생하는 부분을 정확히 파악해야 합니다.
 - `yydebug` 를 사용하면 상태 전환 및 처리 과정을 확인할 수 있습니다.
 3. 의미 행위(semantic actions) 개선
 - 의미 행위 정의가 구문 분석과 올바르게 연동되었는지 확인하고, `node.h` 및 `node.c` 에서 AST 노드 생성을 테스트해야 합니다.
 4. 에러 복구 로직 추가
 - 구문 오류 이후 복구 로직을 추가하여 오류를 무시하고 다음 구문을 처리하도록 개선해야 합니다.

- 이를 통해 `unexpected EOF` 오류를 줄이고 보다 정확한 오류 메시지를 출력할 수 있습니다.

5. 테스트 케이스 확장

- 다양한 COOL 프로그램을 테스트하여 파서가 예상대로 동작하는지 확인하고, 발견된 문제를 하나씩 해결해야 합니다.

- 최종 결론

이번 과제는 파서 작성의 복잡성을 실감하는 동시에, 구문 분석기와 AST 생성에 대한 깊은 이해를 요구하는 도전적인 과제였습니다.

결과적으로 **목표를 달성하지 못했지만**, 이를 통해 파서 작성의 기본 개념을 학습할 수 있었으며, 부족한 부분을 보완하여 더 나은 결과를 기대할 수 있을 것입니다. **향후 개선 방향을 기반으로 지속적으로 디버깅하고 수정해 나가겠습니다.**