**2017년 1학기**

**프로그래밍언어개론 설계 계획서**

**(팀이름) : 자... 이제 누가 버스기사지???**

<작업의 분할>

Item 1

-각자 구현

Item 2

a. define 테이블 생성 및 구현 (동하)

b. insertTable 구현 (동하)

c. lookupTable 구현 (진수)

d.기존 코드에서 INT Type 대신 ID Type입력시 Table에서 꺼내오도록 수정

(cons ~ plus 부분 -> 동하) (minus ~ eq 부분 -> 진수)

=> run\_expr 수정으로 모두 적용

e. lambda 구현

(평일 짬짬이 하며 주말에 같이 동방에서 구현)

Symbol Table 추가

Define에서 Lambda 처리

Lambda 변수 바인딩 및 삭제

Lambda 변수 여러 개 처리

Lambda 옵션 적용

구현 일정 계획

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 작업내용 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 6.1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 팀결성 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Item1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Item2 a |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Item2 b |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Item2 c |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Item2 d |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Item2 e |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| refactoring |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Bug fix |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

* 1. **팀 구성**

|  |  |  |  |
| --- | --- | --- | --- |
| **학번** | **이름** | **역할** | |
| **Item1** | **Item2** |
| **201202141** | **김동하** | **All** | **defTable ,insertTable구현, Symbol Table 생성, Lambda 기본구현, Lambda 옵션 처리,**  **test, refactoring** |
| **201203391** | **임진수** | **All** | **lookupTable구현, run\_expr 수정, Lambda 변수 바인딩 및 삭제. Lambda 다중 변수 처리,**  **test, refactoring** |

\* 팀장 (조교가 연락을 취할 수 있는 대표) : 김동하

* 1. **수정 로그**

17.05.21 Item1 구현 Test 코드 삭제 및 input 추가

17.05.21 define 구현

17.05.22 insertTable 구현

17.05.22 lookupTable 구현

* + 1. 최종 완료
  1. **코드 설명**

**Define ()**

l\_node은 defTable의 key 값이 되고 r\_node의 값은 value 값이 됩니다. Define을 할 때는 기본적을 value 값을 run\_expr 해서 리스트 내부의 값을 실행한 후 quote 된 값만 저장을 합니다. 하지만 저장해야할 value가 lambda 라면 실행을 하지 않고 lambda 식을 저장합니다.

**insertTable ()**

defTable[id] = value 이런 식으로 l\_node의 value(해당 ID 노드의 값)을 id로 받아 key로 사용하고 r\_node를 value로 해서 저장합니다.

**lookupTable ()**

defTable에 id라는 key 값을 가진 페어가 있다면 저장된 value를 리턴하고 없으면 none을 리턴하도록 했습니다.

**Lambda ()**

Lambda의 기본 모형은 argument까지 포함하였을 때 ((lambda (x) (defines) (expression)) argument) 이런 모형입니다. 여기서 기존 다른 오퍼레이션과 다르게 lambda는 리스트 내부가 아닌 밖에 argument를 가져와서 함수에서 사용을 합니다. 그래서 lambda 함수를 실행하기전에 run\_list에서 내부에 lambda가 있는지 검사를 한 후 argument에 대하여 처리를 한 후 lambda 식으로 넘어가서 argument를 받아서 사용하도록 구현하였습니다.

Lambda로 들어오면 l\_node에는 파라미터의 리스트가 오고 r\_node는 옵션 또는 body가 선택이 됩니다. 만약 옵션(안에 define이 들어있는 list)라면 defTable에 define 옵션을 처리해서 그 옵션을 제외하고 다시 run\_expr()을 실행하고 만약 옵션이 복수라면 몇번 반복하여 옵션을 전부처리해줍니다. 그러면 바디에서 실행될 변수가 전부 defTable에 들어가 있는 상태가 됩니다. 옵션을 넣으면서 기존에 전역변수로 선언된 값이 있다면 saveValue에 해당 value를 저장합니다. 그리고 바디부분이라면 결과를 실행하여 결과를 받고 그 후 재귀에서 나오면서 saveValue를 다시 defTable에 복원을 하여서 내부에서 선언된 변수를 내부에서만 실행하고 외부에서 정의된 변수는 영향을 받지 않도록 구현했습니다.

**insertValue ()**

root\_node에서 변수 부분과 값 부분, 빈테이블 saveTable을 파라미터로 받습니다.

받은 변수와 값을 defTable에 넣습니다. 넣기 전에 이미 defTable에 변수의 값이 존재한다면 saveTable에 값을 저장해두고 defTable에 넣습니다.

**removeValue ()**

root\_node에서 변수 부분과 값 부분,insertValue()에서 사용한 saveTable을 파라미터로 받습니다. saveTable에 변수의 값이 존재하지 않는다면 defTable에서 변수 값을 지우고 존재한다면 saveTable의 값으로 변경합니다.

**Func()**

Opcode가 미리 정의된 Symbol table에 존재하지 않는다면 func()로 넘어가도록 symbol table에 값을 넣습니다. 그 값에 의해 func()가 실행됩니다.

Func()에서는defTable에 node의 value의 value 즉 Symbol table의 key 값이 defTable에 존재 한다면 그 값을 꺼내와 치환 후 run\_expr()을 실행합니다.

존재하지 않는다면 Error를 출력합니다.

* 1. **동작 설명**

**ID값 복원 )**

Define으로 정의한 ID를 표현식에서 다시 복원을 하도록 run\_expr 함수에서 기존 id 타입이면 해당 노트를 리턴하고 끝이였는데 그 곳에서 id이면 lookupTable 함수를 이용하여 값이 있을 경우 해당 value를 리턴하여 각 오퍼레이션 함수 내부에서 run\_expr()을 실행하여 new\_node를 만들 때 자동으로 치환 되도록 구현을 하였습니다.

**람다 동작 )**

입력한 값이 Node로 변환되어 run\_expr()로 넘어올 경우 제일 첫 노드의 값에 따라서 반환하게 됩니다. 그중 List일 경우에는 run\_list()를 실행합니다.

run\_list() 에서는 list 내부의 첫 값이 list이고 또 그 list안의 value가 lambda일 경우 insertValue()를 재귀적으로 실행해서 lambda에 파라미터와 입력된 argument를 매칭하여서 defTable에 입력합니다. 그다음 lambda의 식의 결과값을 run\_expr()를 실행하여 받아옵니다. 그후에는 removeValue를 재귀 실행하여 lambda에서만 사용되는 변수 값을 제거합니다. 제거후에는 표현식의 결과를 반환합니다.

**일반 표현식 동작 )**

람다가 아닌 일반 빌트인 함수 같은 표현식을 포함하고 있는 리스트가 run\_list() 실행 파라미터로 들어오면 기존과 같이 fun\_func() 함수를 실행한 결과를 리턴합니다.

**Define 된 함수의 동작 )**

Define 된 함수의 경우 run\_func()에서 opcode를 체크할 때 Table[]에 symbol이 등록 되어 있지 않고 defTable에 들어있기 때문에 실행이 되지않습니다. 그렇기 때문에 Table[]에서 함수를 찾지 못하면 해당 opcode로 심볼테이블 Table[]에 func()변수를 추가로 저장을 합니다. func() 변수는 람다식으로 defTable[]에 저장된 함수를 가져와 실행을 해서 결과를 리턴하여 처리를 합니다. 만약 아직 define 되지 않은 값이면 func()에서 에러 메시지를 출력합니다.

**Recursion 된 함수의 동작 )**

Define 된 함수를 실행하게 되면 defTable[]에 함수가 저장이 되어있습니다. 그래서 define된 함수는 func() 내부에서 defTable[]에 저장된 함수를 가져와서 실행을 합니다.

실행 할 때 run\_list()에서 내부로 saveTable[]에 이미 선언되어 있는 변수를 저장하고 defTable[] 값을 다음 실행할 argument로 바꾸고 함수를 실행합니다. 그러고 결과를 받아오면 defTable[]에 값을 복원하면서 콜스택을 빠져나옵니다.

**변수 scope 구현 )**

전역 변수로 저장된 변수는 defTable[]에 바로 저장을 해서 표현식을 실행할 때 defTable[]에서 ID를 검색할 때 어디서든지 찾아서 쓸 수 있도록 하였고 표현식을 실행하면서 내부에서 선언된 변수는 내부에 코드를 실행 할 때 같은 이름의 전역변수가 있으면 다른 곳에 저장을 하고 defTable[] 업데이트하여서 실행을 하고 함수 실행이 끝나면서 나올 때 이전 변수값을 복원하거나 사용된 변수는 삭제하여서 내부에서 선언된 변수는 내부에서만 사용이 가능하고 이 과정에서 전역변수 내용이 수정되지 않도록 처리했습니다.

* 1. **느낀점**

임진수 :

인터프리터를 구현하면서 racket하고 최대한 비슷한 결과를 내도록 만드는 것에 시간이 많이 들어갔습니다. 기존 인터프리터를 참고하여 만들면서 함수형언어의 특징을 알 수 있었고 재귀적으로 프로그램을 구현하는 방법에 대하여 배울 수 있었습니다.