CBNU
CHUNGBUK NATIONAL UNIVERSITY

컴파일러
2022 프로젝트

제출일 2022.05.31

학과 소프트웨어학과

학번 2020039009

이름 차현아

담당교수 이재성 교수님

추가한 기능

1. 사칙연산
2. 괄호 우선순위
3. if-else문
4. while문

- 추가한 토큰

```
25  더하기  {return(ADD); }
26  빼기    {return(SUB); }
27  곱하기  {return(MUL); }
28  나누기  {return(DIV); }
29  :=      {return(ASSGN); }
30  ;       {return(STMTEND); }
31  "("     {return(LGAL); }
32  ")"     {return(RGAL); }
33  만약에  {return(IF);}
34  반복    {return(WHILE);}
35  아니면  {return(ELSE);}
36  "@"     {return(JUMP);}
37  ">"     {return(GT);}
38  "<"     {return(LT);}
39  ">="    {return(GE);}
40  "<="    {return(LE);}
41  "{"     {return(LPARA);}
42  "}"     {return(RPARA);}
43  시작    {return(START); }
44  IF완료  {return(DONE);}
45  끝      {return(END); }
```

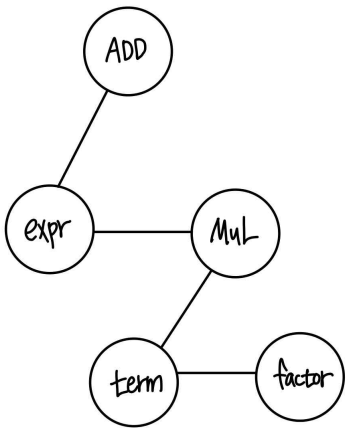
- cbu2.y 파일에서 토큰 정의

```
%token  LGAL RGAL MUL DIV ADD SUB ASSGN LPARA RPARA JUMP WHILE FOR GT LT GE LE ID NUM STMTEND START DONE END IF LOVE STAR ELSE ID2
```

1. 사칙연산 / 괄호 우선순위

```
expr    :   expr ADD term   { $$=MakeOPTree(ADD, $1, $3); }
        |   expr SUB term   { $$=MakeOPTree(SUB, $1, $3); }
        |   term
        ;

term    :  term MUL factor { $$=MakeOPTree(MUL, $1, $3); }
        |  term DIV factor { $$=MakeOPTree(DIV, $1, $3); }
        |  factor
        ;

factor  :   LGAL expr RGAL { $$=$2;}
        |   ID      { /* ID node is created in lex */ }
        |   NUM     { /* NUM node is created in lex */ }
        ;
```

- 괄호->곱하기,나누기->더하기,빼기 순서로 우선순위를 구현하였습니다.
- MakeOPTree 함수를 사용하여 expr = expr + term * factor 식으로 구축되는 AST형태
- dfs 탐색을 사용하여 정한 우선 순위를 확인 할 수 있습니다.



*** 사칙 연산 예시(arith.cbu 파일)**

```
시작
가:=3;
나:=4;
다:=2;
라:= 가 더하기 나 곱하기 다;
마:= 가 더하기 다 곱하기 20 나누기 다 빼기 1;
끝
```

```
C:\Users\user\Desktop\DevKit (1)>cbu arith.cbu

sample CBU compiler v2.0
(C) Copyright by Jae Sung Lee (jasonlee@cbnu.ac.kr), 2022.
2020039009_ChaHyeona.
Successfully compiled. Assembly code is in 'a.asm'.

C:\Users\user\Desktop\DevKit (1)>StackSim a.asm
extended abstract Stack Machine Simulator (StackSim) v1.1
(C)opyright by Jae Sung Lee (jasonlee@cbnu.ac.kr), 2022.

Successfully assembled.
```

<결과>

```
Successfully executed.

[DATA Segment Dump]
Loc#  Symbol        Value
   0  가                3
   1  나                4
   2  다                2
   3  라               11
   4  마               22
[End of DATA Segment]
```

- '라', '마' 값을 통해 연산자 우선순위가 정상적으로 적용되었음을 알 수 있습니다.
- '마' 값을 계산해보면,

다 곱하기 20= 40, 40 나누기 다 = 20, 20 더하기 가 = 23, 23 빼기 1 = 22

*** 사칙 연산 괄호 적용 예시(galho 파일)**

```
시작
가:=3;
다:=4;
마:=2;
나:= 가 더하기 마 곱하기 20 나누기 (다 더하기 다);
라:= 가 곱하기 (마 더하기 다);
바:= (가 더하기 2) 곱하기 (20 나누기 마) 빼기 10;
끝
```

```
C:\Users\user\Desktop\DevKit (1)>cbu arith.cbu

sample CBU compiler v2.0
(C) Copyright by Jae Sung Lee (jasonlee@cbnu.ac.kr), 2022.
2020039009_ChaHyeona.
Successfully compiled. Assembly code is in 'a.asm'.

C:\Users\user\Desktop\DevKit (1)>StackSim a.asm
extended abstract Stack Machine Simulator (StackSim) v1.1
(C)opyright by Jae Sung Lee (jasonlee@cbnu.ac.kr), 2022.

Successfully assembled.
```

<결과>

```
[DATA Segment Dump]
Loc#  Symbol        Value
   0  가                3
   1  다                4
   2  마                2
   3  나                8
   4  라               18
   5  바               40
[End of DATA Segment]
```

- '나' 계산

-> (다 더하기 다)=8, 마 곱하기 20=40,
40 나누기 8=5,가 더하기 5=8 (결과 일치)

- '바' 계산

-> (가 더하기 2)=5, (20 나누기 마)=10,
5 곱하기 10=50, 50 빼기 10= 40 (결과 일치)

2. if-else문

```
stmt    :   ID ASSGN expr STMTEND   { $1->token = ID2; $$=MakeOPTree(ASSGN, $1, $3);}
        |   ID JUMP ID STMTEND { $$=MakeOPTree(JUMP, $1, $3);}
        |   IF LGAL com_pare RGAL LPARA stmt_list RPARA ELSE stmt_list DONE { $$=MakeLongTree(IF, $3, MakeLabel(LOVE, $6), MakeLabel(STAR, $9));}
        |   WHILE LGAL com_pare RGAL LPARA stmt_list RPARA { $$=MakeLongTree(WHILE, MakeLabel(STAR, NULL), $3, MakeLabel(LOVE, $6));}
        ;
```

```
com_pare : expr GT expr { $$=MakeOPTree(GT, $1, $3); }
         | expr LT expr { $$=MakeOPTree(LT, $1, $3); }
         | expr LE expr { $$=MakeOPTree(LE, $1, $3); }
         | expr GE expr { $$=MakeOPTree(GE, $1, $3); }
         ;
```

- stmt에 if-else문을 추가했습니다.
- 규칙은 IF(com_pare) ELSE{stmt_list}IF완료; 형식으로 MakeLongTree 함수와 MakeLabel 함수를 따로 만들어서 구현하였습니다.
- ID JUMP ID STMTEND는 무조건 점프를 해야할 때 사용하는 문장입니다.

```
Node * MakeLabel(int op, Node *operand1)
{
    Node * node;
    node = (Node *) malloc(sizeof (Node));
    node->token = op;
    node->tokenval =op;
    node->son = operand1;
    node->brother = NULL;
    return node;
}
```

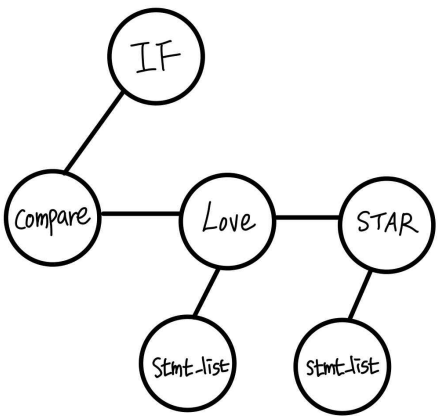
- MakeLabel(int, Node*)의 기능은 새로운 노드를 만들어 인자로 받은 값으로 라벨을 생성하고 stmt_list를 son으로 연결해주어 (LABEL 주소)를 asm코드에서 알맞은 자리에 출력해주기 위함입니다. (LABEL 주소)가 asm에서 적절한 위치에 들어가면, if-else문과 while문의 조건 검사에 활용됩니다.

```
Node * MakeLongTree(int op, Node* operand1, Node* operand2, Node* operand3)
{
    Node * newnode;
    newnode = (Node *)malloc(sizeof (Node));

    Node * newnode1;
    newnode1 = (Node *)malloc(sizeof (Node));

    newnode->token = op;
    newnode->tokenval = op;

    newnode->son = operand1;
    newnode->brother = NULL;

    operand1->brother= operand2;
    operand2->brother= operand3;

    return newnode;
}
```

- 인자로 받은 op를 새로운 노드의 토큰벨류로 지정해주고, operand1을 새로운 노드의 son으로, operand1의 brother를 operand2로, operand2의 brother을 operand3로 연결해주는 트리입니다.

```
$$=MakeLongTree(IF, $3, MakeLabel(LOVE, $6), MakeLabel(STAR, $9));
```

- 위의 MakeLongTree 호출로 구축되는 AST 형태입니다.



- compare에서 조건을 만족하지 않으면 else문인 LOVE로 점프하여 else문이 실행되고, compare에서 조건을 만족하여 {} 안에서 문장들이 실행 된 후, else문 밖으로 무조건 점프하라는 토큰을 만나면 else문을 건너띄고 else문 다음에 있는 라벨 STAR로 점프하게 됩니다.

- case문에 조건문 계산을 위해 추가한 토큰들

```
case GE:
    fprintf(fp, "-₩n GOMINUS memloc₩n");
    break;
case LE:
    fprintf(fp, "-₩n GOPLUS memloc₩n");
    break;
case LT:
    fprintf(fp, "-₩n COPY ₩n GOPLUS memloc₩n GOFALSE memloc₩n");
    break;
case GT:
    fprintf(fp, "-₩n COPY ₩n GOMINUS memloc₩n GOFALSE memloc₩n");
    break;
case LOVE:
    fprintf(fp, "LABEL memloc ₩n");
    break;
case STAR:
    fprintf(fp, "LABEL loc ₩n");
    break;
case JUMP:
    fprintf(fp, "GOTO loc ₩n");
    break;
```

- GE : a>=b 조건문에서 a-b의 값이 음수이면, 조건을 만족하지 않으므로 GOMINUS memloc를 사용하여 memloc으로 점프합니다. a-b의 값이 0이라면, memloc으로 점프하지 않고 조건문 안인 다음 asm코드를 실행합니다.
- GT : a>b 조건문에서는 a-b의 값이 1. 음수인지 확인, 2. 0인지 확인, 총 두 번의 확인을 해야하므로 먼저 COPY를 사용해서 스탭의 top값을 복사해서 푸쉬합니니다. 스택의 top값을 복사해주는 과정을 거치지 않으면 두 번의 검사가 일어날 수 없습니다. COPY를 통해 복사 한 뒤, a-b의 값이 음수이면 조건을 만족하지 않으므로 memloc으로 점프하고, GOFALSE를 사용해서 a-b의 값이 0이어도 조건을 만족하지 않으므로 memloc으로 점프합니다.

- 실제로 LT(<)를 사용했을 때 생성되는 asm코드입니다.
- LT, LE도 GOPLUS가 사용되어 위와 동일한 과정으로 진행됩니다.

- LOVE : LABEL memloc을 스택에 푸시합니다.
- STAR : LABEL loc을 스택에 푸시합니다.
- JUMP : GOTO loc을 스택에 푸시합니다.

```
29    RVALUE 가
30    RVALUE 나
31    GOTO loc
32    LABEL memloc
33    LVALUE 바
34    RVALUE 바
35    RVALUE 가
36    +
37    :=
38    LABEL loc
39    LVALUE 가
40    RVALUE 가
41    RVALUE 가
42    +
43    :=
```

- 앞에서 언급했듯이, 무조건 점프를 할 때 <ID JUMP ID STMTEND>를 사용하여, 해당 문장(토큰 JUMP)을 만나면 GOTO loc을 스택에 푸시합니다.
- 토큰 LOVE를 만나면, LABEL memloc을 스택에 푸시하여 조건식이 F일 때 memloc으로 점프하여 else문을 실행합니다.
- 조건식이 T일 때 if문 안을 다 수행하고 난 뒤, 무조건 점프 문장을 만나면 else문을 수행하지 않고 else 다음으로 뛰어야하므로 LABEL loc을 스택에 푸시합니다. LABEL loc은 조건식이 참이던, 거짓이던 무조건 수행되는 문장입니다.

*** if문 예시(if_true 파일)**

```
시작
가:=3;
나:=5;
다:=8;
바:=0;
만약에 (가<나)
{
      라:= 가 더하기 나;
      마:= 라 더하기 다;
      가@나;
}
아니면 바:= 바 더하기 가;
IF완료
가:=가 더하기 가;
끝
```

```
Successfully executed.

[DATA Segment Dump]
Loc#  Symbol        Value
   0  가                6
   1  나                5
   2  다                8
   3  바                0
   4  라                8
   5  마               16
[End of DATA Segment]
```

- 가(3)<나(5) 조건을 만족하므로, 라= 가 더하기 나 =>8, 마= 라 더하기 다 => 8+8=16, 무조건 점프 문장인 가@나;를 만나서, IF완료 토큰 뒤의 문장 실행, 가 더하기 가 = 6
- 조건을 만족하여 else문은 수행되지 않으므로 바=0

*** if문 예시(if_false 파일)**

```
시작
가:=3;
나:=5;
다:=8;
바:=0;
만약에 (가>나)
{
	라:= 가 더하기 나;
	마:= 라 더하기 다;
	가@나;
}
아니면 바:= 바 더하기 가;
IF완료
가:=가 더하기 가;
끝
```

```
Successfully executed.

[DATA Segment Dump]
Loc#  Symbol        Value
   0  가                6
   1  나                5
   2  다                8
   3  바                3
   4  라                0
   5  마                0
[End of DATA Segment]
```

- 조건을 만족하지 않으므로 else문이 실행되어 바=3이고, IF문 완료 후 무조건 실행되는 문장이 실행되어 가=6입니다.

- if_true 파일로 생성되는 asm코드입니다. 조건식에서 값을 비교하는 부분 부터입니다.

```
13    RVALUE 가
14    RVALUE 나          -
15    =
16     COPY
17     GOPLUS memloc
18     GOFALSE memloc
19    LVALUE 라
20    RVALUE 가
21    RVALUE 나
22    +
23    :=
24    LVALUE 마
25    RVALUE 라
26    RVALUE 다
27    +
28    :=
29    RVALUE 가
30    RVALUE 나
31    GOTO loc
32    LABEL memloc
33    LVALUE 바
34    RVALUE 바
35    RVALUE 가
36    +
37    :=
38    LABEL loc
39    LVALUE 가
40    RVALUE 가
41    RVALUE 가
42    +
43    :=
```

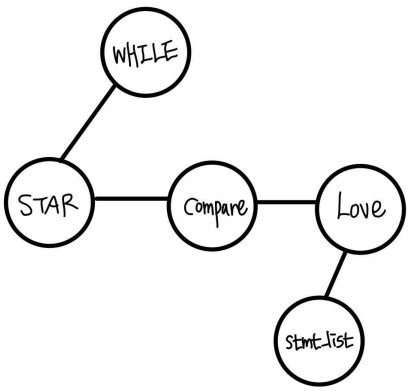
3. while문

```
stmt    :   ID ASSGN expr STMTEND   { $1->token = ID2; $$=MakeOPTree(ASSGN, $1, $3);}
        |   ID JUMP ID STMTEND { $$=MakeOPTree(JUMP, $1, $3);}
        |   IF LGAL com_pare RGAL LPARA stmt_list RPARA ELSE stmt_list DONE { $$=MakeLongTree(IF, $3, MakeLabel(LOVE, $6), MakeLabel(STAR, $9));}
        |   WHILE LGAL com_pare RGAL LPARA stmt_list RPARA { $$=MakeLongTree(WHILE, MakeLabel(STAR, NULL), $3, MakeLabel(LOVE, $6));}
        ;
```

- stmt에 while문을 추가했습니다.
- 규칙은 WHILE(com_pare){stmt_list}; 형식으로 if문에서도 사용했던 MakeLongTree 함수와 MakeLabel함수를 사용합니다.
- ID JUMP ID STMTEND는 무조건 점프를 해야할 때 사용하는 문장입니다.

```
$$=MakeLongTree(WHILE, MakeLabel(STAR, NULL), $3, MakeLabel(LOVE, $6));}
```

- 위의 식으로 구축되는 AST의 형태입니다.



- while문은 조건이 T이면, {}안 문장을 수행하고 무조건 다시 조건을 검사하러 조건식으로 돌아 가야 하기 때문에 WHILE의 son으로 새로운 라벨 STAR을 만들어 배치하였습니다.
- while문 {}안에서 무조건 점프 문장을 만나면, STAR로 점프하여 조건을 확인합니다. 조건이 T이면 다시 {}안을 수행하고, 조건이 F이면 LOVE로 점프합니다.

*** while문 예시(while 파일, <= 조건 만족)**

```
시작
가:=1;
나:=3;
다:=0;
라:=2;
마:=1;
바:=5;
반복 (가<=바)
{
	다:=나 더하기 다;
	가:=가 더하기 마;
	가 @ 나;
}
마:=마 더하기 마;
끝
```

```
[DATA Segment Dump]
Loc#  Symbol        Value
   0  가                6
   1  나                3
   2  다               15
   3  라                2
   4  마                2
   5  바                5
[End of DATA Segment]
```

- 가(1) <= 바(5) 이므로 while문 안으로 들어갑니다.
  다 = 나(3) 더하기 다(0) = 3;
  가 = 가(1) 더하기 마(1) = 2; -> 가@나 (조건식으로 무조건 점프)
- 가(2) <= 바(5) 이므로 다시 while으로 들어갑니다.
  다 = 나(3) 더하기 다(3) = 6; / 가=3; -> 조건 검사
- 가(3) <= 바(5) 이므로 다시 계산 -> 다=9, 가=4 -> 조건 검사
- 가(4) <= 바(5) 이므로 다시 계산 -> 다=12, 가=5 -> 조건 검사
- 가(5) <= 바(5) 이므로 다시 계산 -> 다=15, 가=6 -> 조건 검사
- while문 탈출 -> 마= 마 더하기 마 =2; -> 올바른 결과가 도출됨

```
19  LABEL loc
20  RVALUE 가
21  RVALUE 바
22  -
23   GOPLUS memloc
24  LVALUE 다
25  RVALUE 나
26  RVALUE 다
27  +
28  :=
29  LVALUE 가
30  RVALUE 가
31  RVALUE 마
32  +
33  :=
34  RVALUE 가
35  RVALUE 나
36  GOTO loc
37  LABEL memloc
38  LVALUE 마
39  RVALUE 마
40  RVALUE 마
41  +
42  :=
43  HALT
```

- 조건식을 계산하기 전에 LABEL loc을 두어 조건식을 다시 검사할 수 있도록 하였습니다.
- <= 이므로 a-b가 양수이면 memloc으로 점프하여 while문 안의 식들을 수행하지 않습니다. a-b가 0이면 조건에 걸리지 않으므로 점프하지 않습니다.
- while문 {}의 마지막에는 무조건 점프문을 만나면 GOTO loc을 푸시하여 라벨 loc으로 점프하여 조건 검사를 다시 할 수 있도록 하였습니다.

*** while문 예시(while1 파일, < 조건만족)**

```
시작
가:=1;
나:=3;
다:=0;
라:=2;
마:=1;
바:=5;
반복 (가<바)
{
        다:=나 더하기 다;
        가:=가 더하기 마;
        가 @ 나;
}
마:=마 더하기 마;
끝
```

```
[DATA Segment Dump]
Loc#  Symbol        Value
   0  가                5
   1  나                3
   2  다               12
   3  라                2
   4  마                2
   5  바                5
[End of DATA Segment]
```

- 앞의 예시와 다르게 가(5)<바(5) 일 때 조건식을 만족하지 않으므로 while문 안의 문장들을 수행하지 않는 것을 알 수 있습니다.
- while문이 끝난 뒤의 문장도 '마'에 할당된 값을 통해 정상 수행 됨을 알 수 있습니다.

```
19   LABEL loc
20   RVALUE 가
21   RVALUE 바
22   -
23    COPY
24    GOPLUS memloc
25    GOFALSE memloc
26   LVALUE 다
27   RVALUE 나
28   RVALUE 다
29   +
30   :=
31   LVALUE 가
32   RVALUE 가
33   RVALUE 마
34   +
35   :=
36   RVALUE 가
37   RVALUE 나
38   GOTO loc
39   LABEL memloc
40   LVALUE 마
41   RVALUE 마
42   RVALUE 마
43   +
44   :=
45   HALT
```

- 위의 예시와 동일하게 조건식을 계산하기 전에 LABEL loc을 두어 조건식을 다시 검사할 수 있도록 하였습니다.
- < 이므로 두 번의 검사가 수행되어야합니다. 따라서 COPY를 통해 스택의 top값을 복사해줍니다. a-b가 양수이면 memloc으로 점프하여 while문 안의 식들을 수행하지 않습니다. a-b가 0이어도 조건에 걸리지 않으므로 while문 안의 식들을 수행하지 않고 memloc으로 점프하도록 하였습니다.
- while문 {}의 마지막에는 무조건 점프문을 만나면 GOTO loc을 푸시하여 라벨 loc으로 점프하여 조건 검사를 다시 할 수 있도록 하였습니다.

*** while문 예시(while2 파일, 조건 불만족)**

```
시작
가:=1;
나:=3;
다:=0;
라:=2;
마:=1;
바:=5;
반복 (가>바)
{
        다:=나 더하기 다;
        가:=가 더하기 마;
        가 @ 나;
}
마:=마 더하기 마;
끝
```

```
[DATA Segment Dump]
Loc#  Symbol        Value
   0  가                1
   1  나                3
   2  다                0
   3  라                2
   4  마                2
   5  바                5
[End of DATA Segment]
```

- 조건을 만족하지 않으므로 {}안 문장이 수행되지 않습니다.