컴파일러 기말 프로젝트

부제: Goblin language

충북대학교 소프트웨어학과
2019038105 송수영

제작날짜: 22.05.29

목차

0. 서론

도깨비 말은 한국어의 언어유희의 일종으로, 구어체에서 주로 사용한다. 언어는 알려져있어도 말할 때의 불편함 때문에 사용하지 않는다고 한다. 하지만, 이 언어를 문법으로 사용하게 된 이유는 한국어 컴파일러에서 재미를 더해주기 위해서 사용했다. 하지만, 실제로 문법 작성에 어려움이 있기 때문에 상용화될 수 없을 것 같다.

1. Goblin language 소개

C 문법	Goblin 문법	C 문법	Goblin 문법
+	더버하바기비	>	크브다바
-	빼배기비	<	자박다바
*	고봅하바기비	>=	크브거버나바가밭다바
/	나바누붓세벰	<=	자박거버나바가밭다바
%	나바머버지비	==	가밭다바
sqrt	제베고봅그븐	!=	아반가밭다바
(	(	:=	:=
)	)	;	;
>>	>>	<<	<<

while	초기값;
	바반보복 조건문 {
	stmt_list;
	}
		if	마반야뱍에베 조건문{
			stmt_list;
			}

if
else
	마반야뱍에베또보느븐 조건문{
	stmt_list;
	}아반니비라바며변{
	stmt_list;
	}
		시작	시비자박

		끝	끄븥
		}	}
		{	{

** if/else문의 경우 2번 사용 불가합니다.

** 비트연산자는 숫자만 가능합니다.

2. 테스트

2-1. c language

```c
#include <stdio.h>

int main(void) {

  int i=0;
  int sum=0;

  while(i<10){
    sum = sum+(i%2)+3;
    i++;
  }

  printf("%d %d\n",i,sum);
  return 0;
}
```

2-2. goblin languge

```
시비자박
i:=0;
sum:=0;
바반보복 i 자박다바 10 {
        sum:=sum 더버하바기비 ( i 나바머버지비 2) 더버하바기비 3;
        i:= i 더버하바기비 1;
}
끄붑
```

```
C:\Users\ASUS\Desktop\DevKit1>StackSim a.asm
extended abstract Stack Machine Simulator (StackSim) v1.1
(C)opyright by Jae Sung Lee (jasonlee@cbnu.ac.kr), 2022.

Successfully assembled.



Successfully executed.

[DATA Segment Dump]
Loc#  Symbol        Value
   0  i               10
   1  sum             35
[End of DATA Segment]
```

2-3. C language

```c
#include <stdio.h>

int main(void) {

  int sum=5+3;
  int mul=2*2;
  int sub=7-1;
  int div=9/2;
  int mod=8%4;

  int lbit= sub<<2;
  int rbit= div>>1;

  printf("%d %d %d %d %d %d %d ",sum,mul,sub,div,mod,lbit,rbit);
  if(mul==div)
    printf("%d",mul*mul);
  return 0;
}
```

2-4. Goblin language

```
시비자박

덧셈:= 5 더버하바기비 3;
곱셈:= 2 고봅하바기비 2;
뺄셈:= 7 빼배기비 1;
나눗셈:= 9 나바누붓세벰 2;
나머지:= 8 나바머버지비 4;

왼쪽비트 := 뺄셈 << 2;
오른쪽비트 := 나눗셈 >> 1;

마반야뱍에베 곱셈 가밭다바 나눗셈{
        제곱근 := 제베고봅그븐 곱셈;
}

끄븥
```

```
C:\Users\ASUS\Desktop\DevKit1>StackSim a.asm
extended abstract Stack Machine Simulator (StackSim) v1.1
(C)opyright by Jae Sung Lee (jasonlee@cbnu.ac.kr), 2022.

Successfully assembled.



Successfully executed.

[DATA Segment Dump]
Loc#  Symbol        Value
   0  덧셈              8
   1  곱셈              4
   2  뺄셈              6
   3  나눗셈            4
   4  나머지            0
   5  왼쪽비트         24
   6  오른쪽비트        2
   7  제곱근           16
[End of DATA Segment]
```

2-5. C language

```c
#include <stdio.h>

int main(void) {
  int odd=0;
  int even=0;
  int i=1;

  while(i<10){

    if(!(i%3))
      odd++;
    if(!(i%2))
      even++;
    i++;
  }
  printf("%d %d %d",i,odd,even);
  return 0;
}
```

2-6. Goblin language

```
시비자박
i:=1;
odd:=0;
even:=0;
바반보복 i 자박다바 10{

        마반야뱍에베 ( i 나바머버지비 3 ) 가밭다바 0{
                odd:=odd 더버하바기비 1;
        }
        마반야뱍에베 ( i 나바머버지비 2 ) 가밭다바 0{
                even:=even 더버하바기비 1;
        }

        i:=i 더버하바기비 1;

}
끄븥
```

```
PS C:\Users\ASUS\Desktop\DevKit1> ./StackSim a.asm
extended abstract Stack Machine Simulator (StackSim) v1.1
(C)opyright by Jae Sung Lee (jasonlee@cbnu.ac.kr), 2022.

Successfully assembled.



Successfully executed.

[DATA Segment Dump]
Loc#  Symbol        Value
   0  i                10
   1  odd               3
   2  even              4
[End of DATA Segment]
```

Lex

```
더버하바기비     {return(ADD); }
빼배기비         {return(SUB); }
고봅하바기비     {return(MUL); }
나바누붓세벰     {return(DIV); }
나바머버지비     {return(MOD); }
제베고봅그븐     {return(SQRT);}

크브다바         {return(GT); }
크브거버나바가밭다바 {return (GE);}
자박다바         {return(LT); }
자박거버나바가밭다바 {return (LE);}
가밭다바         {return (EQ);}
아반가밭다바     {return (NQ);}

마반야뱍에베     {return (IF);}
마반야뱍에베또보느븐 {return (IF_ELSE_ST);}
아반니비라바며변 {return (ELSE);}
바반보복         {return (WHILE);}

"{"              {return (LBRACE);}
"}"              {return (RBRACE);}
"("              {return (LPAR);}
")"              {return (RPAR);}
"<<"             {return (LEFT); } //고칠예정
">>"             {return (RIGHT); } //RIGHT
```

주 기능

0. Block 처리

- 중괄호

```
iteration : WHILE condition_stmt LBRACE stmt_list stmt RBRACE { $$=MakeSTMTree(WHILE,$2,$4,$5); }
;
unmatched : IF condition_stmt LBRACE stmt_list RBRACE {$$=MakeOPTree(IF,$2,$4);}
       | IF_ELSE_ST condition_stmt LBRACE stmt_list RBRACE ELSE LBRACE stmt_list RBRACE { $$=MakeSTMTree(IF
_ELSE_ST,$2,$4,$8); }
```

```
stmt_list: stmt_list stmt {$$=MakeListTree($1,$2);}
         |  stmt          {$$=MakeListTree(NULL,$1); }
         | error STMTEND { errorcnt++; yyerrok;}
         ;
```

- 소괄호

```
fact     :   LPAR expr RPAR      { $$=$2;}
         |       ID              { /* ID node is created in lex */ }
         |       NUM             { /* NUM node is created in lex */ }
         ;
```

1. Mod 연산자

```
term     :       term MUL fact    { $$=MakeOPTree(MUL, $1, $3); }
         |       term DIV fact    { $$=MakeOPTree(DIV, $1, $3); }
         |       term MOD fact    { $$=MakeOPTree(MOD, $1, $3); }
         |       SQRT fact        { $$=MakeSQRTree(MUL, $2); }
         |       fact
         ;
```

```
case MOD:
        fprintf(fp,"POP\n"); //B 빼기
        fprintf(fp,"POP\n"); //B 빼기
        if(ptr->son->token==NUM&&ptr->son->brother->token==NUM){ //가능
                fprintf(fp,"PUSH %d\n",ptr->son->tokenval);
                fprintf(fp,"PUSH %d\n",ptr->son->tokenval);
                fprintf(fp,"PUSH %d\n",ptr->son->brother->tokenval);
                fprintf(fp,"/\n"); //나누기연산
                fprintf(fp,"PUSH %d\n",ptr->son->brother->tokenval);
        }else if(ptr->son->token==NUM&&ptr->son->brother->token!=NUM){ //가능
                fprintf(fp,"PUSH %d\n",ptr->son->tokenval);
                fprintf(fp,"PUSH %d\n",ptr->son->tokenval);
                fprintf(fp,"RVALUE %s\n",symtbl[ptr->son->brother->tokenval]);
                fprintf(fp,"/\n");
                fprintf(fp,"RVALUE %s\n",symtbl[ptr->son->brother->tokenval]);
        }else if(ptr->son->token!=NUM&&ptr->son->brother->token==NUM){
                fprintf(fp,"RVALUE %s\n",symtbl[ptr->son->tokenval]);
                fprintf(fp,"RVALUE %s\n",symtbl[ptr->son->tokenval]);
                fprintf(fp,"PUSH %d\n",ptr->son->brother->tokenval);
                fprintf(fp,"/\n");
                fprintf(fp,"PUSH %d\n",ptr->son->brother->tokenval);
        }else{
                fprintf(fp,"RVALUE %s\n",symtbl[ptr->son->tokenval]);
                fprintf(fp,"RVALUE %s\n",symtbl[ptr->son->tokenval]);
                fprintf(fp,"RVALUE %s\n",symtbl[ptr->son->brother->tokenval]);
                fprintf(fp,"/\n");
                fprintf(fp,"RVALUE %s\n",symtbl[ptr->son->brother->tokenval]);
        }
                fprintf(fp,"*\n");
                fprintf(fp,"-\n");
        break;
```

* 나머지 연산자 이용 / 이전 내용을 POP하여, 직접 값을 계산하는 방식으로 사용.

2. 비교 연산자

```
switch(token){ //오른쪽이 왼쪽보다 크다. 0일때 Out
        case GT: //크다 음수일경우 0을 삽입
                fprintf(fp,"-\n");
                if(WhileCnt>-1&&WhileCnt!=OutCnt){
                        fprintf(fp,"COPY\n");
                        fprintf(fp,"GOMINUS LOOPOUT%d\n",++OutCnt);
                        fprintf(fp,"GOFALSE LOOPOUT%d\n",OutCnt);
                }
                else if(IfElseCnt>-1&&ElseCnt!=IfElseCnt){
                        fprintf(fp,"COPY\n");
                        fprintf(fp,"GOMINUS ELSE%d\n",++ElseCnt);
                        fprintf(fp,"GOFALSE ELSE%d\n",ElseCnt);//0
                }
                else if(IfCnt>-1&&IfCnt!=setCnt){
                        fprintf(fp,"COPY\n");
                        fprintf(fp,"GOMINUS OUT%d\n",++setCnt);
                        fprintf(fp,"GOFALSE OUT%d\n",setCnt);
                }

                break;
```

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

3. 비트 연산자

```
bit     :       bit LEFT expr   { $$=MakeOPTree(LEFT,$1,$3);}
        |       bit RIGHT expr  { $$=MakeOPTree(RIGHT,$1,$3);}
        |       expr
        ;
```

```
case RIGHT://비트연산자 숫자되도록 하기.
        fprintf(fp,"POP\n");
        int rsqr=1;
        for(int i=0;i<ptr->son->brother->tokenval;i++)
                rsqr*=2;

        fprintf(fp,"PUSH %d\n",rsqr);
        fprintf(fp,"/\n");
        break;
case LEFT:
        fprintf(fp,"POP\n");
        int lsqr=1;     //2의 제곱근할 값
        for(int i=0;i<ptr->son->brother->tokenval;i++)
                lsqr*=2;
        fprintf(fp,"PUSH %d\n",lsqr);
        fprintf(fp,"*\n");
        break;
```

숫자가 입력되었을 때, 숫자만큼 곱해서 값을 삽입 후 *

4. 조건문

```
#define YYSTYPE Node*
int IfCnt=-1;
int setCnt=-1;
int desCnt=0;
//if문을 관리할 cnt
int Cnt=-1;
int IfElseCnt=-1;
int ElseCnt=-1;
//if/else를 관리할 cnt
```

5. 반복문

```
int WhileCnt=-1;
int LoopCnt=-1;
int OutCnt=-1;
//반복문 관리할 cnt
```

```
void processStatement(Node *ptr){

        switch(ptr->token){
                case IF: //LABEL OUT%d - setCnt
                        fprintf(fp,"LABEL OUT%d\n",setCnt);
                        break;
                case IF_ELSE_ST: //LABEL OUT%d -Cnt
                        fprintf(fp,"LABEL IFELSEOUT%d\n",ElseCnt);
                        break;
                case WHILE:
                        fprintf(fp,"GOTO LOOP%d\n",WhileCnt);
                        fprintf(fp,"LABEL LOOPOUT%d\n",OutCnt);
                        break;
        }
}
```

```
case GT: case LT: case LE: case GE: case EQ: case NQ:
        processCondition(token,ptr);
        break;
case IF:case IF_ELSE_ST:case WHILE:
        processStatement(ptr);
        break;
case ASSGN:
        fprintf(fp,":=\n");
        desCnt++;
        if(WhileCnt>-1&&LoopCnt!=WhileCnt&&desCnt==(checkStmt-1)){

                fprintf(fp,"LABEL LOOP%d\n",++LoopCnt);
        }

        if(IfElseCnt>-1&&ElseCnt>-1&&Cnt!=ElseCnt){
                fprintf(fp,"GOTO IFELSEOUT%d\n",ElseCnt);
                fprintf(fp,"LABEL ELSE%d\n",ElseCnt);
                Cnt++;
        }
        break;
```

마지막으로, 교수님께서 주신 컴파일러는 트리구조로 이루어져 있어서, 반복문, 조건문을 구현하는데 어려움이 있었습니다. 저 같은 경우에는 할당문을 체크하여 LABEL을 붙이는 방식을 이용했습니다. 제 코드를 보게 된다면, 반복문과 조건문을 만들기 위해 Node를 새로 생성하는 방식을 선택하였습니다. 또한, 사칙연산에 제곱근과 모드연산을 추가하였고, 숫자를 이용해서 비트연산자를 사용할 수 있도록 코드를 구현했습니다.