컴파일러의 기초 - Project4

2014-10012 김태완

0. 구현사항

- 기본구현과 심화구현을 모두 수행했다.

1. 심화 구현

[1] while, for, nested while, break, continue

- While들이 서로 같은 label을 가지지 않도록 while_cnt를 이용했고, nested while을 위해 while_stack을 만들어 이후 break, continue에서 어떤 label로 이동해야할지를 알려주었다.

```
298         | WHILE {
299             $<intVal>$ = while_cnt;
300             set_while_stack(while_cnt);
301             while_cnt++;
302             printf("WHILE_START_%d:\n", $<intVal>$);
303         }
304         '(' expr ')' {
305             printf("\tbranch_false WHILE_END_%d\n", $<intVal>2);
306         }
307         stmt {
308             printf("\tjump WHILE_START_%d\n", $<intVal>2);
309             printf("WHILE_END_%d:\n", $<intVal>2);
310             free_while_stack();
311         }
```

- For loop은 break와 continue를 while에서의 동작과 같게 하기 위해 중간에 While과 같은 형식의 Label을 붙였다.

```
312         | FOR '(' expr_e ';' {
313             $<intVal>$ = while_cnt;
314             set_while_stack(while_cnt);
315             while_cnt++;
316             printf("FOR_CHECK_%d:\n", $<intVal>$);
317 
318         } expr_e ';' {
319             printf("\tbranch_false WHILE_END_%d\n", $<intVal>5);
320             printf("\tjump FOR_STMT_%d\n", $<intVal>5);
321             printf("WHILE_START_%d:\n", $<intVal>5);
322 
323         } expr_e ')' {
324             printf("\tjump FOR_MID_%d\n", $<intVal>5);
325             printf("FOR_STMT_%d:\n", $<intVal>5);
326 
327         }stmt {
328             printf("\tjump WHILE_START_%d\n", $<intVal>5);
329             printf("FOR_MID_%d:\n", $<intVal>5);
330             printf("\tjump FOR_CHECK_%d\n", $<intVal>5);
331             printf("WHILE_END_%d:\n", $<intVal>5);
332             free_while_stack();
333         }
```

- break, continue는 어떤 while loop을 탈출 또는 지속할지 알려주기 위해 while stack을 이용했다.

```
334         | BREAK ';' {
335             printf("\tjump WHILE_END_%d\n", get_while_stack());
336         }
337         | CONTINUE ';' {
338             printf("\tjump WHILE_START_%d\n", get_while_stack());
339         }
```

[2] struct assign

- Struct의 assign을 위해 아래와 같이 구현했다. Assign받는 Struct의 memory 주소에 Assign할 Struct의 memory Value를 for loop을 이용해 assign하는 방식이다.

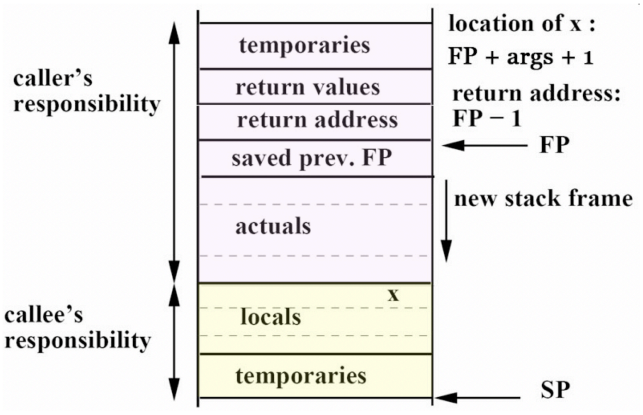
```
362 expr
363         : unary { /* ASSIGN */
364             if($1->type->typeclass != 4) { // not struct
365                 printf("\tpush_reg sp\n");
366                 printf("\tfetch\n");
367             }
368 
369         } '=' expr {
370             if($1->type->typeclass != 4) {
371                 if($4->declclass==0) printf("\tfetch\n"); //V
372                 printf("\tassign\n");
373                 printf("\tfetch\n");
374                 printf("\tshift_sp -1\n");
375             }
376             else {
377                 for(int i=0; i < $1->size; i++) {
378                     printf("\tpush_reg sp\n"); //for VAR
379                     printf("\tpush_const -1\n");
380                     printf("\tadd\n");
381                     printf("\tfetch\n");
382                     printf("\tpush_const %d\n", i);
383                     printf("\tadd\n");
384 
385                     printf("\tpush_reg sp\n"); //for VAL
386                     printf("\tpush_const -1\n");
387                     printf("\tadd\n");
388                     printf("\tfetch\n");
389                     printf("\tpush_const %d\n", i);
390                     printf("\tadd\n");
391 
392                     printf("\tfetch\n");
393                     printf("\tassign\n");
394                 }
395                 printf("\tshift_sp -2\n");
396             }
397         }
398         | or_expr {$$ = $1;}
```

[3] struct parameter

- 함수의 호출은 ppt 의 calling convention 2를 이용했다.
- struct를 parameter로 passing하기 위해 struct를 argument로 쓸 때, struct의 모든 memory에 있는 것을 calling convention2의 actuals 부분에 넣었다.

```
615 args
616         : expr {
617             if($1->declclass==0) { //VAR
618                 if($1->type->typeclass==4) { //Struct
619                     for(int i=1; i<$1->size; i++) {
620                         printf("\tpush_reg sp\n");
621                         printf("\tpush_const -%d\n", i-1);
622                         printf("\tadd\n");
623                         printf("\tfetch\n");
624 
625                         printf("\tpush_const %d\n", i);
626                         printf("\tadd\n");
627                         printf("\tfetch\n");
628                     }
629                     printf("\tshift_sp -%d\n", $1->size-1);
630                     printf("\tfetch\n");
631                     printf("\tshift_sp %d\n", $1->size-1);
632                 }
633                 else printf("\tfetch\n");  //VAR only
634             }
635         }
636         | args ',' expr {
637             if($3->declclass==0) { //VAR
638                 if($3->type->typeclass==4) { //Struct
639                     for(int i=1; i<$3->size; i++) {
640                         printf("\tpush_reg sp\n");
641                         printf("\tpush_const -%d\n", i-1);
642                         printf("\tadd\n");
643                         printf("\tfetch\n");
644 
645                         printf("\tpush_const %d\n", i);
646                         printf("\tadd\n");
647                         printf("\tfetch\n");
648                     }
649                     printf("\tshift_sp -%d\n", $3->size-1);
650                     printf("\tfetch\n");
651                     printf("\tshift_sp %d\n", $3->size-1);
652 
653                 }
654                 else printf("\tfetch\n");  //VAR only
655             }
656         }
```

Example: Calling Convention II



[4] struct return

- struct를 return하기 위해서는 단순히 struct의 주소값을 넘겨주어서 caller function에서 return한 값을 받아오기로 했다.
- Caller function에서 struct의 값을 불러오거나 assign을 위해 stack을 사용하면서 struct의 memory를 덮어쓸 위험이 있는데, 현재 구현에서는 struct를 처리하기 위해 stack이 2word 공간만큼 필요하고, calling convention 2에 따르면, caller로 return한 후 stack pointer와 actuals, locals 공간 사이의 최소 거리가 2word이므로, caller로 return 후 struct memory를 침해할 위험이 없다.

```
276         | RETURN {
277             func_return();
278         } expr ';' {
279             if($3->type->typeclass != 4) {
280                 if($3->declclass==0) printf("\tfetch\n");
281                 printf("\tassign\n");
282             }
283             else { //Struct
284                 printf("\tassign\n");
285             }
286         }
287         | ';'
```

2. 전체적인 디자인

- Start Up Code

```
69 program
70         : {
71             printf("\tshift_sp 1\n");
72             printf("\tpush_const EXIT\n");
73             printf("\tpush_reg fp\n");
74             printf("\tpush_reg sp\n");
75             printf("\tpop_reg fp\n");
76             printf("\tjump main\n");
77             printf("EXIT:\n");
78             printf("\texit\n");
79         } ext_def_list {
80             printf("Lglob.  data %d\n", Global_offset());
81         }
```

- 함수의 마지막 부분 : return하는 상황

```
107         | func_decl compound_stmt {
108             pop_scope();
109             printf("%s_final:\n", $1->name);
110             printf("\tpush_reg fp\n");
111             printf("\tpop_reg sp\n");
112             printf("\tpop_reg fp\n");
113             printf("\tpop_reg pc\n");
114             printf("%s_end:\n", $1->name);
115         }; /* DEFINE func */
116
```

- local_defs가 1을 return했다면 이는 함수에서 선언된 것이다. 따라서 Local Variable을 위한 공간을 할당하기 위해 (함수가 가진 모든 variable size) - (함수가 가진 parameter size)만큼 shift_sp를 해준다.

```
252 compound_stmt
253         : '{' local_defs {
254             if($2==1) {
255                 if(Local_offset()-Local_args() > 0)
256                     printf("\tshift_sp %d\n", Local_offset()-Local_args());
257                 func_start();
258             }
259         } stmt_list '}';
```

-

-

-

-

- if else에서 mid rule action으로 인해 reduce/reduce conflict가 발생했다. 따라서 if_mid이라는 새로운 non-terminal을 만들어 conflict를 해소 했다.

```
268 if_mid  : IF '(' expr ')' {
269             $$ = if_cnt++;
270             printf("\tbranch_false ELSE_%d\n", $$);
271         };
272 stmt
273         : expr ';'
274         | compound_stmt
275         | RETURN ';'
276         | RETURN {
277             func_return();
278         } expr ';' {
279             if($3->type->typeclass != 4) {
280                 if($3->declclass==0) printf("\tfetch\n");
281                 printf("\tassign\n");
282             }
283             else { //Struct
284                 printf("\tassign\n");
285             }
286         }
287         | ';'
288
289         | if_mid stmt %prec THEN {
290             printf("ELSE_%d:\n", $1 );
291         }
292         | if_mid stmt ELSE {
293             printf("\tjump AFTER_%d\n", $1 );
294             printf("ELSE_%d:\n", $1 );
295         } stmt {
296             printf("AFTER_%d:\n", $1 );
297         }
```

- write_string과 write_int의 구현

```
340         | WRITE_STRING '(' STRING {
341             printf("str_%d. string ", string_cnt);
342             printf("%s\n", $3);
343             printf("\tpush_const str_%d\n", string_cnt++);
344             printf("\twrite_string\n");
345
346         } ');' {}
347         | WRITE_INT '(' expr ');'  {
348             if($3->declclass==0) printf("\tfetch\n"); //VAR
349             printf("\twrite_int\n");
350         }
```

- binary는 항상 variable이 아닌 값을 return하도록 했다.

```
binary
        : binary RELOP {
            if($1->declclass==0) printf("\tfetch\n");
            if(!strcmp(yylval.stringVal, ">"))
                $<intVal>$ = 0;
            if(!strcmp(yylval.stringVal, ">="))
                $<intVal>$ = 1;
            if(!strcmp(yylval.stringVal, "<"))
                $<intVal>$ = 2;
            if(!strcmp(yylval.stringVal, "<="))
                $<intVal>$ = 3;
        } binary {
            if($4->declclass==0) printf("\tfetch\n");
            if($<intVal>3==0) printf("\tgreater\n");
            if($<intVal>3==1) printf("\tgreater_equal\n");
            if($<intVal>3==2) printf("\tless\n");
            if($<intVal>3==3) printf("\tless_equal\n");

            $$ = makenumconstdecl($1,0);
        }
        | binary EQUOP {
            if($1->declclass==0) printf("\tfetch\n");
            if(!strcmp(yylval.stringVal, "=="))
                $<intVal>$ = 0;
            if(!strcmp(yylval.stringVal, "!="))
                $<intVal>$ = 1;
        } binary {
            if($4->declclass==0) printf("\tfetch\n");
            if($<intVal>3==0) printf("\tequal\n");
            if($<intVal>3==1) printf("\tnot_equal\n");
            $$ = makenumconstdecl($1,0);
        }
        | binary '+' {
            if($1->declclass==0) printf("\tfetch\n");
        } binary {
            if($4->declclass==0) printf("\tfetch\n");
            printf("\tadd\n");
            $$ = makenumconstdecl($1,0);
        }
        | binary '-' {
            if($1->declclass==0) printf("\tfetch\n");
        } binary {
            if($4->declclass==0) printf("\tfetch\n");
            printf("\tsub\n");
            $$ = makenumconstdecl($1,0);
        }
        | unary %prec '=' { $$ = $1;}
```

- CHAR type const는 모두 ASCII code로 치환하여 INT로 취급했다.

```
unary
        : '(' expr ')'  {$$ = $2;}
        | '(' unary ')' {$$ = $2;}
        | INTEGER_CONST {
            $$ = makenumconstdecl($$=find("int", 3), yylval.intVal);
            printf("\tpush_const %d\n", yylval.intVal);
        }
        | CHAR_CONST {
            $$ = makenumconstdecl($$=find("char", 4), (yylval.intVal));
            printf("\tpush_const %d\n", (yylval.intVal));
        }
```

- ID를 보면 바로 그 주소를 반환하도록 했다.
- '-', '!', INCOP, DECOP등에 대해 구현했다.

```
488         | ID {
489             $$ = find($1->name,strlen($1->name));
490             if($$->declclass==0 || $$->declclass==1) { //VAR or CONST
491                 if($$->scope->level==0)
492                     printf("\tpush_const Lglob+%d\n", offset($$,1));
493                 else {
494                     printf("\tpush_reg fp\n");
495                     printf("\tpush_const %d\n", offset($$,0));
496                     printf("\tadd\n");
497 
498                 }
499             }
500         }
501         | '-' unary %prec '!'  {
502             if($2->declclass==0) printf("\tfetch\n"); //VAR
503             printf("\tpush_const -1\n");
504             printf("\tmul\n");
505             $$=makenumconstdecl($2->type, 0);
506         }
507         | '!' unary   {
508             if($2->declclass==0) printf("\tfetch\n"); //VAR
509             printf("\tpush_const 0\n");
510             printf("\tnot_equal\n");
511             $$=makenumconstdecl($2->type, 0);
512         }
513         | unary INCOP {
514             $$ = makenumconstdecl($1->type, 0);
515             printf("\tpush_reg sp\n");
516             printf("\tfetch\n");
517             printf("\tpush_reg sp\n");
518             printf("\tfetch\n");
519             printf("\tfetch\n");
520             printf("\tpush_const 1\n");
521             printf("\tadd\n");
522             printf("\tassign\n");
523             printf("\tfetch\n");
524             printf("\tpush_const 1\n");
525             printf("\tsub\n");
526         }
```

- '&'에 대해서는 variable을 numconst로 취급하여 주소 성분을 갖게 했다.
- '*'에 대해서는 주소를 fetch하여 value로 바꾸어주었다.
- array접근은 배열의 size단위로 주소를 이동해 값을 불러오도록 했다.
- struct의 접근은 hash.c의 함수를 별도로 만들어 struct의 ID가 가진 offset을 계산해 ID의 주소를 불러왔다.

```
565           | '&' unary %prec '!'  {
566               $$ = makenumconstdecl($2->type, 0);
567               //if VAR, then treat VAR as CONST makes addr
568           }
569           | '*' unary %prec '!' {
570               $$=makevardecl($2->type->ptrto);
571               printf("\tfetch\n");
572           }
573           | unary '[' expr ']' {
574               if($3->declclass==0) printf("\tfetch\n");
575               $$ = arrayaccess($1);
576               printf("\tpush_const %d\n", $$->size);
577               printf("\tmul\n");
578               printf("\tadd\n");
579           }  /* return VAR */
580           | unary '.' ID       {
581               $$ = structaccess($1,$3);
582           } /* return Field */
583           | unary STRUCTOP ID  {
584               $$ = structptraccess($1,$3);
585           }
```

```
189 decl *structaccess (decl *structptr, id *fieldid) {
190 
191     ste *pivot = structptr->type->fieldlist;
192     int offset = 0;
193     while(pivot!=NULL) {
194         if(!strcmp(pivot->name->name, fieldid->name)) {
195             //printf("\tpush_const %d\n", structptr->size - rev_offset - pivot->decl->size);
196             printf("\tpush_const %d\n", offset);
197             printf("\tadd\n");
198             return pivot->decl;
199         }
200         offset+=get_size(pivot->decl);
201         pivot = pivot->prev;
202     }
203     return NULL;
204 }
```