컴파일러의 기초 Project 3

전기정보공학부

2016-13343 유상윤

구현 방법:

Semantic analysis 를 위해 scoped symbol table을 사용했다.

Scoped symbol table을 위한 implementation choice 로는 수업시간에 배운 stack of definitions 를 사용하였다.

다만 string compare를 줄이기 위해 hashed symbol table을 통해 id를 유지하고, 이의 pointer를 사용해서 token id를 관리하였다.

기본적으로 push, pop scope을 통해 scope을 push pop하고, insert를 통해 새로운 definition을 넣고, lookup을 통해 token에 상응하는 definition을 찾는다.

scopedsymtab.c 에 scoped symbol table과 그와 관련된 함수들을 모두 정의되어있다.

subc.h

```
typedef struct id
{
    int tokenType;
    char *name;
} id;

typedef struct decl
{
    int declclass;
    int typeclass;
    struct decl *type;
    int intval;
    struct decl *elementvar;
    int num_index;
    struct ste *fields;
    struct decl *ptrto;
    struct decl *next;
    struct ste *formals;
    struct decl *returntype;
} decl;

typedef struct ste
{
    struct id *id;
    struct decl *decl;
    struct ste *prev;
} ste;

decl *inttype;
decl *chartype;
decl *voidtype;
id *returnid;
```

```
/* For scoped symbol table */
void push_scope();
ste *pop_scope();
void pushstelist(ste *stelist);
decl *declare(id *id, decl *decl);
decl *findcurrentdecl(id *id);
decl *findstructdecl(id *id);

decl *makeargdecl(decl *decl1);
decl *makevardecl(decl *typedecl, bool isptr);
decl *maketypedecl(int typeclass);
decl *makeconstdecl(decl *typedecl);
decl *makearraydecl(decl *num_index, decl *elementvar);
decl *makeptrdecl(decl *typedecl);
decl *makestructdecl(ste *fields);
decl *makeprocdecl();

decl *assign_op(decl *lhs, decl *rhs);
decl *plus_op(decl *decl1, decl *decl2);
decl *minus_op(decl *decl1, decl *decl2);
decl *unary_minus_op(decl *decl);
decl *unary_not_op(decl *decl);
decl *inc_op(decl *decl);
decl *dec_op(decl *decl);
decl *rel_op(decl *decl1, decl *decl2);
decl *equ_op(decl *decl1, decl *decl2);
decl *address_op(decl *vardecl);
decl *pointer_op(decl *vardecl);

decl *structaccess(decl *decl, id *fieldid);
decl *structptraccess(decl *decl, id *fieldid);
decl *arrayaccess(decl *arraydecl, decl *indexdecl);

decl *functioncall(decl *procdecl, decl *actuals);
decl* functionreturn(decl *returndecl);
```

구현 내용:

Scope stack size는 200으로 max 200까지의 scope 깊이를 가질 수 있게 설정하였다.

scopedsymtab.c 에 static function들과 일반 function들이 있다. Static function들은 파일 내부적으로만 사용되고 subc.y에 사용되는 function들 내부에서만 사용된다.

Error code skip:

NULL을 에러가 날 경우 넘기고 NULL을 받아서 semantic analysis를 할 경우 추가적인 error를 체크하지 않고 NULL을 계속 넘긴다.

Multiple Errors:

Error code skip과 같은 logic으로 NULL이 넘어올 경우 추가적인 에러를 감지하지 않아도 동시의 일어나는 error에서는 처음에 발생한 error 만 체크한다.

Undeclared:

findcurrentdecl(id *)를 통해 token을 찾을 때 선언되어 있지 않으면 not declared error를 확인한다.

Redeclaration:

declare(id *, decl *)를 통해 token이 redeclare 될 때 redeclaration error를 확인한다. Redeclaration의 경우 current scope에서 확인한다. Struct은 항상 global 선언되므로 같은 token이 등장하면 redeclared된 것이다. 이를 위해 globaldeclare(id *, decl *)를 별도로 정의하여 사용한다. 이때는 redeclaration을 global scope에서 확인한다.

Type checking:

assign_op, plus_op, 등 xxx_op로 operator의 operand 들을 확인해 type checking 해 알맞은 error를 낸다.

Struct:

Structaccess, structptraccess 를 통해 struct 또는 struct pointer 가 아닐 경우 에러를 뱉는다.

Function:

Makeprocdecl()을 사용해 empty function declaration을 생성 후 scope를 push해 parameter들을 받아 formals에 저장하고, 내부적으로 사용되는 declaration들은 다시 별도의 scope에서만 확인한다.