Code Generation: Declarations

Adrián Novosád
Tomáš Rohovský

Declarations

- Variable declaration has two parts:
  - specifier – a list of various keywords (int, long, extern, struct and so forth)
  - declarator – variable's name, number of stars, array specifiers, parentheses

long int *x, y;

Symbol Table Structures

```
typedef struct symbol
{
        unsigned char  name[SIZE];  /* Input variable name */
        unsigned char  rname[SIZE]; /* Output variable name */
        struct linl        *type;           /* First link in declaration chain */
        struct link        *etype;          /* Last link in declaration chain */
        struct symbol      *args;           /* Funct. arg. list or variable initializer */
        struct symbol      *next; /* cross link to next var. at the same nesting level */
} symbol;

typedef struct link
{
        unsigned class :1;      /* DECLARATOR of SPECIFIER */
        union {
                specifier  s;
                declarator d; }
        select;
        struct link *next;      /* Next element of chain */
}
```

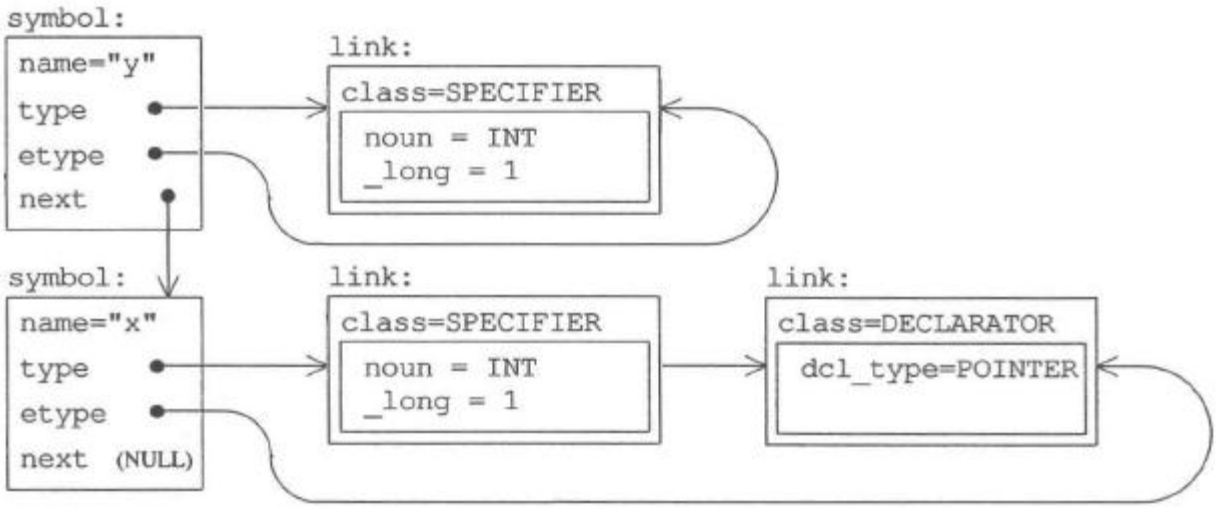
Symbol Table Structures

```
typedef struct specifier
{
        unsigned noun         :3; /* CHAR  INT  STRUCTURE  LABEL */
        unsigned sclasss      :3; /* REGISTER AUTO FIXED CONST TYPEDEF */
        unsigned oclass       :3; /* Ouput storage class: PUB PRI COM EXT */
        unsigned _long        :1; /* 1=long 0=short */
        unsigned _unsigned    :1; /* 1=unsigned 0=signed */
        union { ... } const_val;
} specifier;

typedef struct declarator
{
        int dcl_type; /* POINTER  ARRAY  FUNCTION */
        int num_ele; /* count of elements */
} declarator;
```

Symbol Table Structures: Example

long int *x, y;



Code Generation

- Into machine-independent intermediate language (C-code)
- Directed by the syntax analyzer during the parse of the program – syntax-directed translation
- Rules are associated with actions which handle the code generation
- Yacc and OCCS compiler-development toolkit

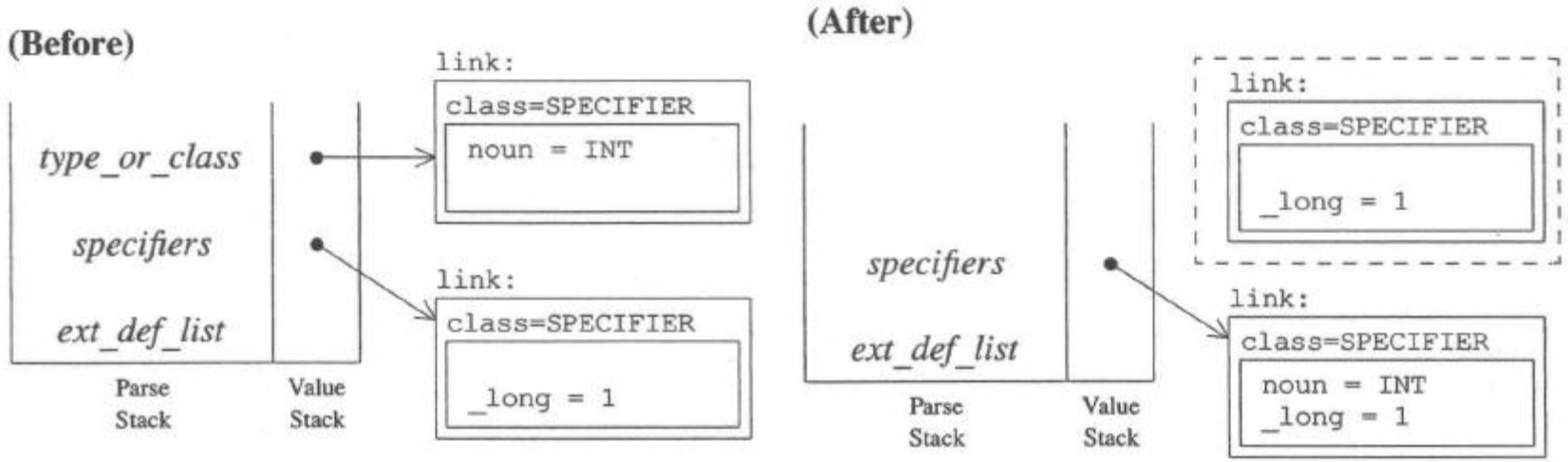
- Declaration processing involves two main tasks:
  - assemble the linked lists that represent the types, attach them to symbols, and put the resulting structures into the symbol table
  - generating C-code definitions for variables

Code Generation: Simple Variables

- For parsing is used Extended PA (bootom-up parsing)
- Example: **long int** *x, y;

Specifier processing

- `new_type_spec(char *lexeme)`
  - create and initialize a link
- `spec_cpy(link *dst, link *src)`
  - merge specifiers

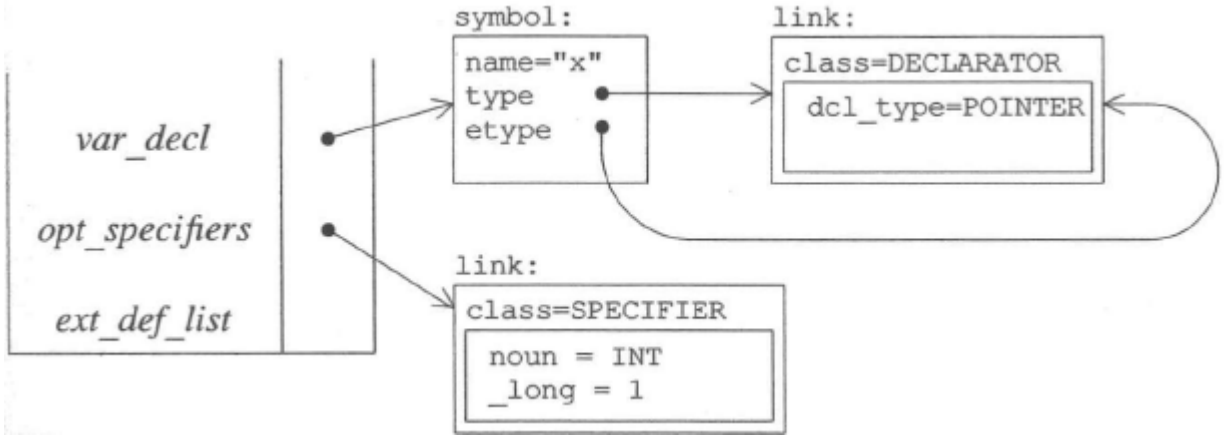


Code Generation: Simple Variables

Example: **long int** *x, y;

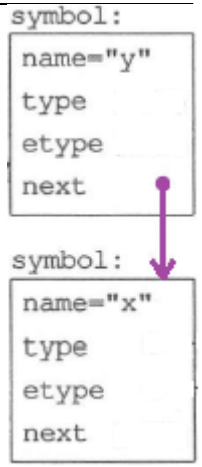
Declarator processing

- `new_symbol(char *name, int scope);`
  - Create a new symbol structure
- `add_declarator(symbol *sym, int type);`
- add a pointer-declarator link to the type chain in the symbol that was created when the name was processed



Code Generation: Simple Variables

Example: **long int** *x, y;



Creating of the cross links

- Cross links join declarations for all variables

at the current scoping level

Merging of the specifier and declarator components

- add_spec_to_decl(link *spec, symbol *chain)

Putting of declarations into the symbol table

- add_symbols_to_table(symbol *sym)

Generating of declarations into the output

- generate_defs_and_free_args(symbol *sym)

Thank you for your attention