Complicated Declarations

- 복잡한 선언문을 설명하는 프로그램(dcl) (Cont'd)
  - 간단한 자료형만 처리 가능
    - ✓ **int**, **char**, **etc**.
    - ✓ 함수의 매개변수 형이나 **const** 선언 등의 한정자는 처리하지 못함
  - 기초적인 기능만을 수행
    - ✓ 오류 회복(error recovery) 기능이 포함되어 있지 않음
    - ✓ 올바른 선언식만 입력해야 함

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "dcl.h"
#include "gettoken.h"

#define MAXTOKEN 100

int tokentype;            /* type of last token */
char token[MAXTOKEN];     /* last token string */
char name[MAXTOKEN];      /* identifier name */
char datatype[MAXTOKEN];  /* data type = char, int, etc. */
char out[1000];           /* output string */

/* convert declaration to words */
int main(void)
{
  while (gettoken() != EOF) {  /* 1st token on line */
    strcpy(datatype, token);   /* is the datatype */
    out[0] = '\0';
    dcl();  /* parse rest of line */
    if (tokentype != '\n')
      printf("syntax error\n");
    printf("%s: %s %s\n", name, out, datatype);
  }

  return 0;
}
```

Complicated Declarations

- 복잡한 선언문을 설명하는 프로그램(dcl) (Cont'd)
  - **gettoken** 함수는 공백 문자들을 무시한 뒤, 입력으로부터 토큰(유의미한 이름)을 찾음
    - ✓ 선언문 분석에 사용되는 이름, 한 쌍의 괄호, 한 쌍의 대괄호, 숫자가 포함된 한 쌍의 대괄호 등을 의미함

```c
#include <ctype.h>
#include <string.h>
#include "gettoken.h"

extern int  tokentype;  /* main.c */
extern char token[];    /* main.c */

int gettoken(void) /* return next token */
{
  int c, getch(void);
  void ungetch(int);
  char *p = token;

  while ((c = getch()) == ' ' || c == '\t') { }
  if (c == '(') {…
  } else if (c == '[') {…
  } else if (isalpha(c)) {…
  } else {…
}
```

Complicated Declarations

- 복잡한 선언문을 설명하는 프로그램(dcl) (Cont'd)
  - **gettoken** 함수는 공백 문자들을 무시한 뒤, 입력으로부터 토큰(유의미한 이름)을 찾음
    - ✓ 선언문 분석에 사용되는 이름, 한 쌍의 괄호, 한 쌍의 대괄호, 숫자가 포함된 한 쌍의 대괄호 등을 의미함

```c
…
  if (c == '(') {
    if ((c = getch()) == ')') {
      strcpy(token, "()");

      return tokentype = PARENS;
    } else {
      ungetch(c);

      return tokentype = '(';
    }
  } else if (c == '[') {
    for (*p++ = c; (*p++ = getch()) != ']'; ) { }
    *p = '\0';

    return tokentype = BRACKETS;
  } else if (isalpha(c)) {
    for (*p++ = c; isalnum(c = getch()); )
      *p++ = c;
    *p = '\0';
    ungetch(c);

    return tokentype = NAME;
  } else {
    return tokentype = c;
  }
```

Complicated Declarations

- 복잡한 선언문을 설명하는 프로그램(dcl) (Cont'd)

```
dcl:        optional *'s direct-dcl
direct-dcl  name
            (dcl)
            direct-dcl()
            direct-dcl[optional size]
```

```c
#include <stdio.h>
#include <string.h>
#include "dcl.h"
#include "gettoken.h"

extern int tokentype;   /* main.c */
extern char token[];    /* main.c */
extern char name[];     /* main.c */
extern char out[];      /* main.c */

/* dcl: parse a declarator */
void dcl(void)
{
  int ns;

  /* count *'s */
  for (ns = 0; gettoken() == '*'; ++ns) { }
  dirdcl();
  while (ns-- > 0)
    strcat(out, " pointer to");
}

/* dirdcl: parse a direct declarator */
void dirdcl(void)…
```


Complicated Declarations

- 복잡한 선언문을 설명하는 프로그램(dcl) (Cont'd)

```
dcl:        optional *'s direct-dcl
direct-dcl  name
            (dcl)
            direct-dcl()
            direct-dcl[optional size]
```

```c
…
/* dirdcl: parse a direct declarator */
void dirdcl(void)
{
  int type;

  if (tokentype == '(') {  /* ( dcl ) */
    dcl();
    if (tokentype != ')')
      printf("error: missing )\n");
  } else if (tokentype == NAME) {  /* variable name */
    strcpy(name, token);
  } else {
    printf("error: expected name or (dcl)\n");
  }
  while ((type=gettoken()) == PARENS || type == BRACKETS) {
    if (type == PARENS) {
      strcat(out, " function returning");
    } else {
      strcat(out, " array");
      strcat(out, token);
      strcat(out, " of");
    }
  }
}
```