Learning summary:
Study: 6 Hours
Exercises: 1.5 Hours

Documentation of Day_20

Exercise 5-18:

Make dcl recover from input errors

Source Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAXTOKEN 100
#define MAXVAL 100
#define BUFSIZE 100

char buf[BUFSIZE];
int bufp = 0;
enum { NAME, PARENS, BRACKETS };

int getch(void);
void ungetch(int c);
void dcl(void);
void dirdcl(void);
int gettoken(void);

int tokentype;
char token[MAXTOKEN];
char name[MAXTOKEN];
char datatype[MAXTOKEN];
char out[1000];

int main()
{
   while (gettoken() != EOF) {
      strcpy(datatype, token);
      out[0] = '\0';
      dcl();
      if (tokentype != '\n'){
         printf("syntax error\n");
         continue;
                  }
      printf("%s: %s %s\n", name, out, datatype);
   }
   return 0;
}
```

```
void dcl(void)
{
   int ns;
   for (ns = 0; gettoken() == '*'; ns++)
      ;
   dirdcl();
   while (ns-- > 0)
      strcat(out, " pointer to");
}

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

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

   while ((c = getch()) == ' ' || c == '\t')
      ;
   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;
    }
}

int getch(void)
{
    return (bufp > 0) ? buf[--bufp] : getchar();
}

void ungetch(int c)
{
    if (bufp >= BUFSIZE)
        printf("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}
```

The main functionality of the code can be summarized as follows:

1. The program reads input declarations and separates the datatype from the variable name.
2. It analyzes the declaration recursively, considering the presence of pointers, arrays, and function return types.
3. It constructs a string representation of the declaration by appending the necessary keywords ("pointer to," "array of," "function returning") based on the declaration structure.
4. Finally, it prints the analyzed declaration along with the variable name and datatype.
5. The code handles basic error checking and reports syntax errors if encountered.

Inputs and Outputs:

1.

```
D:\Day_Wise_Files\Day_20_r\Ex_5-18.exe
char **argv
argv:  pointer to pointer to char
int (*daytab)[13]
daytab:  pointer to array[13] of int
int *daytab[13]
daytab:  array[13] of pointer to int
void *comp()
comp:  function returning pointer to void
void (*comp)()
comp:  pointer to function returning void
char (*(*x())[])()
x:  function returning pointer to array[] of pointer to function returning char
char (*(*x[3])())[5]
x:  array[3] of pointer to function returning pointer to array[5] of char
```

Exercise 5-19:

Modify undcl so that it does not add redundant parentheses to declarations.

The code that I have written is an implementation of a simple C program that analyzes and prints variable declarations. It utilizes a recursive descent parsing approach to handle complex declarations involving pointers, arrays, and functions. The main functionalities are:

1. Reading input declarations and separating the datatype from the variable name.
2. Analyzing the declaration recursively to handle pointers, arrays, and function return types.
3. Constructing a string representation of the declaration by appending the appropriate keywords.
4. Printing the analyzed declaration along with the variable name and datatype.
5. Performing basic error checking and reporting syntax errors if encountered.

Source Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAXTOKEN 100
#define BUFSIZE 100

char buf[BUFSIZE];
int bufp = 0;
```

```
enum { NAME, PARENS, BRACKETS };

int getch(void);
void ungetch(int c);
int gettoken(void);

int tokentype;
char token[MAXTOKEN];
char name[MAXTOKEN];
char datatype[MAXTOKEN];
char out[1000];

int main(void)
{
   int type, paren_flag = 0;
   char temp[MAXTOKEN];

   while (gettoken() != EOF)
   {
      strcpy(out, token);

      while ((type = gettoken()) != '\n')
      {
         if (paren_flag)
         {
            if (type == PARENS || type == BRACKETS)
            {
               sprintf(temp, "(*%s)", out);
               strcpy(out, temp);
            }
            else
            {
               sprintf(temp, "*%s", out);
               strcpy(out, temp);
            }
            paren_flag = 0;
         }
         if (type == PARENS || type == BRACKETS)
         {
            strcat(out, token);
         }
         else if (type == '*')
         {
            paren_flag = 1;
```

```
        }

        else if (type == NAME)
        {
            sprintf(temp, "%s %s", token, out);
            strcpy(out, temp);
        }
        else
        {
            printf("Error: Invalid input at %s\n", token);
            break;
        }
    }

    printf("%s\n", out);
  }

  return 0;
}

int gettoken(void)
{
  int c;
  char *p = token;

  while ((c = getch()) == ' ' || c == '\t')
    ;

  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;
  }
}


int getch(void)
{
  return (bufp > 0) ? buf[--bufp] : getchar();
}

void ungetch(int c)
{
  if (bufp >= BUFSIZE)
    printf("Error: ungetch: too many characters\n");
  else
    buf[bufp++] = c;
}
```

Input and Output:

D:\Day_Wise_Files\Day_20_r\Ex_5-19.exe

```
a * char
char *a
argv * * char
char **argv
daytab * [13] int
int (*daytab)[13]
comp () * void
void *comp()
comp * () void
void (*comp)()
x () * [] * () char
char (*(*x())[])()
x [3] * () * [5] char
char (*(*x[3])())[5]
x () * [] * () char
char (*(*x())[])()
```