

Day 20 Documentation

Md. Mahfuj Hasan Shohug

BDCOM0019

.....

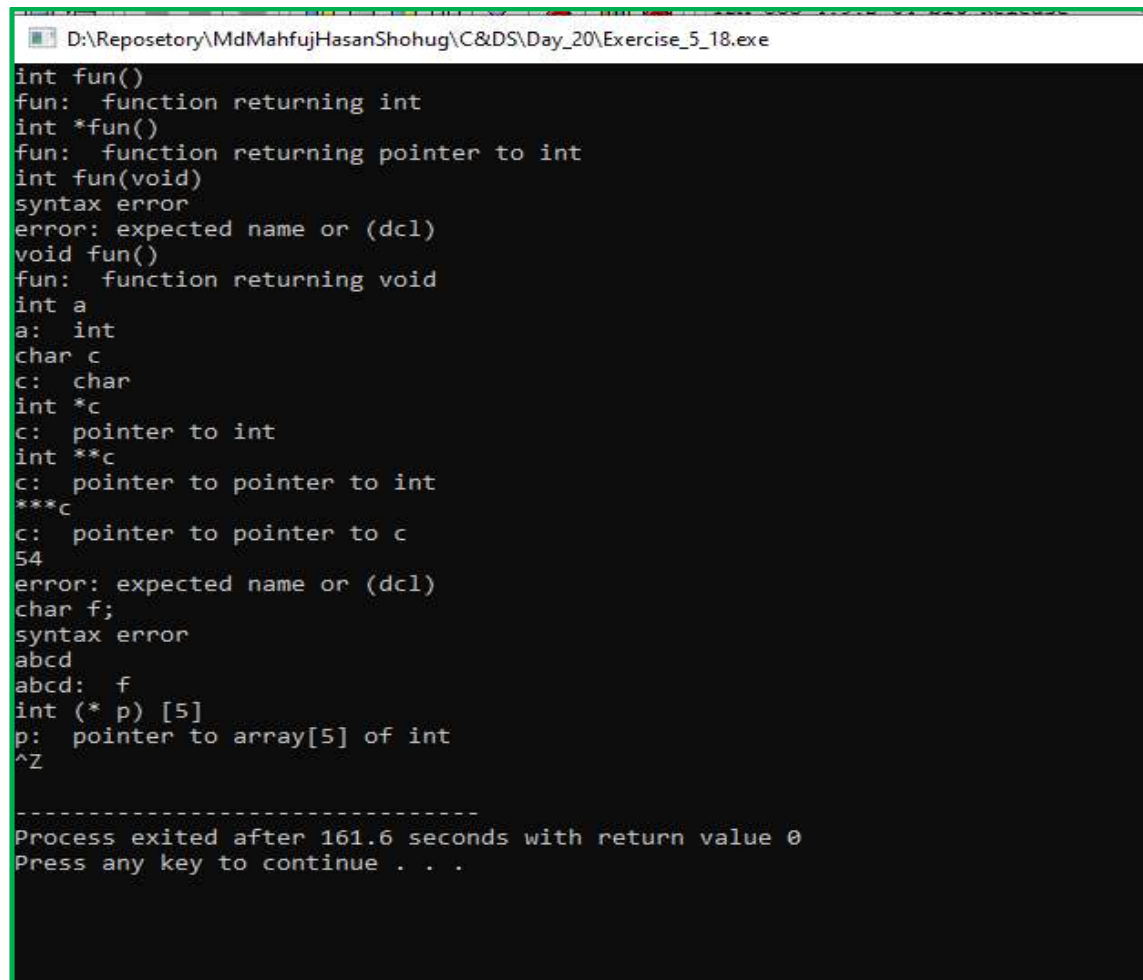
1. Exercise 5-18:

Problem: Make dcl recover from input errors.

Analysis: The main change was error handling and recovery in the dcl() function in my program as compared to the code provided in the book. Continually parsing the declaration while gently handling input errors is the point of view.

- The dirdcl() function handles errors by setting error_flag to 1 when one occurs, such as a missing closing parenthesis or an unexpected token. Until it encounters a closing parenthesis or reaches the end of the input, the function keeps reading tokens.
- Only in cases where there are no errors (error_flag is 0) is output printed. This makes sure that the message are seen.

Here I give some test case of this program:



```
D:\Repository\MdMahfujHasanShohug\C&DS\Day_20\Exercise_5_18.exe
int fun()
fun: function returning int
int *fun()
fun: function returning pointer to int
int fun(void)
syntax error
error: expected name or (dcl)
void fun()
fun: function returning void
int a
a: int
char c
c: char
int *c
c: pointer to int
int **c
c: pointer to pointer to int
***c
c: pointer to pointer to c
54
error: expected name or (dcl)
char f;
syntax error
abcd
abcd: f
int (* p) [5]
p: pointer to array[5] of int
^Z

-----
Process exited after 161.6 seconds with return value 0
Press any key to continue . . .
```

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; /* 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];

int error_flag = 0; // Flag to track errors

/*****
 * Function Name: main
 * Description: Entry point of the program.
 * (void) part specifies an empty parameter list.
 * Returns: 0 on successful execution.
 *****/
int main(void)
{
    while (gettoken() != EOF) {
        strcpy(datatype, token);
        out[0] = '\0';
        error_flag = 0; // Reset error flag for each declaration
        dcl();
        if (tokentype != '\n') {
            printf("syntax error\n");
            error_flag = 1; // Set error flag
        }
        if (!error_flag) { // Print output only if no error encountered
            printf("%s: %s %s\n", name, out, datatype);
        }
    }
    return 0;
}

/*****
 * Function Name: dcl
 * Description: Parses a declarator.
 * Returns: None.
 *****/
void dcl(void)
```

```

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

/*****
 * Function Name: dirdcl
 * Description: Parses a direct declarator.
 * Returns: None.
 *****/
void dirdcl(void)
{
    int type;
    if (tokentype == '(') {
        dcl();
        if (tokentype != ')') {
            printf("error: missing )\n");
            error_flag = 1; // Set error flag
            while (gettoken() != ')') && tokentype != EOF
                ;
        }
    } else if (tokentype == NAME) {
        strcpy(name, token);
    } else {
        printf("error: expected name or (dcl)\n");
        error_flag = 1; // Set error flag
    }
    while ((type = gettoken()) == PARENS || type == BRACKETS) {
        if (type == PARENS) {
            strcat(out, " function returning");
        } else {
            strcat(out, " array");
            strcat(out, token);
            strcat(out, " of");
        }
    }
}

/*****
 * Function Name: gettoken
 * Description: Retrieves the next token from input.
 * Returns: The type of the token.
 *****/
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;
}
}

/*****
 * Function Name: getch
 * Description: Get a (possibly pushed back) character.
 * Returns: The next character from input.
 *****/
int getch(void)
{
    return (bufp > 0) ? buf[--bufp] : getchar();
}

/*****
 * Function Name: ungetch
 * Description: Push character back on input.
 * Parameters:
 *   - c: The character to be pushed back.
 * Returns: None.
 *****/
void ungetch(int c)
{
    if (bufp >= BUFSIZE)
        printf("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}

```

2. Exercise 5-19:

Problem: Modify undc1 so that it does not add redundant parentheses to declarations.

Analysis: In the code from the book, the modification that I done on undc1() function to handle redundant parentheses in declarations. The goal is to avoid adding unnecessary parentheses in the output.

- The modification prevents the addition of extra parentheses to the output. The output is formatted without using additional brackets if a token needs to be delayed in processing and the token after it is likewise a PARENS or BRACKETS token.
- These adjustments make sure that the declaration in the output does not include any extraneous parenthesis.

In the book undc1() function when take the input:

From the input syntax stipulated for undc1, namely:

Input: x () * [] * () char to ooutput: char ((*x())[])(),

Modification: Sample input: x () * * * char

Output before changes: char ((*(*x())))

Output after changes: char (**x())

Test case:

```
D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_20\Exercise_5_19.exe
x () * [] * () char
Update Output: char ((*x())[ ])( )
x () * * * char
Update Output: char ***x()
x [3] * * () * * char
Update Output: char **(**x[3])()
char ((*(*(*x[3]))()))
Error: Invalid input at char
```

```
D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_20\Exercise_5_19.exe
(*(*(*x[3]))())
Error: Invalid input at
Update Output: *
Error: Invalid input at
Update Output:
Error: Invalid input at
Update Output:
Error: Invalid input at [3]
Update Output: x [3]
Error: Invalid input at ( )
Update Output: [3]()
Update Output: ( )

x () * [] * () char
Update Output: char ((*x ( )())[ ])( )
```

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];

/*****
 * Function Name: main
 * Description: Converts word description to a declaration and prints the output.
 * Variables: type - type of token
 *      paren_flag - flag to indicate if parentheses are present
 *      temp - temporary string for constructing the output
 *****/
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("Update Output: %s\n", out);
}

return 0;
}

/*****
* Function Name: gettoken
* Description: Skips blanks and tabs, then finds the next token in the input.
* Variables: c - current character
*           p - pointer to the token string
*****/
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;
}
}

/*****
* Function Name: getch
* Description: Gets the next character from input or the buffer
* Variables: None
*****/
int getch(void)
{
    return (bufp > 0) ? buf[--bufp] : getchar();
}

/*****
* Function Name: ungetch
* Description: Pushes character back onto input or the buffer
* Variables: c - character to be pushed back
*****/
void ungetch(int c)
{
    if (bufp >= BUFSIZE)
        printf("Error: ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}
```