Study Time: 5.5 hours
Exercise Time: 2.5 hours

Documentation of Day 25

Exercise 6-1.

Our version of `getword` does not properly handle underscores, string constants, comments, or preprocessor control lines. Write a better version.

Below I have given the step by step approach of to implement the program:

From book:

If a large structure is to be passed to a function, it is generally more efficient to pass a pointer than to copy the whole structure. Structure pointers are just like pointers to ordinary variables. The declaration

That is why I have implemented the program using pointer.

- The program counts occurrences of C keywords in input text.
- It uses an array of structures to store keywords and their counts.
- Input is processed line by line, character by character.
- Preprocessing directives are skipped.
- Words starts with Underscore are skipped.
- Comments and string literals are ignored.
- Words are extracted and converted to lowercase.
- Binary search is used to find keywords in the array.
- Keyword counts are incremented when found.
- Case insensitivity is ensured by converting words to lowercase.
- The program displays the counts of C keywords.

Source Code:

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

#define MAXWORD 100
#define NKEYS 32

struct key {
    char *word;
    int count;
};

struct key keytab[NKEYS] = {
    { "auto", 0 },
```

```
    { "break", 0 },
    { "case", 0 },
    { "char", 0 },
    { "const", 0 },
    { "continue", 0 },
    { "default", 0 },
    { "do", 0 },
    { "double", 0 },
    { "else", 0 },
    { "enum", 0 },
    { "extern", 0 },
    { "float", 0 },
    { "for", 0 },
    { "goto", 0 },
    { "if", 0 },
    { "int", 0 },
    { "long", 0 },
    { "register", 0 },
    { "return", 0 },
    { "short", 0 },
    { "signed", 0 },
    { "sizeof", 0 },
    { "static", 0 },
    { "struct", 0 },
    { "switch", 0 },
    { "typedef", 0 },
    { "union", 0 },
    { "unsigned", 0 },
    { "void", 0 },
    { "volatile", 0 },
    { "while", 0 }
};

int get_word(char *ptr, char *word, int lim);
int binsearch(char *, struct key *, int);

int main() {
    char line[MAXWORD];
    char word[MAXWORD];
    int in_comment = 0;
    int skip_line = 0;
    int under_char = 0;
    int in_string_literal = 0;

    printf("Enter text (press Ctrl + z to finish):\n");
```

```c
while (fgets(line, sizeof(line), stdin) != NULL) {
    int len = strlen(line);
    if (line[len - 1] == '\n') {
        line[len - 1] = '\0'; // Remove the newline character
    }

    char *ptr = line;

    while (*ptr != '\0') {
        if (!in_comment && !in_string_literal) {
            if (*ptr == '#') {
                // Handle preprocessing directives
                // Skip counting
                break;
            }

            if (*ptr == '/' && *(ptr + 1) == '*') {
                // Start of a multi-line comment
                in_comment = 1;
                ptr += 2;
                continue;
            }

            if (*ptr == '/' && *(ptr + 1) == '/') {
                // Start of a single-line comment
                skip_line = 1;
                break; // Skip the entire line and move to the next iteration
            }

            if (*ptr == '"') {
                // Start of a string literal
                in_string_literal = 1;
                ptr++;
                continue;
            }

            if(*ptr =='_'){
                //Start of a underscore
                under_char = 1;
                ptr++;
                continue;
            }
```

```
            int word_len = get_word(ptr, word, MAXWORD);

            // Convert word to lowercase for case-insensitive comparison
            int i;
            for (i = 0; i < word_len; i++) {
                word[i] = tolower(word[i]);
            }

            int index = binsearch(word, keytab, NKEYS);
            if (index != -1) {
                keytab[index].count++;
            }

            ptr += word_len;
        }

        if (in_comment) {
            if (*ptr == '*' && *(ptr + 1) == '/') {
                // End of a comment
                in_comment = 0;
                ptr += 2;
                continue;
            }
        }

        if(under_char){
            //skip the word starts with underscore
            while(*ptr++ != '\0');
            under_char = 0;
            continue;
        }

        if(skip_line){
            //skip the single-line comment
            while(*ptr++ != '\n');
            skip_line = 0;
            continue;
        }

        if (in_string_literal) {
            if (*ptr == '"') {
                // End of a string literal
                in_string_literal = 0;
                ptr++;
                continue;
```

```
                }
            }
            ptr++;
        }
    }

    printf("\nKeyword counts:\n");
    int j;
    for (j = 0; j < NKEYS; j++) {
        if (keytab[j].count > 0) {
            printf("%4d %s\n", keytab[j].count, keytab[j].word);
        }
    }

    return 0;
}

int binsearch(char *word, struct key tab[], int n) {
    int low = 0;
    int high = n - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int cmp = strcmp(word, tab[mid].word);

        if (cmp < 0) {
            high = mid - 1;
        } else if (cmp > 0) {
            low = mid + 1;
        } else {
            return mid;
        }
    }

    return -1;
}

int get_word(char *ptr, char *word, int lim) {
    int c;
    int word_len = 0;

    while (*ptr != '\0' && isspace(*ptr)) {
        ptr++;
    }
```

```
    while (*ptr != '\0' && !isspace(*ptr) && word_len < lim - 1) {
        word[word_len++] = *ptr++;
    }

    word[word_len] = '\0';
    return word_len;
}
```

➔ **Concepts Explored:**

1. Arrays of Structures: Utilizing an array to store a collection of structures.

2. String Manipulation: Performing operations on strings, such as extracting words and manipulating their contents.

3. Searching and Sorting: Implementing a binary search algorithm to search for keywords and sorting the keyword array for efficient access.

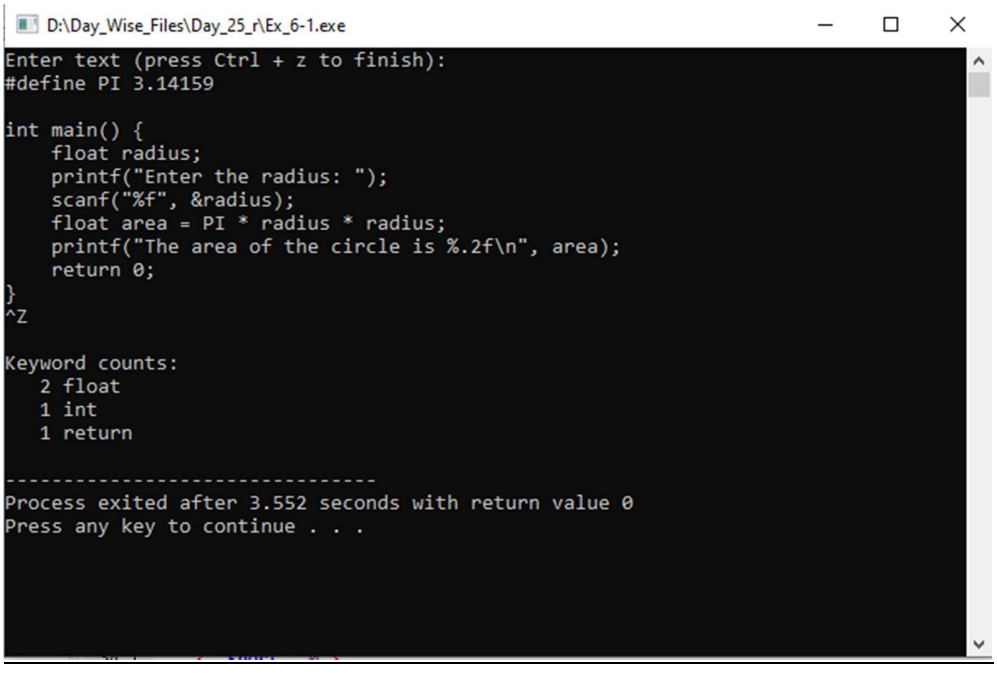
4. Input Processing: Reading and processing user input from the command line.

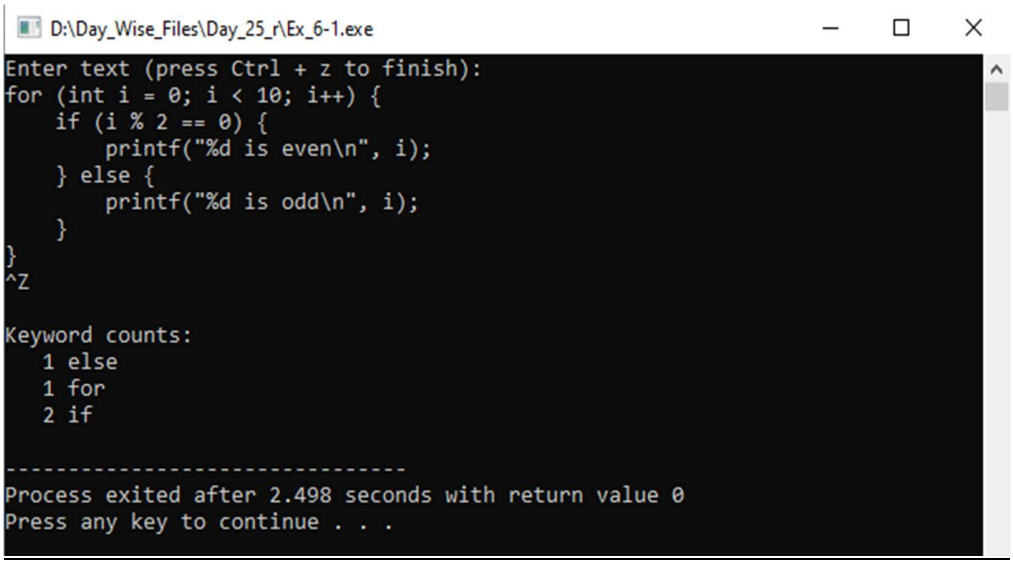
5. Case Insensitivity: Converting words to lowercase for case-insensitive comparison.

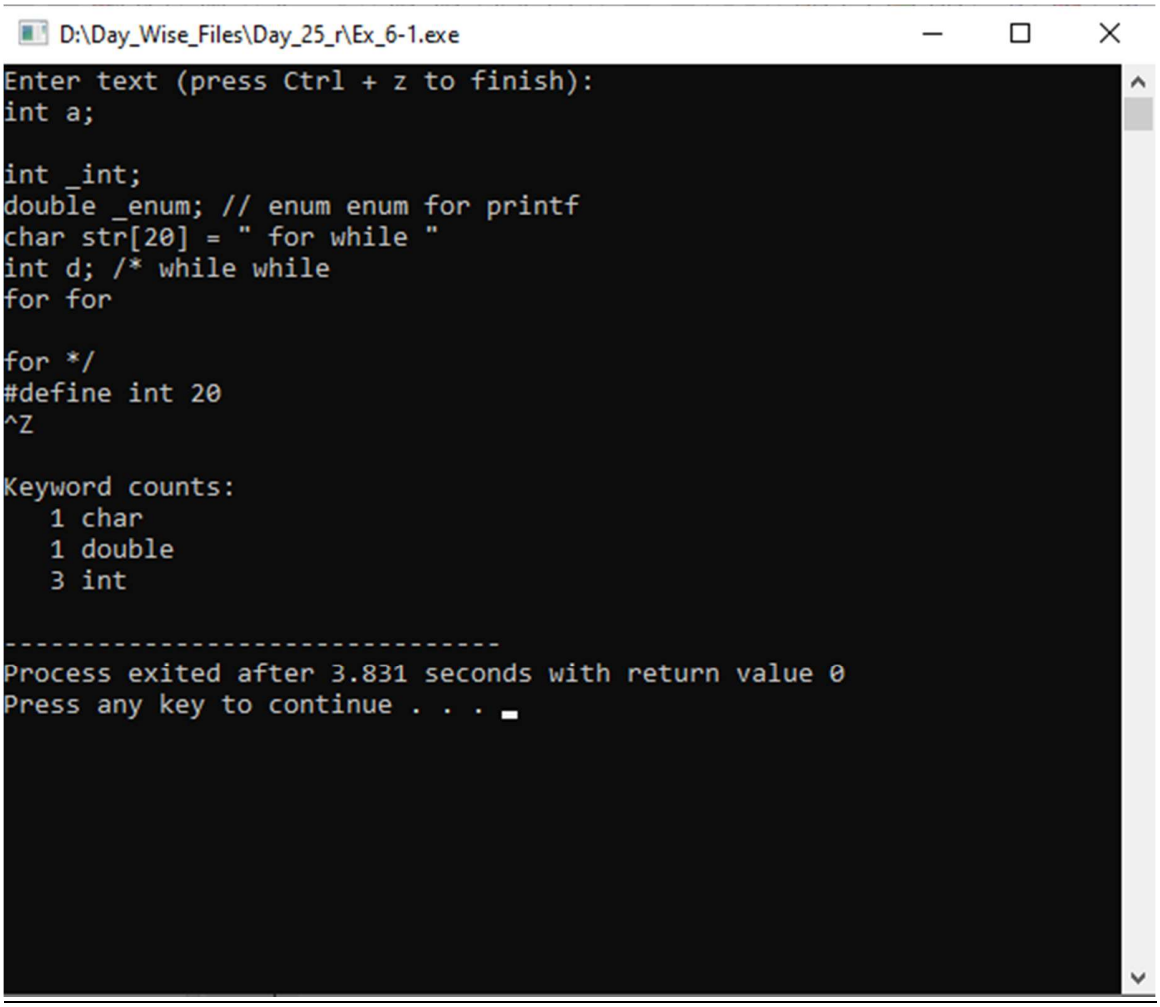
6. Conditional Statements and Loops: Using if statements and loops to control the program flow and iterate through input lines and characters.

These concepts collectively contribute to the implementation of the program and showcase different aspects of C programming, including data structures, string manipulation, input processing, and control flow.

Output:



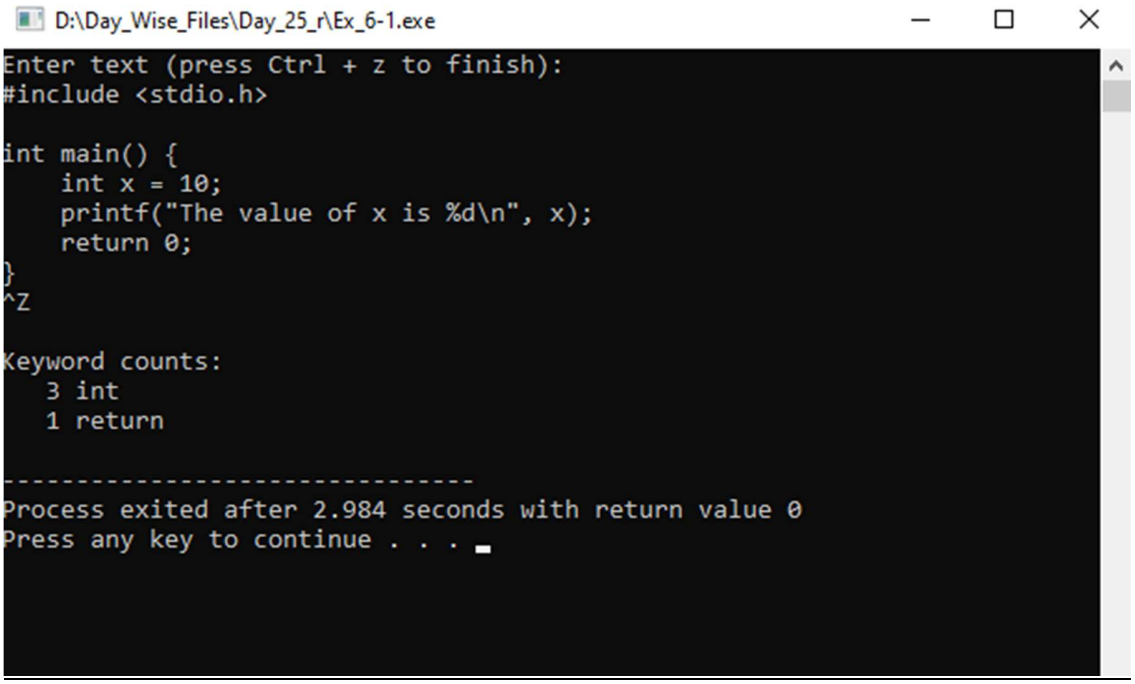

D:\Day_Wise_Files\Day_25_r\Ex_6-1.exe
Enter text (press Ctrl + z to finish):
for (int i = 0; i < 10; i++) {
if (i % 2 == 0) {
printf("%d is even\n", i);
} else {
printf("%d is odd\n", i);
}
}
^Z
Keyword counts:
1 else
1 for
2 if
Process exited after 2.498 seconds with return value 0
Press any key to continue . . .

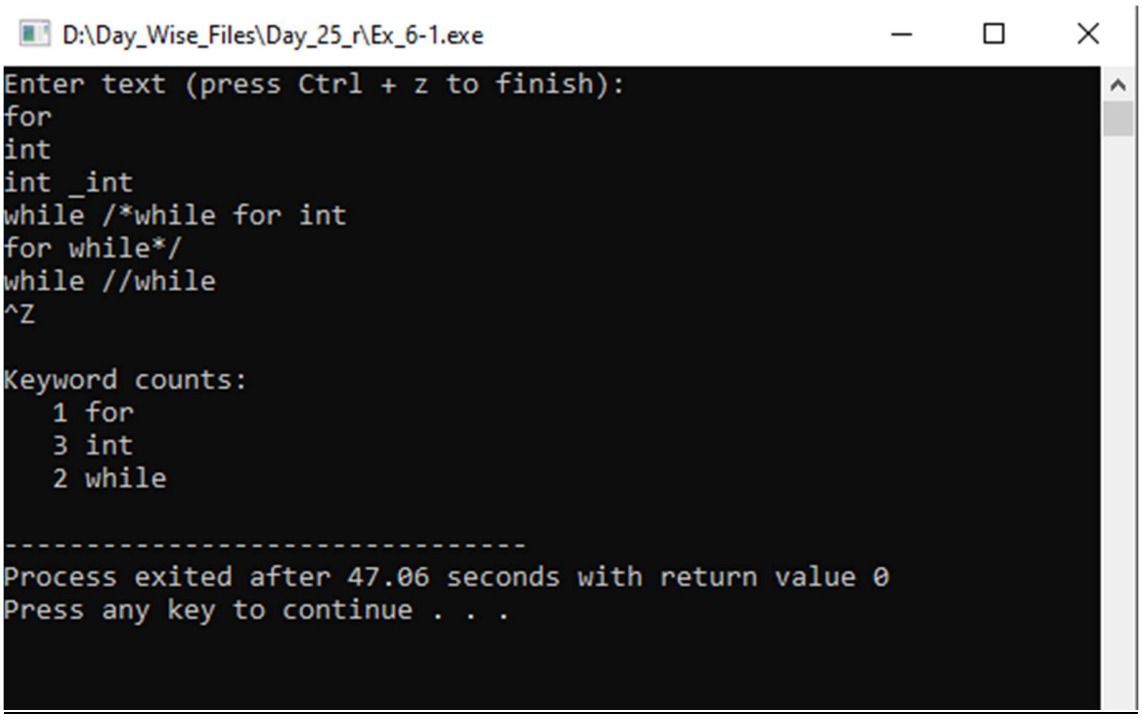

D:\Day_Wise_Files\Day_25_r\Ex_6-1.exe
Enter text (press Ctrl + z to finish):
int a;
int _int;
double _enum; // enum enum for printf
char str[20] = " for while "
int d; /* while while
for for
for */
#define int 20
^Z
Keyword counts:
1 char
1 double
3 int
Process exited after 3.831 seconds with return value 0
Press any key to continue . . .

Here the full source code of this program is the input:

D:\Day_Wise_Files\Day_25_r\Ex_6-1.exe

```
    printf("\nKeyword counts:\n");
    int j;
    for (j = 0; j < NKEYS; j++) {
        if (keytab[j].count > 0) {
            printf("%4d %s\n", keytab[j].count, keytab[j].word);
        }
    }

    return 0;
}

int binsearch(char *word, struct key tab[], int n) {
    int low = 0;
    int high = n - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int cmp = strcmp(word, tab[mid].word);

        if (cmp < 0) {
            high = mid - 1;
        } else if (cmp > 0) {
            low = mid + 1;
        } else {
            return mid;
        }
    }

    return -1;
}

int get_word(char *ptr, char *word, int lim) {
    int c;
    int word_len = 0;

    while (*ptr != '\0' && isspace(*ptr)) {
        ptr++;
    }

    while (*ptr != '\0' && !isspace(*ptr) && word_len < lim - 1) {
        word[word_len++] = *ptr++;
    }

    word[word_len] = '\0';
    return word_len;
}
^Z

Keyword counts:
   7 char
   2 else
   4 for
  67 if
  52 int
   5 return
   4 struct
   6 while

--------------------------------
Process exited after 3.812 seconds with return value 0
Press any key to continue . . .
```


D:\Day_Wise_Files\Day_25_r\Ex_6-1.exe
Enter text (press Ctrl + z to finish):
#include <stdio.h>
int main() {
int x = 10;
printf("The value of x is %d\n", x);
return 0;
}
^Z
Keyword counts:
3 int
1 return
Process exited after 2.984 seconds with return value 0
Press any key to continue . . .


D:\Day_Wise_Files\Day_25_r\Ex_6-1.exe
Enter text (press Ctrl + z to finish):
for
int
int _int
while /*while for int
for while*/
while //while
^Z
Keyword counts:
1 for
3 int
2 while
Process exited after 47.06 seconds with return value 0
Press any key to continue . . .