Study Time: 5.5 hours

Exercise Time: 2.5 hours

Documentation of Day 26

Exercise 6-4.

Write a program that prints the distinct words in its input sorted into decreasing order of frequency of occurrence. Precede each word by its count.

Approach to implement my program:

- Reads input from the user
- Identifies distinct words
- Counts the frequency of each word. It then
- Sorts the distinct words based on their frequency of occurrence in decreasing order and
- Prints the sorted list.

Source Code:

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

struct tree_node
{
    char *word;
    int count;
    struct tree_node *left;
    struct tree_node *right;
};

#define MAXWORD 100
#define BUFSIZE 100
#define MAXNUMBERWORDS 10000

struct tree_node *add_node(struct tree_node *, char *);
void print_tree(void);
int get_word(char *, int);
struct tree_node *talloc(void);
char *string_duplicate(char *);
int getch(void);
void ungetch(int);
void my_qsort(struct tree_node *v[], int left, int right);
void swap_node(struct tree_node *v[], int i, int j);
```

```c
char buf[BUFSIZE];
int bufp = 0;
struct tree_node *tree_nodes[MAXNUMBERWORDS] = { 0 };
int tree_ptr = 0;

/****************************************************************************/
/* FUNCTION NAME : main                                                     */
/*                                                                          */
/* INPUTS        : void                                                     */
/* RETURN        : int                                                      */
/*                                                                          */
/* NOTES         : Builds a binary search tree from words input by the user, */
/*                 then sorts the tree nodes using quicksort and prints the  */
/*                 tree in ascending order.                                  */
/****************************************************************************/
int main()
{
    struct tree_node *root;
    char word[MAXWORD];

    root = NULL;
    while (get_word(word, MAXWORD) != EOF)
        if (isalpha(word[0]) || word[0] == '_')
            root = add_node(root, word);
    my_qsort(tree_nodes, 0, tree_ptr - 1);
    print_tree();
    return 0;
}

/******************************************************************************/
/* STRUCTURE NAME : add_node                                                  */
/*                                                                            */
/* INPUTS        : struct tree_node *p - Pointer to the root of the tree      */
/*                 char *w - Word to be added to the tree                     */
/* RETURN        : struct tree_node * - Pointer to the updated root of the tree */
/*                                                                            */
/* NOTES         : Adds a new node with the given word to the binary search   */
/*                 tree. If the word already exists, increments its count.    */
/******************************************************************************/
struct tree_node *add_node(struct tree_node *p, char *w)
{
    int result;

    if (p == NULL)
    {
        p = talloc();
        p->word = string_duplicate(w);
        p->count = 1;
        p->left = p->right = NULL;
        tree_nodes[tree_ptr++] = p;
    }
    else if ((result = strcmp(w, p->word)) == 0)
        p->count++;
    else if (result < 0)
        p->left = add_node(p->left, w);
    else
        p->right = add_node(p->right, w);
    return p;
}

/******************************************************************************/
/* FUNCTION NAME : print_tree                                                 */
/*                                                                            */
/* INPUTS        : void                                                       */
/* RETURN        : void                                                       */
/*                                                                            */
/* NOTES         : Prints the tree nodes in the order of their counts and     */
/*                 corresponding words.                                       */
/******************************************************************************/
void print_tree(void)
{
    int i;
    for (i = 0; i < tree_ptr; i++)
        printf("%4d %s\n", tree_nodes[i]->count, tree_nodes[i]->word);
}

/******************************************************************************/
/* STRUCTURE NAME : talloc                                                    */
/*                                                                            */
/* INPUTS        : void                                                       */
/* RETURN        : struct tree_node *                                         */
/*                                                                            */
/* NOTES         : Allocates memory for a new tree node.                      */
/******************************************************************************/
struct tree_node *talloc(void)
{
    return (struct tree_node *) malloc(sizeof(struct tree_node));
}
```

```c
/*****************************************************************************/
/* FUNCTION NAME : string_duplicate                                          */
/*                                                                           */
/* INPUTS        : char *s - String to be duplicated                         */
/* RETURN        : char *                                                    */
/*                                                                           */
/* NOTES         : Creates a duplicate of the given string.                  */
/*****************************************************************************/
char *string_duplicate(char *s) {
    char *p = (char *) malloc(strlen(s) + 1);
    if (p != NULL)
        strcpy(p, s);
    return p;
}

/*****************************************************************************/
/* FUNCTION NAME : get_word                                                  */
/*                                                                           */
/* INPUTS        : char *word - Buffer to store the word                     */
/*                 int lim - Maximum length of the word buffer               */
/* RETURN        : int - First character of the word                         */
/*                                                                           */
/* NOTES         : Reads a word from input, consisting of alphabetic letters, */
/*                 digits, or underscores. Returns the first character of the */
/*                 word and stores the complete word in the buffer.          */
/*****************************************************************************/
int get_word(char *word, int lim) {
    int c;
    char *w = word;
    while ((c = getch()) == '\t' || c == ' ')
        ;
    if (c != EOF)
        *w++ = c;
    if (!isalpha(c) && c != '_') {
        *w = '\0';
        return c;
    }
    for ( ; --lim > 0; w++)
        if (!isalnum(*w = getch()) && *w != '_') {
            ungetch(*w);
            break;
        }
    *w = '\0';
    return word[0];
}
```

```c
/*****************************************************************************/
/* FUNCTION NAME : getch                                                     */
/*                                                                           */
/* INPUTS        : void                                                      */
/* RETURN        : int - Next character from input or buffered character     */
/*                                                                           */
/* NOTES         : Retrieves the next character from input or the buffered   */
/*                 characters if available.                                  */
/*****************************************************************************/
int getch(void) {
    return (bufp > 0) ? buf[--bufp] : getchar();
}

/*****************************************************************************/
/* FUNCTION NAME : ungetch                                                   */
/*                                                                           */
/* INPUTS        : int c - Character to be placed back into the buffer       */
/* RETURN        : void                                                      */
/*                                                                           */
/* NOTES         : Places a character back into the input buffer for future  */
/*                 retrieval.                                                */
/*****************************************************************************/
void ungetch(int c) {
    if (bufp >= BUFSIZE)
        printf("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}

/*****************************************************************************/
/* FUNCTION NAME : my_qsort                                                  */
/*                                                                           */
/* INPUTS        : struct tree_node *v[] - Array of tree_node pointers       */
/*                 int left - Left index of the subarray                     */
/*                 int right - Right index of the subarray                   */
/* RETURN        : void                                                      */
/*                                                                           */
/* NOTES         : Sorts the array of tree_node pointers using quicksort     */
/*                 algorithm in descending order based on count field.       */
/*****************************************************************************/
void my_qsort(struct tree_node *v[], int left, int right) {
    int i, last;
    if (left >= right)
        return;
    swap_node(v, left, (left + right) / 2);
    last = left;
    for (i = left + 1; i <= right; i++)
        if (v[left]->count < v[i]->count)
            swap_node(v, ++last, i);
    swap_node(v, left, last);
    my_qsort(v, left, last - 1);
    my_qsort(v, last + 1, right);
}

/****************************************************************************/
/* FUNCTION NAME : swap_node                                                */
/*                                                                          */
/* INPUTS        : struct tree_node *v[] - Array of tree_node pointers      */
/*                 int i - Index of the first node to swap                  */
/*                 int j - Index of the second node to swap                 */
/* RETURN        : void                                                     */
/*                                                                          */
/* NOTES         : Swaps two tree_node pointers in the array.               */
/****************************************************************************/
void swap_node(struct tree_node *v[], int i, int j)
{
    struct tree_node *temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}
```

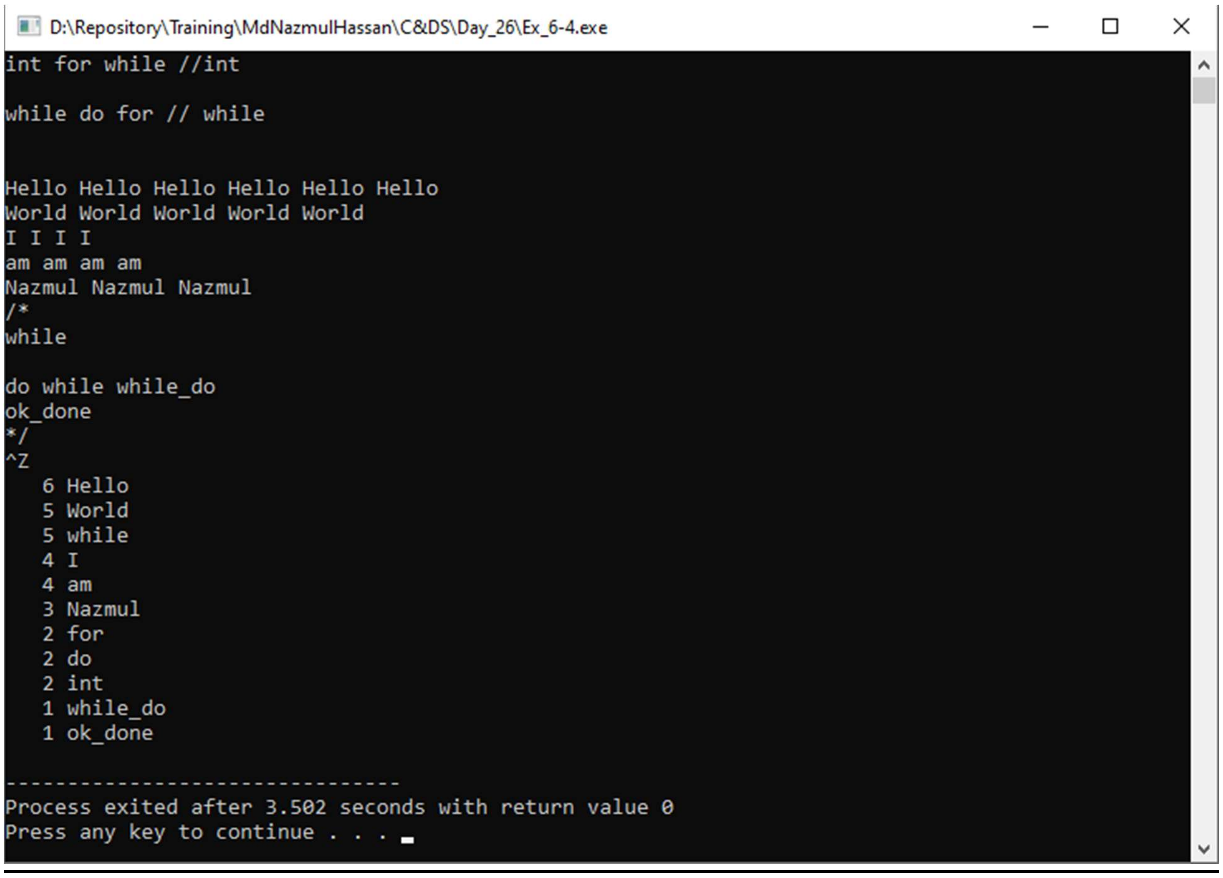
➔ Concepts Explored:

- **Binary search trees:** I have used a binary search tree data structure to store distinct words.
- **Tree traversal:** In-order traversal of the binary search tree to print the words in sorted order.
- **Dynamic memory allocation:** I have used malloc() to allocate memory for tree nodes and free() to release memory, for efficient memory management.
- **String manipulation:** Functions like strcmp(), strcpy(), and string_duplicate() to compare, copy, and duplicate strings.
- **Sorting algorithms:** The quicksort algorithm is implemented to sort the distinct words based on their frequencies.
- **Input processing:** Reads input characters, skips spaces and tabs, and extracts words using functions like getch() and ungetch().

- **Modular programming:** I have organized the code into separate functions, improving code readability and maintainability.

- **Pointers and memory management:** Pointers are used to manipulate tree nodes and manage memory allocation.

Output:


D:\Repository\Training\MdNazmulHassan\C&DS\Day_26\Ex_6-4.exe

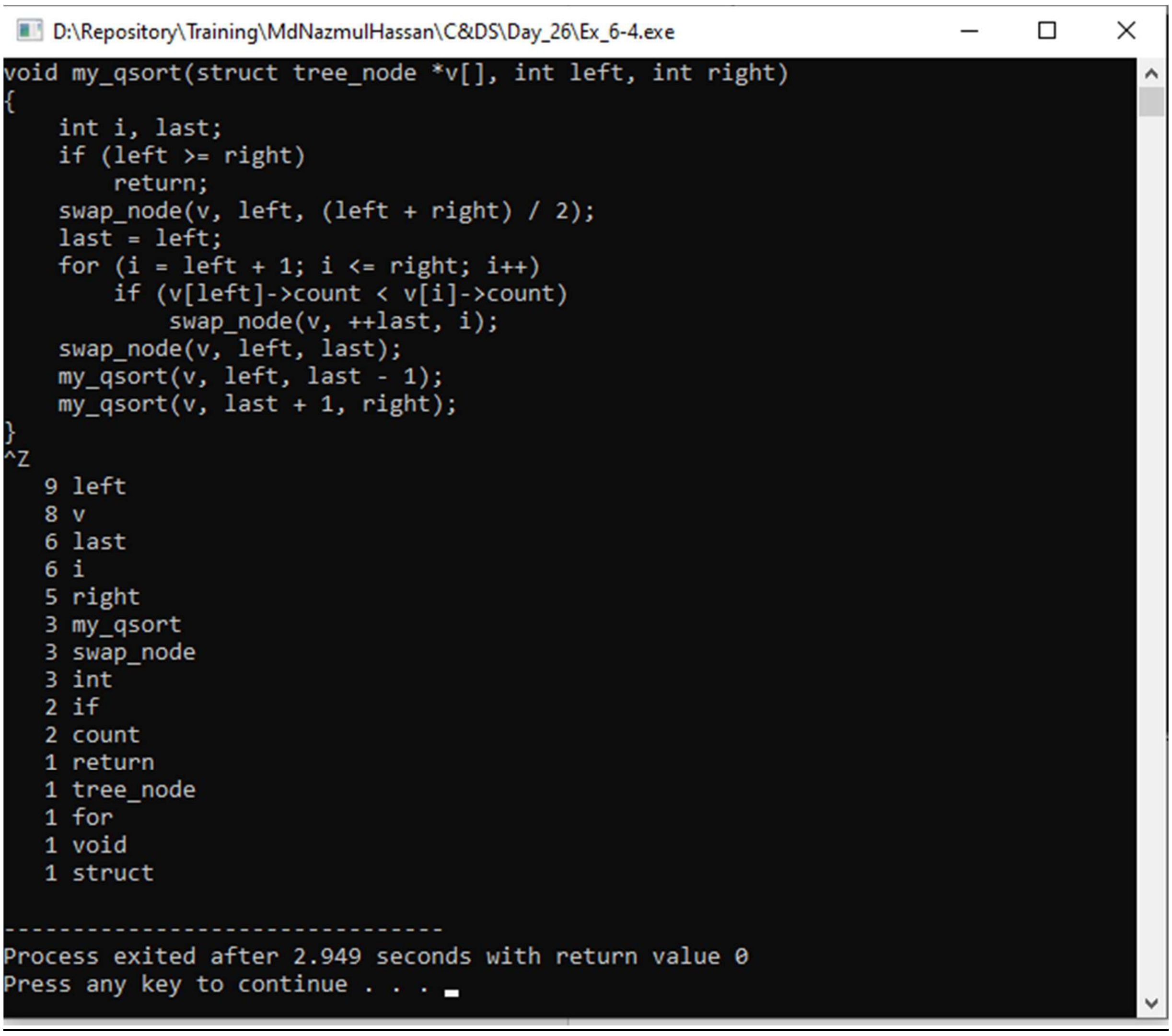
```
int for while //int

while do for // while


Hello Hello Hello Hello Hello Hello
World World World World World
I I I I
am am am am
Nazmul Nazmul Nazmul
/*
while

do while while_do
ok_done
*/
^Z
   6 Hello
   5 World
   5 while
   4 I
   4 am
   3 Nazmul
   2 for
   2 do
   2 int
   1 while_do
   1 ok_done

--------------------------------
Process exited after 3.502 seconds with return value 0
Press any key to continue . . .
```


D:\Repository\Training\MdNazmulHassan\C&DS\Day_26\Ex_6-4.exe

```
void my_qsort(struct tree_node *v[], int left, int right)
{
    int i, last;
    if (left >= right)
        return;
    swap_node(v, left, (left + right) / 2);
    last = left;
    for (i = left + 1; i <= right; i++)
        if (v[left]->count < v[i]->count)
            swap_node(v, ++last, i);
    swap_node(v, left, last);
    my_qsort(v, left, last - 1);
    my_qsort(v, last + 1, right);
}
^Z
   9 left
   8 v
   6 last
   6 i
   5 right
   3 my_qsort
   3 swap_node
   3 int
   2 if
   2 count
   1 return
   1 tree_node
   1 for
   1 void
   1 struct

--------------------------------
Process exited after 2.949 seconds with return value 0
Press any key to continue . . .
```