Dynamic Memory Management Assignment

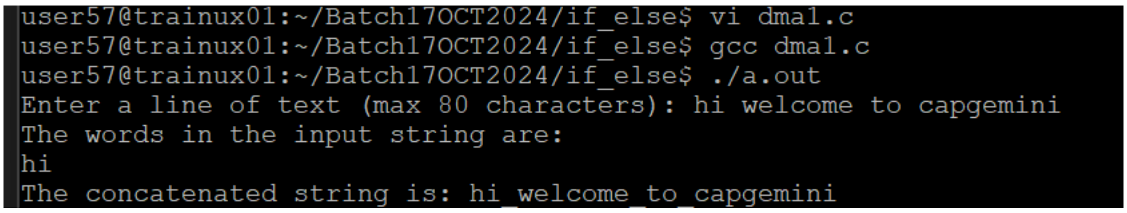
1. Write a program to read a line of text containing 2 or more words, tokenize, display the words, concatenate the words using '_' and display the final string. Consider line length of 80 characters. Provide a modular solution implementing following functions.
   //process the input string and return a concatenated string allocated memory in heap
   char *process_string(char *line);

Sol:

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAX_LEN 80


char *process_string(char *line) {

char *final_str = (char *)malloc(MAX_LEN * sizeof(char));

  if (final_str == NULL) {

    printf("Memory allocation failed!\n");

    exit(1);

  }

  final_str[0] = '\0';

  char *token = strtok(line, " ");

  int first = 1;

  while (token != NULL) {

    if (!first) {

      strcat(final_str, "_");

    }

    strcat(final_str, token);

    first = 0;

    token = strtok(NULL, " ");

  }
```

```
    return final_str;
}
int main() {
    char line[MAX_LEN + 1];
    printf("Enter a line of text (max 80 characters): ");
    fgets(line, sizeof(line), stdin);
    line[strcspn(line, "\n")] = '\0';

    char *result = process_string(line);
    printf("The words in the input string are:\n");

    char *token = strtok(line, " ");
    while (token != NULL) {
        printf("%s\n", token);
        token = strtok(NULL, " ");
    }
  printf("The concatenated string is: %s\n", result);
  free(result);

    return 0;
}
```

Output:



```
user57@trainux01:~/Batch17OCT2024/if_else$ vi dma1.c
user57@trainux01:~/Batch17OCT2024/if_else$ gcc dma1.c
user57@trainux01:~/Batch17OCT2024/if_else$ ./a.out
Enter a line of text (max 80 characters): hi welcome to capgemini
The words in the input string are:
hi
The concatenated string is: hi_welcome_to_capgemini
```

2. WAP to read a URL as input from the user, extract the host name and domain name, store them collectively in an appropriate data structure allocating dynamic memory for its members as per required length. Display the structure contents. Free the memory finally. Some of the functions to be implemented are:
   //validate the received url

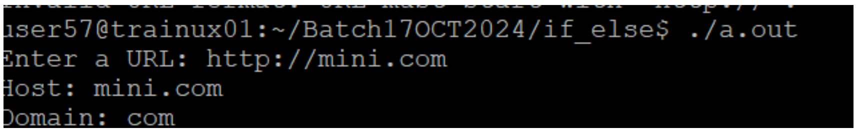
```
int isValidURL(char *url);
//extract and return host name allocated memory in heap
char *gethost(char *url);
//extract and return domain name allocated memory in heap
char *getdomain(char *url);
void display(struct url *obj);
void free(struct url obj);
Input: http://www.altran.com
Output:
Host: altran
Domain: com
Specify the dataset used to test the program
```

Sol:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct url {

  char *host;

  char *domain;

};


int isValidURL(char *url) {

  if (strncmp(url, "http://", 7) == 0) {

    return 1;

  }

  return 0;

}


char *gethost(char *url) {

  char *host_start = url + 7;

  char *host_end = strchr(host_start, '/');

  if (host_end == NULL) {

    host_end = url + strlen(url);
  }



  int length = host_end - host_start;
  char *host = (char *)malloc((length + 1) * sizeof(char));
  if (host == NULL) {
    printf("Memory allocation failed for host!\n");
    exit(1);
  }
  strncpy(host, host_start, length);
  host[length] = '\0';
  return host;
}
char *getdomain(char *url) {
  char *host = gethost(url);
char *dot = strrchr(host, '.');
  if (dot != NULL) {
    char *domain = strdup(dot + 1);
    free(host);
    return domain;
  }
  return NULL;
}
void display(struct url *obj) {
  printf("Host: %s\n", obj->host);
  printf("Domain: %s\n", obj->domain);
}
void free_url(struct url *obj) {
  free(obj->host);
  free(obj->domain);
}
```

```
int main() {
   char url[256];
   struct url u;
   printf("Enter a URL: ");
   fgets(url, sizeof(url), stdin);
   url[strcspn(url, "\n")] = '\0';

   if (isValidURL(url)) {
      u.host = gethost(url);
      u.domain = getdomain(url);
      display(&u);
      free_url(&u);
   } else {
      printf("Invalid URL format. URL must start with 'http://'.\n");
   }
return 0;
}
```

Output:



```
user57@trainux01:~/Batch17OCT2024/if_else$ ./a.out
Enter a URL: http://mini.com
Host: mini.com
Domain: com
```

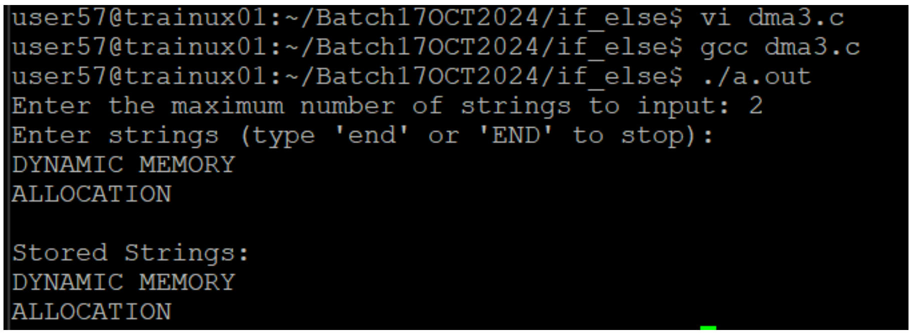
3. WAP to read a maximum of N (N is user input) strings or less from the user at runtime, each string could be of variable length not exceeding a maximum length of 80 characters, allocate memory in heap as per string length and store the strings. Stop reading inputs if input string is “end” or “END”. Display the stored strings. Free the memory before exiting program. Some of the functions to be implemented are:
   [Note : Expected to use char ** and not fixed 2D array]
   //allocate memory for a double pointer to hold n pointers and return
   the pointer
   char **allocate_array_memory(char **ptr, int n);
   //allocate memory for input string and return the pointer
   char *allocate_string_memory(char *string);
   void display(char **arr, int n);

```
void free_array_memory(char **ptr, int n);
void free_string_memory(char *ptr);
```

Sol:

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

char **allocate_array_memory(char **ptr, int n) {
  ptr = (char **)malloc(n * sizeof(char *));
  if (ptr == NULL) {
    printf("Memory allocation failed!\n");
    exit(1);
  }
  return ptr;
}

char *allocate_string_memory(char *string) {
  int length = strlen(string);

  char *str = (char *)malloc((length + 1) * sizeof(char));
  if (str == NULL) {
    printf("Memory allocation for string failed!\n");
    exit(1);
  }
  strcpy(str, string);
  return str;
}

void display(char **arr, int n) {
  printf("\nStored Strings:\n");
  for (int i = 0; i < n; i++) {
    printf("%s\n", arr[i]);
  }
}

void free_array_memory(char **ptr, int n) {
  for (int i = 0; i < n; i++) {
    free(ptr[i]);  // Free each string
  }
  free(ptr);  // Free the array of pointers
}

void free_string_memory(char *ptr) {
  free(ptr);
}

int main() {
    int N;
    printf("Enter the maximum number of strings to input: ");
    scanf("%d", &N);
    getchar();


    char **strings = NULL;
    strings = allocate_array_memory(strings, N);

    int count = 0;
    char input[81];

    printf("Enter strings (type 'end' or 'END' to stop):\n");
    while (count < N) {
        fgets(input, sizeof(input), stdin);
        input[strcspn(input, "\n")] = '\0';

        if (strcmp(input, "end") == 0 || strcmp(input, "END") == 0) {
            break;
        }


        strings[count] = allocate_string_memory(input);
        count++;
    }


    display(strings, count);


    free_array_memory(strings, count);

    return 0;
}
```

Output:



```
user57@trainux01:~/Batch17OCT2024/if_else$ vi dma3.c
user57@trainux01:~/Batch17OCT2024/if_else$ gcc dma3.c
user57@trainux01:~/Batch17OCT2024/if_else$ ./a.out
Enter the maximum number of strings to input: 2
Enter strings (type 'end' or 'END' to stop):
DYNAMIC MEMORY
ALLOCATION

Stored Strings:
DYNAMIC MEMORY
ALLOCATION
```