Date: 19/07/2021

Exercise 8

Name: *Krithika Swaminathan*
Roll No.: *205001057*

1. Frequency of a Word

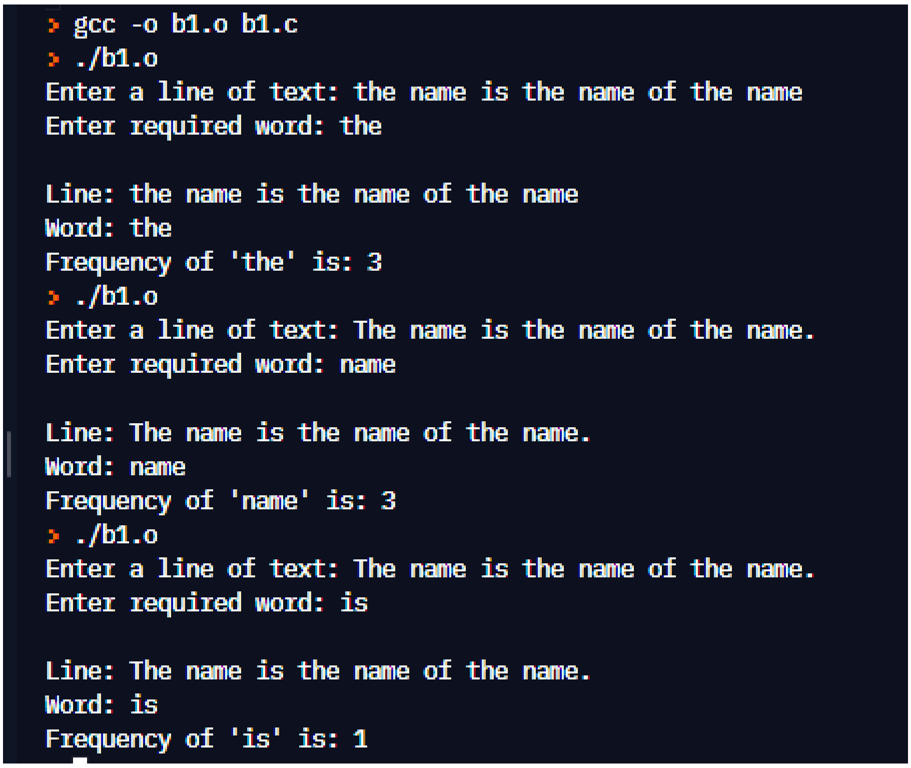
Aim:

To write a program in C that has a user-defined function to search for a given word in a line of text and return the frequency of that word, using pointer notation.

Code:

```
//to search for a given word in a line of text and return the frequency of the word using pointers
#include <stdio.h>
#include <string.h>

#define lim 100
#define wlim 12

int search(char *line, char *word){
  printf("\nLine: %s",line);
  printf("Word: %s\n",word);

  char temp[wlim]="";
  int freq=0;

  for (char *i=line; *(i-1)!='\n'; i++){
    if ((*i==' ' || *i=='.' || *i==',' || *i=='\n') && *(i+1)!=' '){
      if (strcmp(temp,word)==0){
        freq++;
      }
      strcpy(temp,"");
    }
    else{
      strncat(temp,i,1);
    }
  }

  return freq;
}

int main(){
  char line[lim];
  printf("Enter a line of text: ");
  fgets(line,lim,stdin);

  static char word[wlim];
  printf("Enter required word: ");
  scanf("%s",word);

  printf("Frequency of '%s' is: %d\n", word,search(line,word));
  return 0;
}
```

Output:



```
> gcc -o b1.o b1.c
> ./b1.o
Enter a line of text: the name is the name of the name
Enter required word: the

Line: the name is the name of the name
Word: the
Frequency of 'the' is: 3
> ./b1.o
Enter a line of text: The name is the name of the name.
Enter required word: name

Line: The name is the name of the name.
Word: name
Frequency of 'name' is: 3
> ./b1.o
Enter a line of text: The name is the name of the name.
Enter required word: is

Line: The name is the name of the name.
Word: is
Frequency of 'is' is: 1
```

Result:

A program for counting the frequency of a word in a given line of text is written and executed.

2. Parsing into Tokens

Aim:

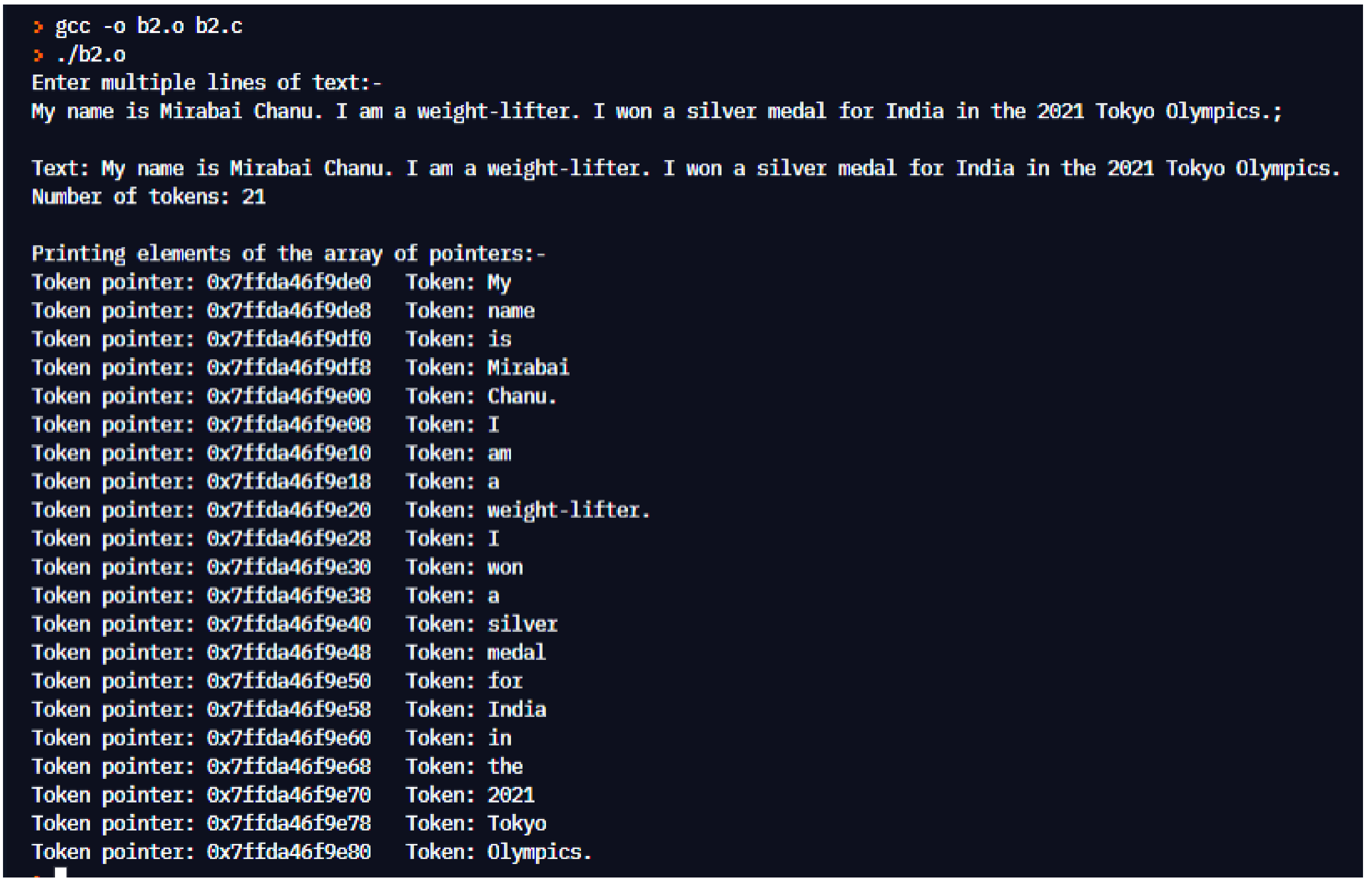
To write a program that parses multiple lines of text to get tokens (words separated by a whitespace) of unspecified maximum length, to store the tokens in a 1D array of pointers using dynamic memory allocation depending on the number of characters in each token.

Code:

```
//given multiple lines of text
//to parse the text into tokens with unspecified max length
//to store the tokens in a 1D array and represent with pointers
//to use dynamic memory allocation depending on number of characters in each token
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#define textlim 150
#define toklim 15

int main(){
  char text[textlim];
  printf("Enter multiple lines of text:-\n");
  scanf("%[^;]s",text);
  printf("\nText: %s\n",text);

  //to find the number of tokens in the text
  int ctr=0, i;
  for (i=0; i<strlen(text); i++){
    if (!(isalnum(text[i]) || text[i]=='\'' || text[i]=='"' || text[i]=='-' ||
text[i]=='\n' ||text[i]=='.')){
      ctr++;
    }
  }
  if (text[i-1]=='.') ctr++;
  printf("Number of tokens: %d\n",ctr);

  //to parse the text for tokens and store the pointers to these tokens in the 1D array
  char *tokens[ctr];
  char temp[toklim]=""; int n;

  int r=0;
  for (i=0; r<ctr; i++){
    if (isalnum(text[i]) || text[i]=='\'' || text[i]=='"' || text[i]=='-' || text[i]=='\n'
|| text[i]=='.'){
      strncat(temp,&text[i],1);
    }
```

```
      else{
        n=strlen(temp);
        tokens[r]=(char *)malloc(n*sizeof(char));
        strncpy(tokens[r],temp,n);
        r++;
        strcpy(temp,"");
      }
    }

    //to print the stored tokens
    printf("\nPrinting elements of the array of pointers:-\n");
    for (r=0; r<ctr; r++){
      printf("Token pointer: %p\t",tokens+r);
      printf("Token: %s\n",*(tokens+r));
    }

    free(tokens);

    return 0;
}
```

Output:



```
> gcc -o b2.o b2.c
> ./b2.o
Enter multiple lines of text:-
My name is Mirabai Chanu. I am a weight-lifter. I won a silver medal for India in the 2021 Tokyo Olympics.;

Text: My name is Mirabai Chanu. I am a weight-lifter. I won a silver medal for India in the 2021 Tokyo Olympics.
Number of tokens: 21

Printing elements of the array of pointers:-
Token pointer: 0x7ffda46f9de0   Token: My
Token pointer: 0x7ffda46f9de8   Token: name
Token pointer: 0x7ffda46f9df0   Token: is
Token pointer: 0x7ffda46f9df8   Token: Mirabai
Token pointer: 0x7ffda46f9e00   Token: Chanu.
Token pointer: 0x7ffda46f9e08   Token: I
Token pointer: 0x7ffda46f9e10   Token: am
Token pointer: 0x7ffda46f9e18   Token: a
Token pointer: 0x7ffda46f9e20   Token: weight-lifter.
Token pointer: 0x7ffda46f9e28   Token: I
Token pointer: 0x7ffda46f9e30   Token: won
Token pointer: 0x7ffda46f9e38   Token: a
Token pointer: 0x7ffda46f9e40   Token: silver
Token pointer: 0x7ffda46f9e48   Token: medal
Token pointer: 0x7ffda46f9e50   Token: for
Token pointer: 0x7ffda46f9e58   Token: India
Token pointer: 0x7ffda46f9e60   Token: in
Token pointer: 0x7ffda46f9e68   Token: the
Token pointer: 0x7ffda46f9e70   Token: 2021
Token pointer: 0x7ffda46f9e78   Token: Tokyo
Token pointer: 0x7ffda46f9e80   Token: Olympics.
```

Exercise 8

Result:

A program for parsing the text and storing the tokens as an array of pointers is written and executed.

3. Building Tables

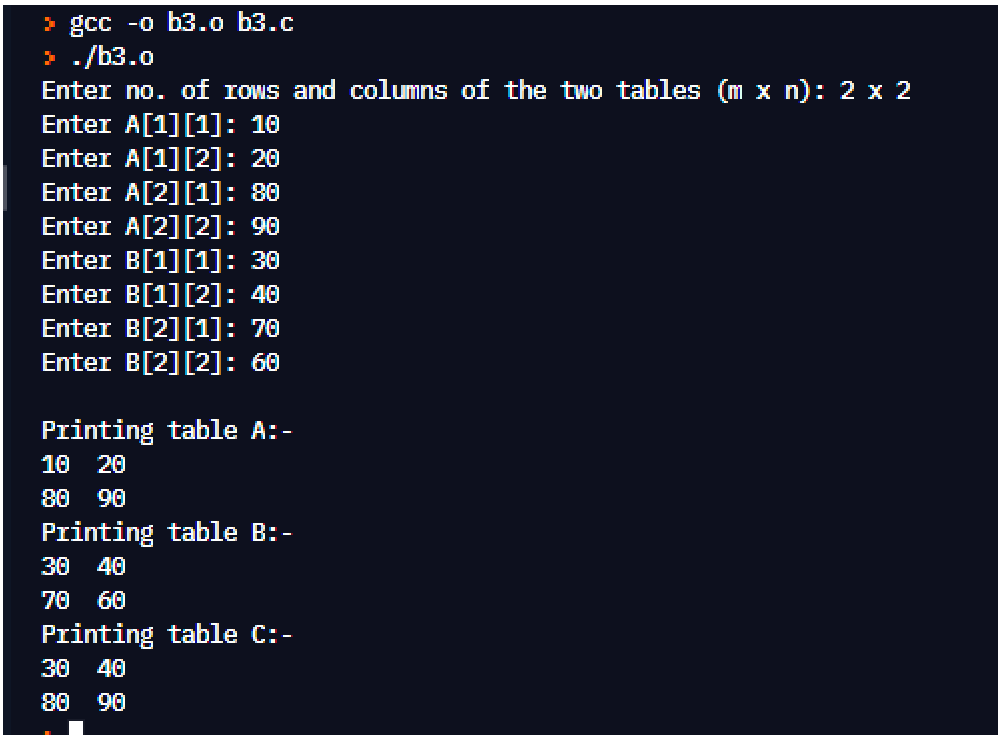
Aim:

To write a program to build two tables and get their elements as input from the user, then build a third table with the respective maximum of elements of the first two tables.

Code:

```
//to build two tables A and B; to build C with each element being corresponding maximum of A and B
//represent each table as a pointer, use dynamic memory allocation
#include <stdio.h>
#include <stdlib.h>

void print(int *table[], int r, int c){
  for (int i=0; i<r; i++){
    for (int j=0; j<c; j++){
      printf("%d\t", *(*(table+i)+j));
    }
    printf("\n");
  }
}

int main(){
  int m,n;
  printf("Enter no. of rows and columns of the two tables (m x n): ");
  scanf("%d x %d", &m,&n);

  int **A=(int**)malloc(m*sizeof(int*));
  int **B=(int**)malloc(m*sizeof(int*));
  int **C=(int**)malloc(m*sizeof(int*));

  //getting inputs for tables A and B
  for (int i=0; i<m; i++){
    A[i]=(int*)malloc(n*sizeof(int));
    for (int j=0; j<n; j++){
      printf("Enter A[%d][%d]: ",i+1,j+1);
      scanf("%d",*(A+i)+j);
    }
  }
  for (int i=0; i<m; i++){
    B[i]=(int*)malloc(n*sizeof(int));
    for (int j=0; j<n; j++){
      printf("Enter B[%d][%d]: ",i+1,j+1);
      scanf("%d",*(B+i)+j);
    }
  }
  //assigning the maximum in tables A and B to table C
  for (int i=0; i<m; i++){
    C[i]=(int*)malloc(n*sizeof(int));
    for (int j=0; j<n; j++){
      if ( *(*(A+i)+j) > *(*(B+i)+j) )
        *(*(C+i)+j)=*(*(A+i)+j);
      else
        *(*(C+i)+j)=*(*(B+i)+j);
      //printing the stored table C elements
      //printf("C[%d][%d]: %d\t", i+1,j+1,*(*(C+i)+j));
    }
  }

  printf("\nPrinting table A:- \n");
  print(A,m,n);

  printf("Printing table B:- \n");
  print(B,m,n);

  printf("Printing table C:- \n");
  print(C,m,n);

  return 0;
}
```

Output:



```
› gcc -o b3.o b3.c
› ./b3.o
Enter no. of rows and columns of the two tables (m x n): 2 x 2
Enter A[1][1]: 10
Enter A[1][2]: 20
Enter A[2][1]: 80
Enter A[2][2]: 90
Enter B[1][1]: 30
Enter B[1][2]: 40
Enter B[2][1]: 70
Enter B[2][2]: 60

Printing table A:-
10  20
80  90
Printing table B:-
30  40
70  60
Printing table C:-
30  40
80  90
```

Exercise 8

Result:

A program for building the tables as required is written and executed.

4. Matrix Multiplication

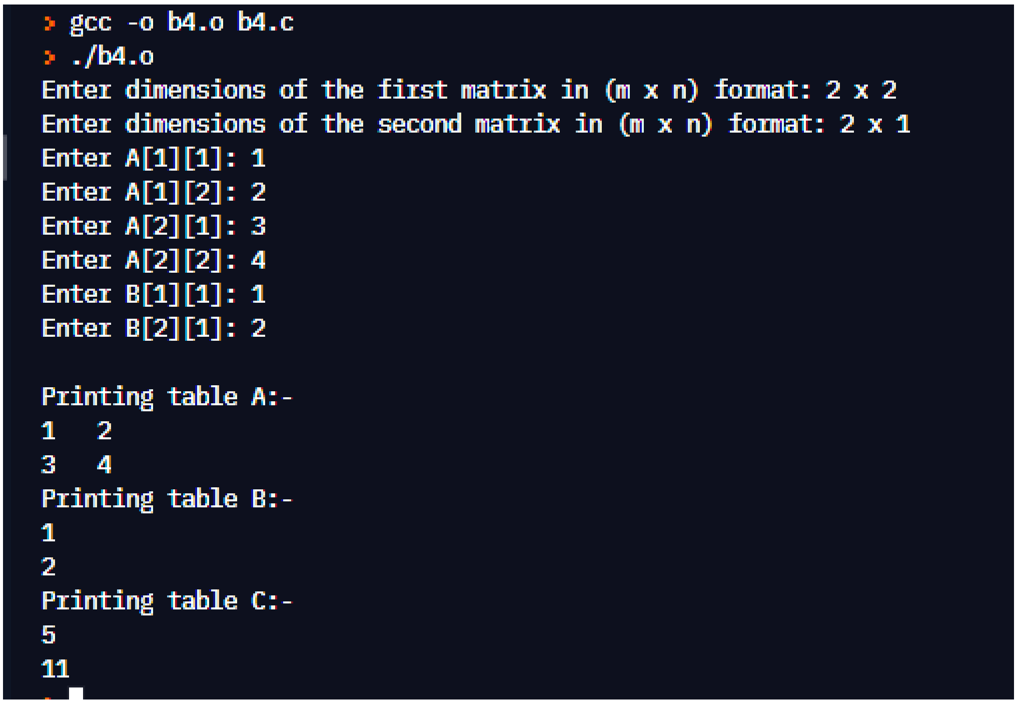
Aim:

To write a user-defined function that performs matrix multiplication using pointers and to write a program to test the above function.

Code:

```c
//to perform matrix multiplication using pointers
//to define func Multiply with parameters being two matrices, return resultant matrix
//to test the function using function pointer
//to accept the input matrices in main
#include <stdio.h>
#include <stdlib.h>

void print(int *mat[], int r, int c){
  for (int i=0; i<r; i++){
    for (int j=0; j<c; j++){
      printf("%d\t", *(*(mat+i)+j));
    }
    printf("\n");
  }
}

int **multiply (int m1, int n1, int m2, int n2, int *a[], int *b[], int *c[]){
  for (int i=0; i<m1; i++){
    for (int j=0; j<n2; j++){
      c[i][j]=0;
      for (int k=0; k<n1; k++){
        c[i][j]+=(a[i][k]*b[k][j]);
      }
    }
  }
  return c;
}

int main(){
  int m1,n1,m2,n2;
  printf("Enter dimensions of the first matrix in (m x n) format: ");
  scanf("%d x %d", &m1,&n1);
  printf("Enter dimensions of the second matrix in (m x n) format: ");
  scanf("%d x %d", &m2,&n2);

  if (n1!=m2){
    printf("Matrix multiplication not possible. Exiting...\n");
    exit(0);
  }
  int **A=(int**)malloc(m1*sizeof(int*));
  //getting inputs for first matrix
  for (int i=0; i<m1; i++){
    A[i]=(int*)malloc(n1*sizeof(int));
    if (A[i]==NULL){
      printf("Error! Memory not allocated. Exiting...\n");
      exit(0);
    }
    for (int j=0; j<n1; j++){
      printf("Enter A[%d][%d]: ",i+1,j+1);
      scanf("%d",*(A+i)+j);
    }
  }

  int **B=(int**)malloc(m2*sizeof(int*));
  //getting inputs for first matrix
  for (int i=0; i<m2; i++){
    B[i]=(int*)malloc(n2*sizeof(int));
    if (B[i]==NULL){
      printf("Error! Memory not allocated. Exiting...\n");
      exit(0);
    }
    for (int j=0; j<n2; j++){
      printf("Enter B[%d][%d]: ",i+1,j+1);
      scanf("%d",*(B+i)+j);
    }
  }

  //allocating data for the product matrix
  int **C=(int**)malloc(m1*sizeof(int*));
  for (int i=0; i<m1; i++){
    C[i]=(int*)malloc(n2*sizeof(int));
    if (C[i]==NULL){
      printf("Error! Memory not allocated. Exiting...\n");
      exit(0);
    }
  }

  //calling the matrix multiplication function and returning the resultant matrix
  C=multiply(m1,n1,m2,n2,A,B,C);

  //printing the matrices
  printf("\nPrinting table A:- \n");
  print(A,m1,n1);

  printf("Printing table B:- \n");
  print(B,m2,n2);

  printf("Printing table C:- \n");
  print(C,m1,n2);
  return 0;
}
```

Output:



```
› gcc -o b4.o b4.c
› ./b4.o
Enter dimensions of the first matrix in (m x n) format: 2 x 2
Enter dimensions of the second matrix in (m x n) format: 2 x 1
Enter A[1][1]: 1
Enter A[1][2]: 2
Enter A[2][1]: 3
Enter A[2][2]: 4
Enter B[1][1]: 1
Enter B[2][1]: 2

Printing table A:-
1   2
3   4
Printing table B:-
1
2
Printing table C:-
5
11
```

Result:

A program to find the product of two given matrices is written and executed.