2024

SANTHIRAM
ENGINEERING
COLLEGE

C.IRFAN HUSSAIN

College Vision:

To become a nucleus for pursuing technical education and pool industrial research and developmental activities with social-conscious and global standards.

Mission

5. To provide Advanced Educational Programs and prepare students to achieve success and take leading roles in their chosen fields of specialization by arising a self-sustained University.
6. To establish postgraduate programs in the current and Advanced Technologies.
7. To establish an R&D Consultancy through developing Industry Institute Interaction, building up exceptional infrastructure.
8. To propel every individual, realize and act for the technical development of the society.

Department Vision

To become a center for quality education in the field of computer science & engineering and to create competent professionals.

Department Mission

5. To provide academic ambience and latest software tools to prepare competent Software Engineers with strong theoretical and practical knowledge.
6. To foster professionalism and strong work ethics in students for the betterment of society.
7. To uplift innovative research in Computer Science and Engineering to serve the needs of Industry, Government and Society
8. To encourage the spirit of entrepreneurship and adaptability in our students in view of the ever-changing scenario of the Software Industry.

1. “Hello World” in C

Objective

In this challenge, we will learn some basic concepts of C that will get you started with the language. You will need to use the same syntax to read input and write output in many C challenges. As you work through these problems, review the code stubs to learn about reading from stdin and writing to stdout. Task This challenge requires you to print Hello,World!

on a single line, and then
print the already provided input string to stdout. If you are not familiar with C, you may want to read about the printf() command.

Example

s =”Life is beautiful”
The required output is:
Hello, World!
Life is beautiful

Function Description

Complete the main() function below.

The main() function has the following input:

● string s: a string

Prints

- *two strings: * "Hello, World!" on one line and the input string on the next line.

Input Format

There is one line of text,s .

Sample Input 0

Welcome to C programming.
Sample Output 0

Hello, World!

Welcome to C programming.

Program:

```
#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>


  int main()

{
   char s[100];
   scanf("%[^\n]%*c",

   &s); printf("Hello,

   World!\n"); printf(s);

   return 0;

 }
```

out put:

Compiler

Message
Success

Input (stdin)

Download

```
Welcome to C programming.
```

Expected
Output
Download

Hello, World!

Welcome to C programming.

2. Playing with

Characters Objective This challenge will help you to learn how to take a character, a string and a

sentence as input in C.

To take a single character ch as input, you can use scanf("%c", &ch

); and printf("%c", ch) writes a character specified by the argument char to stdout

```
char ch; scanf("%c", &ch); printf("%c", ch);
```

This piece of code prints the character ch .

You can take a string as input in C using scanf(“%s”, s). But, it accepts string only until it finds the first space. In order to take a line as input, you can use scanf("%[^\n]%*c", s); where is defined as char s[MAX_LEN] where MAX_LEN is the maximum size of . Here, [] is the scanset character. ^\n stands for taking input until a newline isn't encountered. Then, with this %*c, it reads the newline character and here, the used * indicates that this newline character is discarded. Note: The statement: scanf("%[^\n]%*c", s); will not work because the last statement will read a newline character, \n, from the previous line. This can be handled in a variety of ways. One way is to use scanf("\n"); before the last statement.

Task

You have to print the character, ch, in the first line. Then print in next line. In the last line print the sentence,sen .

Input Format

First, take a character, ch as
input. Then take the string, s as
input.
Lastly, take the sentence sen as input.

Constraints

Strings for s and sen will have fewer than 100 characters, including the
newline.

Output Format

Print three lines of output. The first line prints the character,
ch. The second line prints the string,s .
The third line prints the sentence,sen .

Sample Input 0

```
C
Language
Welcome To C!!
```

Sample Output
0

```
C
Language
Welcome To C!!
```

Program:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
int main()
{
    char ch;
    char s[24];
    char t[100];


    scanf("%c", &ch);

    scanf("%s", s);
    getchar();
    scanf("%[^\n]%*c",
    t);


    printf("%c\n", ch);

    printf("%s\n", s);

    printf("%s\n", t);



    return 0;

}
```

out put:

Compiler Message

Success

Input (stdin)

Download

C

Language

Welcome To

C!!

Expected Output

Download

C

Language

Welcome To

C!!

3. sum and difference of two numbers

Objective

The fundamental data types in c are int, float and char. Today, we're discussing int and float data types.

The printf() function prints the given statement to the console. The syntax is printf("format string",argument_list);. In the function, if we are using an integer, character, string or float as argument, then in the format string we have to write %d (integer), %c (character), %s (string), %f (float) respectively.
The scanf() function reads the input data from the console. The syntax is scanf("format string",argument_list);. For ex: The scanf("%d",&number)

statement reads integer number from the console and stores the given value in variable .

To input two integers separated by a space on a single line, the command is scanf("%d %d", &n, &m), where n and m are the two integers.

Task

Your task is to take two numbers of int data type, two numbers of float data type as input and output their sum:

4. Declare 4 variables: two of type int and two of type float.
5. Read 2 lines of input from stdin (according to the sequence given in the 'Input Format' section below) and initialize your 4 variables.
6. Use the + and - operator to perform the following operations:
   - Print the sum and difference of two int variable on a new line.
   - Print the sum and difference of two float variable rounded to one decimal place on a new line.

Input Format

The first line contains two integers.

The second line contains two floating point numbers.

Constraint

- integer variables
- float variables

Output Format

Print the sum and difference of both integers separated by a space on the first line, and the sum and difference of both float (scaled to 1 decimal place) separated by a space on the second line.

Sample Input

```
10 4
4.0 2.0
```

Sample Output

```
14 6
6.0 2.0
```

Explanation

When we sum the integers 10 and 4 , we get the integer 14 . When we subtract the second number 4 from the first number 10 , we get 6 as their difference. When we sum the floating-point numbers 4.0 and 2.0 , we get 6.0 . When we subtract the second number 2.0 from the first number 4.0 , we get 2.0 as their difference.

program:

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

int main()
{
    int a,b;
    float
    c,d;
    scanf("%d %d",&a,&b);
    scanf("%f %f",&c
    ,&d); int int_sum =

    a+b;
    int int_diff = a-b;
    float float_sum = c+d;
    float float_diff = c-d;
    printf("%d %d\n",int_sum,int_diff);
    printf("%0.1f %0.1f", float_sum,float_diff);
    return 0;
}
```

out put:

Compiler Message

Success

Input (stdin)

Download

```
10 4

4.0 2.0
```

Expected Output

Download

```
14 6

6.0 2.0
```

4. functions in c:

Objective

In this challenge, you will learn simple usage of functions in C. Functions are a bunch of statements grouped together. A function is provided with zero or more arguments, and it executes the statements on it. Based on the return type, it either returns nothing (void) or something.

A sample syntax for a function is

```
return_type function_name(arg_type_1 arg_1, arg_type_2 arg_2, ...) {

...
```

```
        ...

        ...

        [if return_type is non void]

         return something of type `return_type`;

    }
```

For example, a function to read four variables and return the sum of them can be written as

```
         int sum_of_four(int a, int b, int c,
         int d) { int sum = 0;
        sum  +=  a;
        sum  +=  b;
        sum  +=  c;
        sum += d;
        return
        sum;
    }
```

+= : Add and assignment operator. It adds the right operand to the left operand and assigns the result to the left operand.

a += b is equivalent to a = a + b;

Task

Write a function int max_of_four(int a, int b, int c, int d) which reads four arguments and returns the greatest of them.

Note

There is not built in max function in C. Code that will be reused is often put in a separate function, e.g. int max(x, y) that returns the greater of the two values.

Input Format

Input will contain four integers - , one on each line.
Output Format

Print the greatest of the four integers.
Note: I/O will be automatically handled.

Sample Input

3

4

6

5

Sample
Output 6

program:

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

```
int max_of_four(int a, int b, int c, int d) {
    int max = a;
    if (b > max) {
        max = b;
    }
    if (c > max) {

        max = c;
    }
    if (d > max) {

        max = d;
    }

    return max;

}



int main() {

    int a, b, c, d; scanf("%d %d %d %d", &a,

    &b, &c,
    &d); int ans = max_of_four(a, b, c, d);
    printf("%d", ans);


    return 0;

}
```

out put:

Compiler Message
Success

Input (stdin)

```
3
4

6

5
```

Expected
Output 6

5. pointrs in c:

Objective

In this challenge, you will learn to implement the basic functionalities of pointers in C. A pointer in C is a way to share a memory address among different contexts (primarily functions). They are primarily used whenever a function needs to modify the content of a variable that it does not own.

In order to access the memory address of a variable, , prepend it with sign. For example, &val returns the memory address of .

This memory address is assigned to a pointer and can be shared among various functions. For example, will assign the memory address of to pointer . To access the content of the memory to which the pointer points, prepend it with a *. For example, *p will return the value reflected by and any modification to it will be reflected at the source ().

```
    void       increment(int
   *v) { (*v)++;
}

    int main()
   {i nt a;
   scanf("%d",
   &a);
   increment(&a);
   printf("%d", a);
   return 0;
}
```

Task

Complete the function void update(int *a,int *b). It receives two integer pointers, int* a and int* b. Set the value of to their sum, and to their absolute difference. There is no return value, and no return statement is needed.

-
-

Input Format

The input will contain two integers, and , separated by a
newline. Output Format

Modify the two values in place and the code stub main() will print their values.

Note: Input/ouput will be automatically handled. You only have to
complete the function described in the 'task' section.

Sample Input

4

5

Sample
Output 9

1

program:

```
#include <stdio.h>

void update(int *a,int *b) {

        // Complete this function int
          sum,diff;
          sum = *a+*b;
          diff = abs(*a-*b);
             *a = sum;
        *b = diff;

}



int main() {

    int a, b;

    int *pa = &a, *pb = &b;



    scanf("%d %d", &a, &b);
    update(pa, pb);
    printf("%d\n%d", a, b);




    return 0;

}
```

out put:

Compiler Message

Success

Input (stdin)

4

5

Expected Output

9

1

6. conditional statements in c:

Objective

if and else are two of the most frequently used conditionals in C/C++, and they enable you to execute zero or one conditional statement among many such dependent conditional statements. We use them in the following ways: if: This executes the body of bracketed code starting with if evaluates to true.

```
if (condition) {

statement1;

 ...

}
```

4.

if - else: This executes the body of bracketed code starting with if evaluates to true, or it executes the body of code starting with if evaluates to false. Note that only one of the bracketed code sections will ever be executed.

```
if (condition) {

 statement1;
 ...
}
else {
    statement2;
    ...
}
```

5.

if - else if - else: In this structure, dependent statements are chained together and the for each statement is only checked if all prior conditions in the chain are evaluated to false. Once a evaluates to true, the bracketed code associated with that statement is executed and the program then skips to the end of the chain of statements and continues executing. If each in the chain evaluates to false, then the body of bracketed code in the else block at the end is executed. if(first condition) {

```
    ...
}
else if(second condition) {
    ...
}
.
.
.
else if((n-1)'th condition) {
    ....
```

```
}
else {
    ...
}
```

6.

Task

Given a positive integer denoting , do the following:

- If , print the lowercase English word corresponding to the
- number (e.g., one for , two for , etc.).
  If , print Greater than 9.

Input Format

The first line contains a single integer, .

Constraints

-

Output Format

If , then print the lowercase English word corresponding to the number (e.g., one for , two for , etc.); otherwise, print Greater than 9 instead.

Sample

Input 5

Sample Output

five

Sample Input #01

8

Sample Output #01

eight

Sample Input #02

44

Sample Output #02

Greater than 9

program:

```
#include   <assert.h>
#include    <limits.h>
#include    <math.h>
#include  <stdbool.h>
#include <stddef.h>
```

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


char* readline();



int main()

{


        char* n_endptr;

        char* n_str = readline();

        int n = strtol(n_str, &n_endptr, 10);

        if (n_endptr == n_str || *n_endptr != '\0') { exit(EXIT_FAILURE);

  }

       // Write Your Code Here
       if(n>=1 && n<=9)


     {
        if(n==1)
        printf("one");
        if(n==2)
        printf("two");
        if(n==3)

        printf("three");
        if(n==4)
        printf("four");
```

```
            if(n==5)
            printf("five");
            if(n==6)
            printf("six");
            if(n==7)
            printf("seven");
            if(n==8)
            printf("eight");
            if(n==9)
            printf("nine");

        }


        else

            printf("Greater than 9");




        return 0;

}

  char* readline() {

  size_t alloc_length = 1024; size_t
  data_length = 0;
  char* data = malloc(alloc_length);

     while (true) {
  char* cursor = data + data_length;
  char* line = fgets(cursor, alloc_length - data_length,
 stdin);
```

```
            if (!line) { break; }

            data_length += strlen(cursor);

              if (data_length < alloc_length - 1 || data[data_length - 1]
== '\n') { break; }

            size_t new_length = alloc_length << 1; data =
            realloc(data, new_length);

            if (!data) { break; }

            alloc_length = new_length;
      }

            if (data[data_length - 1] == '\n') {
                  data[data_length - 1] = '\0';
      }

      data = realloc(data, data_length);
```

```
    return data;
}
```

out put:

Compiler Message

Success

Input
(stdin)
Downloa
d

Expected
Output
Download

7. For loop in c:

Objective

In this challenge, you will learn the usage of the for loop, which is a programming language statement which allows code to be executed until a terminal condition is met. They can even repeat forever if the terminal condition is never met.
The syntax for the for loop is:

for (<expression_1> ; <expression_2> ; <expression_3>)

<statement>

- expression_1 is used for intializing variables which are generally used for controlling the terminating flag for the loop.

- expression_2 is used to check for the terminating condition. If this evaluates to false, then the loop is terminated.
- expression_3 is generally used to update the flags/variables.

The following loop initializes to 0, tests that is less than 10, and increments at every iteration. It will execute 10 times.

```
for(int i = 0; i < 10; i++) {


  ...

}
```

Task

For each integer in the interval (given as input) :

- If , then print the English representation of it in lowercase. That is "one" for , "two" for , and so on.
- Else if and it is an even number, then print "even".
- Else if and it is an odd number, then print

"odd". Input Format

The first line contains an integer, .

The seond line contains an integer, .

Constraints

Output

For m at

Print the appropriate English representation,even, or odd, based on the conditions described in the 'task' section.

Note:

Sample Input

```
8

11
```

Sample Output

```
eight
nine
even
odd
```

program:

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


int main() {

    int a, b; scanf("%d\n%d",
    &a, &b);


    for (int i = a; i <= b; i++) { if (i == 1) {
                printf("one\n");

                 } else if (i == 2) {
                   printf("two\n");
```

```
        } else if (i == 3) {
            printf("three\n");
        } else if (i == 4) {
            printf("four\n");
        } else if (i == 5) {
            printf("five\n");
        } else if (i == 6) {
            printf("six\n");
        } else if (i == 7) {
            printf("seven\n");
        } else if (i == 8) {
            printf("eight\n");
        } else if (i == 9) {
            printf("nine\n");
        } else if (i % 2 == 0) {
            printf("even\n");
        } else {

            printf("odd\n");


        }

    }



    return 0;

}
```

out put:

Compiler Message

Success

Input (stdin)

Download

```
8
11
```

Expected Output

Download

```
eigh
t
nin
e
eve
n
odd
```

8. Sum of digits of a five digit number:

Objective

The modulo operator, %, returns the remainder of a division. For example, 4 % 3 = 1 and 12 % 10 = 2. The ordinary division operator, /, returns a truncated integer value when performed on integers. For example, 5 / 3 = 1. To get the last digit of a number in base 10, use as the modulo divisor.

Task

Given a five digit integer, print the sum of its digits.

Input Format

The input contains a single five digit number, .

Constraints

Output Format

Print the sum of the digits of the five digit number.

Sample Input 0

10564

Sample Output 0

16

program:

```
#include
<stdio.h>

#include
<string.h>
#include
<math.h>
#include
<stdlib.h>


int main() {

int num, sum = 0;
```

```
    scanf("%d", &num);
    while(num != 0) {
        sum += num %
        10; num /= 10;
    }

    printf("%d", sum);

}
```

out put:

Compiler Message

Success

Input (stdin)

Download

10564

Expected Output

Download

16

9. Bitwise Operators:

In this challenge, you will use logical bitwise operators. All data is stored in its binary representation. The logical operators, and C language, use to represent true and to represent false. The logical operators compare bits in two numbers and return true or false, or , for each bit compared.

- Bitwise AND operator & The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0. It is denoted by &.
- Bitwise OR operator | The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. It is denoted by |.
- Bitwise XOR (exclusive OR) operator ^ The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by .

For example, for integers 3 and 5,
3 = 00000011 (In Binary) 5 = 00000101 (In Binary)

```
AND operation        OR operation        XOR operation
  00000011           00000011            00000011

& 00000101           | 00000101          ^ 00000101
  ________          ________           ________

  00000001 = 1         00000111 = 7        00000110 = 6
```

You will be given an integer , and a threshold, i1nnik$. Print the results of the and, or and exclusive or comparisons on separate lines, in that order.

Example

The results of the comparisons are below:

a	b	and	or	xor
1	2	0	3	3
1	3	1	3	2
2	3	2	3	1

For the and comparison, the maximum is . For the or comparison, none of the values is less than , so the maximum is . For the xor comparison, the maximum value less than is . The function should print: 2

0

2

Function Description

Complete the calculate_the_maximum function in the editor below. calculate_the_maximum has the following parameters:

- int n: the highest number to consider
- int k: the result of a comparison must be lower than this number to be considered

Prints

Print the maximum values for the and, or and xor comparisons, each on a separate line.

Input Format

The only line contains space-separated integers, and .

Constraints

-
-

Sample Input 0

5 4

Sample Output 0

2

3

3

Explanation 0

All possible values of and are:

10.
11.
12.
13.
14.
15.
16.

17.

18.

- The maximum possible value of that is also is , so we print on first line.
- The maximum possible value of that is also is , so we print on second line.
- The maximum possible value of that is also is , so we print on third line.

program:

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


    void calculate_the_maximum(int n, int k) {
  int max_and = 0, max_or = 0, max_xor = 0;
 for (int i = 1; i <= n; i++) {

     for (int j = i + 1; j <= n; j++) {
       int temp_and = i & j;
       int temp_or = i | j;
       int temp_xor = i ^ j;
       if (temp_and > max_and && temp_and < k) {
          max_and = temp_and;
```

```
            }

            if (temp_or > max_or && temp_or < k) {
                max_or = temp_or;
            }

            if (temp_xor > max_xor && temp_xor < k) {
                max_xor = temp_xor;
            }


        }


    }

    printf("%d\n%d\n%d", max_and, max_or, max_xor);


}



int main()
    { int n,
    k;


    scanf("%d %d", &n, &k);
    calculate_the_maximum(n, k);


    return 0;

}
```

out put:

Compiler Message

Success

Input (stdin)

Download

```
5 4
```

Expected Output

Download

```
2
3
3
```

10. Printing Pattern Using Loops:

Print a pattern of numbers from to as shown below. Each of the numbers is separated by a single space.

```
4 4 4 4 4 4 4
4 3 3 3 3 3 4
4 3 2 2 2 3 4
4 3 2 1 2 3 4
4 3 2 2 2 3 4
4 3 3 3 3 3 4
4 4 4 4 4 4 4
```

Input Format

The input will contain a single integer .

Constraints

Sample Input 0

```
2
```

Sample Output 0

```
2 2 2

2 1 2

2 2 2
```

Sample Input 1

```
5
```

Sample Output 1

```
5 5 5 5 5 5 5 5 5

5 4 4 4 4 4 4 4 5

5 4 3 3 3 3 3 4 5

5 4 3 2 2 2 3 4 5

5 4 3 2 1 2 3 4 5

5 4 3 2 2 2 3 4 5
```

```
5 4 3 3 3 3 3 4 5

5 4 4 4 4 4 4 4 5

5 5 5 5 5 5 5 5 5
```

Sample Input 2

```
7
```

Sample Output 2

```
7 7 7 7 7 7 7 7 7 7 7 7 7

7 6 6 6 6 6 6 6 6 6 6 6 7

7 6 5 5 5 5 5 5 5 5 5 6 7

7 6 5 4 4 4 4 4 4 4 5 6 7

7 6 5 4 3 3 3 3 3 4 5 6 7

7 6 5 4 3 2 2 2 3 4 5 6 7

7 6 5 4 3 2 1 2 3 4 5 6 7

7 6 5 4 3 2 2 2 3 4 5 6 7

7 6 5 4 3 3 3 3 3 4 5 6 7

7 6 5 4 4 4 4 4 4 4 5 6 7

7 6 5 5 5 5 5 5 5 5 5 6 7

7 6 6 6 6 6 6 6 6 6 6 6 7

7 7 7 7 7 7 7 7 7 7 7 7 7
```

program:

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


int main()
  { int n;
  scanf("%d", &n);


  int len = 2*n - 1;
  for (int i = 0; i < len; i++) {
      for (int j = 0; j < len; j++) {
          int min = i < j ? i : j;
          min = min < len-i ? min : len-i-1;
          min = min < len-j-1 ? min : len-j-1;
          printf("%d ", n-min);
      }
      printf("\n");
  }
```

```
    return 0;

}
```

out put :

Compiler
Message
Success

Input (stdin)
Download

```
        2
```

Expected
Output
Download

```
2 2 2
2 1 2 2 2
2
```

11. 1D Array In C:

An array is a container object that holds a fixed number of values of a single type. To create an array in C, we can do int arr[n];. Here, arr, is a variable array which holds up to integers. The above array is a static array that has memory allocated at compile time. A dynamic array can be created in C, using the malloc function and the memory is allocated on the heap at runtime. To create an integer array, of size , int *arr = (int*)malloc(n * sizeof(int)), where points to the base address of the array. When you have finished with the array, use free(arr) to deallocate the memory.

In this challenge, create an array of size dynamically, and read the values from stdin. Iterate the array calculating the sum of all elements. Print the sum

and free the memory where the array is stored.

While it is true that you can sum the elements as they are read, without first storing them to an array, but you will not get the experience working with an array. Efficiency will be required later. Input Format

The first line contains an integer, .

The next line contains space-separated integers.

Constraints

Output Format

Print the sum of the integers in the array.

Sample Input 0

```
6

16 13 7 2 1 12
```

Sample Output 0

51

Sample Input 1

7

1 13 15 20 12 13 2

Sample Output 1

76

program:

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


int main() {
/* Enter your code here. Read input from STDIN. Print output to
STDOUT */ int n;
```

```
    scanf("%d", &n);


    int arr[n];

        for (int i = 0; i < n; i++) {
            scanf("%d", &arr[i]);
    }

    int sum = 0;

    for (int i = 0; i < n; i++) {
        sum += arr[i];
    }

    printf("%d",
    sum); return 0;
}
```

out put :

Compiler Message

Success

Input (stdin)

Download

```
        6
```

```
16 13 7 2 1 12
```

Expected Output

Download

```
51
```

12. Array Reversal:

Given an array, of size , reverse it.

Example: If array, , after reversing it, the array should be,

. Input Format

The first line contains an integer, , denoting the size of the array. The next line contains space-separated integers denoting the elements of the array.
Constraints

, where is the element of the array.

Output Format

The output is handled by the code given in the editor, which would print the array.
Sample Input 0

```
6

16 13 7 2 1 12
```

Sample Output 0

12 1 2 7 13 16

Explanation 0

Given array, = . After reversing the array, =

Sample Input 1
7

1 13 15 20 12 13 2

Sample Output 1

2 13 12 20 15 13 1

Sample Input 2

8

15 5 16 15 17 11 5 11

Sample Output 2

11 5 11 17 15 16 5 15

program :

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


int main()

{

    int num, *arr, i;
    scanf("%d", &num);
    arr = (int*) malloc(num * sizeof(int));
    for(i = 0; i < num; i++) {
        scanf("%d", arr + i);

    }



    /* Write the logic to reverse the
    array. */ int temp;
    for(i = 0; i < num/2; i++) {

        temp = arr[i];

        arr[i] = arr[num-1-i];
        arr[num-1-i] = temp;
    }



    for(i = 0; i < num; i++)
```

```
        printf("%d ", *(arr + i));
    return 0;
}
```

out put :

Compiler Message

Success

Input (stdin)

Download

```
6
16 13 7 2 1 12
```

Expected Output

Download

```
12 1 2 7 13 16
```

13. Printing Tokens:

Given a sentence, , print each word of the sentence in a new line.

Input Format

The first and only line contains a sentence, .

Constraints

Output Format

Print each word of the sentence in a new

line. Sample Input 0
This is C

Sample Output 0

This is

C

Explanation 0

In the given string, there are three words ["This", "is", "C"]. We have to

print each of these words in a new line.
Sample Input 1

Learning C is

fun

Sample Output 1

Learning

C

is

fun

Sample Input

2 How is that

Sample Output 2

How

is

that

program:

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


int main() {


    char *s;

    s = malloc(1024 * sizeof(char));
    scanf("%[^\n]", s);
    s = realloc(s, strlen(s) + 1);

    //Write your logic to print the tokens of the sentence
    here. for (char *c = s; *c != NULL; c++) {
    if (*c == ' ') {

        *c = '\n';

    }

}

  printf("%s", s);
      return 0;
}
```

out put:

Compiler Message

Success

Input (stdin)

Download

```
This is C
```

Expected Output

Download

```
Thi
s is
```

C{-truncated-}

14. Digit Frequency:

Given a string, , consisting of alphabets and digits, find the frequency of each digit in the given string.

Input Format

The first line contains a string, which is the given number.

Constraints

All the elements of num are made of english alphabets and digits.

Output Format

Print ten space-separated integers in a single line denoting the frequency of each digit from to .

Sample Input 0

a11472o5t6

Sample Output 0

0 2 1 0 1 1 1 1 0 0

Explanation 0

In the given string:

- occurs two times.
- and occur one time each.
- The remaining digits and don't occur at all.

Sample Input 1
lw4n88j12n1

Sample Output 1

0 2 1 0 1 0 0 0 2 0

Sample Input 2
1v88886l256338ar0ekk

Sample Output 2

1 1 1 2 0 1 2 0 5 0

program:

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

```
#include <stdlib.h>


int main() {


    /* Enter your code here. Read input from STDIN. Print output to
    STDOUT */ char s[1000];
    int freq[10] = {0};


    scanf("%[^\n]", s);

    for (int i = 0; i < strlen(s); i++) {
        if (s[i] >= '0' && s[i] <= '9') {
            freq[s[i] - '0']++;

        }

    }


    for (int i = 0; i < 10; i++) {
        printf("%d ", freq[i]);
    }

    return 0;

}
```

out put:

Compiler Message

Success

Input (stdin)

```
a11472o5t
```

6 Expected

Output

```
0 2 1 0 1 1 1 1 0 0
```

15. Dynamic Array In C:

Snow Howler is the librarian at the central library of the city of HuskyLand.
He must handle requests which come in the following forms:

1 x y : Insert a book with pages at the end of the shelf. 2 x y : Print the number of pages in

the book on the shelf.
3 x : Print the number of books on the shelf.
Snow Howler has got an assistant, Oshie, provided by the Department of
Education. Although inexperienced, Oshie can handle all of the queries of types
2 and 3. Help Snow Howler deal with all the queries of type 1.

Oshie has used two arrays:

```
int* total_number_of_books;

/*

 * This stores the total number of books on each shelf.

 */



int** total_number_of_pages;

/*

 * This stores the total number of pages in each book of each shelf.

 * The rows represent the shelves and the columns represent the books.

 */
```

Input Format

The first line contains an integer , the number of shelves in the library. The second line contains an integer , the number of requests. Each of the following lines contains a request in one of the three specified formats.

Constraints

-
-
- For each query of the second type, it is guaranteed that a book is present on the shelf at index.
- Both the shelves and the books are numbered starting from 0.

- Maximum number of books per shelf .

Output Format

Write the logic for the requests of type 1. The logic for requests of types 2 and 3 are provided.

Sample Input 0

5

5

1 0 15

1 0 20

1 2 78

2 2 0

3 0

Sample Output 0

78

2

Explanation 0

There are shelves and requests, or queries.

- 1 Place a page book at the end of shelf . -

2 Place a page book at the end of shelf . - 3

Place a page book at the end of shelf .

- 4 The number of pages in the book on the shelf is 78.
- 5 The number of books on the shelf is 2.

program:

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

/*
 * This stores the total number of books in each shelf.
*/

int* total_number_of_books;

/*
 * This stores the total number of pages in each book of each shelf.
 * The rows represent the shelves and the columns represent the books.
*/
int** total_number_of_pages;
int main()
int total_number_of_shelves;
scanf("%d",
&total_number_of_shelves);
```

```
    int total_number_of_queries;
    scanf("%d",
    &total_number_of_queries);


    total_number_of_books = (int *)malloc(sizeof(int)*total_number_of_shelves);
    total_number_of_pages = (int **)malloc(sizeof(int *)*total_number_of_shelves);
    for(int i = 0; i<total_number_of_shelves; i++){
        *(total_number_of_books + i) = 0;

    }

    while(total_number_of_queries--){
        int type_of_query;
        scanf("%d", &type_of_query);
        if(type_of_query == 1){
            int x, y;

            scanf("%d %d", &x, &y);

            int booksInShelf = *(total_number_of_books + x);

            *(total_number_of_pages + x) =
(int*)realloc(*(total_number_of_pages+x),sizeof(int)*(booksInShelf+1));

            *(*(total_number_of_pages+x)+booksInShelf) = y;

            *(total_number_of_books + x) += 1;

        } else if (type_of_query == 2) {
            int x, y;
        scanf("%d %d", &x, &y);

        printf("%d\n", *(*(total_number_of_pages + x) + y));

    } else {

        int x;
        scanf("%d",
        &x);
        printf("%d\n", *(total_number_of_books + x));

    }

}



if (total_number_of_books) {
    free(total_number_of_books);
}



for (int i = 0; i < total_number_of_shelves; i++) {
    if (*(total_number_of_pages + i)) {
        free(*(total_number_of_pages + i));

    }

}



if (total_number_of_pages) {
    free(total_number_of_pages);
}
```

```
    return 0;

}
```

out put:

Input (stdin)

Download

```
    99788

    99568

    1  50021  443470714  1  50043

    153031558        1        50084

    325144167        1        50001

    418534703   yk61511   50046

    698808944


    hknmk1 50022 451281656 1

    50017   148646948   1   50066

    132831621        1        50038

    934068093        1        50067

    340781286        1        50098

    928668155        1        50043

    729466976        1        50071

    680798358

    1 50050 117244801 1

    50097 573018848
```

```
1   50095   395333893   1   50066
100747197  1  50092  507445983{-
truncated-}
```

Download to view the full testcase

Expected Output

Download

```
1
304760428 8
13 17 341420436 14
22
298320388
787261177
50 47 54 533521321
976682007
63
925023477
634698109   166617785
390249029{-truncated-}
```

16 .Calculate the Nth term :

Objective

This challenge will help you learn the concept of recursion.

CA function that calls itself is known as a recursive function. The programming language supports recursion. But while using recursion, one needs to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.
To prevent infinite recursion, statement (or similar approach) can be used where one branch makes the recursive call and other doesn't.

```
void recurse() {


    .....

    recurse() //recursive call

    .....

}

int main() {

    .....

    recurse(); //function call

    .....

}
```

Task

There is a series, , where the next term is the sum of pervious three terms.

Given the first three terms of the series, , , and respectively, you have to output the nth term of the series using recursion.

Recursive method for calculating nth term is given below.

Input Format

- The first line contains a single integer, .
- The next line contains 3 space-separated integers, , , and .

Constraints

-
-

Output Format

Print the nth term of the series, .

Sample Input 0

5

1 2 3

Sample Output 0

11

Explanation 0

Consider the following steps:

1.
2.

3.
4.
5.

From steps , , , and , we can say ; then using the values from step , , , and , we get . Thus, we print as our answer

program:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
//Complete the following function.


int find_nth_term(int n, int a, int b, int
  c) { int term, t1 = a, t2 = b, t3 = c;
    if (n == 1)
          term = t1;
      else if (n == 2)
           term = t2;
     else if (n == 3)
          term = t3;
    else {

        for (int i = 4; i <= n; i++) {
            term = t1 + t2 + t3; t1
            =  t2;  t2  =  t3;  t3  =
            term;


        }

    }

    return term;

}



int main() {

    int n, a, b, c;



    scanf("%d %d %d %d", &n, &a, &b,
    &c); int ans = find_nth_term(n, a, b, c);


    printf("%d", ans);
    return 0;
}
```

out put:

Compiler Message

Success

Input (stdin)

Download

```
5
1 2 3
```

Expected Output

Download

```
11
```

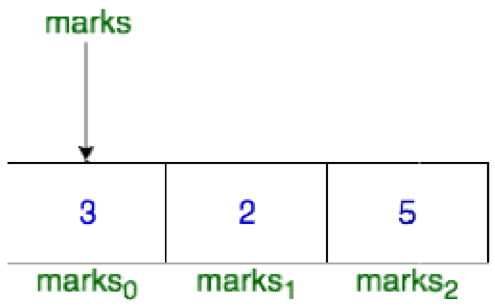
17 .Students Marks Sum:

You are given an array of integers, , denoting the marks scored by students in a class.

- The alternating elements , , and so on denote the marks of boys.
- Similarly, , , and so on denote the marks of girls.

The array name, , works as a pointer which stores the base address of that array. In other words, contains the address where is stored in the memory.

For example, let and stores 0x7fff9575c05f. Then, 0x7fff9575c05f is the memory address of .



Function Description

Complete the function, marks_summation in the editor below.
marks_summation has the following parameters:

- int marks[number_of_students]: the marks for each student
- int number_of_students: the size of marks[]
- char gender: either 'g' or 'b'

Returns

- int: the sum of marks for boys if , or of marks of girls if

Input Format

- The first line contains , denoting the number of students in the class, hence the number of elements in .
- Each of the subsequent lines contains .
- The next line contains .

Constraints

-

- (where)
- = or

Sample Input

0 3

3

2

5

b

Sample Output 0

8

Explanation 0

= [3, 2, 5] and = .

So, .

Sample Input 1

5

1

2

3

4

5

g

Sample Output 1

6

Explanation 1

= [1, 2, 3, 4, 5] and =

So, = = .

Sample Input 2

program:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
//Complete the following function.


int marks_summation(int* marks, int number_of_students, char gender) {

  //Write your code
  here. int sum = 0;
```

```
   for(int i = (gender == 'b' ? 0 : gender == 'g' ? 1 : -1); i < number_of_students;
i+=2) {
    sum += marks[i];
  }
  return sum;
}


int main() {
    int number_of_students;
    char gender;
    int sum;


    scanf("%d", &number_of_students);
    int *marks = (int *) malloc(number_of_students * sizeof (int));


    for (int student = 0; student < number_of_students; student++) {
        scanf("%d", (marks + student));
    }


    scanf(" %c", &gender);
    sum = marks_summation(marks, number_of_students, gender);
    printf("%d", sum);
    free(marks);

    return 0;

}
```

out put:

```
Compiler Message
Success
Input (stdin)

Download

        3
        3
        2
        5
        b
Expected
Output
Download

        8
```

18. Sorting Array Of Strings:

To sort a given array of strings into lexicographically increasing order or into
an order in which the string with the lowest length appears first, a sorting
function with a flag indicating the type of comparison strategy can be written.
The disadvantage with doing so is having to rewrite the function for every new
comparison strategy. A better implementation would be to write a sorting function that

accepts a
pointer to the function that compares each pair of strings. Doing this will mean
only passing a pointer to the sorting function with every new comparison
strategy.

Given an array of strings, you need to implement a function which sorts the
strings according to a comparison function, i.e, you need to implement the
function : void string_sort(const char **arr,const int cnt, int (*cmp_func)(const char* a,
const char* b)){

}

The arguments passed to this function are:

- an array of strings :
- length of string array:
- pointer to the string comparison function:
- You also need to implement the following four string comparison functions:

1. to sort the strings in lexicographically non-decreasing order. 2. to sort the strings in lexicographically non-increasing order. 3. to sort the strings in non-decreasing order of the number of distinct characters present in them. If two strings have the same number of distinct characters present in them, then the lexicographically smaller string should appear first.

4. to sort the strings in non-decreasing order of their lengths. If two strings have the same length, then the lexicographically smaller string should appear first.

Input Format

You just need to complete the function string_sort and implement the four string comparison functions.

Constraints

- No. of Strings
- Total Length of all the strings
- You have to write your own sorting function and you cannot use the inbuilt function
- The strings consists of lower-case English Alphabets only.

Output Format

The locked code-stub will check the logic of your code. The output consists of the strings sorted according to the four comparsion functions in the order mentioned in the problem statement. Sample Input 0

```
4

wkue
qoi
```

```
sbv
fekls
```

Sample Output 0

```
fekls
qoi
sbv
wkue


wkue
sbv
qoi
fekls


qoi
sbv
wkue
fekls


qoi
sbv
wkue
```

fekls

program:

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



int lexicographic_sort(const char* a, const char* b)
    { return strcmp(a, b);
}


int lexicographic_sort_reverse(const char* a, const char* b) {
    return strcmp(b, a);
}


int sort_by_number_of_distinct_characters(const char* a, const char*
    b) { int count_a = 0, count_b = 0;
    int freq_a[26] = {0}, freq_b[26] = {0};
    for (int i = 0; i < strlen(a); i++) {
        if (freq_a[a[i] - 'a'] == 0) {
            count_a++;
            freq_a[a[i] - 'a'] = 1;
        }

    }



    for (int i = 0; i < strlen(b); i++) {
        if (freq_b[b[i] - 'a'] == 0) {
            count_b++;
            freq_b[b[i] - 'a'] = 1;
        }

    }



    if (count_a == count_b) {
        return strcmp(a, b);
    } else {

        return (count_a - count_b);

    }

}



int sort_by_length(const char* a, const char* b) {
    int len_a = strlen(a); int
    len_b = strlen(b);


    if (len_a == len_b) {
        return strcmp(a,
        b);
    } else {

        return (len_a - len_b);

    }

}



void string_sort(char** arr, const int len, int (*cmp_func)(const char* a,
const char* b)) {

    for (int i = 0; i < len; i++) {

        for (int j = i + 1; j < len; j++) {

            if (cmp_func(arr[i], arr[j]) > 0) {
                char* temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }

        }

    }

}
```

```
int main()

{

    int n;
    scanf("%d",
    &n);


    char** arr;

    arr = (char**)malloc(n * sizeof(char*));



    for(int i = 0; i < n; i++){

        *(arr + i) = malloc(1024 * sizeof(char));
        scanf("%s", *(arr + i));
        *(arr + i) = realloc(*(arr + i), strlen(*(arr + i)) + 1);

    }



    string_sort(arr, n, lexicographic_sort);
    for(int i = 0; i < n; i++)
        printf("%s\n", arr[i]);
    printf("\n");


    string_sort(arr, n, lexicographic_sort_reverse);
    for(int i = 0; i < n; i++)
        printf("%s\n", arr[i]);
    printf("\n");



    string_sort(arr, n, sort_by_length);
    for(int i = 0; i < n; i++)
        printf("%s\n", arr[i]);
    printf("\n");


    string_sort(arr, n, sort_by_number_of_distinct_characters);
    for(int i = 0; i < n; i++)
        printf("%s\n", arr[i]);
    printf("\n");
}
```

out put :

Compiler Message

Success

Input (stdin)

Download

```
42
ibahqace
ux prmv
ovsylj
ta
eovfkgik
n
mrhgpe
xi s
gqrhdd
```

19. Permutations Of Strings:

Strings are usually ordered in lexicographical order. That means they are ordered by comparing their leftmost different characters. For example, because . Also because . If one string is an exact prefix of the other it is
lexicographically smaller, e.g., .
Given an array of strings sorted in lexicographical order, print all of its
permutations in strict lexicographical order. If two permutations look the same,
only print one of them. See the 'note' below for an example. Complete the function

next_permutation which generates the permutations in

the described order.

For example, . The six permutations in correct order
are: ab bc cd
ab cd bc
bc ab cd

bc cd ab
cd ab bc
cd bc ab

Note: There may be two or more of the same string as elements of .

For example, . Only one instance of a permutation where all elements match should be printed. In other words, if , then print either or but not both.

A three element array having three distinct elements has six permutations as

oshf own above. In this case, there are three matching pairs permutations where and are switched. We only print the three visibly unique permutations:
ab ab bc
ab bc ab
bc ab ab

Input Format

The first line of each test file contains a single integer , the length of the string array .
Each of the next lines contains a string
. Constraints

-
-
- contains only lowercase English

letters. Output Format
Print each permutation as a list of space-separated strings on a single line.

Sample Input 0

2

ab
cd

Sample Output 0
ab cd cd ab

Sample Input 1

3

a
bc
bc

Sample Output
1 a bc bc bc a
bc
bc bc a

Explanation 1

This is similar to the note above. Only three of the six permutations are printed to avoid redundancy in output.

program:

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


int next_permutation(int n, char **s)

{


    int k = -1;

    for (int i = 0; i < n-1; i++) {    if
        (strcmp(s[i], s[i+1]) < 0)
            k = i;

    }

    if (k == -1) return 0;


    int l = -1;

    for (int i = k+1; i < n; i++) {
        if (strcmp(s[k], s[i]) < 0)
            l = i;
    }



    char   *tmp   =   s[k];
    s[k]   =   s[l];   s[l]   =
    tmp;



    int i = k+1, j = n-1;
    while (i < j) {
        tmp    =    s[i];
        s[i++]  =  s[j];
        s[j--] = tmp;

    }



    return 1;

}
int main()

{

    char
    **s; int
    n;
    scanf("%d", &n);
    s = calloc(n, sizeof(char*));
    for (int i = 0; i < n; i++)
    {

        s[i] = calloc(11, sizeof(char));
        scanf("%s", s[i]);
    }

    do


    {

        for (int i = 0; i < n; i++)

            printf("%s%c", s[i], i ==       n  -  1 ? '\n' : ' ');

    } while (next_permutation(n, s));
    for (int i = 0; i < n; i++)
        free(s[i]);
    free(s);
    return 0;
}
```

out put:

Compiler Message

Success

Input (stdin)

Download

```
2
a
b
c
d
```

Expected Output

Download

```
ab
cd
cd
ab
```

20. Variadic Functions In C:

Variadic functions are functions which take a variable number of
arguments. In C programming, a variadic function will contribute to the
flexibility of the program that you are developing. The declaration of a variadic function
starts with the declaration of at least one
named variable, and uses an ellipsis as the last parameter, e.g.

int printf(const char* format, ...);

In this problem, you will implement three variadic functions named , and to
calculate sums, minima, maxima of a variable number of arguments. The
first

argument passed to the variadic function is the count of the number of arguments, which is followed by the arguments themselves.

Input Format

- The first line of the input consists of an integer .
- Each test case tests the logic of your code by sending a test
- implementation of 3, 5 and 10 elements respectively.
- You can test your code against sample/custom input.

  The error log prints the parameters which are passed to the test implementation. It also prints the sum, minimum element and maximum element corresponding to your code.

Constraints

.

Output Format

"Correct Answer" is printed corresponding to each correct execution of a test implementation."Wrong Answer" is printed otherwise.

Sample Input 0

1

Sample Output 0

Correct Answer

Correct Answer

Correct Answer
program:

```
#include <stdarg.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>



#define MIN_ELEMENT 1

#define MAX_ELEMENT 1000000

int sum (int count,...) {
    va_list args;
```

```
 va_start(args, count);

  int result = 0;

 for (int i = 0; i < count; i++) {

  result += va_arg(args, int);

}

 va_end(args);
 return result;

}

int min(int count,...) {

va_list args;
va_start(args, count);

int result = va_arg(args, int);
for (int i = 1; i < count; i++) {

int current = va_arg(args, int);
if (current < result) {

result = current;

}

}
```

```
    va_end(args);
    return result;




}







int max(int count,...) {
    va_list args;
    va_start(args, count);




    int result = va_arg(args, int);
    for (int i = 1; i < count; i++) {
        int current = va_arg(args, int);
        if (current > result) {
            result = current;
        }



    }








    va_end(args);
    return result;





}








int test_implementations_by_sending_three_elements() {
    srand(time(NULL));
    int elements[3];

    elements[0] = rand() % (MAX_ELEMENT - MIN_ELEMENT          +
1) + MIN_ELEMENT;

    elements[1] = rand() % (MAX_ELEMENT - MIN_ELEMENT          +
1) + MIN_ELEMENT;

    elements[2] = rand() % (MAX_ELEMENT - MIN_ELEMENT          +
1) + MIN_ELEMENT;

    fprintf(stderr,    "Sending    following    three
    elements:\n");  for  (int  i  =  0;  i  <  3;  i++)  {
    fprintf(stderr, "%d\n", elements[i]);

    }
```

```
int elements_sum = sum(3, elements[0], elements[1], elements[2]);


int minimum_element = min(3, elements[0], elements[1], elements[2]);
int maximum_element = max(3, elements[0], elements[1], elements[2]);



fprintf(stderr, "Your output is:\n");


fprintf(stderr, "Elements sum is %d\n", elements_sum);
fprintf(stderr, "Minimum element is %d\n", minimum_element);
fprintf(stderr, "Maximum element is %d\n\n",
maximum_element);



int expected_elements_sum = 0;
for (int i = 0; i < 3; i++) {
 if (elements[i] < minimum_element) {
     return 0;
 }

 if (elements[i] > maximum_element) {

     return 0;

  }


     expected_elements_sum += elements[i];


}


   return elements_sum == expected_elements_sum;

}

 int test_implementations_by_sending_five_elements() {

     srand(time(NULL));
```

```
elements[0] = rand() % (MAX_ELEMENT - MIN_ELEMENT        +

1) + MIN_ELEMENT;

elements[1] = rand() % (MAX_ELEMENT - MIN_ELEMENT         +

1) + MIN_ELEMENT;

 elements[2] = rand() % (MAX_ELEMENT - MIN_ELEMENT       +

 1) + MIN_ELEMENT;
```

```
    elements[3] = rand() % (MAX_ELEMENT - MIN_ELEMENT          +
1) + MIN_ELEMENT;


    elements[4] = rand() % (MAX_ELEMENT - MIN_ELEMENT          +
1) + MIN_ELEMENT;




    fprintf(stderr,    "Sending    following    five
    elements:\n");  for  (int  i  =  0;  i  <  5;  i++)  {
    fprintf(stderr, "%d\n", elements[i]);


    }




    int elements_sum = sum(5, elements[0], elements[1], elements[2],
elements[3], elements[4]);


    int minimum_element = min(5, elements[0], elements[1], elements[2],
elements[3], elements[4]);
```

```
    int maximum_element = max(5, elements[0], elements[1], elements[2],
elements[3], elements[4]);





    fprintf(stderr, "Your output is:\n");


    fprintf(stderr, "Elements sum is %d\n", elements_sum);
    fprintf(stderr, "Minimum element is %d\n", minimum_element);
    fprintf(stderr, "Maximum element is %d\n\n",
    maximum_element);



    int expected_elements_sum = 0;
    for (int i = 0; i < 5; i++) {
        if (elements[i] < minimum_element) {
            return 0;
    }

     if (elements[i] > maximum_element) {
     return 0;
    }

     expected_elements_sum += elements[i];

   }

   return elements_sum == expected_elements_sum;
}

   int test_implementations_by_sending_ten_elements() {
    srand(time(NULL));

    int elements[10];
```

```
elements[0] = rand() % (MAX_ELEMENT - MIN_ELEMENT        +

1) + MIN_ELEMENT;

elements[1] = rand() % (MAX_ELEMENT - MIN_ELEMENT        +

1) + MIN_ELEMENT;

elements[2] = rand() % (MAX_ELEMENT - MIN_ELEMENT        +

1) + MIN_ELEMENT;

elements[3] = rand() % (MAX_ELEMENT - MIN_ELEMENT        +

1) + MIN_ELEMENT;

elements[4] = rand() % (MAX_ELEMENT - MIN_ELEMENT        +

1) + MIN_ELEMENT;
```

```
    elements[5] = rand() % (MAX_ELEMENT - MIN_ELEMENT          +
1) + MIN_ELEMENT;


    elements[6] = rand() % (MAX_ELEMENT - MIN_ELEMENT          +
1) + MIN_ELEMENT;


    elements[7] = rand() % (MAX_ELEMENT - MIN_ELEMENT          +
1) + MIN_ELEMENT;


    elements[8] = rand() % (MAX_ELEMENT - MIN_ELEMENT          +
1) + MIN_ELEMENT;


    elements[9] = rand() % (MAX_ELEMENT - MIN_ELEMENT          +
1) + MIN_ELEMENT;




    fprintf(stderr, "Sending following ten
    elements:\n"); for (int i = 0; i < 10; i++) {
        fprintf(stderr, "%d\n", elements[i]);


    }
    int elements_sum = sum(10, elements[0], elements[1], elements[2],
 elements[3], elements[4],

                              elements[5], elements[6], elements[7], elements[8],
 elements[9]);

    int minimum_element = min(10, elements[0], elements[1], elements[2],
 elements[3], elements[4],

                              elements[5], elements[6], elements[7], elements[8],
 elements[9]);

    int maximum_element = max(10, elements[0], elements[1], elements[2],
 elements[3], elements[4],

                              elements[5], elements[6], elements[7], elements[8],
 elements[9]);

fprintf(stderr, "Your output is:\n");
fprintf(stderr, "Elements sum is %d\n", elements_sum);
fprintf(stderr, "Minimum element is %d\n", minimum_element);
fprintf(stderr, "Maximum element is %d\n\n",
maximum_element);




int expected_elements_sum = 0;
for (int i = 0; i < 10; i++) {
    if (elements[i] < minimum_element) {
        return 0;
    }






    if (elements[i] > maximum_element) {
        return 0;
```

```
            }

             expected_elements_sum += elements[i];

          }


            return elements_sum == expected_elements_sum;


       }



     int main ()

   {
       int number_of_test_cases;

       scanf("%d",

     &number_of_test_cases);

   while (number_of_test_cases--) {

    if (test_implementations_by_sending_three_elements()) {

    printf("Correct Answer\n");

    } else {

   printf("Wrong Answer\n");

  }

     if (test_implementations_by_sending_five_elements()) {
```

```
    printf("Correct Answer\n");


    } else {

    printf("Wrong Answer\n");


   }

    if (test_implementations_by_sending_ten_elements()) {

     printf("Correct Answer\n");

     } else {

     printf("Wrong Answer\n");

    }


  }


  return 0;


 }
```

out put:

Compiler Message

Success

Input (stdin)

Download

Expected Output

Download

Correct

Answer

Correct

Answer

Correct

Answer

Correct

Answer

Correct

Answer

Correct

Answer

Correct

Answer

Correct

Answer

Correct

Answer

Correct

Answer

Correct

Answer

Correct Answer

Correct Answer{-truncated-}

21. Querying the Document:

A document is represented as a collection paragraphs, a paragraph is represented as a collection of sentences, a sentence is represented as a collection of words and a word is represented as a collection of lower-case ([a-z]) and upper-case ([A-Z]) English characters.
You will convert a raw text document into its component paragraphs, sentences and words. To test your results, queries will ask you to return a specific paragraph, sentence or word as described below. Alicia is studying the C

programming language at the University of
Dunkirk and she represents the words, sentences, paragraphs, and documents using pointers:

- A word is described by .
- A sentence is described by . The words in the sentence are separated by one space (" "). The last word does not end with a space(" ").
- A paragraph is described by . The sentences in the paragraph are separated by one period (".").
- A document is described by . The paragraphs in the document are separated by one newline("\n"). The last paragraph does not end with a newline.

For example:

```
Learning C is fun.
Learning pointers is more fun.It is good to have pointers.
```

- The only sentence in the first paragraph could be represented

as: char** first_sentence_in_first_paragraph = {"Learning", "C", "is", "fun"};

- The first paragraph itself could be represented as:

char*** first_paragraph = {{"Learning", "C", "is", "fun"}};

- The first sentence in the second paragraph could be represented as:

char** first_sentence_in_second_paragraph = {"Learning", "pointers", "is", "more", "fun"};

- The second sentence in the second paragraph could be represented as:

char** second_sentence_in_second_paragraph = {"It", "is", "good", "to", "have", "pointers"};

- The second paragraph could be represented as:

char*** second_paragraph = {{"Learning", "pointers", "is", "more", "fun"}, {"It", "is", "good", "to", "have", "pointers"}};

- Finally, the document could be represented as:

char**** document = {{{"Learning", "C", "is", "fun"}}, {{"Learning", "pointers", "is", "more", "fun"}, {"It", "is", "good", "to", "have", "pointers"}}};

Alicia has sent a document to her friend Teodora as a string of characters, i.e. represented by not . Help her convert the document to form by completing the following functions:

- to return the document represented by .
- to return the paragraph.
- to return the sentence in the paragraph.
- to return the word in the sentence of the paragraph.

Input Format The first line contains the integer .

Each of the next lines contains a paragraph as a single string. The next line contains the integer , the number of queries.

Each of the next following formats:

lines or groups of lines contains a query in one of the

- 1 The first line contains :
  - The next line contains an integer , the number of sentences in the paragraph.
  - Each of the next lines contains an integer , the number of words in the sentence.
  - This query corresponds to calling the function .
- 2 The first line contains :
  - The next line contains an integer , the number of words in the sentence of the paragraph.
  - This query corresponds to calling the function
- 3 The only line contains :
  - This query corresponds to calling the function

Constraints

- The text which is passed to the has words separated by a space (" "), sentences separated by a period (".") and paragraphs separated by a newline("\n"). The last word in a sentence does not end with a space.
- The last paragraph does not end with a newline.
- The words contain only upper-case and lower-case English letters.
-
- number of characters in the entire document
- number of paragraphs in the entire

document Output Format

Print the paragraph, sentence or the word corresponding to the query to check the logic of your code.

Sample Input 0

```
2

Learning C is fun.

Learning pointers is more fun.It is good to have pointers.
3
1 2

2

5

6

2 1 1

4

3 1 1 1
```

Sample Output 0

```
Learning pointers is more fun.It is good to have pointers.
Learning C is fun
Learning
```

Explanation 0

The first query corresponds to returning the second paragraph with sentences of lengths and words.
The second query correspond to returning the first sentence of the first paragraph. It contains words.
The third query corresponds to returning the first word of the first sentence of the first paragraph.

program:

```
#include  <stdio.h>  #include  <stdlib.h>
#include  <string.h>  #include<assert.h>
#define MAX_CHARACTERS 1005


#define MAX_PARAGRAPHS 5
```

```
char* kth_word_in_mth_sentence_of_nth_paragraph(char**** document, int k,
int m, int n) {
    return document[n-1][m-1][k-1];
}


char** kth_sentence_in_mth_paragraph(char**** document, int k, int m) {
    return document[m-1][k-1];
}


char*** kth_paragraph(char**** document, int k) {
    return document[k-1];
}


char** split_string(char* text, char delim) {
    assert(text != NULL);
    char** result = malloc(1*sizeof(char*));
    int size = 1;


    char* temp = strtok(text, &delim);
    *result = temp;


    while(temp != NULL) {
        size++;

        result = realloc(result,size*sizeof(char*));
        temp = strtok(NULL, &delim);
        result[size-1] = temp;
    }

    return result;

}



char**** get_document(char* text) {
    assert(text != NULL);


    char** paragraphs = split_string(text, '\n');
    int npar = 0;
    while (paragraphs[npar] != NULL) {
        npar++;
    }

    char**** doc = malloc((npar+1)*sizeof(char***));
    doc[npar] = NULL;


    int i = 0;

    while (paragraphs[i] != NULL) {
        char** sentences = split_string(paragraphs[i], '.');
        int nsen = 0;
        while(sentences[nsen] != NULL) {
            nsen++;
        }



        doc[i] = malloc((nsen+1)*sizeof(char**));
        doc[i][nsen] = NULL;


        int j = 0;

        while (sentences[j] != NULL) {



            doc[i][j] = split_string(sentences[j], ' ');
            j++;
        }
        i++;
    }



    return doc;

}



char* get_input_text() {
    int paragraph_count;
    scanf("%d",
    &paragraph_count);


    char p[MAX_PARAGRAPHS][MAX_CHARACTERS],
    doc[MAX_CHARACTERS];

    memset(doc, 0, sizeof(doc));
    getchar();
    for (int i = 0; i < paragraph_count; i++) {
        scanf("%[^\n]%*c", p[i]);
        strcat(doc, p[i]);

        if (i != paragraph_count - 1)
            strcat(doc, "\n");
    }



    char* returnDoc = (char*)malloc((strlen (doc)+1) * (sizeof(char)));
    strcpy(returnDoc, doc);
    return returnDoc;


}



void print_word(char* word) {
    printf("%s", word);
}
```

```
void print_sentence(char** sentence) {
    int word_count;
    scanf("%d",      &word_count);
    for(int i = 0; i < word_count; i++){
        printf("%s",
        sentence[i]); if( i !=
        word_count - 1)
            printf(" ");

    }

}



void print_paragraph(char*** paragraph)
    { int sentence_count;
    scanf("%d", &sentence_count);

    for (int i = 0; i < sentence_count; i++) {
        print_sentence(*(paragraph + i));
        printf(".");
    }

}



int main()

{

    char* text = get_input_text();
    char**** document = get_document(text);



    int q;
    scanf("%d",
    &q);


    while (q--) {
        int type;
        scanf("%d", &type);



        if (type == 3){
            int k, m,
            n;
            scanf("%d %d %d", &k, &m, &n);


              char* word = kth_word_in_mth_sentence_of_nth_paragraph(document,
k, m, n);


            print_word(word);


        }



        else if (type == 2){
            int k, m;
            scanf("%d %d", &k, &m);

            char** sentence = kth_sentence_in_mth_paragraph(document, k, m);
            print_sentence(sentence);
        }


        else{
            int k;
            scanf("%d",
            &k);
            char*** paragraph = kth_paragraph(document, k);
            print_paragraph(paragraph);
        }

        printf("\n");

    }

}
```

out put:

Compiler Message

Success

Input (stdin)

Download

2 In a prologue a young couple at a diner discuss their life of crime as robbers and the prospect of robbing a

restaurant.They decide to rob the diner and begin the holdup.

Hitmen Jules Winnfield and Vincent Vega arrive at an apartment to retrieve a briefcase for their boss.After Vincent checks the contents of the briefcase Jules shoots one of the associates of Brett. 3

3 4 1 2

2 1 1

23

1 1

2

23

10

Expected Output

Download

and In a prologue a young couple at a diner discuss their life
of crime as robbers and the prospect of robbing a
restaurant
In a prologue a young couple at a diner discuss their life
of crime as robbers and the prospect of robbing a
restaurant.They decide to rob the diner and begin the
holdup.

22. Boxes through a tunnel :

You are transporting some boxes through a tunnel, where each box is a parallelepiped, and is characterized by its length, width and height.

The height of the tunnel feet and the width can be assumed to be infinite.
A box can be carried through the tunnel only if its height is strictly less than the

tunnel's height. Find the volume of each box that can be successfully transported to the other end of the tunnel. Note: Boxes cannot be rotated.

Input Format

The first line contains a single integer , denoting the number of boxes.

lines follow with three integers on each separated by single spaces , and which are length, width and height in feet of the -th box.

Constraints

-
-

Output Format

For every box from the input which has a height lesser than feet, print its volume in a separate line.

Sample Input 0

4

5 5 5

1 2 40

10 5 41

7 2 42

Sample Output 0

125

80

Explanation 0

The first box is really low, only feet tall, so it can pass through the tunnel and its volume is .
The second box is sufficiently low, its volume is .

The third box is exactly feet tall, so it cannot pass. The same can be said about the fourth box.

program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_HEIGHT
41 struct box
{

     /**

    * Define three fields of type int: length, width and height

   */

   int     length;
   int  width;  int
   height;

};

typedef struct box box;
```

```
int get_volume(box b) {

    /**

    * Return the volume of the box

    */

    return b.length * b.width * b.height;

}


int is_lower_than_max_height(box b) {

    /**

    * Return 1 if the box's height is lower than MAX_HEIGHT and 0
    otherwise

    */

    if (b.height < MAX_HEIGHT) {
        return 1;
    } else {

        return 0;

    }

}
    int main()
    int n;
    scanf("%d",
    &n);
    box *boxes = malloc(n * sizeof(box));
    for (int i = 0; i < n; i++) {
        scanf("%d%d%d", &boxes[i].length, &boxes[i].width, &boxes[i].height);

    }

    for (int i = 0; i < n; i++) {

        if (is_lower_than_max_height(boxes[i])) {
            printf("%d\n", get_volume(boxes[i]));
        }

    }

    return 0;

}
```

out put:

Compiler Message

Success

Input (stdin)

Download

```
4

5 5 5

1 2 40

10 5 41
```

7 2 42

Expected Output

Download

125

80

23. Small Triangle,Large Triangles:

You are given triangles, specifically, their sides , and . Print them in the same style but sorted by their areas from the smallest one to the largest one. It is guaranteed that all the areas are different. The best way to calculate a area of a triangle

with sides , and is Heron's

formula: where .

Input Format

The first line of each test file contains a single integer . lines follow with three space-separated integers, , and .

Constraints

-
-
- ,

and Output

Format

Print exactly lines. On each line print space-separated integers, the , and of the corresponding triangle.

Sample Input 0

```
3
7 24 25
5 12 13
3 4 5
```

Sample Output 0

```
3 4 5
5 12 13
7 24 25
```

Explanation 0

The area of the first triangle is . The area of the second triangle is . The area of the third triangle is . So the sorted order is the reverse one.

program:

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

```
struct triangle

{

    int

    a; int

    b; int

    c;

};



typedef struct triangle triangle;



double area(triangle tr) {

    double p = (tr.a + tr.b + tr.c) / 2.0;

    return sqrt(p * (p - tr.a) * (p - tr.b) * (p - tr.c));

}



int compare(const void *a, const void *b) {
    triangle *t1 = (triangle*) a;
    triangle *t2 = (triangle*) b;
    double area1 = area(*t1);
    double area2 = area(*t2);
    if (area1 > area2) {
        return 1;

    } else if (area1 < area2) {
        return -1;

    } else {

        return 0;

    }


}



void sort_by_area(triangle* tr, int n) {
    qsort(tr, n, sizeof(triangle), compare);
}





int main()

{

    int n;
    scanf("%d",
    &n);
    triangle *tr = malloc(n * sizeof(triangle));
    for (int i = 0; i < n; i++) {
        scanf("%d%d%d", &tr[i].a, &tr[i].b, &tr[i].c);

    }

    sort_by_area(tr, n);

    for (int i = 0; i < n; i++) {

        printf("%d %d %d\n", tr[i].a, tr[i].b, tr[i].c);
    }

    return 0;

}
```

out put:

Compiler Message

Success

Input (stdin)

Download

```
3
7 24 25 5 12
13
3 4 5
```

Expected Output

Download

```
3 4 5

5 12 13

7 24 25
```

24. Post Transition:

We live in a big country. This country has towns in it. Each town has some post offices in which packages are stored and transferred.

Post offices have different inner structure. Specifically, each of them has some limitations on the packages it can store - their weight should be between and inclusively, where and are fixed for each office.

Packages are stored in some order in the office queue. That means, that they are processed using this order when sending and receiving.
Sometimes two post offices, even in different towns, may organize the following transaction: the first one sends all its packages to the second one. The second one accepts the packages that satisfy the weight condition for the second office and rejects all other ones. These rejected packages return to the first office back and are stored in the same order they were stored before they were sent. The accepted packages move to the tail of the second office's queue in the same order they were stored in the first office. You should process several queries in your

program. You'll be provided with
structures , and . in order to complete this task, you should fill the following functions:

- given the town , print all packages in this town. They should be printed as follows:
Town_name:

```
    0:
        id_0
        id_1
        ...
    1:
        id_2
        id_3
```

...

...

where , etc are the numbers of post offices and , ... are the ids of packages from the th post office in the order of its queue, , are from the st one etc. There should be one '\t' symbol before post office numbers and two '\t' symbols before the ids.

- given the towns and and post office indices and , manage the transaction described above between the post office # in town and the post office # in town .

- given all towns, find the one with the most number of packages in all post offices altogether. If there are several of them, find the first one from the collection .

- given all towns and a string , find the town with the name . It's guaranteed that the town exists.

Input Format

First line of the input contains a single integer . blocks follow, each describing a town.

Every town block contains several lines. On the first line there is a string - the name of the town. On the second line there is an integer - the number of the offices in the town. blocks follow then, each describing an office. Every office block also contains several lines. On the first line there are three

integers separated by single spaces: (the number of packages in the office), and (described above). blocks follow, each describing a package.

Every package block contains exactly two lines. On the first line there is a string which is an id of the package. On the second line there is an integer which is the weight of the package.

Then, there is a single integer on the line which is the number of queries.
blocks follow, each describing a query.
Every query block contains several lines. On the first line there is an integer , or . If this integer is , on the second line there is a string - the name of town for which all packages should be printed. If this integer is , on the second line there are string , integer , string and integer separated by single spaces. That means transactions between post office # in the town and post office # in the town should be processed.
If the integer is , no lines follow and the town with the most number of packages should be found.
Constraints

- All integer are between and
- , .
- All strings have length
- All towns' names have only uppercase english letters and are unique.
- All packages' ids have only lowercase english letters and are unique.
- For each post office, .
- All queries are valid, that means, towns with the given names always exist, post offices with the given indices always exist.

Output Format

For queries of type , print all packages in the format provided in the problem statement. For queries of type , print "Town with the most number of packages is " on a separate line. Sample Input 0

```
2

A
2
2 1 3

a 2

b 3

1 2 4

c 2
B
1
4 1 4

d 1

e 2

f 3

h 4

5

3

2 B 0 A 1

3

1 A

1 B
```

Sample Output 0

```
Town with the most number of packages is B
Town with the most number of packages is A
A:
    0:
        a
        b
    1:
        c
        e
        f
        h
B:
    0:
        d
```

Explanation 0

Before all queries, town B has packages in total, town has . But after transaction all packages from B's th post office go to the st post office of A, except package d because it's too light.

program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_STRING_LENGTH 6

struct package

{

   char* id;
   int
   weight;
};

typedef struct package
package; struct post_office
{

   int min_weight;
   int max_weight;
   package* packages;
   int packages_count;
};

typedef struct post_office
post_office; struct town
```

```
{

   char* name;
   post_office* offices;
   int offices_count;

};

typedef struct town town;

void print_all_packages(town t)

{

    printf("%s:\n", t.name);

    for (int i = 0; i < t.offices_count; i++)

    {

        printf("\t%d:\n", i);

        for (int j = 0; j < t.offices[i].packages_count; j++)
            printf("\t\t%s\n", t.offices[i].packages[j].id);
    }

}

void send_all_acceptable_packages(town* source, int source_office_index, town*
target, int target_office_index)
```

```
{


    int n = 0;
    for (int i = 0; i < source->offices[source_office_index].packages_count; i++)   if

        (source->offices[source_office_index].packages[i].weight >=
target->offices[target_office_index].min_weight &&

            source->offices[source_office_index].packages[i].weight <=
target->offices[target_office_index].max_weight)

            ++n;

    package* newPackages = malloc(sizeof(package)*(n +
target->offices[target_office_index].packages_count));

    package* oldPackages =
malloc(sizeof(package)*(source->offices[source_office_index].packages_count -
n));

    for (int i = 0; i < target->offices[target_office_index].packages_count; i++)
        newPackages[i] = target->offices[target_office_index].packages[i];
    n = target->offices[target_office_index].packages_count; int
    m = 0;
    for (int i = 0; i < source->offices[source_office_index].packages_count; i++)


        if (source->offices[source_office_index].packages[i].weight >=
target->offices[target_office_index].min_weight &&
            source->offices[source_office_index].packages[i].weight <=
target->offices[target_office_index].max_weight)

        {

            newPackages[n] = source->offices[source_office_index].packages[i];

            ++n;

        }

        else

        {

            oldPackages[m] = source->offices[source_office_index].packages[i];

            ++m;

        }

    target->offices[target_office_index].packages_count = n;
    free(target->offices[target_office_index].packages);
    target->offices[target_office_index].packages = newPackages;
    source->offices[source_office_index].packages_count = m;
    free(source->offices[source_office_index].packages);
    source->offices[source_office_index].packages = oldPackages;


}
```

```
int number_of_packages(town t)

{

    int ans = 0;

    for (int i = 0; i < t.offices_count; i++)
        ans += t.offices[i].packages_count;
    return ans;

}

town town_with_most_packages(town* towns, int towns_count)

{

    int ans;

    int max_packages = -1;

    for (int i = 0; i < towns_count; i++)

        if (number_of_packages(towns[i]) > max_packages)

        {

            max_packages = number_of_packages(towns[i]);
            ans = i;
        }

    return towns[ans];
```

```
}

town* find_town(town* towns, int towns_count, char* name)

{

    for (int i = 0; i < towns_count; i++)
        if (!strcmp(towns[i].name, name))
    return
    &(towns[i]); return
    &towns[0];
}

int main()

{

   int towns_count;
   scanf("%d",
   &towns_count);
   town* towns = malloc(sizeof(town)*towns_count);
   for (int i = 0; i < towns_count; i++) {
      towns[i].name = malloc(sizeof(char) *
      MAX_STRING_LENGTH); scanf("%s", towns[i].name);
      scanf("%d", &towns[i].offices_count);

      towns[i].offices = malloc(sizeof(post_office)*towns[i].offices_count);
```

```
        for (int j = 0; j < towns[i].offices_count; j++) {

            scanf("%d%d%d", &towns[i].offices[j].packages_count,
&towns[i].offices[j].min_weight, &towns[i].offices[j].max_weight);

            towns[i].offices[j].packages =
malloc(sizeof(package)*towns[i].offices[j].packages_count);

            for (int k = 0; k < towns[i].offices[j].packages_count; k++) {

                towns[i].offices[j].packages[k].id = malloc(sizeof(char) *
MAX_STRING_LENGTH);

                scanf("%s", towns[i].offices[j].packages[k].id);
                scanf("%d", &towns[i].offices[j].packages[k].weight);
            }

        }

    }

    int      queries;
    scanf("%d",
    &queries);
    char
    town_name[MAX_STRING_LENGTH];
    while (queries--) {
        int type;
        scanf("%d",
        &type);
```

```
switch (type)
{ case 1:
   scanf("%s", town_name);

   town* t = find_town(towns, towns_count, town_name);
   print_all_packages(*t);
   break
; case 2:
   scanf("%s", town_name);

   town* source = find_town(towns, towns_count, town_name);
   int source_index;
   scanf("%d",
   &source_index);
   scanf("%s", town_name);
   town* target = find_town(towns, towns_count, town_name);
   int target_index;
   scanf("%d", &target_index);


   send_all_acceptable_packages(source, source_index, target, target_index);
   break;
case 3:
```

```
            printf("Town with the most number of packages is %s\n",
town_with_most_packages(towns, towns_count).name);

            break;

        }

    }

    return 0;

}
```

out put:

```
Compiler Message
Success
Input (stdin)

Download

            2

            A
            2
            2 1 3

            a 2

            b 3

            1 2 4

            c
            2
```

```
4 1 4
d 1 e

2

f 3

h 4

5

3

2 B 0 A 1

3

1 A{-truncated-}
```

Download to view the full testcase

Expected Output

Download

```
Town with the most number of packages is B
Town with the most number of packages is A A:


        0:

                a
                b
        1:

                c


                e
                 f
                h
B:
```

25. Structuring the documents:

A document is represented as a collection paragraphs, a paragraph is represented as a collection of sentences, a sentence is represented as a collection of words and a word is represented as a collection of lower-case ([a-z]) and upper-case ([A-Z]) English characters. You will convert a raw text document into its component paragraphs, sentences and words. To test your results, queries will ask you to return a specific paragraph, sentence or word as described below.

Alicia is studying the C programming language at the University of Dunkirk and she represents the words, sentences, paragraphs, and documents using pointers:

- A word is described by:

```
struct word {
    char* data;



};
```

- A sentence is described by:

```
struct sentence {
    struct word* data;
    int word_count;//the number of words in a sentence


};
```

The words in the sentence are separated by one space (" "). The last word does not end with a space.

- A paragraph is described by:

```
struct paragraph {
    struct sentence* data ;


    int sentence_count;//the number of sentences in a paragraph


};
```

The sentences in the paragraph are separated by one period (".").

- A document is described by:

```
struct document {

    struct paragraph* data;



    int paragraph_count;//the number of paragraphs in a document



};
```

The paragraphs in the document are separated by one newline("\n"). The last paragraph does not end with a newline.
For example:

```
Learning C is fun.

Learning pointers is more fun.It is good to have pointers.
```

- The only sentence in the first paragraph could be represented

as: struct sentence first_sentence_in_first_paragraph;
first_sentence_in_first_paragraph.data = {"Learning", "C", "is", "fun"};

- The first paragraph itself could be represented as:

struct paragraph first_paragraph;

first_paragraph.data = {{"Learning", "C", "is", "fun"}};

- The first sentence in the second paragraph could be represented

as: struct sentence first_sentence_in_second_paragraph;

first_sentence_in_second_paragraph.data = {"Learning", "pointers", "is", "more", "fun"};

- The second sentence in the second paragraph could be represented as:

struct sentence second_sentence_in_second_paragraph;

second_sentence_in_second_paragraph.data = {"It", "is", "good", "to", "have", "pointers"};

- The second paragraph could be represented as:

struct paragraph second_paragraph;

second_paragraph.data = {{"Learning", "pointers", "is", "more", "fun"}, {"It", "is", "good", "to", "have", "pointers"}};

- Finally, the document could be represented

as: struct document Doc;
Doc.data = {{{"Learning", "C", "is", "fun"}}, {{"Learning", "pointers", "is", "more", "fun"}, {"It", "is", "good", "to", "have", "pointers"}}};

Alicia has sent a document to her friend Teodora as a string of characters, i.e. represented by not . Help her convert the document to form by completing the following functions:

- to intialise the document. You have to intialise the global variable of type .
- to return the paragraph in the document.
- to return the sentence in the paragraph.
- to return the word in the sentence of the paragraph.

Input Format
The first line contains the integer .

Each of the next lines contains a paragraph as a single string.

The next line contains the integer , the number of queries.

Each of the next lines contains a query in one of the following formats:

- : This corresponds to calling the function .
- : This corresponds to calling the function .
- : This corresponds to calling the function .

Constraints

- The text which is passed to has words separated by a spaces(" "),

  sentences separated by a period(".") and paragraphs separated by a newline("\n").
- The last word in a sentence does not end with a space.
- The last paragraph does not end with a newline.
- The words contain only upper-case and lower-case English letters.
- number of characters in the entire document .
- number of paragraphs in the entire document

. Output Format

Print the paragraph, sentence or the word corresponding to the query to check the logic of your code.

Sample Input 0

2

Learning C is fun.

Learning pointers is more fun.It is good to have pointers.

3

1 2

2 1 1

3 1 1 1

Sample Output 0

Learning pointers is more fun.It is good to have pointers.
Learning C is fun
Learning

Explanation 0

The first query returns the second paragraph.

The second query returns the first sentence of the first paragraph.

The third query returns the first word of the first sentence of the first paragraph.

program:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <assert.h>

#define MAX_CHARACTERS 1005

#define MAX_PARAGRAPHS 5



struct word {
    char* data;
};
```

```
struct sentence {
struct word*
data;
int word_count;//denotes number of words in a sentence


};




struct paragraph {


    struct sentence* data ;


    int sentence_count;//denotes number of sentences in a paragraph


};




struct document {


    struct paragraph* data;
    int paragraph_count;//denotes number of paragraphs in a document


};




struct document get_document(char* text)


{


    struct document doc;


    struct paragraph *cur_paragraph =
    NULL; struct sentence *cur_sentence =
    NULL; char *new_word = NULL;
    doc.data = NULL;
    doc.paragraph_count = 0;
    for (char *s = text; *s; ++s)
    {
        if (*s == ' ' || *s == '.')
        {
            if (cur_paragraph == NULL)
            {
                doc.paragraph_count++;
                doc.data = (struct paragraph *) realloc(doc.data, sizeof(struct
paragraph) * doc.paragraph_count);

                cur_paragraph = doc.data + doc.paragraph_count - 1;
                cur_paragraph->data = NULL;
                cur_paragraph->sentence_count = 0;
                cur_sentence = NULL;
            }

            if (cur_sentence == NULL)

            {
                cur_paragraph->sentence_count++;
                cur_paragraph->data = (struct sentence *)
    realloc(cur_paragraph->data, sizeof(struct sentence)
    * cur_paragraph->sentence_count);

                cur_sentence = cur_paragraph->data +
    cur_paragraph->sentence_count - 1;

                cur_sentence->data = NULL;
                cur_sentence->word_count = 0;
            }

                cur_sentence->word_count++;
```

```
            cur_sentence->data = (struct word *) realloc(cur_sentence->data,
sizeof(struct word) * cur_sentence->word_count);


            cur_sentence->data[cur_sentence->word_count - 1].data = new_word;
            new_word = NULL;
            if (*s == '.')


                cur_sentence = NULL;


            *s = 0;


        }


        else if (*s == '\n')


        {


            cur_sentence = NULL;
            cur_paragraph = NULL;
        }
        else
        {
            if (new_word == NULL)
            {
                new_word = s;
            }
        }
    }
    return doc;
}
```

```
struct word kth_word_in_mth_sentence_of_nth_paragraph(struct document
Doc, int k, int m, int n)


{


    return Doc.data[n - 1].data[m - 1].data[k - 1];


}


struct sentence kth_sentence_in_mth_paragraph(struct document Doc, int
k, int m)


{


    return Doc.data[m - 1].data[k - 1];


}


struct paragraph kth_paragraph(struct document Doc, int k)


{
        return Doc.data[k - 1];
```

```
   }

     void print_word(struct word w) {

       printf("%s", w.data);

  }

            void print_sentence(struct sentence sen) {
               for(int i = 0; i < sen.word_count; i++) {
                print_word(sen.data[i]);



           if (i != sen.word_count - 1) {

           printf(" ");

       }

   }

}

void print_paragraph(struct paragraph para)

   { for(int i = 0; i < para.sentence_count; i++){

    print_sentence(para.data[i]);
    printf(".");
    for(int i = 0; i < doc.paragraph_count; i++) {
        print_paragraph(doc.data[i]);
        if (i != doc.paragraph_count - 1)
            printf("\n");
    }



}






char* get_input_text() {
    int paragraph_count;
    scanf("%d", &paragraph_count);






    char p[MAX_PARAGRAPHS][MAX_CHARACTERS],
    doc[MAX_CHARACTERS];
```

```
memset(doc, 0, sizeof(doc));
getchar();
for (int i = 0; i < paragraph_count; i++) {
    scanf("%[^\n]%*c", p[i]);
    strcat(doc, p[i]);



    if (i != paragraph_count - 1)
        strcat(doc, "\n");
}







char* returnDoc = (char*)malloc((strlen (doc)+1) * (sizeof(char)));
strcpy(returnDoc, doc);
return returnDoc;
```

```
}




int main()

{

    char* text = get_input_text();

    struct document Doc = get_document(text);




    int q;
    scanf("%d",
    &q);



    while (q--) {
        int type;
    scanf("%d", &type);




    if (type == 3){
        int k, m,
        n;
        scanf("%d %d %d", &k, &m, &n);

        struct word w = kth_word_in_mth_sentence_of_nth_paragraph(Doc, k,
m, n);

        print_word(w);

    }




    else if (type == 2) {
        int k, m;
        scanf("%d %d", &k, &m);


        struct sentence sen= kth_sentence_in_mth_paragraph(Doc, k, m);
        print_sentence(sen);
    }




    else{


        int k;
        scanf("%d",
        &k);
        struct paragraph para = kth_paragraph(Doc, k);
        print_paragraph(para);
    }


    printf("\n");
```

```
    }

}
```

out put:

Compiler Message

Success

Input (stdin)

Download

```
2
In a prologue a young couple at a diner discuss their life of crime as robbers and the prospect of robbing a restaurant.They decide to rob the diner and begin the holdup.
Hitmen Jules Winnfield and Vincent Vega arrive at an apartment to retrieve a briefcase for their boss.After Vincent checks the contents of the briefcase Jules shoots one of the associates of Brett.
3 3 4 1 2
2 1 1
1 1
```

Expected Output

Download

and

In a prologue a young couple at a diner discuss their life of crime as robbers and the prospect of robbing a restaurant
In a prologue a young couple at a diner discuss their life of crime as robbers and the prospect of robbing a restaurant.They decide to rob the diner and begin the holdup.

Tab 2