خروجی یاب

خروجی کد زیر را مشخص کنید.

برای هر قسمت از خروجی **توضیح کامل** الزامی است.

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

using namespace std;

void fun(int* p)
{
    int q = 38;
    p = &q;
    (*p) *= 2;
}

int main()
{
    int r = 2;
    int *z = &r;
    (*z)++;
    fun(z);
    printf("%d\n", *z);

    int a[] = {7, 8, 9, 10, 13};
    int *p = a + 1;
    *(a + 3) = *p + 3;
    p[1] = 4 * a[1];
    cout << a[0] << ", " << a[1] << ", " << a[2] << ", " << a[3] << ", " << a[4]

    int arr[] = {12, 13, 14};
    printf("%d, %d, %d\n", sizeof(arr), sizeof(*arr), sizeof(arr[0]));
    char crr[] = {'c', 'b', 'g'};
    printf("%d, %d, %d\n", sizeof(crr), sizeof(*crr), sizeof(crr[0]));
    double drr[] = {12.23, 13.54, 14.098};
    printf("%d, %d, %d\n", sizeof(drr), sizeof(*drr), sizeof(drr[0]));
```

```
33      return 0;
34  }
```

جمع ماتریس‌ها

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

برنامه‌ای بنویسید که جمع دو ماتریس هم‌مرتبه را محاسبه کند.
مدیریت حافظه‌ی ماتریس‌ها در این برنامه باید پویا باشد پس به **تخصیص** و **آزادسازی** حافظه‌ها دقت کنید.

ورودی

در خط اول r و c نوشته می‌شوند که به ترتیب تعداد سطرها و تعداد ستون‌های ماتریس‌ها را نشان می‌دهند نوشته می‌شوند. سپس در r خط بعد، در هر خط c عدد نوشته می‌شود که عدد jام در سطر iام عنصر $a_{i,j}$ را در ماتریس اول نشان می‌دهد. همین روند برای ماتریس دوم تکرار می‌شود.

$$1 \leq r, c \leq 50$$

$$-100 \leq a_{i,j}, b_{i,j} \leq 100$$

خروجی

ماتریس $A + B$ را با توجه به مثال‌ها در خروجی استاندارد چاپ کنید.

مثال

ورودی نمونه ۱

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

خروجی نمونه ۱

```
6 4
```

ورودی نمونه ۲

```
2 2
3 1
7 5
-2 0
-6 -4
```

خروجی نمونه ۲

```
1 1
1 1
```

دترمینان

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۶۴ مگابایت

برنامه‌ای بنویسید که با دریافت یک ماتریس مربعی، **دترمینان** آن را محاسبه کند.
محاسبه‌ی دترمینان را **باید** به صورت **بازگشتی** انجام دهید. در صورت نیاز به ایجاد ماتریس‌های جدید، مدیریت حافظه‌ی آن‌ها را باید به صورت پویا انجام دهید.

راهنمایی: از روش *بسط* استفاده کنید.

ورودی

در خط اول n یا تعداد سطرها و تعداد ستون‌های ماتریس مربعی نوشته می‌شود. سپس در n خط بعد، در هر خط n عدد نوشته می‌شود که عدد jام در سطر iام عنصر $a_{i,j}$ ماتریس را نشان می‌دهد.

$$1 \leq n \leq 10$$

$$-10 \leq a_{i,j} \leq 10$$

خروجی

در یک خط از خروجی استاندارد، دترمینان ماتریس دریافت شده را چاپ کنید.

مثال

ورودی نمونه ۱

```
2
3 1
```

```
7 5
```

خروجی نمونه ۱

```
8
```

ورودی نمونه ۲

```
3
1 18 7
3 10 5
2 2 12
```

خروجی نمونه ۲

```
-456
```

صف

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۶۴ مگابایت

حتما تا کنون در صف‌های مختلفی از جمله صف روز ثبت‌نام ایستاده‌اید.....
عموما صف‌ها سه ویژگی زیر را دارند:

- نفر جدید به **انتهای** صف اضافه می‌شود.
- افراد از **ابتدای** صف خارج می‌شوند.
- طول صف محدودیتی **ندارد**.

برنامه‌ای بنویسید که داده‌هایی به صورت **حروف کوچک انگلیسی** را با دستورات زیر در قالب یک **صف** مدیریت کند:

۱. دستور **enqueue**: مقداری را به **انتهای** صف اضافه می‌کند.
۲. دستور **dequeue**: **اولین** عنصر صف را خارج و چاپ می‌کند. اگر صف خالی باشد، عبارت **empty** چاپ می‌شود.
۳. دستور **print**: عناصر صف را به ترتیب **از ابتدا تا انتها** چاپ می‌کند. در صورتی که صف خالی باشد عبارت **empty** چاپ می‌شود.
۴. دستور **size**: طول صف را چاپ می‌کند.

در برنامه‌ی خود **باید** توابع زیر را پیاده‌سازی کرده و از آن‌ها استفاده کنید:

```
void enqueue(char* &queue, int &n, char data);
char dequeue(char* &queue, int &n);
void print_queue(char* queue, int n);
```

به نکات زیر توجه کنید:

- استفاده از متغیرهای *global* مجاز **نیست**.
- در توابعی که نیاز است آدرسی که پوینتر به آن اشاره می‌کند عوض شود، از **call by reference to pointer** استفاده شده‌است. در صورت نیاز، می‌توانید در این مورد بیشتر بخوانید.

ورودی

در هر خط از ورودی استاندارد، دستوراتی مطابق جدول زیر وارد می‌شوند تا زمانی که مقدار **F** وارد شود.

Function	Input
enqueue	E
dequeue	D
size	S
print	P

خروجی

برای هر دستور به جز **E**، در هر خط از خروجی استاندارد و با توجه به آن دستور، عبارت مناسب را چاپ کنید.

مثال

ورودی نمونه ۱

```
E k
D
D
E x
D
D
E w
```

```
E z
P
E m
S
D
D
S
F
```

خروجی نمونه ۱

```
k
empty
x
empty
w z
3
w
z
1
```

بررسی دستورات:

۱. مقدار **k** در صف قرار می‌گیرد.

۲. اولین عضو صف یعنی **k** خارج‌شده و چاپ می‌شود.

۳. چون صف خالی است، عبارت **empty** چاپ می‌شود.

۴. مقدار **x** در صف قرار می‌گیرد.

۵. اولین عضو صف یعنی **x** خارج شده و چاپ می‌شود.

۶. چون صف خالی است، عبارت **empty** چاپ می‌شود.

۷. مقدار **w** در صف قرار می‌گیرد.

۸. مقدار **z** بعد از **w** در صف قرار می‌گیرد.

۹. عناصر صف از ابتدا چاپ می‌شوند. عناصر فعلی صف **w z** هستند.

۱۰. عنصر **m** بعد از **z** در صف قرار می‌گیرد.

۱۱. طول صف چاپ می‌شود. صف شامل **سه** عنصر **w z m** است.

۱۲. اولین عضو صف یعنی **w** خارج شده و چاپ می‌شود. بعد از آن عناصر صف **z m** خواهند بود.

۱۳. اولین عضو صف یعنی **z** خارج شده و چاپ می‌شود. بعد از آن تنها عنصر صف **m** خواهد بود.

۱۴. طول صف چاپ می‌شود.

۱۵. پایان ورود دستورات.

ورودی نمونه ۲

```
E t
D
D
D
P
E o
D
D
D
E f
S
D
D
E b
E c
E s
E y
P
E c
D
E o
F
```

خروجی نمونه ۲

```
t
empty
```

```
empty
empty
o
empty
empty
1
f
empty
b c s y
b
```

پوینتر به خودِ فانکشن (امتیازی)

تابع **func** با دریافت دو ماتریس به همراه اندازه‌ی آن‌ها، بر روی هر دو خانه‌ی متناظر از ماتریس‌ها *عمل* مشخصی را انجام داده و حاصل را در ماتریس جدیدی به عنوان خروجی برمی‌گراند. این *عمل* درواقع یک تابع دیگر مثل **add** است که به **func** ارسال می‌شود.

تابع **print** یک ماتریس را گرفته و آن را چاپ می‌کند، درایه‌های یک سطر پشت هم و با یک فاصله چاپ شده و بعد از اتمام هر سطر نیز خط جدید چاپ می‌شود.

به شبه کد تابعِ **func** توجه کنید:

```
func(rows, cols, matrix A, matrix B, operation)
  for all (i, j) where 0 <= i < rows and 0 <= j < cols:
    C(i,j) = operation(A(i,j), B(i,j))
  return C
```

پس از بررسی کد زیر، توابع **func** و **print** را به گونه‌ای پیاده‌سازی کنید که **کد به درستی کامپایل شده** و خروجی نیز طبق کامنت‌ها باشد.

به نکات زیر توجه کنید:

- تضمین می‌شود که ماتریس‌های **A** و **B** هم‌مرتبه هستند.
- فقط توابع **func** و **print** را پیاده‌سازی کنید و بقیه کد را تغییر ندهید.

```
#include<iostream>

using namespace std;

int add(int a, int b)
{
    return a + b;
}

int _hash_(int a, int b)
{
```

```
    return ((a % 10) * 10) + (b % 10);
}

/************Your Implementations***************/

//func

//print

/***********************************************/

int main()
{
    int **A = new int*[3];
    for (int i = 0; i < 3; i++)
    {
        A[i] = new int[4];
        for(int j = 0; j < 4; j++)
            A[i][j] = (4 * i) + j + 1;
    }

    int **B = new int*[3];
    for (int i = 0; i < 3; i++)
    {
        B[i] = new int[4];
        for(int j = 0; j < 4; j++)
           B[i][j] = i + j;
    }

    print(3, 4, A);
    /* 1 2 3 4
       5 6 7 8
       9 10 11 12
    */

    print(3, 4, B);
    /* 0 1 2 3
       1 2 3 4
       2 3 4 5
    */
```

```
52
53      int **C = func(3, 4, A, B, add);
54      print(3, 4, C);
55      /* 1 3 5 7
56         6 8 10 12
57         11 13 15 17
58      */
59
60      int **D = func(3, 4, A, B, _hash_);
61      print(3, 4, D);
62      /* 10 21 32 43
63         51 62 73 84
64         92 3 14 25
65      */
66      return 0;
67  }
```

پس از اینکه از صحت کد خود اطمینان پیدا کردید، **فقط** پیاده‌سازی مربوط به دو تابع را در یک فایل **c.** یا **cpp.** کپی کرده و آن را اپلود کنید.

جابه‌جایی‌های عظیم (امتیازی)

تابع **swap** را طوری پیاده‌سازی کنید که کد زیر به درستی کامپایل شده و خروجی آن به شکل مشخص شده باشد. **توجه**: حق تغییر در بقیه‌ی کد را ندارید و **فقط یک** تابع را باید پیاده‌سازی کنید.

```
#include<iostream>
using namespace std;

/*************
implement swap function here
*************/

int main()
{
    int i1 = 111, i2 = 22;
    int *ip1 = &i1, *ip2 = &i2;
    swap('i', &ip1, &ip2);
    *ip1 *= 2;
    *ip2 += 7;
    cout << i1 << ' ' << i2 << '\n'; // 118 44

    char c1 = 'G', c2 = 'P';
    char *cp1 = &c1, *cp2 = &c2;
    swap('c', &cp1, &cp2);
    *cp2 += 5;
    cout << *cp1 << ' ' << *cp2 << '\n'; // P L

    float f1 = 2.86, f2 = 9.68;
    float *fp1 = &f1, *fp2 = &f2;
    swap('f', &fp1, &fp2);
    cout << *fp1 << ' ' << f1 << '\n'; // 9.68 2.86

    return 0;
}
```