به نام خدا

درس مبانی برنامه‌سازی

تمرین ۳

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نیم سال اول ۰۰ـ۹۹

استاد:

رضا فکوری

مهلت ارسال:

۲۸ آذر ـ ساعت ۲۳:۵۹:۵۹

مسئول تمرین‌ها:

امیرمهدی نامجو، پرهام صارمی

مسئول تمرین ۳:

نیما فتحی

طراحان تمرین ۳:

نازنین آذریان ـ پویا اسمعیلی ـ مسیح بیگی ـ پرهام چاوشیان ـ متین داغیانی ـ مازیار شمسی‌پور

فهرست

سوالات ۲

سوال ۱. سوال شانسی ۲

سوال ۲. تغییر جمعیت ۴

سوال ۳. استونکس ۷

سوال ۴. مسافرخانه‌ی جَک زرد ۱۰

سوال ۵. رقم بازی ۱۲

سوال ۶. کمی هم گوگل کنیم! ۱۴

پاسخ‌ها ۱۶

پاسخ ۱. سوال شانسی ۱۶

پاسخ ۲. تغییر جمعیت ۱۸

پاسخ ۳. استونکس ۱۹

پاسخ ۴. مسافرخانه‌ی جَک زرد ۲۴

پاسخ ۵. رقم بازی ۲۷

پاسخ ۶. کمی هم گوگل کنیم! ۳۰

سوالات

سوال ۱. سوال شانسی

یکی از دستیاران آموزشی درس مبانی برنامه‌نویسی می‌خواهد برای این درس تمرین طرح کند اما با توجه به این‌که او به شدت به این کار علاقه‌مند است، هیجان‌زده شده و تعداد بسیار زیادی سوال طرح کرده است. این موضوع باعث شده که کار او برای انتخاب تعدادی از سوالات جهت ارائه به دانشجویان سخت شود. بنابراین تصمیم می‌گیرد که ابتدا سوالات را شماره‌گذاری کند و سپس از بین آن‌ها سوالاتی را که شماره‌شان اعداد تقریبا شانسی هستند انتخاب کنند. از دید TA اعداد شانسی اعدادی هستند که تنها از ارقام ۴ یا ۷ تشکیل شده‌اند. اعداد تقریبا شانسی نیز اعدادی هستند که حداقل یکی از شمارنده‌هایشان عدد شانسی باشد. با توجه به این‌که TA بسیار هیجان‌زده است در تعیین سوالات با شماره تقریبا شانسی به مشکل خورده است، بنابراین به سراغ شما آمده تا در تعیین آن‌ها به او کمک کنید.

ورودی

در خط اول عدد n می‌آید و سپس در هرکدام از n خط بعدی یک عدد می‌آید که باید خروجی متناظر با آن را چاپ کنید.

خروجی

به ازای هر خط اگر عدد ورودی تقریبا شانسی است در خروجی "YES" و در غیر این صورت "NO" را چاپ کنید.

ورودی نمونه

```
3
47
16
78
```

خروجی نمونه

```
YES
YES
NO
```

ورودی نمونه

```
5
753
272
539
710
518
```

خروجی نمونه

```
NO
YES
YES
NO
YES
```

سوال ۲. تغییر جمعیت

توی ده شلمرود حسنی تک و تنها روی چارپایه نشسته بود و به رفت و آمد افراد روستا توجه می‌کرد. او خیلی زود متوجه شد که جمعیت این ده به نحو خاصی تغییر می‌کند، به طوری که هر سال جمعیت به دلیل مرگ و میر یا تولد زیاد یا کم می‌شود. هم‌چنین او متوجه شد هنگامی که جمعیت روستا عددی فرد باشد میزان تولد آن سال به صفر میرسد. حسنی بالاخره بعد از محاسبات فراوان الگوی زیر را برای جمعیت ارائه داد:

۱. اگر جمعیت عددی زوج باشد می‌تواند تا پایان آن سال دقیقا یک و نیم برابر شود.

۲. اگر جمعیت از یک بزرگ‌تر باشد ممکن است تا پایان سال یک نفر کم شود.

هم‌چنین این الگو به صورت اتفاقی تکرار می‌شود به این معنا که اگر ۲ نفر در ده وجود داشته باشد، جمعیت در پایان سال هم می‌تواند ۳ باشد هم ۱. اما اگر ۳ نفر در ده باشند در پایان سال حتما جمعیت ۲ خواهد بود.

حال حسنی می‌خواهد بداند که اگر در آغاز سال x نفر در روستا وجود داشته باشند بعد از گذشت سال های زیاد آیا ممکن است که جمعیت y شود.

توجه کنید در این سوال شما باید از تکه کد زیر استفاده کنید و مجاز به تغییر دادن تابع main نیستید(در صورت تغییر نمره‌ی این سوال برای شما ۰ لحاظ می‌شود).

```
#include <stdio.h>

int main(){
  int x,y,t;
  scanf("%d",&t);
  while(t--){
    scanf("%d%d",&x,&y);
    if (check(x,y))
      printf("YES\n");
    else
      printf("NO\n");
}
  return 0;
}

```

نحوه‌ی ورودی و خروجی گرفتن در این سوال با توجه به این‌که تابع main از قبل پیاده‌سازی شده است مشخص بوده و شما تنها باید تابع check(x, y) را به گونه‌ای پیاده‌سازی کنید که عبارت‌های YES و NO در خروجی با توجه به صورت مسئله درست چاپ شود.

ورودی

همانطور که در تابع main مشخص است، در خط اول به شما ورودی t داده میشود که تعداد سوال هاست. در t خط بعد در هر خط دو عدد x و y وارد می‌شوند.

خروجی

به ازای هر x ، y اگر x می‌تواند پس از گذشت چندسال تبدیل به y شود عبارت "YES" در خروجی ظاهر می‌شود و در غیر این صورت عبارت ."NO"

ورودی نمونه

```
5
1 2
2 5
4 3
500 750
100 99
```

خروجی نمونه

```
NO
NO
YES
YES
```

```
YES
```

سوال ۳. استونکس

محید برای حفظ سرمایه‌اش قصد دارد وارد بازار بورس شود اما کارگزاری‌اش قوانین عجیبی دارد. شما قرار است یک سری عدد به عنوان داده سهام های متفاوت بگیرید و سوددهی آنها را بر اساس قوانین این کارگزاری حساب کنید.بدین صورت که:

- اگر عدد حجم معاملات عددی اول حلقوی باشد سهم به اندازه مجموع ارقام درصد سودده خواهد بود. عدد اول حلقوی به عددی گفته میشود که با شیفت دادن ارقام آن اول بماند. به طور مثال عدد ۱۱۹۳ عدد اول حلقوی است زیرا تمام اعداد ۱۱۹۳ و ۳۱۱۹ و ۹۳۱۱ و ۱۹۳۱ اول هستند.
  مثلا: حجم معاملات = ۱۱۹۳
  پس ۱۴ درصد سود می‌کند. و درغیر این صورت به تعداد مقسوم‌علیه های اولش زیان‌ده خواهد بود.
  مثلا حجم معاملات: ۶، ۲ درصد ضرر ده خواهد بود (۲ـ درصد سود)
- اختلاف حجم تقاضا(t) و حجم عرضه (a) را x می‌نامیم:

$$x = t - a$$

اگر $|x|$ بر تعداد مقسوم‌علیه های خودش بخش‌پذیر بود سهم به اندازه ضرب ارقامش سود یا ضرر می‌دهد.
مثلا ۱۲: ۱۲،۶،۴،۳،۲،۱ در نتیجه ۲ درصد سود یا ضرر می‌دهد.
در غیر این صورت به اندازه مجموع مقسوم علیه‌های اولش سود یا ضرر می‌دهد.
مثلا ۱۴: ۱۴،۷،۲،۱ در نتیجه ۹ درصد سود یا ضرر می‌دهد.
اگر x عددی مثبت باشد سود ده و اگر عددی منفی باشد زیان‌ده خواهد بود.
اگر x یا حجم معاملات ۰ باشد سود آن بخش نیز ۰ است.

سود نهایی از جمع سود های بدست آمده از حجم معاملات و حجم عرضه و تقاضا بدست می‌آید.

ورودی

عدد n به عنوان تعداد سهم‌هایی که می خریم داده می‌شود.
سپس درn خط بعدی در هر خط ۵ عدد صحیح با یک فاصله به صورت زیر داده میشوند:
عدد اول: کد شناسایی سهام
عدد دوم: حجم معاملات
عدد سوم: قیمت
عدد چهارم: حجم تقاضا
«««< HEAD عدد پنجم: حجم عرضه
برای همه ورودی‌ها:

$$10^{-9} \leq input \leq 10^{9}$$

======= عدد پنجم: حجم عرضه
»»»> d۸۴۱۸d۱dfab۲۸۷۳۲fa۴۲c۲۸f۰d۸۷ba۶۲۳d۹۷۰b۱۱ **خروجی**
بعد از هر خط ورودی درصد نهایی سود یا زیان هر سهم نمایش داده شود و در آخر پرسود ترین سهم (بیشترین درصد سود تقسیم بر قیمت) مشخص شود. در صورتی که دو سهم یک مقدار درصد سود تقسیم بر قیمت داشتند سهمی که زودتر آمده انتخاب می‌شود.

ورودی نمونه

```
2
2302 20 1000 18 30
1102 20 10 18 30
```

خروجی نمونه

```
-4%
-4%
Best option: 2302
```

سوال ۴. مسافرخانه‌ی جَک زرد

دیوید جونز قصد دارد تا یک بازی جهان‌باز (open-world) بسازد و در قسمتی از این بازی شخصیت اصلی قرار است دارت بازی کند. او از شما می خواهد که منطق این بازی را به صورت زیر پیاده‌سازی کنید.
۳ دایره با شعاع‌های مختلف و هم‌مرکز داریم (مرکز مبدا مختصات است) که نسبت شعاع‌ها یکسان نیست (اگر یکسان بود برنامه باید عبارت error را چاپ کند و خارج شود). هر شعاع یک امتیاز خاص دارد به طوری که شعاع کوچک‌تر امتیاز بیشتر و شعاع بزرگ‌تر امتیاز کمتری دارد.
قرار است تا مختصات برخورد دارت‌ها به شما داده شود و با توجه به شعاع کوچک‌ترین دایره و نسبت شعاع‌ها و امتیاز هر شعاع، امتیاز هر سری بازی توسط برنامه شما محاسبه شود.

ورودی

خط اول: ۳ عدد که بعد از هر کدام حرف R آمده با یک فاصله وارد می‌شود.
خط دوم: ۳ عدد که بیان‌گر امتیازهاست.
خط سوم: شعاع کوچک‌ترین دایره.
خط چهارم: تعداد پرتاب ها .(n)
n خط بعدی: مختصات برخورد.
همه اعداد ورودی صحیح هستند.
برای همه ورودی‌ها:

$$10^{-9} \leq input \leq 10^{9}$$

خروجی

عددی صحیح که بیانگر امتیاز کسب شده است.

ورودی نمونه

```
100R 200R 900R
100 20 50
10
3
0 0
5 12
100 100
```

خروجی نمونه

```
150
```

پرتاب اول ۱۰۰ امتیاز، پرتاب دوم ۵۰ امتیاز و پرتاب سوم (به دلیل خارج از محوطه بودن) صفر امتیاز دارند. همان‌طور که دیدید نسبت شعاع‌های داده شده و R بیانگر کوچکترین شعاع نیست و به عبارت دیگر ۱و۲و۹ با ۵و۱۰و۴۵ فرقی ندارد و کوچک‌ترین شعاع تعیین کننده اندازه دایره هاست.

سوال ۵. رقم بازی

فاطمه و محمد در حال انجام یک مسابقه هستند. مسابقه به این صورت است که ابتدا یک عدد به آن‌ها داده می‌شود. سپس آن‌ها ارقام این عدد را از سمت چپ (یعنی رقم با بیشترین ارزش) شماره‌گذاری می‌کنند. فاطمه حق دارد اعدادی که در خانه‌های فرد هستند را بردارد و محمد اجازه برداشتن اعداد موجود در خانه‌های زوج را دارد. فاطمه مسابقه را شروع می‌کند و یک عدد دل‌خواه را برمی‌دارد. سپس محمد عددی بر می‌دارد و آن‌قدر این کار را به نوبت انجام می‌دهند تا فقط یک عدد باقی بماند. اگر عدد باقی‌مانده زوج باشد محمد و اگر فرد باشد فاطمه برنده‌ی مسابقه خواهد بود (یعنی آخرین عدد زوج باشد و زوج یا فرد بودن مکان آن عدد مهم نیست.)
برنامه‌ای بنویسید که بتواند برنده این مسابقه را از قبل پیش‌بینی کند. این برنامه ابتدا یک عدد از ورودی می‌گیرد و به آن تعداد مسابقه انجام خواهد شد. در هر مسابقه یک عدد داده می شود و شما به ازای هر مسابقه باید برنده را اعلام کنید.
اگر فاطمه برنده شد عدد یک و اگر محمد برنده شد عدد دو را در خروجی چاپ کنید.
توجه داشته باشید که جایگاه اعداد فقط در مرحله اول تعیین شده و ثابت می‌ماند و در هر مرحله تغییر نمی‌کند.

ورودی

ابتدا عدد صحیح t ، تعداد تست‌ها داده می‌شود. سپس درهر یک از t خط بعدی عدد صحیح حداکثر ۱۸ رقمی n داده می‌شود که نشان‌دهنده‌ی یک مسابقه است.

خروجی

به ازای هر تست ورودی یک‌بار یکی از دو عدد ۱ یا ۲ چاپ خواهد شد.

ورودی نمونه

```
4
2
3
102
2069
```

خروجی نمونه

```
2
1
1
2
```

- در تست اول عدد ۲ داده شده است. چون فقط یک عدد داده شده هیچ‌کس نمی‌تواند عددی بردارد و عدد ۲ باقی می ماند و چون زوج است محمد برنده خواهد شد.
- به صورت مشابه با دادن عدد ۳ برنده فاطمه خواهد بود.
- در عدد ۱۰۲ فاطمه می‌تواند یکی از دو خانه‌ی فرد را انتخاب و حذف کند اما محمد فقط امکان حذف عدد ۰ را دارد (چون تنها عددی است که شماره خانه‌اش زوج است). بنابراین فاطمه برای آن‌که برنده شود در مرحله اول عدد ۲ را می‌سوزاند. سپس محمد در نوبت خود تنها انتخابش یعنی عدد ۰ را می‌سوزاند و چون عدد ۱ که باقی مانده فرد است فاطمه برنده خواهد شد.
- در عدد ۲۰۶۹ فاطمه می‌تواند اعداد ۲ و ۶ را بسوزاند اما محمد فقط یکی از دو عدد موجود در خانه‌های زوج را می تواند بسوزاند تا عدد باقی‌مانده برنده را تعیین کند. بنابراین محمد عدد ۹ را می‌سوزاند تا در مرحله‌ی آخر عدد ۰ باقی‌مانده و خودش برنده شود.

سوال ۶. کمی هم گوگل کنیم!

شما باید تابع findDivisiorSum را به صورتی پیاده‌سازی کنید که تعدادی عدد از ورودی گرفته و مجموع شمارنده‌های آن‌ها را محاسبه کند. سپس به صورت زیر خروجی را برگرداند. (متغیر s خارج از برنامه شما به صورت گلوبال تعریف شده است و شما می توانید در هر جای برنامه از آن استفاده کنید):

- اگر مقدار s برابر ۰ بود، تابع شما ۴ ورودی گرفته که ورودی اول آن یک کاراکتر و ورودی‌های بعدی اعداد شما هستند. اگر مقدار کاراکتر M بود، بیشترین و اگر m بود کمترین مقدار را از بین مجموع شمارنده‌های ورودی‌ها برگردانید.
- اگر مقدار s برابر ۰ نبود، ورودی اول تابع n بوده و پس از آن n ورودی می‌آید. این بار نیز مجموع شمارنده را محاسبه کرده و کمترین آن‌ها را برگردانید.

اعداد ورودی تابع شما به صورت calc(num1, num2, +) یا هر عملیات ریاضی که جایگزین + شده است، می‌باشند. عملیات ریاضی به صورت کاراکتر نبوده و شما باید کدی بنویسید که دقیقا همین فرمت را پردازش کند. (در صورتی که عملیات ریاضی تقسیم باشد، عدد اول حتما بر عدد دوم بخش‌پذیر است).

مثال

حالت اول (متغیر s با مقدار ۰ در برنامه وجود دارد):

```
findDivisorSum('M', calc(5, 10, +), calc(17, 13, *), calc(45, 5, /))
```

حالت دوم (متغیر s با مقدار ۱ در برنامه وجود دارد):

```
findDivisorSum(2, calc(12, 2, *), calc(50, 17, %))
```

شما تنها باید تابع‌های خواسته شده را پیاده‌سازی کنید و کد شما نباید تابع main داشته باشد. همچنین باید کد زیر در ابتدای برنامه ی شما قرار بگیرد.

```
#include "grader.h"

```

به علاوه تابع زیر باید در برنامه ی شما وجود داشته باشد:

```
long long run(char type, int num1, int num2, int num3, int num4) {
return findDivisorSum(type,
calc(num1, num2, +),
calc(num1, num2, -),
calc(num3, num4, *));
}

```

نکات ارسال پاسخ:

۱. پسوند فایل نهایی خود را به cpp. تغیر داده و به عنوان کد C++ ارسال کنید. (کد شما باید به زبان C باشد ولی ارسال به عنوان C++ اشکالی ندارد.)

۲. برای قسمت اول و دوم سوال کد یکسان ارسال کنید. توجه کنید که در صورت مغایرت کد های ارسالی نمره ای از سوال دریافت نخواهید کرد.

پاسخ‌ها

پاسخ ۱. سوال شانسی

برای حل این سوال از دو تابع استفاده می‌کنیم:

۱. isLucky

این تابع تشخیص می‌دهد که آیا عددی که به آن ورودی داده شده شانسی است یا خیر. برای این کار تمامی ارقام عدد ورودی را بررسی می‌کند. اگر حداقل یکی از ارقام ۴ یا ۷ نباشد، مقدار غلط و در غیر این صورت درست باز می‌گرداند.

۲. isAlmostLucky

بررسی می‌کند که آیا عدد تقریبا شانسی است یا خیر. برای این کار شانسی بودن تمامی شمارنده‌های عدد ورودی‌اش را بررسی می‌کند و اگر حداقل یکی از آن‌ها شانسی بود صحیح و در غیر این صورت غلط بازمی‌گرداند. در پایان به ازای تمامی اعداد ورودی تابع isAlmostLucky صدازده شده و خروجی مناسب چاپ می‌شود.

```c
#include <stdio.h>


int isLucky(int num)
{
  while(num > 0)
  {
    int r = num % 10;
    if(r != 4 && r != 7)
    {
      return 0;
    }
  num = num / 10;
  }
  return 1;
}

int isAlmostLucky(int num)
{
  int i = 1;
  while(i <= num)
```

```
  {
    if((num % i == 0) && isLucky(i))
      return 1;
    i++;
  }
  return 0;
}

int main()
{
  int num = 0;
  int n = 0;
  scanf("%d", &n);
  for(int i = 0; i < n; i++){
    scanf("%d", &num);
    if(isAlmostLucky(num))
    {
      printf("YES\n");
    }
  else
    printf("NO\n");
  }
  return 0;
}

```

پاسخ ۲. تغییر جمعیت

روشن است که یک عدد به اندازه کافی بزرگ پس از گذشت سال های زیاد میتواند به هر عدد دیگری تبدیل شود.(هر سال یک نفر از جمعیت روستا کم شود)
حال اگر عدد از ۴ بزرگتر باشد میتواند به اندازه‌ی دلخواه ما بزرگ شود (هر سال یک و نیم برابر میشود و اگر فرد بود یک نفر کم میشود و سپس دوباره یک و نیم برابر میشود) ، به عبارت دیگر هر عدد بزرگتر یا مساوی ۴ به هر عدد دلخواه دیگر میتواند تبدیل شود و مسئله تنها به بررسی حالت های اعداد ۲ ، ۳ ، ۴ منتهی می‌گردد.

```c
#include <stdio.h>

int check(int x, int y){
  if ( x==1 && y!=1)
    return 0;
  if ( x==3 && y!=3 && y!=2 && y!=1)
    return 0;
  if ( x==2 && y!=1 && y!=3 && y!=2 )
    return 0;
  return 1;
}

int main(){
  int x,y,t;
  scanf("%d",&t);
  for(int i=0; i < t; i++){
    scanf("%d%d",&x,&y);
    printf(check(x,y)? "YES\n" : "NO\n");
  }
  return 0;
}
```

پاسخ ۳. استونکس

نمونه پاسخ مورد قبول:

```
#include<stdio.h>

int calculateProfitFromVolume(int);
int calculateProfitFromDemandSupply(int,int);
int isNumberCircularPrime(int);
int getSumOfDigits(int);
int getNumberOfPrimefactors(int);
int getNumberOfDigits(int);
int pow(int, int);
int isNumberPrime(int);
int rotate(int, int);
int isNumberDivisibleByTheNumberOfItsDivisors(int);
int getNumberOfDivisors(int);
int getMultiplicationOfDigits(int);
int getSumOfPrimeFactors(int);

int main() {
  int n = 0;
  float max = 0;
  int maxId = 0;
  scanf("%d", &n);
  for (int i = 0; i < n; i++) {
    int id = 0;
    int volume = 0;
    int price = 0;
    int demand = 0;
    int supply = 0;
    scanf("%d %d %d %d %d", &id, &volume, &price, &demand, &supply);
    int profit = calculateProfitFromVolume(volume) +
 calculateProfitFromDemandSupply(demand,supply);
    printf("%d%%\n", profit);
    float profitPerPrice = (float)profit / price;
    if (profitPerPrice > max || i == 0) {
      max = profitPerPrice;
      maxId = id;
    }
  }
  printf("Best option: %d", maxId);
  return 0;
}
```

```
int calculateProfitFromVolume(int volume) {
  if (isNumberCircularPrime(volume)) return getSumOfDigits(volume);
  else return getNumberOfPrimefactors(volume) * -1;
}

int isNumberCircularPrime(int number) {
  int digits = getNumberOfDigits(number);
  for (int i = 0; i < digits; i++) {
    if (isNumberPrime(number)) {
      number = rotate(number, digits);
    }
    else return 0;
  }
  return 1;
}

int getNumberOfDigits(int number) {
  if (number == 0) return 1;
  int count = 0;
  while (number != 0) {
    number /= 10;
    count++;
  }
  return count;
}

int isNumberPrime(int number) {
  if (number <= 1) return 0;
  for (int i = 2; i <= number / 2; ++i) {
    if (number % i == 0) return 0;
  }
  return 1;
}

int rotate(int number, int numberOfDigits)
{
  int lastDigit = number % 10;
  return (lastDigit * pow(10, numberOfDigits - 1)) + (number/10);
}

int pow(int base, int power) {
  int res = 1;
  for (int i = 0; i < power; i++) res *= base;
```

```
  return res;
}

int getSumOfDigits(int number) {
  int sum = 0;
  while (number > 0)
  {
    sum += number % 10;
    number /= 10;
  }
  return sum;
}

int getNumberOfPrimefactors(int number) {
  if (isNumberPrime(number)) return 1;
  int count = 0;
  for (int i = 2; i <= number / 2; i++) {
    if (number % i == 0 && isNumberPrime(i)) count++;
  }
  return count;
}

int calculateProfitFromDemandSupply(int demand,int supply) {
  int x = demand - supply;
  int isProfitable = (x > 0 ? 1 : -1);
  if (x < 0) x = -1 * x;
  if (isNumberDivisibleByTheNumberOfItsDivisors(x)) return
 getMultiplicationOfDigits(x) * isProfitable;
  else return getSumOfPrimeFactors(x) * isProfitable;
}

int isNumberDivisibleByTheNumberOfItsDivisors(int x) {
  if (x == 0) return 0;
  if (x % getNumberOfDivisors(x) == 0) return 1;
  else return 0;
}

int getNumberOfDivisors(int number) {
  int count = 0;
  for (int i = 1; i <= number; i++) {
    if (number % i == 0) count++;
  }
  return count;
}
```

```
int getMultiplicationOfDigits(int number) {
  if (number == 0) return 0;
  int multiplication = 1;
  while (number > 0)
  {
    multiplication *= (number % 10);
    number /= 10;
  }
  return multiplication;
}

int getSumOfPrimeFactors(int number) {
  if (isNumberPrime(number)) return number;
  int sum = 0;
  for (int i = 2; i <= number / 2; i++) {
    if (number % i == 0 && isNumberPrime(i)) sum += i;
  }
  return sum;
}
```

برای حل این سوال، محاسبه دو نوع سود نیاز است. سود حاصل از حجم معاملات و سود حاصل از اختلاف عرضه و تقاضا، پس برای بدست اوردن سود هر بخش یک تابع تعریف می‌کنیم. calculateProfitFromVolume() سود حاصل از حجم معاملات و calculateProfitFromDemandSupply() سود حاصل از اختلاف عرضه و تقاضا را حساب می‌کند.

برای بدست آوردن سود حاصل از حجم معاملات توابع زیر تعریف می‌شوند.

- isNumberCircularPrime() : اگر عدد ورودی اول حلقوی باشد عدد یک و اگر نباشد عدد صفر را برمی‌گرداند.
- getNumberOfDigits() : تعداد ارقام عدد ورودی را برمی‌گرداند.
- isNumberPrime() : اگر عدد ورودی اول باشد عدد یک و اگر نباشد عدد صفر را برمی‌گرداند.
- rotate() : عدد ورودی را یک رقم به راست چرخانده و برمی‌گرداند.
- getSumOfDigits() : مجموع ارقام عدد ورودی را برمی‌گرداند.

- getNumberOfPrimefactors() : تعداد مقسوم الیه های اول عدد ورودی را برمی‌گرداند.
- pow() : عدد اول ورودی را به توان عدد دوم می رساند و عدد حاصل را بر میگرداند.

برای بدست آوردن سود حاصل از اختلاف عرضه و تقاضا توابع زیر تعریف می شوند.

- isNumberDivisibleByTheNumberOfItsDivisors() : اگر عدد ورودی بر تعداد مقسوم الیه هایش بخش پذیر باشد عدد یک و در غیر این صورت عدد صفر را برمی‌گرداند.
- getNumberOfDivisors() : تعداد مقسوم الیه های عدد ورودی را برمی‌گرداند.
- getMultiplicationOfDigits() : حاصل ضرب ارقام عدد ورودی را برمی‌گرداند.
- getSumOfPrimeFactors() : مجموع مقسوم الیه های اول عدد ورودی را برمی‌گرداند.

توجه کنید که هر تابع فقط یک کار ساده را انجام می‌دهد، به طور مثال در تابع isNumberCircularPrime() اگر از تابع‌های getNumberOfDigits() و isNumberPrime() و rotate() استفاده نمی‌کردیم اندازه تابع بزرگ و خوانایی کد بسیار کم می‌شد.

برای بدست آوردن پرسودترین سهم بعد از محاسبه سود سهم، حاصل تقسیم سود بر قیمت که می‌تواند اعشاری باشد را بدست می آوریم و با بزرگ‌ترین عددی که تا الان بدست آمده (max) مقایسه می کنیم. اگر سود برقیمت فعلی بزرگ‌تر از max بود، آن عدد را به جای max و شناسه سهم را به عنوان maxId جدید قرار می‌دهیم. توجه کنید که برای بار اول مقایسه صورت نمی گیرد و max و maxId فقط مقدار دهی می شوند.

پاسخ ۴. مسافرخانه‌ی جَک زرد

نمونه پاسخ قابل قبول:

```c
#include<stdio.h>

int getMaximum(int, int, int);
int getMiddle(int, int, int);
int getMinimum(int, int, int);
int calcScore(int, int, float, float, float, int, int, int);
int areEqual(int, int, int);

int main() {
  int a1 = 0, a2 = 0, a3 = 0;
  scanf("%d%*c %d%*c %d%*c", &a1, &a2, &a3);
  if (areEqual(a1, a2, a3)) {
    printf("error");
    return 0;
  }

  int temp1 = 0, temp2 = 0, temp3 = 0;
  scanf("%d %d %d", &temp1, &temp2, &temp3);
  int s1 = getMaximum(temp1, temp2, temp3);
  int s2 = getMiddle(temp1, temp2, temp3);
  int s3 = getMinimum(temp1, temp2, temp3);

  int realR1 = 0, n = 0;
  scanf("%d", &realR1);
  scanf("%d", &n);
  float realR2 = ((float)a2 / a1) * realR1;
  float realR3 = ((float)a3 / a1) * realR1;
  int score = 0;
  for (int i = 0; i < n; i++) {
    int x, y;
    scanf("%d %d", &x, &y);
    score += calcScore(x, y, realR1, realR2, realR3, s1, s2, s3);
  }
  printf("%d", score);
  return 0;
}

int areEqual(int a, int b, int c) {
  if (a == b || a == c || b == c) return 1;
  else return 0;
```

```
}

int calcScore(int x, int y, float r1, float  r2, float r3, int score1,
    int score2, int score3) {
  if (x * x + y * y <= r1 * r1) return score1;
  if (x * x + y * y <= r2 * r2) return score2;
  if (x * x + y * y <= r3 * r3) return score3;
  else return 0;
}

int getMaximum(int n1, int n2, int n3) {
  if (n1 > n2&& n1 > n3) return n1;
  if (n2 > n1&& n2 > n3) return n2;
  else return n3;
}

int getMinimum(int n1, int n2, int n3) {
  if (n1 < n2 && n1 < n3) return n1;
  if (n2 < n1 && n2 < n3) return n2;
  else return n3;
}

int getMiddle(int n1, int n2, int n3) {
  if (n1 < getMaximum(n1, n2, n3) && n1 > getMinimum(n1, n2, n3)) {
    return n1;
  }
  else if (n2 < getMaximum(n1, n2, n3) && n2 > getMinimum(n1, n2, n3)) {
    return n2;
  }
  else return n3;
}

```

ابتدا با تابع areEqual() نسبت شعاع‌ها را بررسی می‌کنیم این تابع در صورتی که ورودی هایش برابر باشند عدد یک و در غیر این صورت عدد صفر را برمی‌گرداند. در صورت یک بودن خروجی error چاپ شده و اجرا تمام می‌شود. سپس امتیاز شعاع‌ها را با استفاده از توابع زیر مرتب می‌کنیم.

- getMaximum() : بزرگترین عدد بین ورودی ها را برمی‌گرداند.
- getMiddle() : عدد وسطی بین ورودی ها را برمی‌گرداند.

- getMinimum() : کوچکترین عدد بین ورودی ها را برمی‌گرداند.

در ادامه برنامه شعاع های واقعی محاسبه می‌شوند بدین صورت که نسبت شعاع دوم و سوم بر نسبت شعاع اول تقسیم کرده و در شعاع کوچکتر ضرب می‌کنیم. توجه کنید که شعاع دوم و سوم می‌توانند اعشاری باشند.
سپس امتیاز هر برخورد در تابع calcScore() محاسبه می‌شود بدین صورت که به جای بدست آوردن فاصله از مبدا، مجذور آن را بدست می‌آوریم و شعاع داده شده را نیز در خودش ضرب می‌کنیم که توان دو شود:

$$r = \sqrt{x^2 + y^2} \rightarrow r^2 = x^2 + y^2$$

سپس با توجه به بازه ای که فاصله تا مبدا در آن قرار می‌گیرد امتیاز داده می‌شود.

پاسخ ۵. رقم بازی

در یک مرحله از بازی اگر تعداد رقم ها زوج باشد آخرین عددی که باقی می ماند در مرتبه زوج قرار دارد. به همین صورت اگر تعداد ارقام فرد باشد آخرین عددی که باقی می ماند در مرتبه فرد قرار دارد.
محمد در صورتی برنده می شود که آخرین رقم باقی مانده زوج باشد، بنابراین در دو حالت محمد برنده می شود:

۱. اگر تعداد ارقام زوج باشد:

چون آخرین عدد باقی مانده در مرتبه زوج قرار دارد محمد خودش انتخاب خواهد کرد که کدام عدد مرتبه زوج باقی بماند یعنی میتواند عدد مرتبه زوج مورد نظرش را تا آخرین مرحله بر ندارد تا آن عدد به عنوان آخرین عدد باقی بماند.بنابراین اگر در خانه های مرتبه زوج تنها یک عدد زوج داشته باشیم هم محمد برنده می شود چون آن عدد را تا انتها بر نمیدارد و به این ترتیب آخرین عدد باقی مانده زوج خواهد شد و محمد برنده می شود.

۲. اگر تعداد ارقام فرد باشد:

چون آخرین عدد باقی مانده در مرتبه فرد قرار دارد این بار فاطمه انتخاب خواهد کرد که کدام عدد مرتبه فرد آخرین باقی مانده باشد و از آنجایی که می خواهد خودش ببرد کاری میکند که آخرین عدد باقی مانده فرد باشد و اگر تنها یک عدد فرد در خانه های مرتبه فرد قرار داشته باشد فاطمه موفق خواهد شد بنابراین در این شرایط محمد تنها زمانی برنده می شود که تمام خانه های مرتبه فرد زوج باشند.

می دانیم اگر محمد برنده نشود فاطمه برنده خواهد شد.
در نتیجه ابتدا تابعی مینویسیم که با گرفتن عدد ورودی و تعداد رقم های آن محاسبه کند که چه کسی برنده خواهد شد. در تابع تعداد ارقام زوج مرتبه فرد و تعداد ارقام زوج مرتبه زوج را محاسبه می کنیم. بنابر توضیحات بالا در دو حالت محمد برنده می شود:

۱. اگر تعداد کل ارقام زوج باشد و تعداد ارقام زوج مرتبه زوج حداقل یک باشد(یعنی بزرگتر از ۰ باشد).

۲. اگر تعداد کل ارقام فرد باشد و تعداد ارقام زوج مرتبه فرد برابر کل تعداد اعداد مرتبه

فرد باشد(یعنی از نصف کل اعداد بیشتر باشد)

اگر هر یک از دوشرط بالا رخ داد خروجی تابع محمد و در غیر این صورت خروجی فاطمه خواهد بود.
برای استفاده از این تابع برای حل سوال کافیست تعداد ارقام هر مرحله را بدست آوریم که برای این کارتابعی می نویسیم که عدد ورودی را تقسیم بر ۱۰ میکنیم و این کار را آنقدر انجام می دهد تا به ۰ برسد. تعداد دفعاتی که تقسیم انجام داده برابر با تعداد ارقام هر عدد خواهد بود.
برای بدست آوردن هر خط خروجی تعداد ارقام ورودی را با تابع بالا بدست می آوریم و خود ورودی و تعداد ارقامش را به تابع محاسبه برنده میدهیم تا برنده را به ما خروجی دهد.
نکته: ورودی را long long گرفتیم تا بتوانیم تا اعداد ۱۸ رقمی را محاسبه کنیم.

```c
#include <stdlib.h>

int findWinner(long long, int);

int countDigits(long long);

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++) {
        long long input;
        scanf("%lld", &input);
        printf("%d\n", findWinner(input, countDigits(input)));
    }
    return 0;
}

int countDigits(long long n) {
    int counter = 0;
    while (n >= 0) {
        counter++;
        n /= 10;
        if (n == 0) {
            break;
        }
    }
    return counter;
}
```

```

int findWinner(long long input, int n) {
    int evenNumbersInEvenPositions = 0;
    int evenNumbersInOddPositions = 0;

    for (int i = 0; i < n; i++) {
        if (input % 2 == 0) {
            if (i % 2 == 0) {
                if (n % 2 != 0) {
                    evenNumbersInOddPositions++;
                } else {
                    evenNumbersInEvenPositions++;
                }
            } else {
                if (n % 2 != 0) {
                    evenNumbersInEvenPositions++;
                } else {
                    evenNumbersInOddPositions++;
                }
            }
        }
        input /= 10;
    }

    if (n % 2 != 0) {
        if (evenNumbersInOddPositions > n / 2) {
            return 2;

        }
    } else {
        if (evenNumbersInEvenPositions > 0) {
            return 2;
        }
    }
    return 1;
}


```

پاسخ ۶. کمی هم گوگل کنیم!

ابتدا یک تابع برای محاسبه ی مجموع شمارنده های یک عدد می نویسیم. این تابع یک int ورودی گرفته و مجموع شمارنده های آن را در یک متغیر long long ریخته و برمی گرداند.
برای این که این تابع بتواند سریع تر مجموع را محاسبه کند، به جای این که حلقه تا n جلو برود، آن را تا رادیکال n جلو برده، سپس هر شمارنده ای که پیدا شد، اگر دقیقا برابر با رادیکال n نباشد، n تقسیم بر آن عدد نیز یک شمارنده ی بزرگتر از رادیکال n است. بنابراین حلقه سریع تر تمام می شود و همه ی شمارنده ها نیز محاسبه می شوند.

```
long long sumDivisor(int n) {
    long long sum = 0;
    for (int i = 1; i * i < n; i++) {
        if (i * i == n) {
            sum += i;
        } else if (n % i == 0) {
            sum += i + (n / i);
        }
    }
    return sum;
}

```

سپس ۲ تابع findDivisorSum را برای هر حالت s می نویسیم و برای این که ۲ تابع با اسم یکسان بتوانند ورودی های متفاوت داشته باشند، آنها را درون #if می گذاریم. با این کار، اگر مقدار s صفر باشد فقط تابع اول و اگر مقدار آن یک باشد فقط تابع دوم کامپایل می شود. بنابراین مشکلی برای داشتن دو تابع با اسم یکسان وجود ندارد چون فقط یکی از آنها کامپایل می شود.

```
#if s == 0
long long findDivisorSum(char c, int num1, int num2, int num3) {
    if(c == 'm') {
        long long s1 = sumDivisor(num1);
        long long s2 = sumDivisor(num2);
        long long s3 = sumDivisor(num3);
        long long res = s1;
        if (s2 < res) {
```

```
            res = s2;
        }
        if (s3 < res) {
            res = s3;
        }
        return res;
    } else {
        long long s1 = sumDivisor(num1);
        long long s2 = sumDivisor(num2);
        long long s3 = sumDivisor(num3);
        long long res = s1;
        if (s2 > res) {
            res = s2;
        }
        if (s3 > res) {
            res = s3;
        }
        return res;
    }
}

```

در این تابع مجموع سه عدد محاسبه شده و براساس مقدار c یکی از آنها برگردانده شده است.

در تابع بعد می خواهیم تعداد ورودی های نامعین داشته باشیم. برای این کار، از کتابخانه ی stdarg.h استفاده می کنیم. برای تعیین نامعین بودن تعداد ورودی ها، تابع را به شکل زیر تعریف می کنیم.

```
    long long findDivisorSum(int n, ...) {}

```

برای استفاده از stdarg ابتدا یک متغیر از نوع va_list ساخته سپس این متغیر و تعداد ورودی ها را به تابع va_start می دهیم. در نهایت هر بار می توان با دادن این متغیر و نوع ورودی (در این تابع همه ی ورودی ها int هستند)، به تابع va_arg این مقدار را خوانده و به وسیله ی تابع sumDivisor مقدار مجموع شمارنده های آن را محاسبه می کنیم.

```
#else
long long findDivisorSum(int n, ...) {
    va_list ap;
```

```
    va_start(ap, n);
    long long min = -1;
    for (int i = 0; i < n; i++) {
        int num = va_arg(ap, int);
        long long sum = sumDivisor(num);
        if (min == -1 || sum < min) {
            min = sum;
        }
    }
    return min;
}
#endif

```

برای تعریف calc، کافیست یک ماکرو تعریف کنیم که باعث می شود قبل از کامپایل، خود کامپایلر به جای calc(a, b, op) عبارت a op b را قرار دهد. بنابراین بدون تعریف کردن تابع جدید، این عبارت به درستی پردازش می شود.

```
    #define calc(a, b, op) a op b

```