به نام خدا

درس مبانی برنامه‌سازی

تمرین ۱

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نیم سال اول ۰۰ـ۹۹

استاد:

رضا فکوری

مهلت ارسال:

۷ آذر ـ ساعت ۲۳:۵۹:۵۹

مسئول تمرین‌ها:

امیرمهدی نامجو، پرهام صارمی

مسئول تمرین ۱:

عرشیا اخوان

طراحان تمرین ۱:

امیرحسین باقری، علی حاتمی، مریم‌السادات رضوی، علیرضا حسین‌خانی

فهرست

نکات قابل توجه ۲

سوالات ۳
سوال ۱. ورژن نجات بخش ۳
سوال ۲. دیوارهای همگرا ۶
سوال ۳. نگهبان مزاحم ۸
سوال ۴. کلید رهایی ۱۰

پاسخ‌ها ۱۲
پاسخ ۱. ورژن نجات بخش ۱۲
دیوار نوشته اول ۱۲
دیوار نوشته دوم ۱۴
دیوار نوشته سوم ۱۶
دیوار نوشته چهارم ۱۸
دیوار نوشته پنجم ۱۹
پاسخ ۲. دیوارهای همگرا ۲۰
پاسخ ۳. نگهبان مزاحم ۲۱
پاسخ ۴. کلید رهایی ۲۴

نکات قابل توجه

- توجه کنید که برای این سوالات حق استفاده از حلقه، شرط، تابع، آرایه و عملگر تقسیم را ندارید. در صورت استفاده از موارد ذکر شده، حتی در صورت اخذ نمره کامل در سامانه کوئرا، نمره صفر برای آن سوال در نظر گرفته خواهد شد.
- برای سوالات ۲ تا ۴، باید کدی متناسب با خواسته برنامه بنویسید و آن را در سامانه کوئرا ارسال کنید. سامانه کوئرا به طور خودکار کد شما را اجرا کرده و به آن نمره خواهد داد. تا قبل از اتمام مهلت تمرین می‌توانید به هر تعداد که می‌خواهید پاسخ‌های جدید ارسال کرده و در صورت وجود اشتباه پاسخ خود را اصلاح کنید.
- پیشنهاد می‌کنیم که ابتدا ۳ سوال ۲ تا ۴ را حل کنید و سپس به حل سوال اول بپردازید.
- برای سوال اول فایل تحلیل کد خود را در قالب zip آپلود کنید. می‌توانید تایپ کنید و یا دست نویس تحویل دهید. اگر دست‌نویس تحویل می‌دهید فایل دست نویس خود را به صورت خوانا نوشته و اسکن شده آن را ارسال فرمایید.
- ورودی های لازم برای نمونه کدهای سوال اول ذیل قطعه کد به شما داده شده است. توجه کنید که لزوما همه ورودی‌های داده شده در سوال مورد استفاده قرار نمی‌گیرند.

سوالات

سوال ۱. ورژن نجات بخش

همانطور که میدانید سازنده زبان سی Dennis MacAlistair Ritchie است. دنیس که اعلام کرده است ورژن جدید این زبان موجب تحول عظیمی در برنامه نویسی و به تبع آن در دنیا خواهد شد اکنون به مصر سفر کرده است تا از اهرام سحر آمیز مصر باستان دیدن کند. در خلال این سفر ناگهان دنیس ناپدید می‌شود. کارآگاهان بسیاری به دنبال وی راهی اهرام مصر می‌شوند اما هر کدام در همان مراحل ابتدایی متوقف می‌شوند. هم اکنون که دنیا با بحران های زیادی همچون گرمایش زمین و کرونا مواجه است گروهی از برنامه نویسان معتقدند باید با یک تفکر برنامه نویسی به دنبال دنیس رفت تا وی ورژن نجات بخش سی را به دنیا عرضه کند.

بسیاری از برنامه‌نویسان به دنبال دنیس رفته‌اند اما هر کدام به دلیل نداشتن تبحر کافی در دیوار های اهرام مصر مرده‌اند. حالا حدس‌های بسیاری بر اینکه دنیس نیز مرده است وجود دارد.

اما شما معتقدید دنیس قطعا پیش از مرگ خود نسخه را در جایی از هرم قرار داده است. بنابراین شما این خطر را به جان می‌خرید و پا در هرم خوفو، بزرگ‌ترین هرم مصر می‌گذارید زیرا آخرین نشانه های دنیس در آنجا گزارش شده‌اند.

در ابتدا شما به دیواری مستحکم برمی‌خورید. دیوار اسرار آمیز که تنها زیر نور ماه می‌توان نوشته‌هایش را خواند و فرصت ورود به آن زیاد نیست. تنها تا ددلاین تمرین ۱ مبانی بچه های شریف فرصت هست (اهرام به شیوه خودشان کار می‌کنند) حالا شما باید مراحل زیر را بگذرانید و با حل کردن هر نوشته دیوار جواب را به دیوار بگویید تا دیوار اسرارآمیز کنار برود و شما بتوانید به مراحل بعدی جست‌وجوی خود ادامه دهید.

روی هر کدام از دیوارهای هرم کدی نوشته شده است که دنیس آن را طراحی کرده است تا دست نااهلان به ورژن نهایی نرسد. شما باید برای ورود به دیوار ابتدا لیاقت خود را ثابت کنید و این کد ها را تحلیل کنید به دیوار بگویید تا وارد آن شوید.

دیوار نوشته اول

```
#include <stdio.h>

int main() {
  int a = 1;
  printf("Dennis says\n") || scanf("%d",&a);
  printf("%d\n",a<<4 | a>>4 && printf("The only way to learn a new
   programming language is by writing programs in it?\n"));
  a = printf("") & printf("walls are coming \n") || scanf("%d",&a);
  (a & 1) ? printf("BE\n") : printf("prepared\n");
  printf("%d",a);
}
```

ورودی :

```
  5 12
```

دیوار نوشته دوم

```
#include <stdio.h>

int main() {
  int x = 2,y = 3,z = 4, w = 12.8;
  x = 230 >> !y + x | z && 50 % 13 / (int)w++  + x + y * w;
  printf("%d", x);

  x = 1, y = 3;
  z = y <<  x;
  y = z * (y-- * (++x));
  x = x >> y;
  printf("%d", y);

  int a = 5;
  printf("%d",a<<2 + 3>>a | a<<2 ^ a<<3 & a<<9 - 1) ;
}
```

دیوار نوشته سوم

```
#include <stdio.h>

int main() {
  int b, c, d, e, f, g;
  printf("%d", e);
  printf("%d%d%d%d",  scanf("%d %d",&b, &c),
  scanf("%d %d",&d, &e),
  scanf("%d %*d", &f, &g, &e),
  printf("AB\x9ZYX\bGHI\0JKLMNOP"));
  printf("%d", e);
}
```

ورودی :

```
1 1 1 l 1 1
```

دیوار نوشته چهارم

```
#include <stdio.h>

int main() {
  float ff = -11654.2;
  short s = (short) ff;
  unsigned short us = (unsigned short)s;
  char character = (char) us;
  printf("%c", character);
}
```

دیوار نوشته پنجم

```
#include <stdio.h>

int main() {
  int a = 5;
  printf("%d",a<<2 + 3>>a | a<<2 ^ a<<3 & a<<9 - 1);
  a = 17;
  printf("%d",a<<2 + 3>>a | a<<2 ^ a<<3 & a<<9 - 1);
}
```

سوال ۲. دیوارهای همگرا

پس از عبور از دیوار اول شما خیال می‌کنید کار تمام شده و همین حالاست که روح دنیس با در دست داشتن لوح حاوی کد به استقبال‌تان بیاید. در حالی که به موفقیت خود می‌بالید خود را در تالاری وسیع و خالی می‌یابید که دیوار های آن با کتیبه های عجیب و غریب مزین شده. سرگردان به این سو و آن‌سو می‌روید تا بلکه راه خروجی از این تالار پیدا کنید که ناگهان احساس می‌کنید دیوار های اطراف‌تان دیوار های معمولی نیستند و با سرعتی نه چندان آهسته در حال حرکت به سمت شما هستند. رعب و وحشت وجود شما را فرا گرفته. در همین حال صدایی ناشناس از مکانی که متوجه نمی‌شوید کجاست شروع به پخش شدن می‌کند و در کمال تعجب شما زبان صدا را متوجه می‌شوید.

صدا برای شما توضیح می‌دهد که آقای ریچی در آخرین حضورش در این تالار کیبرد قدیمی خود که متعلق که دهه ۱۹۶۰ میلادی‌ست اینجا جا گذاشته و حال تنها با وارد کردن ترجمه رمزی که روی کتیبه مرمرین حک شده است دیوار ها متوقف شده و دریچه خروج باز می‌شود...

این رمز عبارت است از یک عبارت ۱۳ کاراکتری که در ابتدای رشته عجیبی از حروف کوچک و بزرگ انگلیسی و علائم و نمادها آمده است. نکته اینجاست که وارد کردن این ۱۳ کاراکتر به خودی خود هیچ کمکی به شما نمی‌کند چرا که با استفاده از تکنیکی موسوم به کد سزار (که ره‌آورد سفر دیپلماتیک ژولیوس سزار، رقیب سرسخت فراعنه در تسخیر حکومت کل جهان، به اهرام است) رمزنگاری شده. میزان انتقال این رمزنگاری، یک عدد صحیح است که بعد از یک کاراکترِ فاصله، در رشته حک شده روی کتیبه وجود دارد و در حقیقت، این تنها عدد موجود در آن رشته است.

شما در آن شرایط سخت تصمیم بزرگی می‌گیرید. نوشتن برنامه ای پیش از له شدن میان دیوار ها، که با ورودی گرفتن متن کتیبه، رمز ترجمه شده را در خروجی چاپ کند...

کد سزار : یک روش رمزگذاری که هر کاراکتر را به اندازه x کاراکتر در جدول اسکی جا به جا میکند تا به کاراکترِ رمز متناظر برسد. (از جست‌وجو و کسب اطلاعات بیشتر درباره این تکنیک دریغ نکنید :))

ورودی

ورودی تنها شامل یک خط است که در آن رشته حک شده روی کتیبه مرمرین آمده است. شما اطمینان دارید که کاراکتر های جا به جا شده از جدول اسکی بیرون نمی‌زنند. همچنین تضمین می‌شود همواره بعد از رمز و قبل از کلید رمزنگاری لااقل یک کاراکتر وجود دارد.

خروجی

در تنها خط خروجی رمز ۱۳ کاراکتری ترجمه شده چاپ می‌شود.

مثال

ورودی نمونه ۱

```
AbkkfpFpAb^a<This_is_the_tomb_of_Pharaoh 3Be_Careful
```

خروجی نمونه ۱

```
DennisIsDead?
```

در این نمونه ۱۳ کاراکتر اول کتیبه که همان رمز باشد `AbkkfpFpAb^a<` است و کلید رمزنگاری هم عدد ۳+ است بنابراین هر کاراکتر از رمز به ۳ کاراکتر بعد از خودش ترجمه می‌شود.

ورودی نمونه ۲

```
HtijNx\fnynsl_RITCHIES_SOUL_IS_IN_TORMENT -5Hurry_UP!
```

خروجی نمونه ۲

```
CodeIsWaiting
```

در این نمونه هم رمز `HtijNx\fnynsl` و کلید رمز ۵_ است. پس هر کاراکتر به ۵ کاراکتر قبل از خودش در جدول اسکی ترجمه می‌شود.

سوال ۳. نگهبان مزاحم

شما در حال جست‌وجو برای کشف آخرین نسخه زبان C هستید! متاسفانه نگهبانان اهرام متوجه حضور شما شده‌اند و به شما توصیه میکنیم قبل از این که به اتهام ورود غیرمجاز به اهرام بازداشت شوید، رمز محرمانه‌ی ورودی را محاسبه کنید! در داستان‌های مصر باستان نوشته شده که این رمز روش محاسبه‌ی جالبی داشته است و برای هر راهرو چیزی نبوده به جز دترمینان ماتریس ۳ × ۳ ای که روی دیوار ورودی آن نوشته شده است! پس از حل تمامی دترمینان‌ها، نگهبانان ۹ کلید به شما می‌دهند که در سوال بعد به آن‌ها نیاز خواهید داشت :) موفق باشید!

نکته مهم

دقت کنید! در این تمرین مجاز به استفاده از عملگر ـ (تفریق) نیستید! درصورت استفاده نیمی از نمره کسر خواهد شد!

ورودی

ورودی شامل سه سطر است که در هر سطر سه عدد صحیح با فاصله از هم آمده است.

$$a_{ij} \in \mathbb{Z}$$

$$-1000 \leq a_{ij} \leq 1000$$

خروجی

خروجی برنامه شما باید دترمینان ماتریس ورودی باشد.

مثال

ورودی نمونه ۱

```
1 2 3
4 5 6
7 8 9
```

خروجی نمونه ۱

```
0
```

ورودی نمونه ۲

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

خروجی نمونه ۲

```
-140
```

میتوانید از ابزارهای آنلاین برای صحت‌سنجی دترمینان محاسبه شده توسط برنامه خود استفاده کنید :)

سوال ۴. کلید رهایی

شما اکنون وارد تالار اصلی هرم شده‌اید پس از گذراندن مراحل قبل، حال در تالار مرکزی ایستاده‌اید و به آنچه روبه‌روی‌تان قرار دارد می‌نگرید یک تالار خالی که تنها یک مومیایی داخل آن قرار دارد. دیگر خسته و ناامید شده‌اید و نمی‌دانید که اکنون باید چه‌کار کنید! درب پشت شما نیز قفل شده است و راه بازگشت ندارید. همان‌طور که به اطراف تالار نگاه می‌کنید زمان نیز می‌گذرد و ماه در آسمان پدیدار می‌شود و از روزنه‌ای بالای تالار، پرتوی نوری روی مومیایی می‌افتد و شما متوجه نوشته های روی مومیایی می‌شوید. شما می‌دانید این نیز ماجراجویی و خطر دیگری‌ست اما قصد ندارید که عقب بکشید و دنیا را به حال خود رها کنید.

نوشته‌های روی مومیایی به یک قفل اشاره دارند که قفل همان مومیایی‌ست. بله! درست حدس زدید! این مومیایی خود دنیس است و ورژن نهایی نیز درون آن قرار دارد. شما باید از آخرین تست آقای دنیس نیز به سلامت عبور کنید و دنیا را نجات دهید.

نوشته ها از دسته کلیدی حکایت می‌کنند که همراه شماست و شما آنها را از مراحل قبل با خود آورده‌اید کلید ها از ۰ تا ۸ شماره‌گذاری شده‌اند و شما نمی‌توانید همه آن‌ها را امتحان کنید زیرا با اولین انتخاب غلط، شما به طرز اسرارآمیزی خواهید مرد.

مومیایی‌ها علاقه زیادی به اعداد یک‌رقمی دارند و اعداد اول را به شیوه خودشان یک‌رقمی می‌کنند به طوری که همه رقم های عدد مورد نظر را جمع می‌کنند و این کار را تا جایی تکرار می‌کنند که به عددی یک رقمی برسند. شما نیز باید عدد روی مومیایی را به شیوه بالا رمز کنید و عدد یک رقمی متناسب با آن را پیدا کنید. این عدد، همان شماره کلید مربوطه است.

کلید را پیدا کنید و دنیا را نجات دهید.

ورودی

ورودی تنها شامل یک عدد اول است.

$$1 \leq n \leq 10000000000000000000$$

خروجی

خروجی برنامه یک عدد یک رقمی ایست که کلید رهایی ماست.

مثال

ورودی نمونه ۱

```
5
```

خروجی نمونه ۱

```
5
```

ورودی نمونه ۲

```
559177
```

خروجی نمونه ۲

```
7
```

جمع ارقام ۵، ۵، ۹، ۱، ۷ و ۷ می‌شود ۳۴ سپس جمع ارقام ۳ و ۴ ، ۷ می‌شود.

پاسخ‌ها

پاسخ ۱. ورژن نجات بخش

دیوار نوشته اول

```
#include <stdio.h>

int main() {
  int a = 1;
  printf("Dennis says\n") || scanf("%d",&a);
  printf("%d\n",a<<4 | a>>4 && printf("The only way to learn a new
   programming language is by writing programs in it?\n"));
  a = printf("") & printf("walls are coming \n") || scanf("%d",&a);
  (a & 1) ? printf("BE\n") : printf("prepared\n");
  printf("%d",a);
}
```

ورودی :

```
  5 12
```

اولین printf خروجی غیر صفر دارد و برای عملگر OR منطقی مقدار «درست»، و چون طرف چپ عملیات «یا» برابر با مقدار درست است سمت راست، دیگر اجرا نخواهدشد. عبارتی را که printf بعدی میخواهد آنرا چاپ کند پرانتز گذاری می‌کنیم:

```
    ((a << 4) | (a >> 4)) && printf("The only way to learn a new
   programming language is by writing programs in it?\n")
```

در حینی که برنامه در حال محاسبه این عبارت ریاضی است به printf برمی‌خورد و ابتدا آنرا چاپ می‌کند (سمت چپ AND عبارتی غیر صفر است)، سپس مقدار آنرا در عبارت قرار می‌دهد. مقدار نهایی عبارت که برابر با مقدار «درست» است (یعنی ۱) در خروجی چاپ خواهدشد.

در خط بعدی ابتدا سمت چپ OR منطقی اجرا می‌شود. سمت چپ AND بیتی یک printf خالی است که مقدار ۰ را برمی‌گرداند و سمت دیگر آن printf دیگری است که اجرا شده و مقدار غیر صفر برمی‌گرداند. AND بیتی این دو عدد برابر با صفر خواهدبود پس سمت راست OR منطقی نیز اجرا می‌شود و عدد ۵ از خروجی خوانده می‌شود. خروجی

عملیات اسکن ۱ است چون scanf توانسته است یک متغیر را با موفقیت بخواند. نتیجه نهایی OR منطقی در این عبارت «درست» خواهد بود و ۱ درون a ریخته می‌شود. AND بیتی a و ۱ برابر با ۱ خواهد بود که «درست» طلقی می‌شود و بنابر این BE چاپ خواهدشد. در نهایت a چاپ شود و خروجی نهایی برابر خواهد بود با:

```
 Dennis says
 The only way to learn a new programming language is by writing
programs in it?
 1
 walls are coming
 BE
 1
```

دیوار نوشته دوم

```
#include <stdio.h>

int main() {
  int x = 2,y = 3,z = 4, w = 12.8;
  x = 230 >> !y + x | z && 50 % 13 / (int)w++  + x + y * w;
  printf("%d", x);

  x = 1, y = 3;
  z = y <<  x;
  y = z * (y-- * (++x));
  x = x >> y;
  printf("%d", y);

  int a = 5;
  printf("%d",a<<2 + 3>>a | a<<2 ^ a<<3 & a<<9 - 1) ;
}
```

در خط اول متغیر ها از جنس int تعریف و مقداردهی اولیه می‌شوند. در انتصاب متغیر w کست‌ضمنی اتفاق می‌افتد و مقدار ۱۲ درون آن قرار می‌گیرد. در خط دوم اگر با توجه به اولویت عملگر ها عبارتمان را پرانتز گذاری کنیم به عبارت زیر خواهیم رسید:

```
(230 >> (!y + x | z )) && (((50 % 13) / ((int)w++)  + x) + (y * w)))
```

متغیر y دارای مقداری غیر از صفر است و «درست» طلقی می‌شود و با عملیات نقیض منطقی (!) تبدیل به ۰ یا همان نادرست می‌شود. OR بیتی ۲ و ۴ برابر با ۶ خواهدبود و سمت چپ AND خواهدشد:

```
230 >> 6
```

که مقدار آن برابر با ۳ خواهد بود (محاسبه این مقدار لزومی نداشت چون ۲۳۰ از ۶۴ بیشتر است و ۶ بار شیفت دادن بیت‌های آن آنرا تبدیل به صفر نخواهد کرد). چون سمت چپ AND یک عدد غیر از صفر است، «درست» طلقی می‌شود. سمت راست نیز برابر با ۴۱ خواهدبود که آن هم مقداری غیر از صفر است و درنهایت مقدار AND منطقی این دو عدد برابر با مقدار «درست» یعنی ۱ خواهد بود. این مقدار یک به x منتصب می‌شود و در خط بعد عدد ۱ در خروجی چاپ می‌شود.

در خط بعد مقداری جدید به x و y انتصاب می‌یابد. ضرب ۳ در ۲ (هر شیفت به سمت چپ در صورت عدم سرریز، عدد را دو برابر می‌کند :)) که مقدار ۶ خواهد داشت درون z ریخته می‌شود. در عملیات کاهش پسوندی ابتدا مقدار متغیر در عبارتی که در حال محاسبه است جاگذاری می‌شود و سپس یک واحد کاهش پیدا می‌کند (یعنی مقدار y در عبارت جاگذاری می‌شود و بلافاصله پیش از جاگذاری دیگر متغیر ها مقدار آن یک واحد کاهش پیدا می‌کند). در افزایش پیشوندی عکس این روند اتفاق می‌افتد. پس در آخر عبارت ۶ در ۳ در ۲ محاسبه و درون y ریخته می‌شود (دقت کنید که مقدار y قبل از این انتصاب برابر با ۲ بود که حاصل از عملیات کاهش پیشوندی بوده است). حال x که در این لحظه مقدار ۲ درون آن است ۳۶ بار به راست شیفت داده می‌شود و نتیجه‌اش (یعنی ۰) درون x ریخته می‌شود. در خط بعدی y که برابر با ۳۶ است در خروجی چاپ خواهد شد.

برای بدست آوردن عبارتی که در آخرین printf چاپ می‌شود، ابتدا آنرا پرانتز گذاری می‌کنیم :

```
((a << (2 + 3)) >> a) | ( (a<<2 ) ^ ((a << 3 ) & (a << (9 - 1) )))
```

که پس از این پرانتزگذاری به راحتی می‌توان دریافت که مقدار آن برابر با ۲۱ خواهد بود و ۲۱ در خروجی چاپ خواهدشد. پس خروجی نهایی که در خروجی نمایش داده می‌شود :

```
13621
```

دیوار نوشته سوم

```
#include <stdio.h>
int main() {
  int b, c, d, e, f, g;
  printf("%d\n", e);
  printf("%d %d %d %d",  scanf("%d %d",&b, &c),
  scanf("%d %d",&d, &e),
  scanf("%d %*d", &f, &g, &e),
  printf("AB\x9ZYX\bGHI\0JKLMNOP"));
  printf("\n%d", e);
}
```

ورودی :

```
1 1 1 l 1 1
```

در خط اول متغیر هایی تعریف شده اند. و چون مقداری به آنها داده نشده است مقداری که از قبل در خانه های حافظه قرار داشته است همچنان باقی می‌ماند.(برای درک بیشتر c garbage values in را سرچ کنید).

پس بنابراین در خط بعدی مقداری نامشخص جاپ می‌شود و به خط بعدی می‌رود.سپس به printf می‌رسیم که ۴ مقدار را قرار است که بر روی کنسول چاپ کند. ما فرض می‌کنیم که برنامه از سمت راست به چپ شروع به اجرای مقادیر عبارات درون printf می‌کند(این اتفاق، اتفاقی وابسته به کامپایلر است و شرح دقیق در زبان c برای اجرای آن نیامده است. ممکن است در اگر برنامه را توسط کامپایلر دیگری اجرا می‌کردیم به جواب دیگری می‌رسیدیم که البته تمام آنها جوابی درست هستند).

ابتدا `printf("AB\x9ZYX\bGHI\0JKLMNOP")` اجرا می‌شود. AB چاپ می‌شود. سپس طبق تعریف `\x` مقدار مربوط به کد اسکی عدد مبنای ۱۶ جلوی خود را چاپ می‌کند که در جدول اسکی معادل یک horizantal tab است. سپس ZYX چاپ می‌شود. اسکیپ کاراکتر b که یک بک‌اسپیس است. سپس GHI و بعد از `\0` دیگر چیزی چاپ نمی‌شود و printf مقدار ۱۰ کاراکتر چاپ کرده است و آنرا برمی‌گرداند. کاراکتر هایی که چاپ کرده (آنها را از فرمت خوانده و عملیات مربوط به آن را انجام داده) عبارتند از:

```
A, B, \x9, Z, Y, X, \b, G, H, I
```

سپس scanf ها اجرا می‌شوند. اولین scanf از سمت چپ مقدار یک را میخواند و در f میریزد سپس مقدار بعدی را می‌خواند اما در متغیری نمی‌ریزد و خروجی آن تعداد متغیر هایی است که مقدار آن با موفقیت از ورودی خوانده شده و به منتصب شده اند (یعنی یک). اینکه متغیر های بیشتری به تابع پاس داده‌شده‌است تاثیری در عملکرد تابع نخواهد داشت و ملاک عملکرد تابع فرمت اولیله‌ای است که به آن داده‌می‌شود. سپس scanf بعدی اجرا می‌شود سومین یک را از ورودی خوانده و در d می‌ریزد سپس به حرف l می‌رسد و چون قصد خواندن عدد را دارد همانجا متوقف می‌شود و مقدار یک برمیگرداند (تعداد متغیر هایی که با موفقیت از ورودی خوانده است) و مقدار l در ورودی باقی می‌ماند تا در آینده خوانده‌شود ولی scanf بعدی هم انتظار عدد دارد و نمی‌تواند حرف l را بخواند متوقف شده و مقدار صفر بر می‌گرداند حال دیگر مقادیری که printf اولیه قرار است چاپ کند مشخص شده و این خط نیز به پایان می‌رسد و در خط آخر نیز چون همچنان e مقداردهی نشده‌است، مقداری که در آن وجود دارد چاپ می‌شود (که معادل با اولین خطی است که در خروجی داریم).

یک نکته

کاراکتر \b در حقیقت چیزی را پاک نمی‌کند و تنها cursor را یک خانه به عقب می‌برد. حال اگر در ادامه کاراکتری چاپ شود از آن نقطه به بعد روی کاراکتر(های) قبلی نوشته می‌شود و اگر کاراکتری وارد نشود همانطور باقی‌خواهدماند.

دیوار نوشته چهارم

```
#include <stdio.h>

int main() {
  float ff = -11654.2;
  short s = (short) ff;
  unsigned short us = (unsigned short)s;
  char character = (char) us;
  printf("%c", character);
}
```

عددی که در ابتدا در اختیار داریم ۱۱۶۵۴ـ است. چون جزء صحیح این مقدار در short جای می‌گیرد در هنگام کست‌شدن، قسمت صحیح آن (۱۱۶۵۴ـ) درون متغیر s ریخته می‌شود. مقدار باینری آن که مکمل دو است برابر خواهد بود با:

```
1101 0010 0111 1010
```

زمانی که این متغیر به نوع unsigned خود کست می‌شود، همان بیت ها درون متغیر جدید کپی می‌شود اما این بار مکمل دو نخواهد بود و یک عدد بدون علامت است:

```
b'1101 0010 0111 1010' = 53882
```

حال کست به char انجام می‌شود. char یک متغیر یک بایتی است و برای مقدار دهی برنامه هشت بیت سمت راست عددمان را برمی‌دارد که برابر خواهد بود با:

```
b'0111 1010' = 122
```

و عدد ۱۲۲ در جدول اسکی شماره کاراکتر z است پس این کاراکتر در خروجی چاپ خواهد شد.

دیوار نوشته پنجم

```
#include <stdio.h>

int main() {
  int a = 5;
  printf("%d",a<<2 + 3>>a | a<<2 ^ a<<3 & a<<9 - 1);
  a = 17;
  printf("%d",a<<2 + 3>>a | a<<2 ^ a<<3 & a<<9 - 1);
}
```

تنها کافیست اولویت عملگر هارا رعایت کنیم. برای این منظور عبارات را پرانتزگذاری می‌کنیم:

```
a = 5
(((a<<(2 + 3))>>a) | ((a<<2) ^ ((a<<3) & (a<<(9 - 1))))) = 21
```

```
a = 17
(((a<<(2 + 3))>>a) | ((a<<2) ^ ((a<<3) & (a<<(9 - 1))))) = 68
```

پاسخ ۲. دیوارهای همگرا

نمونه پاسخ مورد قبول:

```
#include <stdio.h>
int main() {
  int k;
  char c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13;
  scanf("%c%c%c%c%c%c%c%c%c%c%c%c%c%*s%d%*s",
  &c1,&c2,&c3,&c4,&c5,&c6,&c7,&c8,&c9,&c10,&c11,&c12,&c13,&k);
  printf("%c%c%c%c%c%c%c%c%c%c%c%c%c",
  c1+k,c2+k,c3+k,c4+k,c5+k,c6+k,c7+k,c8+k,c9+k,c10+k,c11+k,c12+k,c13+k);
}
```

در ابتدا باید ۱۳ کاراکتر اول کتیبه را به عنوان رمز دریافت کنیم. برای این کار کافی‌ست ۱۳ متغیر از نوع char تعریف کنیم و هر یک از آن‌ها را با استفاده از specifier مربوط به کاراکتر، %c ، دریافت کنیم. در ادامه کتیبه، بعد از ۱۳ کاراکتر ابتدایی، یک رشته با طول نامشخص از حروف و علائم آمده‌است، پس از آن یک فاصله و سپس کلید رمزگذاری. بدیهی‌ست که ما احتیاجی به ذخیره آن رشته نامشخص نداریم و کافی‌ست از روی آن عبور کنیم و آن را نادیده بگیریم. برای این‌کار باید از نماد * و specifier مربوط به رشته، %s ، استفاده کنیم. %s در حالت عادی یک رشته از کاراکتر ها را تا جایی که به اولین whitespace برسد می‌خواند. حالا با اضافه کردن علامت * بین % و s در واقع این رشته خوانده‌شده نادیده گرفته‌می‌شود و در جایی ذخیره نخواهدشد. اکنون می‌توانیم ادامه ورودی، که ما تنها به عدد رمز در ابتدای آن را احتیاج داریم، را بخوانیم. پس از خواندن کتیبه از ورودی، برای ترجمه آن ۱۳ کاراکتر با کلید رمز، کافی‌ست هر کاراکتر را با عدد رمز جمع کنیم. با چاپ کردن این کاراکتر ها با فرمت %c پس از جمع کردن با کلید رمز، در حقیقت داریم هر کاراکتر را به اندازه این کلید در جدول اسکی جابه‌جا می‌کنیم.

پاسخ ۳. نگهبان مزاحم

برای حل این سوال کافی بود درایه های ماتریس را ورودی بگیریم و با محاسبه دترمینان، خروجی را چاپ کنیم (با کد زیر تا اینجا نیمی از سوال حل شده!)

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

int main()
{
  int a11, a12, a13, a21, a22, a23, a31, a32, a33;
  scanf("%d %d %d\n%d %d %d\n%d %d %d", &a11, &a12, &a13, &a21, &a22, &
   a23, &a31, &a32, &a33);
  int result = a11*a22*a33 + a12*a23*a31 + a13*a21*a32 - (a13*a22*a31 +
   a11*a23*a32 + a12*a21*a33);
  printf("%d", result);
  return 0;
}
```

اما بخش اصلی و نکته دار سوال، مربوط به محدودیت استفاده از کاراکتر ـ و هدف از این کار، حل سوال به کمک جمع و تفریق در سیستم مکمل دو بوده است که اغلب کامپیوترهای امروزی از آن برای ذخیره اعداد استفاده می‌کنند. برای محاسبه قرینه یک عدد باینری در سیستم مکمل دو، میتوانیم تمامی بیت‌ها را قرینه کرده (معادل کاری که عمگر نقیض بیتی انجام میدهد) و سپس عدد ۱ را به آن اضافه کنیم؛ البته روشهای دیگری نیز وجود دارند (مثلا میتوانیم عدد منفی یک را در سیستم مکمل دو بسازیم (همه بیتهای اینتیجر یک میشوند) و سپس در عددی که میخواهیم قرینه شود ضرب باینری کنیم)

$$-a = \sim a + 1$$

یادآوری میشود که در سیستم مکمل دو برای تفریق دو عدد کافی است عدد اول را با قرینه عدد دوم جمع کنیم. در قسمت زیر یک نمونه کد صحیح و پس از آن چند نمونه از کدهایی که شما ارسال کردید و صحیح هستند آورده شده است:

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

```
int main()
{
  int a11, a12, a13, a21, a22, a23, a31, a32, a33;
  scanf("%d %d %d\n%d %d %d\n%d %d %d", &a11, &a12, &a13, &a21, &a22, &
    a23, &a31, &a32, &a33);
  int result = a11*a22*a33 + a12*a23*a31 + a13*a21*a32 + (~(a13*a22*a31
    + a11*a23*a32 + a12*a21*a33) + 1);
  printf("%d", result);
  return 0;
}
```

در هر سه کد زیر که دوستان شما ارسال کرده اند ابتدا منهای یک در سیستم مکمل دو (همه بیتها باید ۱ باشند) به روشهای جالبی! ساخته شده و برای قرینه کردن اعداد در آنها ضرب شده است.

```
#include <stdio.h>
int main(){
  unsigned short us = 32768;
  short b = (short)us / 32768;
  int a1,a2,a3,a4,a5,a6,a7,a8,a9;
  scanf("%d %d %d %d %d %d %d %d %d",&a1,&a2,&a3,&a4,&a5,&a6,&a7,&a8,&a9
    );
  int matrix;
  matrix=(a1 *(a5 * a9 + b*a6*a8) + b*a2*(a4*a9 +b*a6*a7) +a3*(a4*a8 +b*
    a5*a7));
  printf("%d" , matrix);
}
```

```
#include <stdio.h>
int main() {
  int s; short int m = 1;
  short int a; short int b; short int c;
  short int d; short int e; short int f;
  short int g; short int h; short int i;
  scanf("%hd%hd%hd%hd%hd%hd%hd%hd%hd", &a, &b, &c, &d, &e, &f, &g, &h, &
    i);
  m = 1 << 15;
  m += 32767;
  s = a * e * i + m * a * h * f + b * f * g + m * b * d * i + c * d * h
    + m * c * e * g;
  printf("%d", s);
```

```
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  long long int z = '«'/85;
  long long int a11, a12, a13, a21, a22, a23, a31, a32, a33;
  scanf("%lld%*c%lld%*c%lld", &a11, &a12, &a13);
  scanf("%lld%*c%lld%*c%lld", &a21, &a22, &a23);
  scanf("%lld%*c%lld%*c%lld", &a31, &a32, &a33);
  long long int det = a11*a22*a33+a12*a23*a31+a13*a21*a32+z*(a13*a22*a31
   +a12*a21*a33+a11*a23*a32);
  printf("%lld", det);
  return 0;
}
```

پاسخ ۴. کلید رهایی

نمونه پاسخ مورد قبول:

```
#include <stdio.h>

int main(void)
{
  long long int n;
  scanf("%lld",&n);
  printf("%lld",n%9);
}
```

عدد ورودی اول است بنابراین باقی مانده به ۹ آن هیچ وقت صفر نمی‌شود.می‌دانید برای باقی مانده ۹ ابتدا ارقام را جمع و سپس باقی مانده به ۹ می‌گیریم. حال دیگر کمی فکر کنید. آنقدر جمع می‌کنیم تا به یک عدد یک رقمی برسیم. یعنی دقیقا داریم باقی مانده به ۹ را محاسبه می‌کنیم.