باسمه تعالی

دوره‌ی آموزشی المپیاد کامپیوتر

# آزمون میان دوره‌ای درس زبان

چهارشنبه ۲ مرداد ۱۳۸۶

نصیری‌شرق — وقت: ۳ + ۱۲۰ دقیقه

لطفاً قبل از آغاز حل مسائل به نکات زیر توجه کنید:

۱) در سؤال‌هایی که با علامت (♠) مشخّص شده و درباره‌ی صحت کارکرد یک برنامه پرسیده شده است

- در صورتی که کد کامپایل نمی‌شود، محل و علّت خطای کامپایل[۱] را مشخّص کرده و معلوم کنید که چگونه باید این خطا را رفع کرد.
- در غیر این صورت اگر کد کامپایل می‌شود اما در هنگام اجرا دچار خطای زمان اجرا[۲] می‌شود، محل و علّت آن خطا را مشخّص کرده و مشخّص کنید که چگونه می‌توان از این خطا اجتناب کرد.
- در غیر این صورت اگر برنامه به‌درستی کامپایل می‌شود اما به هر دلیلی (اعم از پایان نپذیرفتن به‌خاطر افتادن در حلقه‌ی نامتناهی، فرمت اشتباه خروجی و ...) خواسته‌ی برنامه‌نویس را برآورده نمی‌کند، این مورد را به همراه محل و علّت دقیق و نحوه‌ی رفع مشکل بنویسید.
- در غیر این صورت اگر برنامه خروجی درست را تولید می کند اما ناکارآمد است (برای مثال، پیچیدگی زمانی آن به سادگی قابل کاهش است)، با ذکر نحوه‌ی بهبود کد به این مهم اشاره کنید.
- نهایتاً در صورتی که کد درست کار کرده و بهینه هم هست، با ذکر این مورد ادعای خود مبنی بر صحّت و بهینگی آن را تا حد امکان اثبات کنید. ضمناً در صورت نیاز خروجی برنامه را نیز بنویسید.

۲) وقت خود را بین سؤالات تنظیم کنید. جواب بعضی از سؤالات تنها یک یا دو خط بوده و مستقل از نمره و طول صورت سؤال است!

«موفق باشید!»

[۱] Compilation Error
[۲] Runtime Error

## مسئله‌ی صفر) فشرده‌سازی حسین! .....۶ + ۳ + ۵ + ۱۰ + ۲ + ۱۰ + ۳ + ۱ = ۴۰ نمره

حسین می‌خواهد تعداد زیادی عدد صحیح را که هرکدام در بازه‌ی [۰, $۲^{۲۸}$ − ۱] قرار دارند (با یک ترتیب خاص) نگه‌داری کند. بالطبع حسین هم، مانند ما، می‌داند که یک روش نگه‌داری این اعداد استفاده از یک آرایه‌ی int است؛ اما از آن‌جا که تعداد زیادی از اعداد حسین کوچک هستند (و مثلاً در حالت طبیعی در char یا short جا می‌شوند)، حسین نمی‌خواهد برای همه‌ی اعداد ۴ بایت حافظه مصرف کند.

پس از بررسی‌های زیاد، حسین تصمیم گرفت هر عدد را به‌صورت «چند بایت متوالی» کد کند که اوّلاً حافظه‌ی کم‌تری (در مقایسه با حالت تمام int مصرف شود)، ثانیاً آرایه‌ی بایت‌های متوالی (حاصل از چسباندن کدهای این اعداد) قابل تفکیک باشد. دقت کنید که اگر برای مثال، حسین تنها به نوشتن اعداد در حداقل تعداد بایت و چسباندن آن‌ها به‌هم اکتفا کند، آن‌وقت معلوم نیست آرایه‌ی {..., ۱۱۱۱۱۱۱۰, ۰۰۰۰۰۰۰۱} با دو عنصر ۱ و ۲۵۴ (از چپ به راست) شروع می‌شود یا یک عنصر ۵۱۰ یا مقادیر دیگر.

روش حسین چنین است که برای هر یک از اعداد، ابتدا آن عدد را در مبنای دو می‌نویسد. سپس بیت‌های آن‌را در دسته‌های ۷ بیتی (و نه ۸ بیتی) دسته‌بندی می‌کند. نهایتاً قبل از ۷ بیتِ دسته‌ی آخر (کم‌ارزش‌ترین) یک بیت صفر و قبل از ۷ بیتِ سایر دسته‌ها (در صورت وجود) یک بیت ۱ قرار می‌دهد. برای مثال عدد $(۱۰۱۰\ ۱۱۰۱۰۱۱)_۲$ = ۱۳۸۷ به‌صورت دو دسته‌ی {۰۰۰۰۱۰۱۰, ۱۱۰۱۰۱۱} در می‌آید که حسین قبل از دسته‌ی‌آخر (۱۱۰۱۰۱۱) یک بیت صفر و قبل از سایر دسته‌ها یک بیت ۱ قرار می‌دهد تا در پایان کار ۱۳۸۷ را با دو بایت ۱۰۰۰۰۱۰۱۰, ۰۱۱۰۱۰۱۱ نمایش بدهد. هم‌چنین عدد $(۱۰\ ۱۰۰۱۰۰۰\ ۱۰۰۰۰۰۱۱)_۲$ = ۴۲۰۵۱ با ۳ بایت به‌صورت ۱۰۰۰۰۰۰۱۰, ۱۱۰۰۱۰۰۰, ۰۱۰۰۰۰۱۱ نمایش داده می‌شود.

الف) (۱ نمره) چرا در این روش مقدار حافظه‌ی مصرفی از حالتی که تمام اعداد در int ذخیره شوند، بیش‌تر نمی‌شود؟

ب) (۳ نمره) آیا یک آرایه‌ی بایت‌های متوالی از اعداد کد شده به‌این روش، قابل تفکیک (بازگشت به اعداد اوّلیه) است؟ برای این‌کار الگوریتمی به‌زبان فارسی نوشته و آن را روی آرایه‌ی {۱۱۰۰۱۰۱۱, ۱۰۰۰۰۰۰۰, ۰۰۰۰۰۰۰۱, ۰۱۱۱۱۱۱۰, ۰۰۰۰۰۰۰۱, ۱۰۰۰۱۰۰۰, ۰۱۱۱۰۱۱۱} اجرا کنید تا اعداد اوّلیه به‌دست بیایند. لازم نیست اعداد اوّلیه را به مبنای ۱۰ برگردانید!

ج) (۱۰ نمره) تابع int CodeThis(int number, unsigned char c[]) را برای کد کردن اعداد بنویسید.

در این تابع number عدد مورد نظر (مثل ۱۳۸۷ در بالا) است. این تابع باید بایت‌های ساخته شده را (از پرارزش به کم‌ارزش) در آرایه‌ی c ریخته و نهایتاً به‌عنوان خروجی تعداد این بایت‌ها را برگرداند. برای مثال خروجی برنامه‌ی روبه‌رو باید 138 107 باشد، که $(۱۰۰۰۱۰۱۰)_۲$ = ۱۳۸ و $(۰۱۱۰۱۰۱۱)_۲$ = ۱۰۷.

```
#include <cstdio>

int CodeThis(int number, unsigned char c[]) {
  /* You must write the body of this func! */
}

int main() {
   unsigned char code[4];
   int codelen = CodeThis(1387, code);
   for (int i=0; i<codelen; i++)
      printf("%d ", (int)code[i]);
   printf("\n");
   return 0;
}
```

د) (۲ نمره) یکی از مزایای این شیوه‌ی ارسال و دریافت متغیرها به‌عنوان ورودی و خروجی تابع CodeThis (این‌که خروجی تابع تعداد بایت‌های کد شده است) امکان کد کردن متوالی چند عدد در یک دستور است، طوری که کدهای آن‌ها پشت سر هم قرار بگیرند.

در یک دستور (یک خط کد C) با ۳ بار فراخوانی تابع CodeThis، ۳ عدد ۶۶ و سپس ۳۶۵ و نهایتاً ۷۸۵ را به‌صورت کد شده در یک آرایه‌ی ۵ بایتی قرار دهید، طوری که ابتدا یک بایت کد عدد ۶۶، سپس دو بایت ۳۶۵ و بعد از آن نیز دو بایت ۷۸۵ قرار بگیرد.

ه) (۱۰ نمره) تابع int DecodeThis(unsigned char c[], int clen, int a[]) را برای تبدیل یک آرایه‌ی کد به آرایه‌ای از اعداد صحیح (عکس عمل CodeThis) بنویسید.

در این تابع c آرایه‌ی بایت‌های کد مورد نظر و clen تعداد بایت‌های آن است. این تابع باید اعداد اولیه را به‌دست آورده و آن‌ها را به همان ترتیب در آرایه‌ی a ریخته و به‌عنوان خروجی تعداد این اعداد را برگرداند. برای مثال خروجی برنامه‌ی زیر باید 13 1387 باشد.

```
#include <cstdio>

int CodeThis(int number, unsigned char c[]) {
  /* You must write the body of this func! */
}

int DecodeThis(unsigned char c[], int clen, int a[]) {
  /* You must write the body of this func, too! */
}

int main() {
  unsigned char code[100];
  int codelen = CodeThis(1387, code);
  int newcodelen = CodeThis(13, code+codelen);
  // now code is {138 107 13}
  //           = {10001010 01101011 00001101}

  int a[100];
  int numlen = DecodeThis(code, newcodelen, a);
  for (int i=0; i<numlen; i++)
    printf("%d ", a[i]);
  return 0;
}
```

و) (۵ نمره) برنامه‌ی بنویسید که ابتدا یک عدد $n$ و سپس $n$ عدد صحیح مثبت و کوچک‌تر از $۲^{۲۸}$ از ورودی بخواند و آن‌ها را در یک آرایه‌ی unsigned char c[] کد کند و در خروجی ابتدا درصد حجم کاهش یافته در مقایسه با روش نگه‌داری به‌صورت int چهاربایتی تا ۳ رقم اعشار چاپ کند. سپس اعداد را از آرایه‌ی c بازگرداند و آن‌ها را در خروجی چاپ کند. فرض کنید $n \leq ۱۰۰۰$ است.

شما می‌توانید این برنامه را مستقل از قسمت‌های ج و د و با فرض وجود آن دو تابع بنویسید.

ز) (۳ نمره) می‌دانیم هر عنصر آرایه‌ی c یک بایت ۸ بیتی است که $۲^{۸} = ۲۵۶$ حالت می‌تواند داشته باشد؛ اما یکی از این ۲۵۶ حالت هرگز در هیچ‌کجای رشته‌ی کد شده (c) ظاهر نمی‌شود! آن عدد کدام است؟

ح) (۶ نمره) حسین باز هم حجم کمتری می‌خواست! با بررسی دنباله‌ی اعداد، حسین متوجه شد که تعدادی عدد صفر متوالی (در اعداد صحیح ورودی) وجود دارد. با کمک قسمت و، آیا می‌توانید الگوریتم حسین را طوری بهبود ببخشید که در صورتی که در اعداد وردی تعدادی صفر متوالی وجود داشت، اندازه‌ی کد از اینی که هست هم کوچک‌تر شود؟ دقت کنید که در الگوریتم جدید طول رشته هرگز نباید از طول رشته‌ی الگوریتم قبلی بیش‌تر بشود و بهبودی شما باید برای تمام ورودی طول کم‌تر یا مساوی را ضمانت کند.

در این قسمت نیازی به نوشتن کد نیست و صرف توضیح فارسی کفایت می‌کند.

## ۱) ماجرای کامبیز و مقدار کد ASCII کاراکتر 'c' ..... ۲ + ۴ = ۶ نمره

می‌خواهیم بدانیم کد ASCII کاراکتر 'c' چند است.

الف) در خط اوّل تابع main برنامه‌ی روبه‌رو حداکثر ۱۷ کاراکتر بنویسید که مقدار کد ASCII کاراکتر 'c' را محاسبه و چاپ کند. دقت کنید که نوشتن عدد ۹۹ (به‌صورت مستقیم) مجاز نیست و برای نوشتن هم باید الزاماً از تابع printf استفاده شود.

```
#include <cstdio>
int main() {
   /* < 20 characters here */
   return 0;
}
```

فقط همین حداکثر ۱۷ کاراکتر را در برگه‌ی پاسخ‌نامه بنویسید.

ب) (♠) کامبیز که از روش قسمت الف اطلاع ندارد، قطعه کد مقابل را برای این منظور نوشته است. آیا این کد درست کار می‌کند؟

[راهنمایی: کامبیز هم، مانند شما، می‌داند که char، یک نوع علامت‌دار (Signed Type) است. ]

```
#include <stdio>
int main() {
   int v = -1;
   for (char i=32; i<=127; i++)
      if (i == 'c')
         v = i;
   printf("%d\n", v);
   return 0;
}
```

## ۲) برای یک‌صدم بیش‌ترا ..... ۳ + ۲ + ۱ + ۱ = ۷ نمره

الف) آیا برنامه‌ی روبه‌رو درست کامپایل و اجرا می‌شود؟ در صورتی که خطای کامپایل دارد آن را یافته و رفع کنید.

```
#include <cstdio>

const int N = 3;
int *p, q;

int main() {
  scanf("%d", &q);
  p = &q;
  q = 100/*p;
  printf("100/%d = %d\n", *p, q);
  return 0;
}
```

ب) خروجی/عکس‌العمل برنامه‌ی روبه‌رو به‌ازای ورودی‌های زیر چیست؟

ب – یک) ۵

ب – دو) ۱۰۰

ب – سه) ۲۰۰

ب – چهار) صفر

## ۳) تمام آن‌چه راجع به اندازه می‌دانیم! ..... ۲ + ۱ × ۵ = ۷ نمره

(♠) آیا برنامه‌ی روبه‌رو درست کامپایل و اجرا می‌شود؟ خروجی برنامه‌ی روبه‌رو چیست؟

```
#include <cstdio>

void f(int x) {
   printf("%d\n", sizeof x);
}

int main() {
   int *a = new int[10];
   char* cp, c2;
   char ch;

   printf("%d %d %d %d %d\n",
      sizeof -1, sizeof true,
      sizeof  a, sizeof cp, sizeof c2);
   f(ch);
   return 0;
}
```

## ۴) ماتریس بی‌چاره! ..... ۱۰ نمره

(♠) خروجی برنامه‌ی روبه‌رو چیست؟
لطفاً در برگه‌ی پاسخ خود تمام فاصله‌های خالی (space) را با کاراکتر underline (_) نشان دهید. دقّت کنید که بخشی از نمره‌ی این سؤال به همین مورد اختصاص دارد!

```
#include <cstdio>

const int N = 3;
int **p, *q, a[N][N];

int main() {
   for (int i=0; i<N; i++)
      for (int j=0; j<N; j++)
         a[i][j] = 10*i+j;
   q = &a[0][0];
   for (int i=0; i<N; i++) {
      p = &(q = &a[0][0]);
      0[p[0]+i*(N+1)] = 0;
   }
   for (int i=0; i<N; i++) {
      for (int j=0; j<N; j++)
         printf("%2d ", a[i][j]);
      printf("\n");
   }
   return 0;
}
```

## ۵) تابع خوب برای بچه‌های خوب! ..... ۳ + ۴ = ۷ نمره

تابع GoodFunc زیر را در نظر بگیرید.

الف) (♠) آیا این تابع کار خاصی انجام می‌دهد؟

ب) آیا تابعی در زبان C وجود دارد که عمل‌کرد مشابهی با این تابع داشته باشد؟

```
void GoodFunc(char s1[], char s2[]) {
     for (int i=0; s1[i] = s2[i]; i++);
}
```

## ۶) چه‌کسی عاشق C است؟ ..... ۲ + ۵ + ۲ + ۱ = ۱۰ نمره

الف) تابع JJ در حالت عادی چه‌کاری را انجام می‌دهد؟
ب) هدف نویسنده‌ی این کد چه بوده است؟
ج) (♠) خروجی این برنامه چیست؟
د) با توجه به هدف نویسنده‌ی این کد، آیا این برنامه از نظر زمان اجرا بهینه است؟

```
#include <cstdio>
#include <cstring>
char s[20] = "I LOVE C!";
void JJ(int i1, int i2) {
    s[i1] = s[i1] + s[i2];
    s[i2] = s[i1] - s[i2];
    s[i1] = s[i1] - s[i2];
}
int main() {
    for (int i=0; i<strlen(s); i++)
        JJ(i, strlen(s)-1-i);
    printf("%s\n", s);
    return 0;
}
```

## ۷) عملیّات شوم! ..... ۵ + ۲ + ۲ + ۲ + ۲ = ۱۳ نمره

برنامه‌ی سمت چپ را در نظر بگیرید.
آیا این برنامه به‌درستی کامپایل می‌شود؟ در صورت منفی بودن جواب، خطای کامپایل آن را رفع کنید.

```
#include <cstdio>
void show(x) { printf("%d\n", x); }
int main() {
  /* place any of below codes here */
  return 0;
}
```

برای هر یک از تکّه کدهای زیر، در صورتی که آن را در قسمت مشخص شده در main برنامه‌ی بالا قرار دهیم، خروجی یا خطا را بنویسید.

**کد الف.** توجه کنید که اولویت << و >> برابر است. عیناً مانند + و -.

```
int a = 0x5C;
show(a >> 4 << 4);
```

**کد ب.** توجه کنید که اولویت ~ بیش‌تر از / است.

```
int b = ~unsigned(0) / 2;
show(b);
```

**کد ج.** (در C چگونه می‌شد باینری نوشت؟!)

```
int c = 01010 / 10;
show(c);
```

**کد د.** اولویت عملگرهای unary بالاتر از عملگرهای binary است.

```
unsigned char d = !55 ^ ~55;
show(d);
```

**کد ه** خیلی هم پیچیده نیست!

```
unsigned char e = 1;
for (int i=0; i+1<7; i++)
  e = (e << 1) + ((e >> i) & 1);
show(e);
e++;
unsigned char f = 2 * e - 2;
show(f);
```

«پایان!»