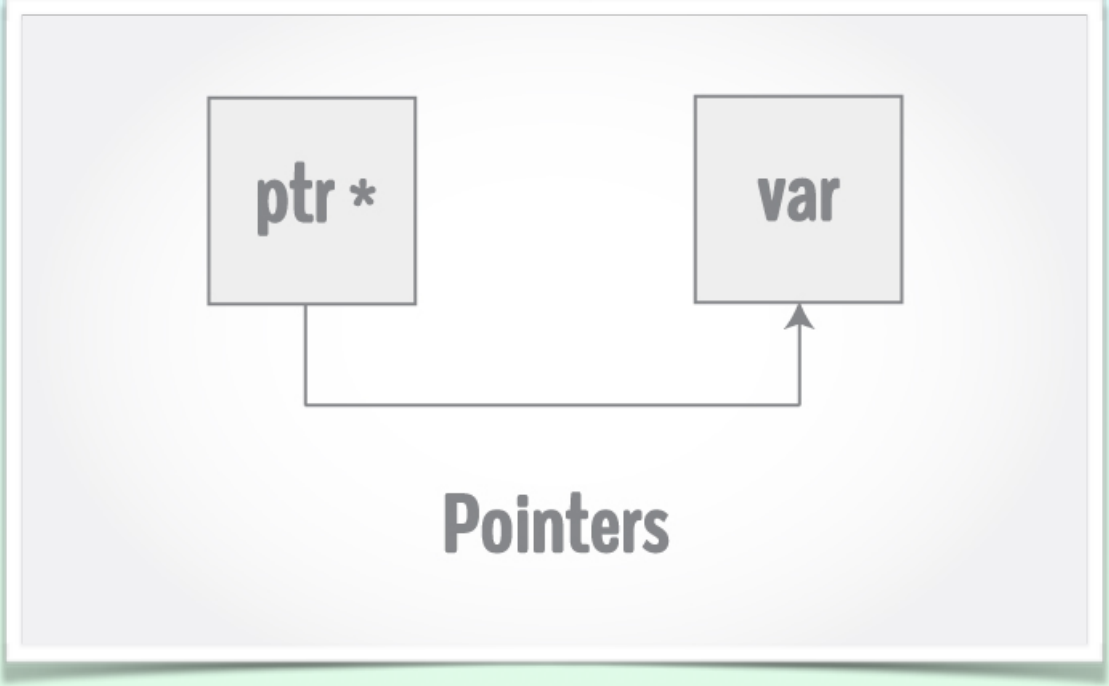
به نام خداوند بخشنده‌ی مهربان

شماره‌ی ۶

۳ دی ۱۳۹۹

کد نامه

ویژه‌ی دانش‌جویان مبانی برنامه‌سازی نیم‌سال اول ۱۴۰۰-۱۳۹۹ دانشکده‌ی مهندسی کامپیوتر دانشگاه صنعتی شریف



در این شماره از کدنامه، می‌خوانید:

اشاره‌گرهای پرحاشیه!

آیا می‌دانستید؟

اشاره‌گر در بسیاری از زبان‌های دیگر، به عنوان یک نوع داده‌ی مجزا وجود ندارد، اما در عمل و در پشت صحنه، از آن استفاده می‌شود!

اشاره‌گرهای پرحاشیه! (قسمت اول)

در نحوه‌ی استفاده از آدرس متغیرها در حافظه، مهارت کسب کنید

تاکنون در کلاس‌های درس و حل تمرین، به اندازه‌ی کافی با متغیرها و آرایه‌ها سوالات متعدد حل کرده‌اید؛ در نتیجه خوب است که با نحوه قرارگیری این متغیرها و آرایه‌ها در حافظه‌ی رایانه بیش‌تر آشنا شوید. دسترسی به آدرس متغیرها، کاربردهای متعددی در نوشتن ساده‌تر برنامه‌های بعضا پیچیده دارد و در انجام پروژه‌ی درس نیز کاربرد فراوانی خواهد داشت. با کدنامه همراه باشید.

مطالب تدریس شده در کلاس درس توسط استاد در هفته‌ی گذشته

29 آذر

الگوریتم‌های آرایه

کسب مهارت در طراحی الگوریتم به کمک آرایه‌ها (مانند الگوریتم‌های مرتب‌سازی)

1 دی

کاراکترها و رشته‌ها

استفاده از نوع خاصی از آرایه‌ها (از نوع مجموعه‌ای از کاراکترها) برای حل مسائل متعدد

3 دی

کاراکترها و رشته‌ها

استفاده از نوع خاصی از آرایه‌ها (از نوع مجموعه‌ای از کاراکترها) برای حل مسائل متعدد

اشاره‌گرهای پرحاشیه! (قسمت اول)

سید پارسا نشایی

نگاهی به درون حافظه

متغیرهایی که تعریف می‌کنید، هر یک بسته به اندازه‌ی خود، مقداری فضا در حافظه‌ی رم سیستم اشغال می‌کنند. حافظه‌ی رم، به شکل خانه–خانه طبقه بندی شده است. مثلا، قطعه کد زیر را در نظر بگیرید:

```
int x = 5;
```

بعد از اجرای کد فوق، یک مکان در حافظه به طول مشخصی به‌طور اختصاصی برای متغیر x در نظر گرفته می‌شود. مثلا فرض کنید کامپیوتر تصمیم گرفته x در خانه ۱۰۰۰ ام حافظه قرار گیرد. پس از تعریف x به شکل فوق، مقدار ۵ در خانه‌ی ۱۰۰۰ ام حافظه نوشته می‌شود.

می‌توانید آدرس یک متغیر را به کمک نماد & در زبان C به دست آورید. به عنوان مثال، دستور زیر آدرس متغیر x را در خروجی چاپ می‌کند (با هر بار اجرای برنامه، ممکن است عددی متفاوت دریافت کنید) – توجه کنید که به دلیل سیستم مکمل ۲، ممکن است عددی نامربوط دریافت کنید:

```
printf("%d", &x);
```

ذخیره آدرس یک متغیر در یک متغیر دیگر

احتمالا این سوال به ذهنتان رسیده است که اگر توانسته‌ایم آدرس یک متغیر در حافظه را چاپ کنیم، پس حتما می‌توانیم آن را در یک متغیر دیگر با دستوری مثل `int y = &x;` ذخیره کنیم، اما اگر اقدام به کامپایل چنین دستوری کنید، با خطا از سوی کامپایلر مواجه خواهید شد. دلیل این خطا آن است که هر نوع داده در کامپیوتر اندازه‌ی مشخصی دارد؛ مثلا ۳۲ بیت برای اعداد صحیح، ۸ بیت برای کاراکترها و موارد مشابه. محدوده‌ی مجاز برای آدرس یک متغیر، از رایانه‌ای به رایانه‌ی دیگر متفاوت است و ممکن است به ظرفیت حافظه RAM رایانه و نیز چند عامل دیگر بستگی داشته باشد؛ در نتیجه لازم بوده در زبان C، نوع داده‌ی خاصی برای متغیرهایی که «آدرس یک متغیر دیگر» را در خود نگه می‌دارند، به وجود آید. برای این‌که مشخص کنیم یک متغیر از نوع متغیرهایی است که قرار است آدرس متغیری دیگر را نگه دارند، از نماد * در هنگام تعریف استفاده می‌کنیم:

```
int * y = &x;
```

در اصل، متغیر y از نوع `int *` تعریف شده است؛ یعنی متغیری که می‌تواند آدرس یک متغیر از نوع `int` را در خود نگه دارد. مشابها، `char *` یعنی متغیری که آدرس یک کاراکتر را در خود نگه می‌دارد. برای سایر انواع داده نیز می‌توان به شکل مشابه، عمل کرد. به متغیرهایی که به این شکل تعریف می‌شوند، **اشاره‌گر (Pointer)** می‌گوییم، زیرا می‌توان گفت این متغیرها با داشتن آدرس یک متغیر دیگر، به مکان آن متغیر در حافظه‌ی کامپیوتر، «اشاره» می‌کنند.

دسترسی به متغیر به کمک آدرس آن

می‌توانیم از روی آدرس متغیر در حافظه، به مقدار خود متغیر دسترسی داشته باشیم. برای این‌کار، کافی است یک نماد ستاره در پشت اسم متغیری که حامل آدرس است، قرار دهیم (این نماد ستاره با نمادی که هنگام تعریف y استفاده کردیم، اشتباه نشود!). به عنوان مثال، اگر x متغیری با مقدار ۵ و y متغیری باشد که شامل آدرس x است (و یا به قولی، به x اشاره می‌کند)، آن‌گاه *y به معنای مقدار متغیری است که آدرس آن در y واقع باشد. می‌دانیم متغیری که آدرس‌اش در y واقع است، x است، پس *y در حقیقت مقدار داخل x، یعنی ۵ را بر می‌گرداند.

اگر متوجه نشده‌اید، به مثال زیر دقت کنید.

مثال: کارکرد و خروجی برنامه‌ی زیر را نوشته و با استدلال، چگونگی به دست آمدن خروجی نوشته شده را تحلیل کنید.

```
int a;
int * b;
scanf("%d", &a);
b = &a;
printf("%d", (*b) + 2);
```

پاسخ: متغیر a یک متغیر عادی int است، اما b یک متغیر از نوعی است که بتواند آدرس یک متغیر int را در خود نگه دارد (به عبارت دیگر، b از نوع اشاره‌گر به int است). در خط سوم، مقدار a از ورودی خوانده می‌شود. در خط چهارم، آدرس متغیر a در حافظه (مثلا فرض کنید ۱۰۰۰ است) در b نوشته می‌شود. در خط آخر، عبارت *b به معنای مقدار متغیری است که آدرس آن در b قرار دارد. متغیری که آدرس‌اش در b است، همان a است، پس *b به معنای همان مقدار a است که برابر

ورودی کاربر بود. سپس، دو تا بیش‌تر از این مقدار ورودی، در خط آخر چاپ می‌شود. پس، برنامه این‌گونه عمل می‌کند که یک عدد را از ورودی گرفته و دو واحد بیش‌تر از آن را در خروجی چاپ می‌کند.

معنای & هنگام استفاده از scanf

تاکنون در برنامه‌هایی که نوشته‌اید، از نماد & هنگام گرفتن ورودی، به کرات استفاده کرده‌اید. هم‌اکنون یاد گرفته‌اید که آمدن نماد & قبل از اسم یک متغیر، به معنای آدرس آن متغیر است. در حقیقت، تابع scanf به گونه‌ای طراحی شده است که آدرس متغیری که قرار است مقداردهی شود را دریافت کرده و به کمک آن، خود متغیر را را مقداردهی کند.

چند کاربرد از آدرس متغیرها

تعویض مقدار متغیرها درون توابع

می‌دانیم که اگر مقدار یک متغیر را درون یک تابع عوض کنید، مقدار آن بیرون از تابع عوض نخواهد شد:

```
void f(int i) {
	i++;
}
int main() {
	int n = 5;
	f(n);
	printf("%d", n); // prints 5
	return 0;
}
```

دلیل این اتفاق، آن است که هنگام صدا زدن f، یک **کپی** از متغیر ساخته می‌شود و به تابع فرستاده می‌شود. در داخل تابع، این کپی متغیر است که تغییر می‌کند و پس از اتمام تابع، کپی متغیر از بین می‌رود؛ در نتیجه، مقدار متغیر بدون تغییر می‌ماند. اصطلاحا به این روش، Pass by Value می‌گویند که به آن معناست که مقدار (Value) متغیر، کپی می‌شود.

حال فرض کنید به جای خود n، متغیری را به تابع می‌دادیم که آدرس n داخل آن بود (یعنی به n اشاره می‌کرد). در این حالت، i یک کپی از آدرس n خواهد بود (یعنی i هم به n اشاره می‌کند) و در نتیجه عبارت *i برابر مقدار متغیری است که آدرس آن داخل i قرار دارد، یعنی همان متغیر n. در نتیجه، اگر *i را تغییر دهیم، مقدار n در تابع main نیز تغییر می‌کند!

```
void f(int * i) {
	(*i)++;
}
int main() {
	int n = 5;
	f(&n);
	printf("%d", n); // prints 6
	return 0;
}
```

به این روش، Pass by Reference گفته می‌شود، زیرا تنها یک اشاره‌گر (رفرنس) به متغیر اصلی، به تابع فرستاده می‌شود. کاربرد این روش در زمانی است که می‌خواهیم مقدار یک متغیر را از درون یک تابع تغییر دهیم.

تمرین: سعی کنید به کمک آدرس متغیرها، تابعی بنویسید که بیش از یک خروجی عددی برگرداند.

آرایه به عنوان آدرس

در زبان C، آرایه‌ها یک خاصیت بسیار مهم دارند: هر گاه اسم آرایه را به تنهایی (یعنی بدون علامت کروشه و اندیس جلوی آن) بنویسید، این اسم به معنای «آدرس اولین خانه‌ی آرایه» خواهد بود؛ یعنی اگر A یک آرایه

ارتباط با کدنامه

خوش‌حال می‌شویم اگر پیشنهادات و انتقاداتی نسبت به کدنامه دارید یا سوال خاصی دارید که تمایل دارید در کدنامه پاسخ داده شود، به دستیاران آموزشی درس اطلاع دهید.

باشد، دستور زیر، آدرس خانه‌ی اول آرایه (یعنی آدرس A[0]) را چاپ می‌کند:

```
printf("%d", A);
```

با استفاده از این نکته، می‌توان به شکل جدیدی با آرایه‌ها کار کرد. به مثال زیر توجه کنید:

مثال: عضو پنجم یک آرایه از اعداد صحیح (یعنی A[4]) را بدون استفاده از نمادهای [و] چاپ کنید.

پاسخ: اعضای یک آرایه در حافظه، پشت سر هم ذخیره می‌شوند. یاد گرفتیم که A شامل آدرس A[0] است؛ در نتیجه اگر یک واحد به A اضافه کنیم (یعنی A + 1)، آدرس خانه‌ی بعدی حافظه (که A[1] است) را دریافت می‌کنیم. اگر به همین شکل پیش برویم، اگر ۴ واحد به A اضافه کنیم (یعنی A + 4)، آدرس خانه‌ی A[4] را تحویل می‌گیریم. می‌دانیم برای این‌که از آدرس یک متغیر به مقدارش برسیم، باید از نماد ستاره قبل از متغیر حاوی آدرس استفاده کنیم؛ در نتیجه، عبارت *(A+4) معادل عبارت A[4] خواهد بود و پاسخ این مثال، برابر است با:

```
printf("%d", *(A+4));
```

پرسش: می‌دانیم هر عدد صحیح در رایانه‌ی به‌خصوصی، ۴ بایت جا اشغال می‌کند؛ پس قاعدتا برای این‌که از عدد صحیح اول به عدد صحیح پنجم برسیم، باید ۴ * ۴ = ۱۶ واحد به جلو برویم و نه ۴ واحد؛ پس چرا در کد، A+4 نوشته شده؟

پاسخ: کامپایلر خودش متوجه می‌شود که در حال کار با چه نوع داده‌ای هستیم و خود به خود تعداد بایت‌هایی که در کد نوشته‌ایم (در این مثال، ۴ ای که در کد نوشته‌ایم) را در اندازه‌ی نوع داده (که برای عدد صحیح، ۴ بایت است)، ضرب می‌کند. اگر بخواهیم این رفتار کامپایلر را دور بزنیم و به طور صریح مشخص کنیم که چند بایت به جلو برود، باید از نوع داده‌ای به اسم void * استفاده کنیم (که هیچ ربطی به نوع داده‌ی void ندارد!). به مثال زیر توجه کنید:

```
void * x = A;

printf("%d", *(x+4));
```

در این مثال، چون A در یک متغیر از نوع void * ریخته شده، در نتیجه در خط دوم، به جای این‌که ۴ * ۴ یا همان ۱۶ بایت به جلو رفته و A[4] چاپ شود، تنها ۴ بایت (که معادل یک عدد int است) به جلو رفته و A[1] چاپ خواهد شد.

اشاره‌گرها در حلقه‌ها

مثال: کارکرد و خروجی برنامه‌ی زیر را نوشته و با استدلال، چگونگی به دست آمدن خروجی نوشته شده را تحلیل کنید.

```
char a[100];

scanf("%s", a);

for (int i = 0; i < 100; i++) {

      if ((*(a+i)) == '\0') break;

      printf("%c\n", *(a+i));

}
```

پاسخ: ابتدا یک آرایه‌ی رشته‌ای تعریف شده و رشته از کاربر گرفته می‌شود، سپس در یک حلقه، تک تک کاراکترها را تا نرسیدن به انتهای رشته بررسی و هریک را در یک خط مجزا چاپ می‌کنیم. a+i به معنی آدرس خانه‌ی A[i] است، در نتیجه *(a+i) به معنای خود محتوای داخل A[i] خواهد بود.

حال، یک پرسش برای شما طرح می‌کنیم. سعی کنید با توجه به آن‌چه در این شماره و نیز شماره‌های پیشین آموخته اید، به این پرسش، پاسخ دهید.

پرسش: آیا می‌توانید دلیل خطای کامپایل برنامه‌ی زیر را بیان کنید؟ (راهنمایی: به توضیحات کامپایلر توجه کنید)

```
char a[100];

scanf("%s", a);

for (int i = 0; i < 100; i++) {

if ((*(a+i)) == '\0') break;

      void * x = a;

      printf("%c\n", *(x+i));

}

printf("%s", a);
```