به نام خداوند بخشنده‌ی مهربان

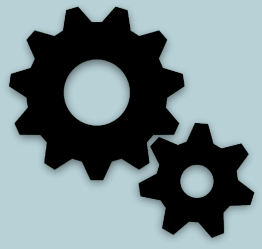
شماره‌ی ۳ ۵ آذر ۱۳۹۹

کد نامه

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

در این شماره از کدنامه، می‌خوانید:

تمیز بنویسیم!



تابع و کاربرد آن

تمیز بنویسیم!

با یکی از مهم‌ترین عادت‌ها در برنامه‌نویسی آشنا شوید

آن چیزی که یک برنامه نویس معمولی را از یک برنامه نویس خبره جدا می‌کند، توجه به جزئیاتی هم‌چون کدنویسی تمیز است، به طوری که هر فرد دیگری بتواند تنها با نگاه کردن به کد شما، مفهوم آن را درک کند. تمیز نویسی کد، در دروسی مانند برنامه‌سازی پیشرفته در نیم‌سال آینده ضروری و الزامی است، و لازم است که از اکنون به تمیزنویسی در برنامه‌ها عادت کنید. با کدنامه همراه باشید.

مطالب تدریس شده در کلاس درس توسط استاد در هفته‌ی گذشته

1 آذر

دستورات شرط و حلقه

آشنایی با مهم‌ترین دستورات در برنامه‌سازی به زبان C

3 آذر

دستورات شرط و حلقه

آشنایی با مهم‌ترین دستورات در برنامه‌سازی به زبان C

5 آذر

تابع

آشنایی اولیه با مفهوم تابع و دلیل نیاز به آن در برنامه‌سازی ساخت‌یافته

تمیز بنویسیم!

متین داغیانی

کد کثیف، کد تمیز

تا حالا پیش آمده که یک متن به خصوص را بخوانید و از همان اول از کارتان پشیمان شده باشید؟ اشتباهات املایی فراوان، جملات طولانی، فاصله‌ی کم خطوط، اندازه‌ی کوچک متن و... همان چیزهایی هستند که باعث شده‌اند احتمالا جواب شما به این سوال مثبت باشد. برعکس آن هم درست است. شاید تعداد انگشت‌شماری از متون هم در ذهنتان باشند که وقتی آنها را می‌خواندید، به معنای واقعی کلمه از آن لذت برده باشید، یا به قول معروف با آن ارتباط برقرار کرده باشید.

برنامه‌نویسی هم چندان بی‌شباهت به نوشتن نیست. فرقی نمی کند که یک برنامه‌ی ساده ی چند خطی باشد، یا یک پروژه‌ی بزرگ. چیزی که یک برنامه نویس معمولی را از یک برنامه نویس خبره جدا می‌کند، توجه به همان جزئیاتی است که باعث تفاوت تجربه شما از خواندن یک متن اعصاب خرد کن و یک شاهکار ادبی می‌شود.

چرا باید تمیز کد بزنیم؟

بتوانیم کدمان را بخوانیم!

یکی از تجربیات ناخوشایند هر برنامه‌نویس آن است که بعد از مدتی دوباره به سراغ برنامه‌ای که نوشته است برود و خود را میان انبوهی از خطوط بی‌معنی پیدا کند که حتی خودش هم متوجه نشود که چطور آنها را نوشته است! به جرئت می‌توان گفت که رعایت برخی اصول نه چندان پیچیده، کمک می‌کنند تا این تجربه‌ی تلخ، کم‌تر برایمان پیش بیاید.

دیگران بتوانند کدمان را بخوانند!

برنامه‌نویسی، در بسیاری از مواقع، یک کار گروهی است و شما ناگزیر هستید با دیگران تعامل و همکاری داشته باشید. یکی از مواردی که باعث شکست بسیاری از پروژه‌های بزرگ تیمی می‌شود، عدم درک متقابل میان اعضا است. اگر نتوانید متوجه شوید که کدی که همکارتان نوشته است چگونه کار می‌کند، هرگز نخواهید توانست مشکل آن را برطرف کنید و یا قابلیت‌های آن را افزایش دهید.

کد کثیف، محکوم به نابودی است!

نادیده گرفتن جزئیات و نکات ایمنی و ساختمانی، همان عاملی است که باعث می‌شود یک ساختمان به ظاهر زیبای تازه‌ساز، بعد از چند سال، تبدیل به یک مخروبه شود. حتی اگر بتوان قسمتی از آن را با هزینه‌ی گزاف، بازسازی و تعمیر کرد، مشکل از جایی دیگر سر در می‌آورد؛ درست مانند کد کثیف!

بر خلاف آن، داشتن چارچوب و رعایت قواعد کد تمیز، همان چیزی است که مسیر توسعه‌ی یک برنامه را هموارتر می‌کند و به طور چشم‌گیری از هزینه‌های بالقوه می‌کاهد.

با کدنامه، تمیز بنویسیم!

حالا که در حال فراگیری برنامه‌نویسی هستیم، خیلی بهتر است که در کنارش اصولی را نیز یاد بگیریم که به ما کمک می‌کنند تا بهتر کد بزنیم؛ یا به عبارت دیگر، **تمیز بنویسیم**.

در شماره‌های آینده‌ی کدنامه، به صورت مداوم، مطالبی در اختیارتان قرار خواهند گرفت تا نکات موسوم به Clean Code را به طور خلاصه و جامع با هم مرور کنیم. در این شماره از کدنامه نیز تعدادی از نکات مهم در کدنویسی تمیز را با یک‌دیگر می‌بینیم.

چند نکته‌ی مهم در کدنویسی تمیز

متغیرها، اسم‌اند!

سعی کنید در نام‌گذاری متغیرهایتان از اسم‌ها استفاده کنید. استفاده از فعل یا صفت به تنهایی توصیه نمی‌شوند. هم‌چنین، اسامی مخفف یا خاص را در نام‌گذاری به کار نبرید.

```
int best_number; // Correct!
int best;       // Wrong!

int the_sample_result;  // Correct!
int smplerslt;          // Wrong!
```

اسمهای معنادار، بهترین راهنماها

بر روی نام‌گذاری متغیرها فکر کنید. سعی کنید از اسامی یا گروه‌های اسمی استفاده کنید که بتوانند به این سوالات پاسخ دهند:

- چرا تعریف شده است؟
- چه کار می‌کند؟

```
int d;                          // Wrong!
int elapsed_time_in_days; //Correct!
```

از نامهای طولانی نترسید!

طولانی بودن نام متغیر، بهتر از بی‌معنا بودن آن است. در یک برنامه شاید صدها یا هزاران متغیر وجود داشته باشند و اگر تمام آنها را با حروف الفبا یا

ترکیب آنها با اعداد تعریف کرده باشید، شاید بعدا نتوانید متوجه شوید که کارکرد متغیر چیست و چه اطلاعاتی را در خود ذخیره می‌کند. البته نباید در این موضوع، زیاده‌روی کنید. بر اساس استانداردها، بهتر است طول نام هر متغیر، کمتر از ۳۱ کاراکتر باشد. به عنوان مثال، اگر متغیری قرار است حاصل جمع متغیرهای a و b را در خود نگه دارد، استفاده از sum_of_a_and_b بهتر از استفاده از s به عنوان نام است.

جدا کردن کلمات

در زبان‌های مختلف برنامه‌نویسی، انواع مختلفی از قواعد برای جداکردن واژه‌های تشکیل‌دهنده نام متغیرها وجود دارد. در بیشتر کتابخانه‌های استاندارد زبان C، از **حروف کوچک** برای تعریف متغیرها و از کاراکتر _ برای جداکردن واژه‌ها استفاده شده است. به عنوان مثال، استفاده از my_new_variable به myNewVariable یا mynewvariable، ترجیح می‌دهیم.

عبارات شرطی و حلقه‌ها

عبارت‌های شرطی از پراستفاده‌ترین جملات در مکالمات عادی و روزانه ما و هم‌چنین زبان‌های برنامه‌سازی هستند. حلقه‌ها نیز از پررنگ‌ترین ویژگی‌های هر زبان برنامه‌نویسی محسوب می‌شوند که امکان مدیریت عملیات پی‌درپی و متوالی را در اختیار توسعه‌دهنده‌ها قرار می‌دهند. البته این ابزارهای مفید می‌توانند زمانی که با تعداد بسیاری پرانتز، آکولاد، حلقه‌های تو در تو و... دست و پنجه نرم می‌کنید، بسیار گیج‌کننده نیز باشند!

شرط‌های پرمعنا

در زبان C، اعداد ۰ و ۱ علاوه بر اینکه به عنوان متغیرهای int ظاهر می‌شوند، می‌توانند توصیف‌کننده‌ی متغیرهای صحیح/غلط نیز باشند. در واقع مقدار **صفر** به معنای غلط و مقدار **یک** به معنای درست است. برای آنکه هنگام خواندن یک عبارت شرطی بتوانیم این دو کاربرد را از یک‌دیگر متمایز کنیم، به این صورت عمل می‌کنیم که اگر قصد مقایسه‌ی یک عدد با عدد صفر را داشتیم، حتما عبارت a == 0 را به !a ترجیح می‌دهیم، اما اگر متغیر ما، یک متغیر حالت بود (برای مثال، found)، از آن در عبارت if به شکل if(!found) و نه به شکل if (found == 0) استفاده می‌شود.

فاصله‌گذاری اجتماعی در اول خط!

برای تمایز میان قسمت‌ها و بلاک‌های مختلف (در شرط‌ها، حلقه‌ها و...)، بهتر است از تورفتگی‌های مناسب استفاده کنیم. به دو مثال زیر در زمینه استفاده از indent در ابتدای خط، توجه کنید:

مثال غلط:

```
if (a == 5) {
printf(“Hello!”);
}
```

مثال درست:

```
if (a == 5) {
        printf(“Hello!”);
}
```

شرط‌های مرکب

از جمله مواردی که می‌تواند به شدت کدمان را ناخوانا کند، شرط‌های مرکب یا چندخطی است. برای آنکه بتوانیم بهتر آن‌ها را تحلیل و بررسی کنیم، بهتر است هر شرط را در یک خط بنویسیم (**تمام** عملگرها باید یا در ابتدا و یا در انتهای خطوط آورده شوند).

```
if (condition1 ||
   (condition2 && condition3) ||
   condition4) {
        printf(“Hello!”);
}
```

به این نکته هم توجه داشته باشید که به طور کلی، بهتر است از به کار بردن شرط‌های طولانی و پیچیده، اجتناب کنید و سعی کنید تا جای ممکن، با اصلاح منطق برنامه‌تان، آن‌ها را ساده‌تر کنید.

مارپیچ شرط‌ها!

اگرچه در استفاده از شرط‌ها، حلقه‌ها و ترکیب آن‌ها با یک‌دیگر، محدودیتی وجود ندارد، با این حال بهتر است تا جای ممکن از عبارات تو در تو و طولانی پرهیز کنیم. در بسیاری از اوقات، می‌توان با کمی فکر، راه مناسب‌تری برای ساده‌تر کردن کدمان پیدا کرد. یکی از این راه‌ها، استفاده از توابع است که با آن در صفحه‌ی بعد آشنا می‌شوید.

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.

تابع و کاربرد آن

علیرضا حسین‌خانی

به زبان ساده، یک تابع، بخشی از کد شما است که کار مخصوصی را انجام می‌دهد. احتمالا با برخی از توابع معروف در زبان C آشنا شده‌اید؛ مثلا printf که برای چاپ کردن کاراکترها و رشته‌های متنی از آن استفاده می‌کنیم. فرض کنید این تابع وجود نداشت! در این‌صورت، شما برای هربار چاپ کردن یک متن، مجبور بودید کدهای مربوط به ارتباط با صفحه نمایش که سازندگان سیستم‌عامل رایانه‌تان قبلا نوشته اند، مجددا از اول بنویسید! :(حالا تصور کنید کلا چیزی به‌نام تابع وجود نداشت. در این‌صورت، چند اتفاق ناگوار! رخ می‌داد: اول این‌که حجم کدها خیلی خیلی بیش‌تر می‌شد و باید کدهایی که در قسمت‌های مختلف برنامه کاربرد دارند را متناوبا کپی می‌کردیم (اصطلاحا reusability of code به معنای توانایی بازاستفاده از کد از بین می‌رفت). دوم این‌که با افزایش تعداد خطوط، فهمیدن هدف و کارکرد کد، سخت‌تر می‌شد (اصطلاحا readability of code به معنای توانایی خواندن کد به شدت کاهش پیدا می‌کرد) و در صورتی هم که برنامه‌ی ما باگ یا اشکالی می‌داشت، این تعداد خطوط زیاد و درهم بودن کد، کار دیباگ و اشکال‌زدایی کد را سخت می‌کرد. همه‌ی برنامه‌های جهان که به اندازه‌ی برنامه‌ی یک تمرین مبانی برنامه‌سازی نیستند؛ حجم تعداد زیادی از برنامه‌ها به چندین هزار و حتی میلیون خط کد می‌رسد و نگه‌داری تمامی این کدها در main، اصلا منطقی نیست. پس اساسا! توابع در کنار ما هستند تا کار ما را راحت‌تر کنند: با افزایش خوانایی کد، استفاده مجدد در همان برنامه یا برنامه‌های دیگر، کم کردن حجم برنامه و راحت‌تر کردن دیباگ و اشکال زدایی. یکی از نشانه‌هایی که به ما نشان می‌دهد که خوب است از تابع استفاده کنیم، کدهای تکراری هستند. سعی کنید همیشه کدهای تکراری را با فراخوانی توابع جایگزین کنید!

اجزای تابع

۱. نام و بدنه‌ی تابع: نام تابع، مثل برچسبی است که روی تابع‌مان زده‌ایم تا هر جا به آن نیاز داشتیم، بتوانیم از این تابع به راحتی استفاده کنیم. سعی کنید اسم‌های معناداری برای توابع انتخاب کنید تا بعدا دچار مشکل در کدنویسی نشوید! در بدنه‌ی تابع، خود کارهایی که باید توسط تابع انجام شوند، به صورت یک سری دستورالعمل نوشته می‌شوند.

۲. لیست پارامترها: همان‌طور که گفتیم، توابع کار به خصوصی را انجام می‌دهند؛ مثلا جمع کردن دو عدد یا چاپ کردن یک متن. طبعا باید آن دو عدد یا آن رشته‌ی متنی (و به طور کلی متغیرهای مورد نیاز برای انجام کار) را به نحوی به توابع بدهیم و این متغیرها، همان پارامترها (یا به تعبیری ورودی‌ها (آرگومان‌ها)ی تابع) هستند. البته ممکن است که یک تابع به هیچ ورودی‌ای نیاز نداشته باشد؛ در این‌صورت، به پرانتز باز و بسته خالی جلوی اسم تابع، اکتفا می‌کنیم. دقت کنید که هنگام نوشتن لیست پارامترها، باید نوعشان و اسمی که به آن‌ها می‌دهیم (تا بتوانیم داخل تابع از آن‌ها استفاده کنیم) را نوشته و مشخص کنیم.

۳. مقدار بازگشتی: برخی از توابع – مثل تابع جمع کردن دو عدد – نیاز دارند که نتیجه کارشان را به برنامه‌ی اصلی تحویل دهند (یک مقدار را برگردانند یا به اصطلاح return کنند). اگر هیچ مقداری قرار نیست از تابع ما return شود، از کلمه void در قسمت نوع بازگشتی استفاده می‌کنیم (در این‌صورت می‌توانیم در انتهای بدنه‌ی تابع، از کلمه‌ی return خالی استفاده کنیم و یا اصلا حرفی از return نزنیم) و در غیر این‌صورت، نوع مقدار بازگشتی را در قسمت نوع بازگشتی (مثلا int یا char) می‌نویسیم. انواع بازگشتی، مشابه انواع متغیرها هستند.

برای صدازدن یک تابع، کافی است نام تابع را نوشته و در پرانتز، ورودی‌های لازم را به تابع ارسال کنیم.

سوال: آیا می‌توان چند تابع با نام‌های یکسان داشت؟

پاسخ: در زبان C خیر، اما در خیلی از زبان‌های برنامه‌نویسی دیگری که بعدا خواهید دید – مثل C++ و Java – این مشکل حل شده؛ مثلا در Java شما مجاز هستید توابع هم‌نام تعریف کنید، به این شرط که لیست ورودیشان تفاوت داشته باشد (هرگونه تفاوت در نوع ورودی‌ها، تعدادشان یا ترتیبشان کافی است). اما در زبان C، مجاز به این کار نیستید.

سوال: prototype چیست؟

پاسخ: ما می‌توانیم در زبان C و در خطوط اولیه برنامه قبل از تابع main، برای توابعمان «پروتوتایپ» بنویسیم. پروتوتایپ به زبان ساده، همان خط اول تعریف تابع (یا به اصطلاح signature تابع) است، به علاوه‌ی سمی‌کالن در انتهایش! مثلا:

```
int sum(int num_one, int num_two);
```

وظیفه‌ی اصلی پروتوتایپ این است که به کامپایلر اعلام کند که ممکن است با چنین تابعی مواجه شود. با این‌کار، در ادامه کامپایل، هر جا که تابع مدنظر صدا زده شود، کامپایلر signature تابع را چک می‌کند و اگر مشکلی وجود داشت (مثلا تعداد ورودی های داده شده یکی کم بود) به شما خطای کامپایل داده و از اجرای برنامه جلوگیری می‌کند؛ در غیر این‌صورت، برنامه کامپایل می‌شود و اگر در صدا زدن تابع مدنظرمان بی‌دقتی کرده باشیم، خروجی های عجیبی دریافت می‌کنیم!