به نام خداوند بخشنده‌ی مهربان

شماره‌ی ۵ ۲۱ آبان ۱۴۰۰

کد نامه

ویژه‌ی دانش‌جویان مبانی برنامه‌سازی نیم‌سال اول ۱۴۰۱-۱۴۰۰ دانشکده‌ی مهندسی کامپیوتر دانشگاه صنعتی شریف

سفری به دنیای حافظه

با مقدمات چگونگی ذخیره کدها و داده‌ها در حافظه رم آشنا شوید

برنامه‌های در حال اجرا، هر کدام مقداری حافظه لازم داشته تا داده‌هایی که به آن‌ها نیاز دارند را در حین اجرای برنامه، در آن ذخیره کنند. آیا تا به حال به این فکر کرده‌اید که هنگامی که برنامه‌ای اجرا می‌شود، اطلاعات آن کجا ذخیره می‌شوند؟ آیا می‌دانید که تفاوت بین متغیرهای محلی، استاتیک و گلوبال در حافظه چیست؟ در این مقاله، قصد داریم به این سوال پاسخ دهیم. با کدنامه همراه باشید.

تقویم درس در هفته‌ی پیش‌رو

۲۳ آبان

آرایه

۲۵ آبان

الگوریتم‌های آرایه

۲۷–۲۸ آبان

مهلت تمرین ۴ و
انتشار تمرین ۵ – آرایه

؟ چه منابعی برای مطالعه‌ی مبانی برنامه‌سازی توصیه می‌شوند؟

مرجع رسمی درس، کتاب دایتل و دایتل است، بنابراین در صورت وجود زمان کافی، مطالعه‌ی سوالات پایانی هر فصل کتاب توصیه می‌شود. سایت‌هایی چون GeeksForGeeks و هم‌چنین سوالات مطرح شده در جلسات «کد با هم» که به زودی آغاز می‌شوند، همگی برای تمرین بیش‌تر مناسب هستند. البته، واضح است که حل درست، دقیق و کامل تمرین‌های درس بدون تقلب از یک‌دیگر، از مهم‌ترین فعالیت‌هایی است که برای یادگیری بیش‌تر و نیز کسب مهارت در حل سریع‌تر سوالات، لازم است. مقاله‌های نوشته شده در کدنامه نیز مرجع مناسبی برای دوره و کسب اطلاعات بیش‌تر در خصوص سرفصل‌های درس‌اند.

سفری به دنیای حافظه

محمد خسروی

حافظه دسترسی تصادفی

حافظه دسترسی تصادفی (Random Access Memory) یا همان RAM، یک نوع حافظه کامپیوتر است که داده‌های برنامه‌های در حال اجرا در آن ذخیره می‌شوند. RAM به ما این امکان را می‌دهد که داده‌ها را با سرعتی بالاتر از دیسک سخت بخوانیم و بنویسیم. داده‌هایی که برنامه در حین کار کردن استفاده می‌کند، باید ابتدا به داخل رم منتقل شوند و سپس مورد استفاده قرار گیرند. اما داده‌ها تا ابد در RAM باقی نمی‌مانند! RAM وابسته به انرژی الکتریکی است و به محض قطع شدن جریان، تمام اطلاعات ذخیره شده در آن از دست می‌روند. در زمان اجرای یک برنامه C، حافظه به قسمت‌های مختلفی که هر کدام برای منظوری خاص استفاده می‌شوند تقسیم می‌شود. در ادامه، این بخش‌ها را معرفی و اطلاعاتی که در آن‌ها ذخیره می‌شود را شرح می‌دهیم:

- بخش «متن» یا Text Segment: بخشی است که در آن دستورهای زبان ماشین ذخیره می‌شوند. دستورهای زبان ماشین، همان ترجمه‌ی کدی که شما نوشته‌اید به زبان ماشین (صفر و یک) است.
- بخش داده‌های مقدار دهی اولیه شده: این بخش که به اختصار، بخش داده (Data Segment) نامیده می‌شود، محلی است که در آن متغیرهای استاتیک و global ای (جلوتر با آن‌ها آشنا می‌شوید) که مقداردهی اولیه شده‌اند ذخیره می‌شوند. این بخش را می‌توان به دو بخش متغیرهای قابل تغییر و متغیرهای غیرقابل تغییر (ثابت)، تقسیم بندی کرد.
- بخش داده‌های مقداردهی اولیه نشده: این بخش که **BSS** نامیده می‌شود، متغیرهای global و استاتیکی را در خود ذخیره می‌کند که توسط برنامه نویس مقدار دهی اولیه نشده‌اند. مقدار اولیه‌ی متغیرها در این بخش، به طور خودکار، صفر می‌شود.
- پشته یا Stack: پشته محل ذخیره متغیرهایی است که درون توابع (متغیرهای محلی) تعریف می‌کنیم. علاوه بر متغیرها، اطلاعات دیگر مربوط به توابع نیز در پشته ذخیره می‌شوند. داده‌ها در پشته به صورت موقت ذخیره و به محض اتمام برنامه، به صورت خودکار پاک می‌شوند. پشته‌ها حافظه‌ی محدودی دارند که اندازه آن وابسته به سیستم عامل است.

ساختار پشته

همانطور که از اسمش به نظر می‌رسد، داده‌ها در پشته به صورت «پشته‌ای» ذخیره می‌شوند؛ به عبارتی، هر داده‌ای که زودتر وارد شود، دیرتر خارج می‌شود. برای فهم بهتر استک، یک جعبه تصور کنید که در آن، چندین جسم را، روی هم، یک به یک در آن قرار می‌دهیم. جسمی که آخر از همه داخل جعبه قرار می‌گیرد، هنگامی که شروع به خالی کردن جعبه کنیم، زودتر از بقیه از آن بیرون می‌آید.

Heap

هیپ (Heap)، قسمتی است که در آن اختصاص حافظه پویا انجام می‌شود. اختصاص حافظه در هیپ به صورت خودکار مدیریت نمی‌شود و آزادی عمل بیشتری برای برنامه‌نویس به وجود می‌آورد. هیپ شبیه یک استخر بزرگ از حافظه است که شما می‌توانید هرچه‌قدر که می‌خواهید از آن حافظه بگیرید. در آینده با متغیرهای ذخیره شده در این بخش، بیش‌تر آشنا می‌شوید. برخی از نکات مهم در خصوص هیپ و پشته به شرح زیرند:

- پشته بسیار سریع‌تر از هیپ عمل می‌کند.
- از آن‌جایی که تخصیص حافظه در پشته به صورت خودکار انجام می‌شود، حافظه‌ی ما تکه تکه نمی‌شود (یعنی همه‌ی اطلاعات در یک جا ذخیره می‌شود)، اما از آن‌جا که در هیپ، خود ما هستیم که حافظه را تخصیص می‌دهیم، امکان تکه تکه شدن داده‌ها وجود دارد.
- اندازه‌ی حافظه‌ای که پشته می‌تواند اشغال کند، بسیار محدود است، اما هیپ ظرفیت بیش‌تری دارد.

محدوده‌ی متغیرها در توابع

وقتی که شما تعدادی متغیر را درون یک تابع تعریف می‌کنید، این متغیرها فقط داخل همان تابع قابل استفاده هستند. در واقع، این متغیرها فقط در پشته‌ی آن تابع وجود دارند و محدوده‌ی متغیرهای تعریف شده در تابع، در داخل همان تابع است. اما وقتی که متغیری به شکل global (یعنی بیرون توابع) تعریف می‌شود، آن متغیر در همه جای برنامه‌ی شما قابل استفاده است. در واقع، این متغیر، داخل **بخش داده** ذخیره می‌شود و به همین خاطر، محدوده آن، کل برنامه شماست.

کاربرد متغیر global و استاتیک

هنگامی که می‌خواهیم از یک متغیر در بخش‌های مختلف کد خود استفاده کنیم، آن را به صورت global تعریف می‌کنیم، و هنگامی که می‌خواهیم مقدار یک متغیر که درون تابع تعریف شده است، پس از اتمام تابع، همچنان باقی بماند و از بین نرود، از متغیر استاتیک استفاده می‌کنیم. با متغیرهای استاتیک در ادامه‌ی درس آشنا خواهید شد.

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!

دست‌های پشت پرده در رایانه!

محمدمهدی برقی

کامپیوتر موجودی استثنایی، باهوش و دوست‌داشتنی، اما زبان‌نفهم است! موجودی که تا امروز، کلی با آن کار داشتید و از این به بعد، با توجه به این حقیقت که در فرم انتخاب رشته‌ی خود، رشته‌ی مهندسی کامپیوتر را برگزیده‌اید، قرار است بخشی اساسی از زندگی‌تان را تشکیل دهد؛ اما لازم است بدانیم که این موجود، چه‌طور کدهایی که به آن می‌دهیم را متوجه می‌شود و تفسیرشان می‌کند. آیا کامپیوتر زبان انسان‌ها را می‌فهمد؟ چه کسی به آن‌ها زبان برنامه‌نویسی C را یاد داده است؟

کامپایلر

جواب تمام سوالاتی که در بند فوق گفته شد، در یک کلمه به اسم «کامپایلر» است. کامپایلر (Compiler) – یا در برخی زبان‌ها، مفسر (Interpreter) همان دست‌های پشت‌پرده‌ی داستان ماست. شاید دوست داشته باشید بدانید پس از زدن دکمه Build and Run (و یا اجرای دستورات مشابه در Command Line، ترمینال و یا دکمه‌های مشابه در سایر IDEها، مانند CodeBlocks، VS Code، CLion، Xcode و...)، ترجمه‌ی کد به زبان کامپیوتر (زبانی که توسط کامپوتر قابل فهم باشد)، چگونه انجام می‌شود.

واضح‌تر صحبت کنیم!

همان‌گونه که در شماره‌های پیشین کدنامه – به خصوص در مقاله‌ی «تاریخچه‌ی برنامه‌نویسی کامپیوتر» در شماره‌ی ۳ کدنامه‌ی امسال – مطالعه کرده‌اید، زبان‌های برنامه‌نویسی، به چند سطح تقسیم می‌شوند؛ از سطح یک که نزدیک‌ترین زبان‌ها به سخت‌افزار رایانه (پردازنده) هستند (مثل زبان ماشین و صفر و یک‌ها) تا سطوح بالاتر، مثل MATLAB و Python که وقتی با آن‌ها کد می‌زنید، گویا در حال نوشتن یک متن به زبان انگلیسی هستید! زبان C در میان این زبان‌ها، یک زبان سریع است؛ اما– کار با– آن خیلی آسان نیست، ولی در عین حال دستورات آن اصلا سخت و غیر قابل فهم نیستند. اما، برای کامپیوتر، هر چیزی به جز صفر و یک خالص، غیر قابل فهم است، پس به مترجمی– که اسمش را کامپایلر می‌گذاریم – نیاز داریم.

توجه! IDE با IDE، کامپایلر با کامپایلر!

در حقیقت، IDE ها (مثل کدبلاکس، ویژوال استودیو یا ایکس‌کد) دفتری برای نوشتن کدهایمان هستند که مخصوص کدنویسی طراحی شده‌اند. همان‌طور که برای کلاس زبان خارجی در ایام طفولیت (!) به دفتر دو خط نیاز داشتیم، از اکنون برای هر کلاس برنامه‌نویسی، به IDE مخصوص آن زبان نیاز داریم.

اما در مقام مقایسه، کامپایلر فقط و فقط یک مترجم است که از روی دفتر شروع به خواندن کرده و هر آن‌چه را که می‌خواند، ترجمه می‌کند. هر زبانی، یک یا چند کامپایلر مخصوص به خود را دارد (مانند کامپایلر gcc در زبان C). برخی زبان‌ها، به جای کامپایلر از مفسر استفاده می‌کنند که با تفاوت‌های این دو مفهوم در آینده آشنا خواهید شد.

توجه کنید که با آن که گفته شد «هر زبان IDE مخصوص به خود را دارد»، بسیاری از IDEها نیز وجود دارند که پشتیبانی کامل از چندین زبان برنامه‌نویسی را ارائه می‌کنند و نیز تعداد زیادی زبان برنامه‌نویسی وجود دارند که توسط بیش از یک IDE پشتیبانی می‌شوند. برخی از برنامه‌نویسان به جای استفاده از یک IDE، در یک محیط خام متن‌نویسی (مانند Notepad یا TextEdit) کد می‌نویسند، اما این محیط‌ها امکاناتی مانند رنگ‌بندی اتوماتیک کلمات کلیدی و یا تکمیل خودکار عبارت‌ها را به صورت پیش‌فرض در اختیار برنامه‌نویس قرار نمی‌دهند.

از IDE به کامپایلر: صدای منو داری؟

وقتی در IDE تان، دکمه Build and Run را می‌زنید، IDE بلافاصله یک تماس با کامپایلر می‌گیرد و از او می‌خواهد که کد را بفهمد و ترجمه کند. کامپایلر هم بلافاصله و با توجه به نوع زبان، شروع به بررسی تک‌تک خطوط کد می‌کند. سپس، کامپایلر کدها را به زبان قابل فهم برای کامپیوتر تبدیل کرده و آن‌ها را اجرا می‌کند.

برنامه‌ی اجرایی

خروجی عملیات کامپایلر، یک فایل اجرایی (exe) است که IDE با دانستن مکان آن، فایل را اجرا می‌کند. مکان گرفتن ورودی و نشان دادن خروجی‌ها، ممکن است پنجره‌ی سیاه رنگ (یا سفید رنگ!) ترمینال یا Command Line باشد (در IDE هایی مثل کدبلاکس و ویژوال استودیو) و یا محیطی درون خود IDE باشد (در IDE هایی هم‌چون CLion و ایکس‌کد). با اجرا شدن برنامه‌ی اجرایی، می‌توانید نتیجه‌ی ساعت‌ها و حتی روزها تلاش بی‌وقفه‌تان را دیده و از آن لذت ببرید! :)

در این مقاله سعی شد خلاصه‌ای هرچند کوتاه از کامپایلر و نحوه کارکرد آن و تفاوت‌هایش با IDE بیان شود. امیدواریم که با خواندن این مقاله، تا حدود خوبی با دست‌های پشت‌پرده در رایانه آشنایی پیدا کرده باشید. در شماره‌های آینده‌ی کدنامه، ادامه‌ی جزئیات روند ترجمه و اجرای برنامه‌ها، بیان خواهند شد.