به نام خدا

مبانی برنامه‌نویسی

آرش شفیعی

برنامه‌نویسی سی

برنامه‌نویسی سی

- در این قسمت قصد داریم همهٔ ویژگی‌های مهم و ساختار کلی زبان سی را بدون ورود به جزئیات شرح دهیم.
- بنابراین در این قسمت هدف توصیف شرح کلی زبان سی است به طوری که بتوانیم با استفاده از آن الگوریتم‌های ساده را پیاده‌سازی کنیم. در فصل‌های بعدی بیشتر به شرح جزئیات خواهیم پرداخت.
- در این قسمت به طور خلاصه در مورد متغیرها و ثابت‌ها، محاسبات عددی، ساختارهای کنترلی، توابع و مقدمات ورودی و خروجی صحبت خواهیم کرد و قسمت‌های پیشرفته‌تر مانند اشاره‌گرها و ساختمان‌ها را در قسمت‌های بعد توضیح خواهیم داد.

برنامه‌نویسی سی

- اولین برنامه‌ای که در اولین درس همهٔ زبان‌های برنامه‌نویسی نوشته می‌شود برنامه‌ای است که جملهٔ "hello, world" را بر روی خروجی چاپ می‌کند.
- این برنامه در زبان سی به صورت زیر است.

```c
#include <stdio.h>
int main() {
  printf("hello, world\n");
}
```

- این برنامه در یک فایل سورس با پسوند c. ذخیره می‌شود. برای کامپایل این برنامه می‌توان از ابزار gcc استفاده کرد. دستور gcc hello.c یک فایل آبجکت a.out تولید می‌کند که قابلیت اجرا شدن دارد و جمله "hello, world" را چاپ می‌کند.

برنامه‌نویسی سی

- یک برنامهٔ سی به طور کلی از تعدادی تابع تشکیل شده است. یک تابع با استفاده از تعدادی دستورات محاسباتی و ساختارهای کنترلی تعدادی ورودی را دریافت کرده و عملیاتی را بر روی ورودی اعمال و صفر یا یک خروجی تولید می‌کند.
- تابع `main` تابع اصلی برنامه است یعنی یک برنامه با اجرای تابع `main` آغاز می‌شود. توابع می‌توانند نام‌های دلخواه داشته باشند، اما تابع `main` یک تابع ویژه است که باید برای شروع برنامه وجود داشته باشد. در دستورات تابع `main` می‌توانند توابع دیگر فراخوانی شوند.
- در ابتدای برنامه کتابخانه‌های مورد نیاز اضافه می‌شوند. هر کتابخانه شامل توابعی است که کاربرد خاصی دارند و وظایف خاصی انجام می‌دهند. برای مثال کتابخانه `stdio` یک کتابخانه استاندارد برای انجام عملیات ورودی خروجی [1] است.

[1] standard input output library

برنامه‌نویسی سی

- در اولین برنامه در تابع `main` از تابع `printf` برای چاپ یک رشته استفاده کردیم. یک تابع معمولاً تعدادی پارامتر [1] دریافت می‌کند و عملیات خود را بر روی ورودی‌ها انجام می‌دهد و خروجی تولید می‌کند. یک تابع همچنین می‌تواند بدون پارامتر باشد. متغیرهایی که به پارامترهای یک تابع ارسال می‌شوند را آرگومان [2] می‌نامیم.
- همهٔ دستورات یک تابع در بین در آکولاد بازو بسته {} قرار می‌گیرند.
- به تابع `printf` یک رشته (`"hello, world\n"`) به عنوان آرگومان ارسال می‌شود. یک رشته عبارتی است که از تعدادی حرف (کاراکتر) تشکیل شده است. کاراکتر `\n` برای اتمام خط فعلی و ادامهٔ چاپ با شروع خط بعدی استفاده می‌شود.
- هنگام چاپ رشته همچنین می‌توان از کاراکترهای ویژهٔ دیگر مانند `\t` برای فاصلهٔ ستونی، `\"` برای چاپ علامت نقل قول و `\\` برای چاپ بک‌اسلش [3] استفاده کرد.

[1] parameter
[2] argument
[3] backslash

متغیرها و ساختارهای کنترل

- حال می‌خواهیم برنامه‌ای نویسیم که دمای اندازه‌گیری شده در واحد فارنهایت را به سلسیوس تبدیل کند. فرمول تبدیل فارنهایت به سلیوس به صورت زیر است.

$$C = (5/9)(F-32)$$

متغیرها و ساختارهای کنترل

- این برنامه به صورت زیر نوشته می‌شود.

```
۱  #include <stdio.h>
۲  /* print Fahrenheit-Celsius table for fahr = 0, 20, ..., 300 */
۳  int main () {
۴    int fahr, celsius;
۵    int lower, upper, step;
۶    lower = 0; /* lower limit of temperature scale */
۷    upper = 300; /* upper limit */
۸    step = 20; /* step size */
۹    fahr = lower;
۱۰   while (fahr <= upper) {
۱۱       celsius = 5 * (fahr - 32) / 9;
۱۲       printf ("%d\t%d\n", fahr, celsius);
۱۳       fahr = fahr + step;
۱۴   }
۱۵ }
```

- پس از معرفی کتابخانه `stdio` در ابتدای برنامه (برای انجام عملیات چاپ بر روی صفحه نمایش)، توضیحاتی [1] درمورد برنامه در خط دوم داده شده است. این توضیحات، تنها برای شرح عملکرد برنامه استفاده می‌شوند و کامپایلر در هنگام کامپایل برنامه آنها را نادیده می‌گیرد. هر متنی بین دو علامت `*/` و `/*` قرار بگیرد جزء توضیحات محسوب می‌شود. توضیحات باعث می‌شوند عملکرد برنامه توسط برنامه‌نویسان دیگر بهتر فهمیده شود.

[1] comment

متغیرها و ساختارهای کنترل

- با استفاده از متغیرها [1] می‌توان مقادیری را توسط یک نام معین ذخیره و نگهداری کرد. مقدار متغیرها در طول برنامه قابل تغییر است.
- در زبان سی متغیرها به همراه نوع‌شان تعریف می‌شوند. یک متغیر می‌تواند از نوع عدد صحیح، عدد اعشاری، کاراکتر یا غیره باشد.
- در برنامه تبدیل فارنهایت به سلسیوس می‌خواهیم مقادیر فارنهایت و سلسیوس را نگهداری کنیم. همچنین می‌خواهیم کمترین و بیشترین مقدار دما به فارنهایت که در برنامه قابل تبدیل است را نگهداری کنیم و به علاوه می‌خواهیم برای نمونه برداری دما به فارنهایت فاصله بین دو دما را در یک متغیر ذخیره کنیم.
- بنابراین به ۵ متغیر نیاز داریم که همه از نوع صحیح هستند.

```
1 int fahr, celsius;
2 int lower, upper, step;
```

[1] variables

متغیرها و ساختارهای کنترل

- نوع `int` برای عدد صحیح، نوع `float` برای اعداد اعشاری، نوع `double` برای اعداد اعشاری با دقت دو برابر، و نوع `char` برای حروف به کار می‌رود.
- هر نوع متغیر اندازهٔ معینی دارد. برای مثال متغیر `int` در ماشین‌های ۳۲ بیتی ۲ بایت و در ماشین‌های ۶۴ بیتی ۴ بایت است.
- در ماشین‌های ۳۲ بیتی آدرس‌ها ۳۲ هستند و بنابراین تعداد $۲^{۳۲}$ مکان حافظه را می‌توان آدرس دهی کرد. در ماشین‌های ۶۴ بیتی تعداد $۲^{۶۴}$ آدرس حافظه وجود دارد.
- بنابراین یک ماشین ۳۲ بیتی حداکثر ۴ گیگابایت حافظه (معادل $۲^{۳۲}$ بایت) را می‌توانند آدرس دهی کند و یک ماشین ۶۴ بیتی حداکثر ۱۶ میلیون ترابایت.

متغیرها و ساختارهای کنترل

- یک متغیر صحیح در یک کامپیوتر ۶۴ بیتی ۴ بایت است. بنابراین برای اعداد صحیح مثبت می‌توان از صفر تا ۴۲۹۴۹۶۷۲۹۵ را در آنها ذخیره کرد.
- نوع‌های داده‌ای دیگر نیز وجود دارند که بعدها بیشتر درمورد آنها صحبت خواهیم کرد.
- در برنامهٔ تبدیل فارنهایت به سانتیگراد یک حلقه وجود دارد که در آن حلقه می‌خواهیم یک روند را تکرار کنیم. به ازای هر یک از مقادیر در واحد فارنهایت، عملیات لازم برای تبدیل به سانتی‌گراد یکسان است، پس این عملیات به ازای همهٔ اعداد باید تکرار شود. این حلقه باید تا زمانی تکرار شود که مقدار متغیر `fahr` کمتر از کران بالای `upper` باشد. بنابراین حلقه به صورت `while (fahr <= upper){}` نوشته می‌شود یعنی تا زمانی که `fahr` کوچکتر یا مساوی `upper` باشد عملیات تکرار شود.

- اگر بدنهٔ حلقه شامل تنها یک دستور باشد می‌توانیم از علامت آکولاد استفاده نکنیم. همچنین معمولاً برای خوانایی یک برنامه از دندانه‌گذاری [1] استفاده می‌کنیم، بدین معنی که دستورات درون حلقه، با چند خط فاصله شروع می‌شوند.
- تبدیل فارنهایت به سلسیوس با استفاده از دستور `celsius = 5 * (fahr - 32) / 9` انجام می‌شود. دلیل اینکه مقدار `(fahr - 32)` را ابتدا در ۵ ضرب و سپس بر ۹ تقسیم می‌کنیم این است که در غیراینصورت مقدار `5/9` که تقسیم دو عدد صحیح است برابر است با عدد صحیح صفر خواهد بود و مقدار `(fahr - 32)*(5/9)` همیشه صفر خواهد بود.

[1] indentation

- در برنامه تبدیل فارنهایت به سلسیوس همچنین از تابع `printf` استفاده کردیم. این تابع برای چاپ یک رشته استفاده می‌شود. هریک از کلمات این رشته، یا به عبارت دیگر زیررشته‌ها، که با علامت درصد آغاز می‌شود، با مقادیر پارامترهای دوم به بعد جایگزین می‌شود.
- در مثال قبل دو زیر رشتهٔ `%d` که در آرگومان اول به تابع چاپ ارسال شده است با مقادیر آرگومان‌های دوم و سوم جایگزین می‌شود و بنابراین مقادیر فارنهایت و سلسیوس در خروجی چاپ می‌شوند.
- تابع `printf` جزئی از زبان سی نیست بلکه تابعی در کتابخانه استاندارد مربوط به ورودی و خروجی `stdio` است.

متغیرها و ساختارهای کنترل

- ورودی اول تابع `printf` رشته‌ای است که در خروجی استاندارد چاپ می‌شود. این رشته می‌تواند شامل زیر رشته‌هایی باشد که نحوهٔ نمایش (فرمت) [1] خروجی را تعیین می‌کند. این زیر رشته‌ها که با علامت % شروع می‌شوند را تعیین کنندهٔ فرمت [2] می‌نامیم.
- تعیین کنندهٔ فرمت با مقادیر ورودی‌های دوم به بعد تابع `printf` جایگزین می‌شود و اعداد و رشته‌ها را با فرمت تعیین شده در خروجی استاندارد چاپ می‌کند.
- یک تعیین کنندهٔ فرمت می‌تواند `c%` برای چاپ کاراکتر، `d%` برای چاپ اعداد صحیح دهدهی، `f%` برای چاپ اعداد اعشاری باشد.

[1] format
[2] format specifier

متغیرها و ساختارهای کنترل

- برنامهٔ تبدیل فارنهایت به سلسیوس می‌تواند خروجی را به گونه‌ای چاپ کند که هر دو ستون اعداد (مقادیر فارنهایت و سلسیوس) از سمت راست همتراز شوند. برای این کار از دستور زیر استفاده می‌کنیم.

```
printf("%3d %6d \n" , fahr, celsius);
```

- در این دستور عدد ۳ قبل از حرف d تعیین می‌کند که عرض ستون چاپ برای عدد صحیح اول ۳ باشد.
- همچنین در این برنامه می‌توانیم از نوع اعداد اعشاری به جای اعداد صحیح برای مقادیر استفاده کنیم.

متغیرها و ساختارهای کنترل

- برای چاپ کردن اعداد اعشاری از f% استفاده می‌کنیم. همچنین می‌توانیم تعیین کنیم اعداد اعشاری با چه دقتی چاپ شوند. برای مثال در دستور زیر دمای سلسیوس با ۱ رقم بعد از اعشار چاپ می‌شود.

```
printf("%3.0f %6.1f \n" , fahr, celsius);
```

- همچنین برای تبدیل فارنهایت و سلسیوس می‌توانیم بنویسیم.

```
celsius = ( 5.0 / 9.0 ) * ( fahr - 32.0 )
```

- در اینجا با تقسیم عدد اعشاری 5.0 بر 9.0 یک عدد اعشاری به دست می‌آید و محاسبات به درستی انجام می‌شود.

متغیرها و ساختارهای کنترل

- یک برنامه می‌تواند معمولاً به شکل‌های مختلف نوشته شود. برای مثال چندین نوع ساختار حلقه در زبان سی وجود دارد که می‌توان آنها را به جای یکدیگر استفاده کرد.
- برنامهٔ تبدیل دماهای فارنهایت به سلسیوس به شکلی دیگر در زیر نوشته شده است.

```
۱ #include <stdio.h>
۲ /* print Fahrenheit-Celsius table */
۳ int main ()
۴ {
۵   int fahr;
۶   for (fahr = 0; fahr <= 300; fahr = fahr + 20)
۷     printf ("%3d %6.1f\n", fahr, (5.0 / 9.0) * (fahr - 32));
۸ }
```

نماد ثابت

- در برنامه قبل از اعداد ۳۰۰ و ۲۰ در برنامه بدون نام استفاده کردیم. این نوع برنامه‌نویسی گرچه برنامه را مختصر می‌کند، ولی باعث می‌شود خواندن برنامه برای دیگر برنامه‌نویسان مشکل شود.
- با استفاده از کلید واژهٔ `#define` می‌توانیم یک نماد ثابت [1] تعریف کنیم.

[1] constant

نماد ثابت

- برای مثال در برنامه قبل می‌توانستیم اعداد کران بالا و کران پایین و مقدار افزایش را به صورت زیر تعریف کنیم.

```
۱  #include <stdio.h>
۲  #define LOWER 0 /* lower limit of table */
۳  #define UPPER 300 /* upper limit */
۴  #define STEP 20 /* step size */
۵  /* print Fahrenheit-Celsius table */
۶  int main ()
۷  {
۸    int fahr;
۹    for (fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP)
۱۰     printf ("%3d %6.1f\n", fahr, (5.0 / 9.0) * (fahr - 32));
۱۱ }
```

ورودی و خروجی

- ورودی و خروجی در زبان سی به صورت جریان‌های متنی [1] یا استریم‌های متنی هستند. یک استریم متنی دنباله‌ای است از تعدادی رشته که می‌توانند از یکدیگر با کاراکتر خط جدید [2] جدا شده باشند. هر رشته شامل تعدادی حرف یا کاراکتر است.
- در کتابخانهٔ استاندارد تعدادی تابع برای دریافت ورودی و چاپ خروجی طراحی شده‌اند. برای مثال تابع `getchar()` حرف بعدی را از ورودی استاندارد می‌خواند و به عنوان یک متغیر از نوع کاراکتر بازمی‌گرداند. همچنین تابع `putchar(c)` محتوای یک کاراکتر را در خروجی استاندارد چاپ می‌کند.

[1] text streams
[2] newline character

ورودی و خروجی

- برای مثال می‌خواهیم برنامه‌ای بنویسیم که یک کاراکتر را از ورودی دریافت کند و در صورتی که ورودی، کاراکتر پایان فایل [1] نبود، آن کاراکتر را در خروجی چاپ کند.

[1] end of file character

ورودی و خروجی

- این برنامه به صورت زیر است :

```
۱  #include <stdio.h>
۲  /* copy input to output; */
۳  int main ()
۴  {
۵    int c;
۶    c = getchar();
۷    while (c != EOF)
۸      {
۹        putchar(c);
۱۰       c = getchar();
۱۱     }
۱۲ }
```

ورودی و خروجی

- در برنامهٔ قبل خروجی `getchar()` را به جای یک کاراکتر برابر با یک عدد صحیح `int` قرار دادیم. باید دقت کرد که یک کاراکتر در واقع یک عدد را در حافظه نگهداری می‌کند بنابراین `char` و `int` می‌توانند به جای یکدیگر استفاده شوند. البته نوع کاراکتر یک بایت است و نوع عدد صحیح ۲ یا ۴ بایت.
- در ورودی استاندارد از `Ctrl + D` برای وارد کردن کاراکتر پایان فایل استفاده می‌کنیم.

- برنامه قبل را می‌توانیم به صورت مختصرتر به صورت زیر بنویسیم :

```
۱ #include <stdio.h>
۲ /* copy input to output; 2nd version */
۳ int main ()
۴ {
۵   int c;
۶   while ((c = getchar ()) != EOF)
۷     putchar (c);
۸ }
```

- در واقع یک عبارت انتساب دارای یک مقدار است که مقدار آن برابر با مقدار سمت چپ عبارت انتساب است. بنابراین می‌توانیم مقدار (`c = getchar()`) را با `EOF` مقایسه کنیم.
- همچنین پرانتزگذاری عبارت `c = getchar() != EOF` با اهمیت است. عملگرها در زبان دارای تقدم یا اولویت [1] هستند. عملگرهای با اولویت بالاتر در یک عبارت زودتر اجرا می‌شوند. اولویت `!=` از اولویت `=` بالاتر است. بنابراین `c = getchar() != EOF` معادل است با

```
c = (getchar() != EOF)
```

- معنی این عبارت این است که مقدار عبارت `getchar() != EOF` سنجیده شود (این مقدار برابر با صفر است اگر نابرابری برقرار نباشد و برابر با یک است اگر نابرابری برقرار باشد) و سپس مقدار به دست آمده (صفر یا یک) در متغیر `c` ذخیره شود.

[1] precedeuce

- حال می‌خواهیم برنامه‌ای بنویسیم که کاراکترهای ورودی را بخواند و تعداد آنها را در پایان چاپ کند. این برنامه به صورت زیر است.

```
۱  #include <stdio.h>
۲  /* count characters in input; 1st version */
۳  int main ()
۴  {
۵    long nc;
۶    nc = 0;
۷    while (getchar() != EOF)
۸      ++nc;
۹    printf ("%ld\n", nc);
۱۰ }
```

ورودی و خروجی

- عملگر ++ یک واحد به مقدار متغیر اضافه می‌کند. می‌توانیم عبارت nc++ را به صورت nc = nc+1 نیز بنویسیم.
- همچنین ++nc یک واحد به متغیر می‌افزاید. تفاوت افزایش پیشوندی و پسوندی در این است که در عبارت پیشوندی x = ++nc ابتدا مقدار متغیر یک واحد افزایش پیدا می‌کند و سپس عملیات انتساب انجام می‌شود ولی در عبارت پسوندی x = nc++ ابتدا عملیات انتساب انجام می‌شود و سپس متغیر یک واحد افزایش می‌یابد.
- همچنین عملگر -- یک واحد از مقدار یک متغیر می‌کاهد.
- برای ذخیرهٔ متغیر nc از نوع int استفاده نکردیم زیرا ممکن است تعداد کاراکترها آنقدر زیاد باشد که در متغیر int نگنجد. یک متغیر از نوع long یا عدد صحیح بزرگ در کامپیوترهای ۳۲ بیتی ۴ بایت و در کامپیوترهای ۶۴ بیتی ۸ بایت است.
- برای چاپ یک متغیر از نوع long در تابع چاپ از تعیین کنندهٔ فرمت %ld استفاده می‌کنیم.

- می‌توانستیم از نوع داده‌ای double نیز به صورت زیر در یک حلقهٔ for استفاده کنیم.

```
۱ #include <stdio.h>
۲ /* count characters in input; 2nd version */
۳ int main ()
۴ {
۵   double nc;
۶   for (nc = 0; getchar () != EOF; ++nc);
۷   printf ("%.0f\n", nc);
۸ }
```

- بدنهٔ حلقهٔ for در این مثال خالی است، زیرا در دستور حلقه همهٔ کارها انجام می‌شود.
- همچنین در مثال‌های قبل اگر هیچ کاراکتری وارد نشود، تعداد کاراکترهای ورودی صفر اعلام خواهد شد که نتیجه مورد نظر است.

ورودی و خروجی

- حال می‌خواهیم برنامه‌ای بنویسیم که تعداد خط‌های برنامه را بشمارد. در اینجا باید بررسی کنیم آیا یک حرف وارد شده برابر کاراکتر خط جدید \n است یا خیر.
- این برنامه به صورت زیر نوشته می‌شود.

```
۱  #include <stdio.h>
۲  /* count lines in input */
۳  int main ()
۴  {
۵    int c, nl;
۶    nl = 0;
۷    while ((c = getchar ()) != EOF)
۸      if (c == '\n')
۹        ++nl;
۱۰   printf ("%d\n", nl);
۱۱ }
```

- عملگر == به معنای برابری است، در حالی که دیدیم عملگر = به معنی انتساب است. ممکن است با اشتباه در استفاده این عملگرها برنامه‌ای به دست آید که به درستی کامپایل می‌شود ولی نتیجه درست به دست نمی‌دهد.
- یک کاراکتر را با یک علامت نقل قول نشان می‌دهیم. بنابراین `'\n'` به معنای کاراکتر خط جدید است. هر کاراکتری یک کد معادل استاندارد اسکی [1] دارد. برای مثال کد معادل `'A'` برابر است با ۶۵.
- کاراکتر `'\n'` برابر با کد اسکی ۱۰ است. توجه کنید که `'\n'` یک کاراکتر است نه دو کاراکتر. برای نمایش حرف `n` از `'n'` و برای نمایش حرف \ از `'\\'` استفاده می‌کنیم.

[1] American Standard Code for Information Interchange (ASCII)

ورودی و خروجی

- حال فرض کنید می‌خواهیم تعداد خطوط، کلمات و حروف را در ورودی بشماریم. کلمات از یکدیگر با کاراکتر خط فاصله یا کاراکتر خط جدید یا ستون جدید جدا می‌شوند.

ورودی و خروجی

- این برنامه به صورت زیر نوشته می‌شود. برنامه wc در لینوکس همین عملیات را انجام می‌دهد.

```
۱ #include <stdio.h>
۲ #define IN 1     /* inside a word */
۳ #define OUT 0    /* outside a word */
۴ /* count lines, words, and characters in input */
۵ int main ()
۶ {
۷   int c, nl, nw, nc, state;
۸   state = OUT;
۹   nl = nw = nc = 0;
```

```
11    while ((c = getchar ()) != EOF)
12      {
13            ++nc;
14            if (c == '\n')
15                ++nl;
16            if (c == ' ' || c == '\n' || c == '\t')
17                state = OUT;
18            else if (state == OUT)
19                {
20                      state = IN;
21                      ++nw;
22                }
23      }
24    printf ("%d %d %d\n", nl, nw, nc);
25  }
```

ورودی و خروجی

- در این برنامه با استفاده از یک متغیر `state` بررسی می‌کنیم آیا در حین پردازش یک کلمه هستیم یا کلمهٔ قبلی به پایان رسیده است. برای این کار از دو مقدار `IN` و `OUT` که در واقع مقادیر ۰ و ۱ هستند استفاده می‌کنیم. استفاده از این نام‌ها به جای مقادیر برنامه را خواناتر می‌کند.

- گرچه در برنامه‌های کوچک ممکن است خوانایی برنامه نسبت به حجم برنامه و میزان استفاده از حافظه اهمیت کمتری داشته باشد ولی در برنامه‌های بزرگ‌تر خوانایی برنامه اهمیت بسیار زیادی پیدا می‌کند زیرا هزینهٔ نگهداری برنامه نیز یکی از معیارهای مهم در طراحی برنامه است. هرچه یک برنامه خواناتر باشد، پیدا کردن مشکلات و همچنین ایجاد تغییرات در آن آسان‌تر می‌شود و بنابراین هزینهٔ نگهداری آن پایین می‌آید.

- در ابتدای برنامه از عبارت انتساب `nl = nw = nc = 0;` استفاده کردیم. با توجه به این‌که وابستگی عملگر [1] تساوی از راست به چپ است، این عبارت به صورت `nl = (nw = (nc = 0));` ترجمه می‌شود.

[1] operator associativity

- عملگر || به معنی فصل منطقی است که یای منطقی نیز نامیده می‌شود. عطف منطقی به صورت && است. اولویت عطف بالاتر از فصل است.
- مقدار یک عبارت منطقی از چپ به راست ارزیابی می‌شود و به محض اینکه مقدار منطقی آن تعیین شده محاسبات ادامه پیدا نمی‌کند. برای مثال در برنامه قبل در عبارت
`if ( c==' ' || c=='\n' || c='\t' )`
اگر مقدار c برابر با کاراکتر خط فاصله بود ارزیابی عبارت منطقی ادامه پیدا نمی‌کند زیرا مقدار آن در هر صورت برابر با ۱ خواهد بود.

ساختار شرطی

- در زبان سی ساختار شرطی به صورت زیر نوشته می‌شود.

```
if (expression) {
    statement1
} else {
    statement2
}
```

- این دستورات بدین معنی است که اگر مقدار منطقی `expression` برابر با مقدار درست (`true`) باشد عبارت یا مجموعه عبارات `statement1` اجرا می‌شوند و در غیر اینصورت، یعنی اگر مقدار منطقی عبارت `expression` برابر با مقدار نادرست (`false`) باشد، عبارت یا مجموعه عبارات `statement2` اجرا می‌شوند.

ساختار شرطی

- یک عبارت منطقی یا بولی [1] عبارتی است که از متغیرهای منطقی و عملگرهای منطقی تشکیل شده است.
- مقدار یک متغیر منطقی می‌تواند درست (`true`) یا نادرست باشد (`false`). در زبان سی عدد صفر مقدار نادرست، و عدد یک مقدار درست را نشان می‌دهد. به طور کل مقدار منطقی همهٔ مقادیر غیرصفر درست و مقدار منطقی عدد صفر نادرست است.
- چند عملگر مهم منطقی برای توصیف عبارات منطقی وجود دارند که عبارتند از : عملگر عطف (و) منطقی (AND) ، عملگر فصل (یا) منطقی (OR) ، و عملگر نقیض (NOT) .
- در زبان سی عملگر عطف منطقی توسط && ، فصل منطقی توسط || ، و نقیض توسط علامت ! نشان داده می‌شوند.

[1] logical or boolean expression

ساختار شرطی

- نتیجهٔ عطف دو متغیر منطقی تنها در صورتی درست است که مقدار هر دو عملوند درست باشد.
- نتیجهٔ فصل دو متغیر منطقی در صورتی درست است که مقدار حداقل یکی از دو عملوند درست باشد.
- بنابراین می‌توانیم نتایج اعمال عملگرهای عطف و فصل بر روی دو عملوند منطقی را به صورت زیر بنویسیم.

x	y	x && y	x || y
درست	درست	درست	درست
درست	نادرست	نادرست	درست
نادرست	درست	نادرست	درست
نادرست	نادرست	نادرست	نادرست

- نقیض مقدار درست، نادرست، و نقیض مقدار نادرست، درست است.

x	!x
درست	نادرست
نادرست	درست

ساختار شرطی

- فرض کنید در یک برنامه می‌خواهیم اگر نمره‌ای بزرگتر یا مساوی ۱۰ و کوچکتر یا مساوی ۲۰ باشد، عبارت «قبول» (pass) و در صورتی که نمره کوچکتر از ۱۰ و بزرگتر یا مساوی ۰ باشد، عبارت «رد» (fail) و در غیر اینصورت عبارت «غیرمعتبر» (invalid) را چاپ کنیم.

```
۱ if (mark <= 20 && mark >= 10) {
۲     printf("pass\n");
۳ } else if (mark >=0 && mark < 10) {
۴     printf("fail\n");
۵ } else {
۶     printf("invalid\n");
۷ }
```

آرایه‌ها

- حال فرض کنید می‌خواهیم برنامه‌ای بنویسیم که میانگین ۵ عدد را پیدا کند.
- می‌توانیم برای هر یک از این ۵ عدد یک متغیر در نظر بگیریم، اما ممکن است در آینده بخواهیم میانگین ۵۰۰۰ عدد را محاسبه کنیم.
- در چنین مواردی از آرایه‌ها استفاده می‌کنیم. یک آرایه متغیری است که می‌تواند تعدادی مقادیر از یک نوع را در خود نگهداری کند.

آرایه‌ها

- برنامه محاسبه میانگین را به صورت زیر می‌نویسیم.

```c
#include <stdio.h>
int main() {
    float marks[5];
    marks[0] = 19;
    marks[1] = 20;
    marks[2] = 18;
    marks[3] = 15;
    marks[4] = 16.5;
    float sum = 0;
    for (int i=0; i<5; i++) {
            sum = sum + marks[i];
    }
    float average = sum / 5;
    printf("Average : %.2f\n", average);
}
```

آرایه‌ها

- عناصر آرایه را به صورت زیر نیز می‌توانیم مقداردهی اولیه کنیم.

```
۱  #include <stdio.h>
۲  int main() {
۳      float marks[5] = {19, 20, 18, 15, 16.5};
۴      float sum = 0;
۵      for (int i=0; i<5; i++) {
۶              sum = sum + marks[i];
۷      }
۸      float average = sum / 5;
۹      printf("Average : %.2f\n", average);
۱۰ }
```

آرایه‌ها

- این برنامه ممکن است در آینده تغییر کند و بخواهیم میانگین را برای تعداد بیشتری اعداد محاسبه کنیم، بنابراین بهتر است یک ثابت برای مقدار ۵ تعریف کنیم. با این کار خوانایی برنامه افزایش پیدا می کند و همچنین تغییر برنامه راحت تر انجام می‌شود.

```
۱  #include <stdio.h>
۲  #define NUM 5
۳  int main() {
۴      float marks[NUM] = {19, 20, 18, 15, 16.5};
۵      float sum = 0;
۶      for (int i=0; i<NUM; i++) {
۷              sum = sum + marks[i];
۸      }
۹      float average = sum / NUM;
۱۰     printf("Average : %.2f\n", average);
۱۱ }
```

آرایه‌ها

- حال فرض کنید می‌خواهیم برنامه‌ای بنویسیم که تعداد رخداد هر رقم و حروف خطوط فاصله را در یک متن بشمارد.
- برای شمردن ارقام به ۱۰ متغیر نیاز داریم. به جای ۱۰ متغیر می‌توانیم از یک آرایه با ۱۰ عنصر به صورت زیر استفاده کنیم.

آرایه‌ها

```c
#include <stdio.h>
/* count digits, white space, others */
int main ()
{
  int c, i, nwhite, nother;
  int ndigit[10];
  nwhite = nother = 0;
  for (i = 0; i < 10; ++i)
    ndigit[i] = 0;
```

```
11    while ((c = getchar ()) != EOF)
12      if (c >= '0' && c <= '9')
13        ++ndigit[c - '0'];
14      else if (c == ' ' || c == '\n' || c == '\t')
15        ++nwhite;
16      else
17        ++nother;
18    printf ("digits =");
19    for (i = 0; i < 10; ++i)
20      printf (" %d", ndigit[i]);
21    printf (", white space = %d, other = %d\n", nwhite, nother);
22  }
```

آرایه‌ها

- متغیری از نوع آرایه به صورت `int ndigit[10]` تعریف کردیم که شامل ۱۰ عنصر است. هر عنصر آن یک عدد صحیح است. برای مثال `ndigit[0]` یک عدد صحیح است برای شمردن ارقام صفر، و به همین ترتیب `ndigit[9]` یک عدد صحیح است برای شمردن ارقام ۹ در متن وارد شده.
- اندیس یک آرایه همیشه یک عدد صحیح است.
- همانطور که گفتیم یک کاراکتر یک مقدار عددی دارد بنابراین می‌توانیم بنویسیم `if(c >='0' && c<='9')` بدین معنی که اگر معادل عددی حرف وارد شده از معادل عددی حرف `'0'` بیشتر و از معادل عددی حرف `'9'` کمتر است.
- فاصلهٔ یک کاراکتر با کاراکتر `'0'` معادل مکان آن در آرایه خواهد بود زیرا معادل عددی کاراکترهای ارقام به ترتیب به صورت اعداد متوالی هستند. پس می‌توانیم بنویسیم `ndigit[c - '0']`
- برای دستورات شرطی پی‌درپی از `if` و `else if` استفاده کنیم.

توابع

- یک تابع تعدادی دستورات را به صورت یک گروه در می‌آورد و با یک نام نامگذاری می‌کند به طوری که با فراخوانی نام تابع تمام دستورات اجرا می‌شوند.
- در فراخوانی یک تابع درواقع نیازی نداریم بدانیم تابع و عملیات آن چگونه اجرا می‌شوند، بلکه تنها کافی است بدانیم آن تابع چه کاری انجام می‌دهد.
- با استفاده از توابع همچنین می‌توانیم که برنامه نظم بیشتری بدهیم.

توابع

- فرض کنید می‌خواهیم تابعی بنویسیم که مقدار x^y را محاسبه می‌کند. در واقع این تابع دو متغیر `x` و `y` را دریافت می‌کند و مقداری را از تابع `power(x,y)` بازمی‌گرداند.

توابع

- این تابع به صورت زیر پیاده‌سازی و فراخوانی می‌شود.

```c
#include <stdio.h>
int power (int m, int n);
/* test power function */
int main ()
{
  int i;
  for (i = 0; i < 10; ++i)
    printf ("%d %d %d\n", i, power (2, i), power (-3, i));
  return 0;
}
```

توابع

```
۱۱ /* power: raise base to n-th power; n >= 0 */
۱۲ int power (int base, int n)
۱۳ {
۱۴   int i, p;
۱۵   p = 1;
۱۶   for (i = 1; i <= n; ++i)
۱۷     p = p * base;
۱۸   return p;
۱۹ }
```

توابع

- یک تابع در حالت کلی تشکیل شده است از یک نوع داده بازگشتی، نام تابع، متغیرهای ورودی که پارامتر نامیده می‌شوند و بدنهٔ تابع.
- یک تابع در یک فایل تعریف می‌شود و نمی‌توان یک تابع را به دو قسمت تقسیم کرد. وقتی یک تابع در یک فایل جداگانه از تابع فراخوانی کننده قرار می‌گیرند، برای استفاده از آن باید فایل‌های آبجکت ساخته شده به یکدیگر پیوند داده شوند.
- تابع `power` در مثال قبل دارای دو پارامتر ورودی از نوع عدد صحیح و یک مقدار بازگشتی از نوع عدد صحیح است.
- متغیرهایی که در تعریف تابع به کار می‌روند را پارامتر و متغیرهایی که در فراخوانی تابع به تابع ارسال می‌شوند را آرگومان می‌نامیم.
- کلمهٔ کلیدی `return` برای بازگرداندن یک مقدار از تابع به کار می‌رود.

توابع

- در مثال قبل تابع main هم یک مقدار باز می‌گرداند. مقدار صفر نشان دهنده این است که برنامه بدون خطا متوقف شده است.
- یک تابع را می‌توانیم در یک قسمت اعلام کنیم و سپس در قسمتی دیگر تعریف کنیم. برای اعلام [1] تابع باید نوع خروجی، نام تابع و پارامترهای ورودی و نوع آنها مشخص شود. به مجموعهٔ نوع، نام، و پارامترهای ورودی تابع، پروتوتایپ تابع گفته می‌شود، پس در اعلام تابع تنها پروتوتایپ تابع [2] مشخص می‌شود.
- یک تابع پس از اعلام باید تعریف شود، در تعریف تابع [3] بدنهٔ تابع نیز باید مشخص شود.
- در اعلام تابع، الزامی به ذکر نام پارامترها وجود ندارد. بنابراین می‌توانیم بنویسیم.

```
int power (int, int);
```

[1] declare
[2] function prototype
[3] definition

فراخوانی با مقدار

- در زبان سی فراخوانی توابع به طور پیش فرض با مقدار است، بدین معنی که مقدار آرگومان‌ها در مقدار پارامترها کپی می‌شوند و تنها مقدار آرگومان‌ها در دسترس است نه مکان حافظه و آدرس آنها بنابراین متغیرهای ارسال شده به عنوان آرگومان در بدنه تابع در دسترس نیستند.
- فراخوانی با مقدار [1] در کنار فراخوانی با ارجاع [2] دو نوع فراخوانی در زبان سی هستند. فراخوانی به طور پیش‌فرض با مقدار است. در مورد فراخوانی با ارجاع در آینده توضیح خواهیم داد.

[1] call by value
[2] call by reference

فراخوانی با مقدار

- در مثال زیر، مقدار n در تابع کاهش می‌یابد ولی از آنجایی که n یک کپی از آرگومان دوم ارسال شده به تابع است، مقدار آرگومان دوم تغییر نخواهد کرد.

```
۱ /* power: raise base to n-th power; n >= 0; version 2 */
۲ int power (int base, int n)
۳ {
۴   int p;
۵   for (p = 1; n > 0; --n)
۶     p = p * base;
۷   return p;
۸ }
```

فراخوانی با مقدار

- وقتی نیاز داشته باشیم مقدار یک آرگومان را تغییر دهیم باید از فراخوانی با ارجاع استفاده کنیم. در فراخوانی با ارجاع پارامترها به صورت اشاره‌گر تعریف می‌شوند.
- وقتی یک آرایه به عنوان آرگومان به یک تابع ارسال می‌شود، فراخوانی با ارجاع است و نام آرایه به آدرس اولین عنصر آرایه که به عنوان آرگومان ارسال شده است اشاره می‌کند. عناصر یک آرایه از آرگومان به پارامتر کپی نمی‌شوند.

فراخوانی با مقدار

- بنابراین تابع زیر مقدار عناصر آرایه را تغییر می‌دهد.

```
۱  void change(int arr[], int n) {
۲      for (int i=0; i<n; i++)
۳          arr[i]*=2;
۴  }
۵  int main () {
۶      int x[] = {1,2,3,4};
۷      change(x, 4);
۸      for (int i=0; i<4; i++)
۹          printf("x[%d] = %d\n", i, x[i]);
۱۰ }
```

فراخوانی با مقدار

- اما تابع زیر مقدار ورودی تابع را تغییر نمی‌دهد.

```
۱ void change(int n) {
۲     n*=2;
۳ }
۴ int main () {
۵     int x = 4;
۶     change(x);
۷     printf("x = %d\n", x);
۸ }
```

آرایهٔ حروف

- آرایه‌ای که در زبان سی بیشترین استفاده را دارد، آرایهٔ حروف است. یک آرایهٔ حروف، آرایه‌ای است که نوع داده‌ای عناصر آن حروف یا کاراکترها هستند.
- می‌خواهیم برنامه‌ای بنویسیم که متنی را به صورت دنباله‌ای از خطوط دریافت می‌کند و بلندترین خط را چاپ می‌کند.

آرایهٔ حروف

- این برنامه را می‌توانیم ابتدا به صورت شبه‌کد بنویسیم :

۱. تا وقتی که یک خط دیگر وجود دارد.
۲. اگر طول خط فعلی از طول بلندترین خطی که تاکنون وارد شده بیشتر است.
۳. خط فعلی را ذخیره کن.
۴. طول خط فعلی را ذخیره کن.
۵. بلندترین خط را چاپ کن.

آرایهٔ حروف

- طرح کلی این برنامه را در چند خط بیان کردیم. در هریک از این خطوط تنها آنچه باید انجام شود، بدون شرح چگونگی انجام آن توصیف شد. پس هریک از این قسمت‌ها را می‌توانیم به صورت یک تابع بنویسیم.
- ابتدا به تابعی نیاز داریم که خط بعدی را دریافت کند.
- تابعی با نام `getline` تعریف کنیم و در تعریف این تابع سعی می‌کنیم آن را تا حد امکان به صورت عمومی تعریف کنیم یعنی تابع را به صورتی تعریف کنیم که در برنامه‌های مختلف بتواند استفاده شود.

آرایهٔ حروف

- برای مثال تابع `getline` را طوری طراحی می‌کنیم که طول یک خط را بازگرداند و در صورتی که به پایان متن رسیده‌ایم و خطی وجود ندارد مقدار صفر را بازگرداند. توجه کنید اگر در خطی هیچ کاراکتری وجود نداشته باشد، کاراکتر `'\n'` وجود دارد پس طول آن برابر با ۱ است.
- همچنین وقتی طول یک خط از طولانی‌ترین خط بیشتر باشد باید آن را در مکانی ذخیره کنیم. پس نیاز به یک تابع کپی `copy` داریم که یک خط را در یک متغیر که آرایه‌ای از کاراکترها ذخیره کند.

آرایهٔ حروف

- این برنامه را به صورت زیر می‌توانیم بنویسیم.

```
۱ #include <stdio.h>
۲ #define MAXLINE 1000 /* maximum input line length */
۳ int _getline (char line[], int maxline);
۴ void _copy (char to[], char from[]);
```

آرایهٔ حروف

```
۶  /* print the longest input line */
۷  int main ()
۸  {
۹    int len; /* current line length */
۱۰   int max; /* maximum length seen so far */
۱۱   char line[MAXLINE]; /* current input line */
۱۲   char longest[MAXLINE]; /* longest line saved here */
```

آرایهٔ حروف

```
16    max = 0;
17    while ((len = _getline (line, MAXLINE)) > 0)
18      if (len > max) {
19          max = len;
20          _copy (longest, line);
21      }
22    if (max > 0)      /* there was a line */
23      printf ("%s", longest);
24    return 0;
25  }
```

آرایهٔ حروف

```
/* getline: read a line into s, return length */
int _getline (char s[], int lim)
{
  int c, i;
  for (i = 0; i < lim - 1 && (c = getchar ()) != EOF && c != '\n'; ++i)
    s[i] = c;
  if (c == '\n') {
      s[i] = c;
      ++i;
    }
  s[i] = '\0';
  return i;
}
```

آرایهٔ حروف

```
۳۹ /* copy: copy 'from' into 'to'; assume to is big enough */
۴۰ void _copy (char to[], char from[])
۴۱ {
۴۲   int i;
۴۳   i = 0;
۴۴   while ((to[i] = from[i]) != '\0')
۴۵     ++i;
۴۶ }
```

آرایهٔ حروف

- اولین پارامتر تابع `getline` یک آرایه است که عناصر آن را کاراکترها تشکیل داده‌اند. و دومین پارامتر آن یک عدد صحیح است که ماکزیمم طول آرایه را مشخص می‌کند. از آنجایی که طول آرایه در `main` مشخص می‌شود و تابع `getline` از آن مطلع نیست، بنابراین نیاز داریم این مقدار را به عنوان آرگومان به تابع `getline` ارسال کنیم.
- همچنین تابع `getline` یک عدد صحیح به تابع `main` بازمی‌گرداند که طول خط خوانده شده از ورودی است.
- تابع `getline` کاراکتر `'\0'` را در پایان آرایه قرار می‌دهد. این کاراکتر را کاراکتر تهی [1] نیز می‌نامیم. کاراکتر تهی بدین معنی است که رشته به پایان رسیده است. دنباله‌ای از حروف که با علامت پایان مشخص می‌شوند را رشته [2] می‌نامیم.

[1] null character
[2] string

آرایهٔ حروف

- در زبان سی و کتابخانه‌های استاندارد زبان سی این قرار داد استفاده شده است که یک رشته با کاراکتر تهی پایان می‌یابد. برای مثال رشتهٔ `"hello\n"` را در حافظه ذخیره می‌کنیم، درواقع `"hello\n\0"` در حافظه ذخیره می‌شود.
- برای مثال در تابع `printf` می‌توانیم از تعیین کنندهٔ فرمت `%s` برای چاپ رشته استفاده کنیم. آرایهٔ حروف مورد نظر توسط تابع `printf` دریافت می‌شود و چاپ رشته تا جایی ادامه می‌یابد که به کاراکتر تهی برخورد کنیم.
- همچنین در تابع `copy` کپی کردن کاراکترها تا جایی ادامه می‌یابد که به کاراکتر تهی برخورد کنیم.

آرایهٔ حروف

- توجه کنید که طول آرایهٔ x در مثال زیر ۶ است، زیرا کاراکتر '\0' به انتهای آن افزوده می‌شود، اما طول آرایهٔ y برابر با ۵ است.

```
۱  void copy(char to[], char from[]) {
۲      int i=0;
۳      while ((to[i] = from[i])!='\0')
۴          i++;
۵  }
۶  int main () {
۷      char x[] = "hello";
۸      char y[] = {'h', 'e', 'l', 'l', 'o'};
۹      char z[10];
۱۰     copy(z, x);
۱۱     printf("%s\n", z);
۱۲ }
```

حوزهٔ تعریف

- در برنامهٔ قبل متغیرهای `line` و `longest` را در تابع `main` تعریف کردیم. این متغیرها فقط در تابع `main` تعریف شده‌اند و بیرون از تابع دسترسی به آنها امکان پذیر نیست.
- متغیرهایی که در یک تابع تعریف می‌شوند را متغیرهای محلی [1] تابع می‌نامیم. به عبارت دیگر این متغیرها تنها در حوزهٔ تابع [2] تعریف شده‌اند.
- یک متغیر محلی با فراخوانی تابع تعریف می‌شود و در حافظه قرار می‌گیرد و به محض اتمام اجرای تابع از حافظه حذف می‌شود و مقدار خود را از دست می‌دهد و دسترسی به آن امکان‌پذیر نیست. بنابراین در ابتدای تابع نیاز به مقداردهی اولیه این متغیرهای محلی داریم.

[1] local variables
[2] scope

حوزهٔ تعریف

- یک متغیر را می‌توانیم بیرون از توابع نیز تعریف کنیم. چنین متغیرهایی را متغیر عمومی [1] می‌نامیم. به متغیرهای عمومی متغیرهای خارجی [2] نیز گفته می‌شود.
- یک متغیر عمومی با شروع برنامه تعریف می‌شود و با اتمام برنامه از بین می‌رود بنابراین مقدار خود را در طول برنامه نگه‌می‌دارد.
- وقتی یک متغیر خارج از یک تابع تعریف شده باشد، می‌توانیم با استفاده از کلمهٔ کلیدی `extern` آن را اعلام کنیم.

[1] global variable
[2] external variable

حوزهٔ تعریف

- در مثال از متغیرهای عمومی به جای متغیرهای محلی استفاده شده است.

```
۱  #include <stdio.h>
۲  #define MAXLINE 1000     /* maximum input line size */
۳  int max;
۴  char line[MAXLINE];
۵  char longest[MAXLINE];    /* maximum length seen so far */
۶  /* current input line */
۷  /* longest line saved here */
۸  int _getline (void);
۹  void _copy (void);
۱۰ /* print longest input line; specialized version */
۱۱ int main ()
۱۲ {
۱۳   int len;
۱۴   extern int max;
```

حوزهٔ تعریف

```
۱۶    extern char longest[];
۱۷    max = 0;
۱۸    while ((len = _getline ()) > 0)
۱۹      if (len > max)
۲۰        {
۲۱      max = len;
۲۲      _copy ();
۲۳        }
۲۴    if (max > 0)        /* there was a line */
۲۵      printf ("%s", longest);
۲۶    return 0;
۲۷  }
```

```
28 /* getline: specialized version */
29 int
30 _getline (void)
31 {
32   int c, i;
33   extern char line[];
34   for (i = 0; i < MAXLINE - 1
35                       && (c = getchar ()) != EOF && c != '\n'; ++i)
36     line[i] = c;
37   if (c == '\n')
38     {
39       line[i] = c;
40       ++i;
41     }
42   line[i] = '\0';
43   return i;
44 }
```

حوزهٔ تعریف

```
۴۴ /* copy: specialized version */
۴۵ void _copy (void)
۴۶ {
۴۷   int i;
۴۸   extern char line[], longest[];
۴۹   i = 0;
۵۰   while ((longest[i] = line[i]) != '\0')
۵۱   ++i;
۵۲ }
```

حوزهٔ تعریف

- اگر تعریف متغیر قبل از تابع استفاده‌کننده از آن صورت گرفته باشد، نیازی به اعلام متغیر با کلمه `extern` نداریم. همچنین اگر یک برنامه از چند فایل تشکیل شده باشد و یک متغیر در یک فایل تعریف شده باشد و بخواهیم در یک فایل دیگر از آن استفاده کنیم، نیاز داریم با استفاده از کلمهٔ `extern` آن را اعلام کنیم.
- توابعی مانند `printf` که از آنها استفاده کردیم، در فایل‌های دیگر تعریف شده‌اند و بنابراین در ابتدای برنامه فایل `stdio.h` را ضمیمه کردیم. توابع ورودی و خروجی توسط توسعه دهندگان زبان سی در این فایل تعریف شده‌اند.
- نوع `void` برای یک متغیر به معنی نوع تهی است. تابعی که هیچ متغیری بازنمی‌گرداند، مقدار بازگشتی آن از نوع `void` است.

حوزهٔ تعریف

- وقتی یک متغیر را تعریف می‌کنیم، در واقع متغیر باید ساخته شود و بر روی حافظه قرار بگیرد. وقتی یک متغیر را اعلام می‌کنیم در واقع به کامپایلر می‌گوییم آن متغیر قبلا تعریف شده و اکنون می‌خواهیم از آن استفاده کنیم.

- تا آنجایی که امکان دارد بهتر است از متغیرهای عمومی و خارجی استفاده نکنیم. دلیل اول این است که استفاده زیاد از متغیرهای خارجی از خوانایی برنامه می‌کاهد. دلیل دوم این است که گاهی ممکن است توابع مختلف مقدار یک متغیر عمومی را به نحوی تغییر دهند که برنامه نویس نسبت به آن آگاه نباشد و این امر باعث ایجاد برنامه‌ای شود که از نظر منطقی دچار مشکل شود. سومین دلیل این است که ممکن است چند تابع در یک اجرای همزمان (در برنامه نویسی همروند) به طور همزمان یک متغیر را تغییر دهند که باعث ایجاد اشکال در اجرای برنامه شود. چهارمین دلیل این است که توابعی که پارامتر دریافت می‌کنند به طور عمومی تعریف می‌شوند و در همهٔ برنامه‌ها قابل استفاده هستند چون به صورت یک تابع مستقل عمل می‌کنند. بنابراین ترجیح می‌دهیم برنامه قبلی را با متغیرهای محلی پیاده‌سازی کنیم و نه متغیرهای عمومی.

تمرین

- برنامه‌ای بنویسید که عدد n را دریافت کند و مجموع $n + \cdots + ۲ + ۱$ را بازگرداند.
- برنامه‌ای بنویسید که عدد n را دریافت کند و زوج و فرد بودن آن را تعیین کند.
- برنامه‌ای بنویسید که عدد n را دریافت کند و !n را بازگرداند.
- برنامه‌ای بنویسید که عدد n را دریافت کرده و عدد فیبوناچی n ام را بازگرداند.
- برنامه‌ای بنویسید که انتگرال $f(x) = x^۲$ را به طور تقریبی از ۰ تا n محاسبه کند.
- برنامه‌ای بنویسید که عدد n را دریافت کرده، و ارقام آن را از سمت راست به چپ چاپ کند.
- برنامه‌ای بنویسید که عدد n را دریافت کرده، آن را به مبنای ۲ تبدیل کند.
- برنامه‌ای بنویسید که یک عدد دودویی را دریافت کرده، به مبنای ۱۰ تبدیل کند.
- برنامه‌ای بنویسید که دو عدد را دریافت کرده، بزرگترین مقسوم علیه مشترک آنها را محاسبه کند.
- برنامه‌ای بنویسید که دو عدد را دریافت کرده، کوچک ترین مضرب مشترک آنها را محاسبه کند.
- برنامه‌ای بنویسید که یک عدد را دریافت کرده، بررسی کند آیا عدد دریافت شده اول است یا خیر.
- برنامه‌ای بنویسید که مجموع سری $... + ۱/۸ + ۱/۴ + ۱/۲ + ۱$ را محاسبه کند.