طراحی زبان‌های برنامه‌سازی

تمرین اول:

۱-

سیستم های توکار با قابلیت ثابت یا قابل برنامه ریزی، برای یک کارکرد خاص یا عملکردهای درون یک سیستم بزرگتر طراحی شده اند که مصرف توان کمتر، اندازه کوچک تر، بازه عملیاتی انعطاف پذیر و هزینه کمتر به ازای هر واحد دارند. برای چنین زبان‌هایی استفاده همزمان کاربران و استفاده از فایل مطرح نیست بلکه مدیریت خطا بسیار مهم است و باید به صورت بلادرنگ (real time) پاسخ دهد. همچنین معمولا به صورت توزیع شده است و نیاز به برنامه نویسی همروند می‌باشد.

توسعه دهندگان از انواع زبان‌های برنامه‌نویسی در سیستم‌های توکار استفاده می‌کنند. پرکاربردترین زبان‌ها شامل C, C++, MicroPython, Python, Java است که حدود ۸۰درصد از سیستم‌های توکار از زبان برنامه‌نویسی C استفاده می‌کنند. همچنین JavaScript نیز در برخی از سیستم‌های توکار استفاده می‌شود. توسعه دهندگان از زبان‌های دیگری براساس نیازخاص استفاده می‌کنند. این زبان‌ها در همه سیستم‌های توکار رایج نیستند اما در موارد خاص محبوب هستند مثل Ada, Assembly, Go

۲- زبان‌هایی مثل TCL, Rexx and BLISS از نوع Type Less هستند ولی زبان هایی مثل Java, C++ بدون نوع Type Less هستند. در زبان‌های Type Less اساسا تنها یک نوع داده وجود دارد به طوری که یک متغیر می‌تواند هر نوع داده‌ای را ذخیره کند و سپس با اجرای برنامه نوع داده‌های ذخیره شده در یک متغیر تغییر می کند. در واقع برای همه متغیرها به یک اندازه حافظه می‌گیرد و بعد موقع انجام عملیات به صورت اتوماتیک برای مثال اگر لازم هست به int تبدیل می‌کند و محاسبات انجام می‌شود و سپس به حالت دیفالت برمی‌گردد.

درواقع چنین زبان‌هایی شامل نکات زیر است:

- متغیرها در زبان‌های Type Less می‌توانند هرنوع داده‌ای را ذخیره کنند.
- در زبان‌های Type Less از انواع داده‌های سنتی برای دسته‌بندی داده‌ها استفاده نمی شود.
- زبان‌های Type Less ایمنی نوع را در اولویت قرار نمی‌دهند.

۳- در زبان‌های late binding می‌توان توابعی با نام یکسان و تعداد متغیر متفاوت داشت زیرا در زمان اجرا (run time)، تابع موردنظر باتوجه به آرگومان‌های ورودی تابع مشخص می‌شود. (در واقع کامپایلر نوع شیء را در زمان اجرا شناسایی و سپس تابع موردنظر فراخوانی می‌شود.) که در ادامه با یک مثال توضیح داده می‌شود:

```
class Foo{
    public $name = 'foo';

    public function getName(){
        echo "this is {$this->name}";
    }
}
class Bar extends Foo{
    public $name = 'bar';
}
```

اگر از کلاس Bar که از کلاس Foo در کد بالا یک نمونه بسازیم و متد getName را صدا بزنیم باید خروجی به صورت this is bar باشد.

```
$obj = new Bar();
$obj -> getName(); // output: this is bar
```

وقتی یک متد non-static صدا زده می‌شود و یا از فیلدهای non-static استفاده شود. مثلا در اینجا کامپایلر در ابتدا سینتکس را چک می‌کند و تشخیص می‌دهد که آبجکت Bar متد getName را صدا زده. بنابراین فیلد name را از کلاس Bar دریافت می‌کند چون ابجکتی که getName رو صدا زده Bar هست که به آن early binding گفته می‌شود. یعنی کامپایلر در زمان کامپایل ابجکت رو چک می‌کند و فیلدهای ابجکت رو به متدی که صدا می‌زند bind می‌کند و اگر ابجکت فیلدهای موردنظر را در کلاس خودش نداشت از کلاس پدر می‌گیرد.

حال اگر متدها یا فیلدها static باشند فرق می‌کند:

```php
class Foo{
    public static $name = 'foo';

    public function getName(){
        echo 'this is '.self::$name;
    }
}
class Bar extends Foo{
    public static $name = 'bar';
}

$obj = new Bar();
$obj -> getName();
```

اگر از کلاس Bar مثل دفعه قبل نمونه بگیریم و متد getName را صدا بزنیم خروجی this is foo خواهد بود زیرا پروپرتی های استاتیک زمان run-time کنترل می‌شوند درواقع زمانی که کامپایل شده و برنامه در حال اجرا است. بنابراین کامپایلر نمی‌تواند در همان ابتدا فیلد کلاس Bar را به آبجکت Bar که متد getName را صدا می‌زند بدهد و در زمان اجرا پراپرتی‌های کلاسی که متد را صدا می‌زند bind می‌شود. اگر بخواهیم به فیلد name در Bar دسترسی داشته باشیم باید متد getNmae درکلاس Bar تعریف شود یا از static هنگام صدا زدن فیلد در متد استفاده شود. (مثل کد زیر)

```php
class Foo{
    public static $name = 'foo';

    public function getName(){
        echo 'this is '.static::$name;
    }
}
class Bar extends Foo{
    public static $name = 'bar';
}

$obj = new Bar();
$obj -> getName();
```

۴- یک فرآیند زمان کامپایل است که تایپ‌های متغیرهای مشخص نشده را بازسازی می‌کند. یک مشکل در استنتاج نوع، تشخیص گیج کننده و گاه غیر شهودی است که توسط جستجوگر نوع در نتیجه خطاهای نوع ایجاد می شود. اصلاحی از الگوریتم یکسان سازی مورد استفاده در استنتاج نوع هیندلی-میلنر ارائه شده است، که اجازه می دهد تا استدلال خاصی که منجر به یک متغیر برنامه دارای یک نوع خاص است برای توضیح نوع ثبت شود.

```
fun f(x, y):
if x:
   y++
else:
   y--
```

در این تابع استنباط می‌شود که x از نوع Boolean و y از نوع Number است.

Haskell و ML دو زبانی هستند که در آنها این جنبه از کامپایل از محبوبیت خاصی برخوردار شده است. همچنین استنتاج نوع در زبان های نمونه سازی و اسکریپت نویسی که به طور فزاینده ای مورد استقبال قرار می‌گیرند، کاربرد دارد.