پروژه‌ی درس طراحی کامپایلر

زبان Blend

دکتر جابری پور
نسخه ۱/۳۵

۱۰ بهمن ۱۳۹۵

۱ مقدمه

درس کامپایلر که یکی از دروس اصلی برنامه درسی رشته‌های علوم کامپیوتر می‌باشد، همواره شامل پروژه‌ای نیز بوده است. عمدتاً دو هدف از طرح پروژه برای این درس وجود دارد؛ آشنایی با طراحی یک زبان برنامه‌سازی و پیاده‌سازی کامپایلر آن و همچنین پیاده‌سازی یک سیستم نرم‌افزاری قابل توجه.

متاسفانه پیاده‌سازی یک زبان برنامه‌سازی واقعی که عموماً مورد استفاده واقع می‌شود بسیار دشوار است. بنابراین لازم است زبانی ساده با قابلیت‌های کافی برا کسب تجارب مورد اشاره طراحی شود.

این پروژه از شما پیاده‌سازی گام به گام یک کامپایلر را می‌خواهد، به این صورت که نیازهای هر بخش را تعریف کرده و از شما می‌خواهد به صورت مجزا آنها را پیاده کنید. شما می‌توانید از زبان‌های C++ یا Java برای پیاده‌سازی کامپایلر خود بهره ببرید. همچنین، با توجه به آنچه در کلاس آموزش داده می‌شود، نسخه جدید نرم‌افزار PGen در اختیار شما قرار می‌گیرد. این نرم‌افزار به شما کمک می‌کند تا به سادگی ساختاریاب خود را پیاده‌سازی کنید. سعی شده نرم‌افزار طوری شود که با مطالب مطرح شده سر کلاس بیشترین تطبیق را داشته باشد. یکی از معدود محدودیت‌های این پروژه لزوم پیاده‌سازی پروژه با استفاده از این ابزار است. یکی دیگر از این موارد، استفاده از LLVM و ابزارهای مشابه است. به این ترتیب، مثلاً پیاده‌سازی اسکنر به کمک Flex نتنها ایرادی ندارد بلکه شدیداً توصیه می‌شود!

۲ ابزارها

دو نسخه (نسخه ۲ و ۳) از نرم‌افزار PGen هم اکنون در دسترس است که هر دو از طریق صفحه درس در اختیار شما قرار می‌گیرد.

نسخه ۲ حاوی باگ‌های متعددی در ساخت جدول پارس است ولی چندسالی در دانشگاه شریف از آن استفاده شده است. نسخه ۳ سعی کرده مشکلات نسخه ۲ را حل کرده و نرم‌افزار پایدار و قابل اطمینانی را فراهم آورد؛اما از همان الگوریتم‌های ساختاریابی نسخه ۲ استفاده می‌کند. الگوریتم‌های ساختاریابی به زبان‌های C++ و Java توسعه یافته‌اند. سعی کردیم حتی‌الامکان نسخه ۳ تا حدی تست شود؛ تاکنون اشکالی در آن مشاهده نشده ولی با توجه به اینکه توسعه آن اخیراً پایان یافته، ممکن است حاوی تعدادی باگ باشد. لازم به ذکر است که برای اجرای نسخه ۳، نیاز به داشتن JDK 1.8.44 به بالا را دارید. راهنمای استفاده از نسخه ۳ داخل نرم‌افزار و نسخه ۲ به صورت فایل pdf است؛ برخی نکات تکنیکی این راهنما در نسخه ۳ هم قابل استفاده است. توجه کنید که نسخه ۳ یک بهینه‌سازی از نسخه ۲ است و اصول اولیه نسخه ۲ را حفظ کرده است؛ بنابراین در صورتی که قصد دارید از نسخه ۳ استفاده کنید هم راهنمای کوتاه نسخه ۲ را مطالعه کنید. ضمناً نسخه ۲ حاوی چند مثال آموزشی نیز هست.

کد نسخه ۳، برروی GitHub قرار گرفته است. در صورت لزوم می‌توانید آن را برای خودتان اصلاح کنید و نظرتان را به توسعه دهنده آن منتقل کنید.

۳ بررسی اجمالی

زبان Blend یک زبان Impratrive است. (همانند Java یا C++) این زبان چندان شئ‌گرا نیست. فقط به مانند C۸۹ از انواعی که کاربر تعریف کرده پشتیبانی می‌کند.

سعی شده این زبان، نه خیلی بزرگ باشد و نه خیلی کوچک. تا هم پیاده‌سازی آن فواید فوق را داشته باشد و هم به اتمام رساندن آن مثل یک کابوس نباشد!

Blend شامل تعدادی تابع، ساختار و محیط می‌باشد. این زبان یک تابع main دارد که اجرای برنامه از آن نقطه شروع می‌شود.

این زبان از محیطی پشتیبانی می‌کند که امکان دسته‌بندی متغیرها را به برنامه‌نویس می‌دهد. همچنین این زبان Type Safe است.

۴ ویژگی‌های زبان

در این قسمت اجزا و ویژگی‌های زبان را به صورت کلی مرور می‌کنیم. مفاهیم این بخش در بخش ۶ به صورت دقیق تعریف می‌شوند.

همانطور که در بخش قبل اشاره شد، این زبان شامل تعدادی تابع، محیط و ساختار است. در اینجا این موارد را به صورت غیر رسمی توضیح می‌دهیم.

۱.۴ ساختار

این ویژگی همانند زبان C است. یک ساختار متشکل از تعدادی متغیر پایه یا تعریف شده توسط کاربر است. هر ساختار به شکل

```
structure <structure_name>
{
   type1 attrb1;
   type2 attrb2;
   ...
}
```

تعریف می‌شود.

دسترسی به عناصر ساختار نیز مانند C به کمک عملگر dot ممکن است. تنها تفاوتی که این ساختارها با زبان C دارند، آن است که هر نوع غیر پایه(User-Defined) که در هر جای برنامه تعریف یا اعلان می‌شود(اعم از اعلان در یک ساختار دیگر یا تعریف داخل یک تابع یا محیط)، به صورت ضمنی در حکم reference می‌باشد. این ارجاع به طور پیش‌فرض به NULL اشاره کرده و در صورت دسترسی به آن برنامه باید با یک خطای زمان اجرا خارج شود.

برای تغییر این مقدار، باید یک حافظه به آن تخصیص داده شود، شمای کلی این عمل به صورت زیر است.

```
s = assign <a1, a2,,a4,...>;
```

مقادیر در صورت وجود داخل حافظه مربوطه به ترتیب تعریف قرار می‌گیرند. همچنین اگر یکی اعضا، خالی در نظر گرفته شود، در صورتی که از انواع اولیه باشد، مقدار حافظه آن دست‌نخورده باقی می‌ماند و غیر این صورت همان NULL مقدار ارجاع آن خواهد بود.

حافظه تخصیص داده شده به یک ساختار باید به کمک کلید واژه release پس داده شود:

```
release s;
```

در غیر اینصورت نشت حافظه رخ داده است.

۲.۴ تابع

توابع این زبان تعدادی ورودی و همچنین تعدادی خروجی دارند. نوع تعریف توابع به زبان C مشابهت دارد.

```
function (type1, type2, ...) <function_name>([specifier] type1 arg1,
    [specifier] type2 arg2, ...)
{
  // function body...
  return (a, b, c,...);
}
```

همچنین اگر تابعی خروجی نداشته باشند داخل پرانتز فقط `void` قرار می‌گیرد. ارسال تمامی آرگومان‌ها به هر تابعی، به طور پیش‌فرض به شکل Call By Value [۱] است. مگر آنکه تعیین کننده‌ها چیز دیگری را معین کنند. در این زبان دو نوع تعیین‌کننده داریم:

- out: این تعیین‌کننده، بیان می‌کند، در صورت تغییر مقدار متغیر مذکور در بدنه تابع، خارج از تابع نیز مقدار آن تغییر خواهد کرد.
- late: ارزیابی این آرگومان تا زمانی که در بدنه تابع به آن نیاز نشود، به تاخیر می‌افتد. یعنی اگر هنگام فراخوانی تابع، بجای متغیر a که قبل از آن در اعلان تابع عبارت مذکور آمده است، $f(x)$ قرار دهیم، تا زمانی که در بدنه تابع، متغیر a را نیاز نداشته باشیم، عبارت $f(x)$ ارزیابی نمی‌شود.توجه کنید که ممکن است یک عبارت هرگز مورد ارزیابی قرار نگیرد.

خروجی تابع نیز پس از عبارت `return` داخل پرانتز خواهد آمد. جهت استفاده از خروجی توابع یک نوع انتساب خاص طراحی شده است که به شکل زیر می‌باشد، این انتساب مختص توابع و چندتایی‌هاست. (برای توضیح بیشتر به ۳.۶.۴ مراجعه کنید.)

```
(a, b, c) := function(1, 2, 3);
// also OK:
(a, b, c) := (1, 2, 3);
```

۳.۴ محیط

همانطور که اشاره شد، در این زبان امکان تعریف محیط نیز وجود دارد. دلیل وجود این ویژگی، ایجاد امکان دسته‌بندی متغیرهاست. محیط در هرجای برنامه قابل تعریف است؛ ولی فقط در همان قسمت یا در قسمت‌های زیرین قابل دسترسی است؛ یعنی اگر یک محیط درون یک تابع تعریف شده، در تابع دیگر قابل استفاده نیست ولی اگر محیطی در قسمت سراسری تعریف شود، در همه توابع قابل استفاده است.

نحوه تعریف یک محیط به شکل:

```
environment userEnv
{
  type1 dcl1;
  type2 dcl2;
}
```

[۱] یعنی تغییرات در بدنه تابع روی متغیرها اثری ندارد.

همچنین جهت استفاده از محیط در هر نقطه از برنامه به شکل زیر عمل می‌شود:

```
::userEnv.dcl1
```

۴.۴ انواع

انواع این زبان بر دو دسته، پایه یا ابتدایی و ثانویه است. انواع ابتدایی شامل:

۱. bool مقادیر بولی، ۱بایت، شامل مقادیر ۰ و ۱.

۲. integer اعداد صحیح، ۴بایت، ذخیره‌سازی به شیوه مکمل ۲.

۳. long integer اعداد صحیح با طول ۸ بایت. به شیوه مکمل ۲.

۴. characters کاراکتر، ۱بایت، ذخیره‌سازی به شیوه اعداد صحیح.

۵. real اعداد حقیقی، ۴ بایت، ذخیره‌سازی به شیوه IEEE754 با دقت معمولی.

۶. string رشته، حاوی تعدادی کاراکتر، شیوه ذخیره‌سازی: به تشخیص کامپایلر نویس! البته تضمین می‌شود طول یک رشته بیش از ۱۰۰۰۰ کاراکتر نیست.

مابقی انواع به کمک ساختارها (به بخش ۱.۴ رجوع کنید) توسط کاربر تعریف می‌شوند.

کلیه انواع ابتدایی (بجز رشته) به یکدیگر قابل تبدل می‌باشند. در تبدیل انواع صحیح ،کاراکتر و اعشاری به بولی و بالعکس، عدد ذخیره شده به شکل ۰ یا ۱ خواهد بود. در تبدیل اعداد اعشاری به انواع صحیح و کارکتر نیز، عدد گرد شده و بخش کم ارزش، براساس میزان فضای نوع مقصد، در آن ذخیره می‌شود. در تبدیل عکس نیز عدد باید به نزدیکترین عدد اعشاری ممکن تبدیل شود.

۵.۴ توابع تعبیه شده

مانند بسیاری از زبان‌های برنامه‌سازی، این زبان شامل تعدادی تابع است. این توابع تعریف نمی‌شوند، بلکه از پیش در زبان موجود هستند.

- main شروع برنامه از این تابع است. این تابع با سایر توابع دیگری که در این لیست می‌آید متفاوت است. زیرا برنامه نویس خودش این تابع را تعریف می‌کند:

```
function (int) main()
{
// function body
}
```

- IO توابع ورودی–خروجی زبان. دو تابع از این نوع وجود دارد:

  - `(void)read(id)`

  این تابع، بر اساس نوع ورودی، آن نوع را از ورودی خوانده و داخل حافظه ارسال شده قرار می‌دهد. نوع ورودی جزء انواع پایه‌ای است. (به بخش ۴.۴ مراجعه کنید)

- `(void)write(expr)`
یک عبارت ارزیابی شده و مقدار آن در خروجی استاندارد نوشته می‌شود.

- توابع رشته

- `(int)strlen(string)`
این تابع طول یک رشته را در قالب یک عدد صحیح باز می‌گرداند.
- `(string)concat(string, string)`
این تابع دو رشته را الصاق کرده و حاصل را باز می‌گرداند.

۶.۴ دستورالعمل‌ها و عبارات

در این بخش، قسمت اصلی زبان یعنی عبارات آن را معرفی می‌کنیم.

۱.۶.۴ ثوابت

ساده‌ترین نوع عبارات هستند:

- ثوابت بولی مثل true و false
- ثوابت عددی، از جمله ثوابت صحیح(شامل اعداد ۴ بایتی و ۸ بایتی) و ثوابت حقیقی
- ثوابت کارارکتری مثل `'c'`
- ثوابت رشته‌ای مانند `"this is a string"`

۲.۶.۴ شناسه‌ها

نام متغیرهای محلی، نام پارامترهای فرمال تابع و نام عناصر ساختار جزء شناسه‌ها هستند. همچنین برای معرفی نام توابع و ساختارها هم از شناسه استفاده می‌شود، ولی توابع و ساختارها از این جهت جزء عبارات نیستند. این شناسه‌ها نباید با کلمات کلیدی زبان تداخل داشته باشند.

۳.۶.۴ انتساب

انتساب به فرم `id := expr` می‌باشد. در انتساب در صورت لزوم عمل تبدیل نوع برای انواع پایه، صورت می‌پذیرد. مقدار این عبارت در صورتی تبدیل نوع رخ ندهد ۱ بوده و درغیر اینصورت ۰ خواهد بود. همچنین همانطور که در بخش ۷.۴ آمده به کمک کلید واژه‌ای خاص می‌توان به ابعاد آرایه مقدار دهی کرد. همچنین اختصاص فضا به ساختارها (رجوع کنید به ۱.۴) یک نوع انتساب است.

همچنین نوع خاصی از انتساب نیز در این زبان تعبیه شده که برای توابع (بخش ۲.۴)و چندتایی ها مورد استفاده قرار می‌گیرد. (برای تعریف دقیق چندتایی به گرامر ۱.۶ مراجعه کنید)

بعلاوه توجه کنید انتساب در محدوده سراسری، نباید نیازمند انجام محاسبه باشد. وجود چنین انتسابی خطاست. به طور مثال مواردی مانند `a := 2` قابل قبود است ولی موردی مثل `a := 2 + 3` یا `a := b` خطاست. به طور کلی، برای انتساب در محوده سراسری نباید نیاز به تولید کد باشد.

۴.۶.۴ فراخوانی تابع

در این زبان توابع به شکل `id(expr, expr, ....)` فراخوانی می‌شوند. در این عبارت، آرگومان‌ها از چپ به راست مورد ارزیابی قرار می‌گیرند. همچنین مقدار این عبارت برابر خروجی تابع خواهد بود.

۵.۶.۴ عبارات شرطی

عبارات شرطی در زبان به صورت

```
if(expr)
  { expr }
else
  { expr }
```

است. معنا و تفسیر این عبارت، به شکل استاندارد است. یادآوری می‌شود که بخش `else` تنها در صورتی که مقدار عبارت برابر ۰ یا `false` باشد اجرا می‌شود. این عبارت مقدار بازگشتی ندارد.

۶.۶.۴ حلقه‌ها

در این زبان دو نوع حلقه وجود دارد:

۱. `while` این حلقه به شکل زیر نوشته می‌شود:

```
while(expr) { expr}
```

۲. `do-while` این حلقه به شکل زیر نوشته می‌شود:

```
do{expr} while(expr)
```

دستورات بدنه در این حلقه حداقل یکبار اجرا می‌شود.

لازم به ذکر است دستورات `break` و `continue` فقط داخل بدنه حلقه معتبرند. این عبارت مانند، عبارات شرطی نوع خروجی ندارد. همچنین قواعد مدار کوتاه در این عبارات برقرار نیستند، یعنی یک عبارت تماماً مورد ارزیابی قرار می‌گیرد.

۷.۶.۴ قطعه

یک قطعه فرم زیر را دارد:

```
{expr; expr;...}
```

یعنی یک قطعه حاوی تعدادی عبارت است، این عبارات به ترتیب مورد ارزیابی قرار می‌گیرند.

۸.۶.۴ تعریف متغیر

تعریف متغیر به شکل زیر است:

```
type a = expr
```

که در آن استفاده از یک عبارت جهت انتساب اختیاری است. در اینجا باید کنترل شود که مقدار عبارت قابل انتساب به نوع مربوطه باشد. همچنین توجه کنید که متغیرهای محلی و محیط‌های محلی فقط در آن بلاک و بلاک‌های زیری آن قابل استفاده هستند. همچنین به بخش ۳.۴ نگاه کنید.

Case ۹.۶.۴

این عبارت به شکل

```
case expr_0 of
  constant_val_1: {expr_1}
  ...
  constant_val_n: {expr_n}
endcase
```

می‌باشد. استفاده از Case فقط برای عبارات با مقادیر صحیح ممکن است و ساختار آن نیز باید به کمک جدول پرش پیاده‌سازی شود. برخلاف زبان C نیاز به کلید واژه break نیست.

assign و release ۱۰.۶.۴

واژه assign جهت گرفتن حافظه پویا مورد استفاده قرار می‌گیرد. هر حافظه‌ای که توسط این عبارت گرفته می‌شود باید توسط reslse پس داده شود. کلیه انواع تعریف شده توسط کاربر و آرایه‌ها به این طریق حافظه‌دهی می‌شوند. جزئیات را در ۱.۴ و ۷.۴ ببینید.

isvoid ۱۱.۶.۴

به شکل

```
isvoid(id)
```

استفاده می‌شود. ورودی آن یک ID بوده و خروجی آن `true` (در صورتی که برای شئ یا آرایه متناظر با شناسه حافظه تخصیص داده شده باشد) یا `false` (در صورتی که ورودی، نوع پایه باشد یا شئ یا آرایه‌ای باشد که به آن حافظه تخصیص داده نشده)

عملگر حسابی و مقایسه‌ای ۱۲.۶.۴

این زبان شامل تعدادی عملگر دوگانی به شرح زیر است:

- Add: $+$
- Subtract: $-$
- Multiply: $*$
- Divide: $/$
- Bitwise AND: $\&$

- Exclusive Add: ^
- Bitwise OR: |
- Logical AND: &&
- Logical OR: ||
- Mod: ٪

لازم به ذکر است که عملگرهای بیتی فقط بر روی انواع صحیح قابل تعریف‌اند.
همچنین عملگرهای یگانی این زبان شامل:

- Minus: −
- Logical Not: !
- Complement: ~

است.
همه عملگرهای غیر منطقی بالا (که مقدار بولین باز می‌گردانند) بسته به نوع عملوندشان (صحیح یا حقیقی) نوع بازگشتی‌شان تعیین می‌شود.
این زبان عملگرهای مقایسه‌ای زیر را دارد:

- Less than: <
- Less or Equal: ⩽
- Equal: =
- Not Equal: ! =
- Greater or Equal: ⩾
- Greater than: >

این عملگرها و عملگر یگانی نقیض، مقدار بولین باز می‌گردانند. مابقی عملگرها بسته به نوع مقدار آن نوع را باز می‌گردانند.

۱۳.۶.۴ دستورات goto و اعلان label

به کمک این دو دستور مانند آنچه در بخش ۱.۶ آمده می‌توان نقاطی را تعریف کرد که کنترل اجرا قابل انتقال به آنها باشد؛ یعنی ادامه اجرای برنامه از آن نقطه باشد (مانند آنچه در زبان C وجود دارد.) لازم به ذکر است که در تقدم و تأخر تعریف برچسب و دستور پرش محدودیتی وجود ندارد؛ تنها محدودیت موجود وجود دو دستور در یک تابع می‌باشد.

۷.۴ آرایه‌ها

در این زبان می‌توان آرایه‌های چند بعدی را تعریف کرد. این تعریف، با کمی تغییر بسیار شبیه به زبان C است.

```
array type arrID := [expr, expr, expr] assign
```

که در آن بخش انتساب اختیاری است. توجه کنید، که یک آرایه می‌تواند تغییر ساختار دهد. بنابراین قبل از تغییر ساختار باید حافظه متناظر با آن آزاد شود. این کار به صورت

```
release arrID
```

انجام می‌شود. دقت کنید که میزان آزادسازی حافظه توسط برنامه‌نویس اعلان نمی‌شود. توجه کنید که وظیفه شماست کد این کار را تولید کنید.

۵ ساختار واژگانی

واحدهای واژگانی این زبان متشکل از اعداد، صحیح و حقیقی، شناسه‌ها، واژه‌های کلیدی و رشته‌ها می‌شود. همچنین این زبان حساس به کوچک و بزرگ بودن حروف است.

۱.۵ ثوابت و شناسه‌ها

۱.۱.۵ اعداد صحیح

هر رشته کاراکتری از ارقام (۰ تا ۹) که ناتهی باشد را یک عدد صحیح می‌نامیم. این اعداد به صورت پیش‌فرض عدد صحیح ۴بایتی در نظر گرفته می‌شوند؛ اما در صورتی که طول بیشتری داشتند، صحیح ۸بایتی در نظر گرفته خواهند شد، در اغیر این صورت، تعریف چنین اعدادی خطا محسوب می شود. همچنین اعداد، در مبنای ۱۰ در نظر گرفته می‌شوند، مگر آنکه قبل از رشته اعداد صحیح 0x بیاید، که به معنای آن است که عدد در مبنای ۱۶ نوشته شده.

۲.۱.۵ اعداد حقیقی

هر رشته ناتهی از ارقام که شامل کاراکتر نقطه(.) باشد را یک عدد حقیقی در نظر می‌گیریم.

۳.۱.۵ کاراکاتر

یک کاراکتر! جهت استفاده به صورت ثابت، داخل Single Quotation قرار می‌گیرد.

۴.۱.۵ رشته

رشته‌ها داخل Double Quotation محصور می‌شوند. نحوه نوشتن کاراکترهای خاص ثوابت رشته‌ای مانند زبان C++ است.

۵.۱.۵ شناسه

شناسه‌ها با حروف الفبای انگلسی شروع می‌شوند و ترکیبی از عدد، شناسه و زیرخطUnderscore می‌باشند.

۲.۵ Comment

۱.۲.۵ تک خطی

هرگاه دو علامت خط تیره ((Dash(-) به شکل متوالی بیایند، خط جاری، از نقطه وقوع تا انتها Comment حساب می‌شود.

۲.۲.۵ چند خطی

هر آنچه بین --> و <-- قرار گیرد (اعم از خط و کاراکتر) به عنوان Comment محسوب می‌شود.

لطفاً به این مثال توجه کنید:

```
int a; --this is single line comment
string s;    <-- a multiple
  line comment -->
```

۳.۵ White Space

کاراکترهای زیر، فاصله سفید محسوب می‌شوند:

- Blank, ASCII 32
- \n, ASCII 10
- \f, ASCII 12
- \r, ASCII 13
- \t, ASCII 9
- \v, ASCII 11

۴.۵ کلمات کلیدی

کلمات کلیدی این زبان به شرح زیر است:

array، assign، bool، break، case، char، continue، do، else، endcase، environment، false، function، goto، if، int، isvoid، label، late، long، of، out، real، release، return، string، structure، true، void، while

۶ نحو زبان Blend

در این بخش در حد مناسبی آنچه تا کنون درباره زبان گفتیم را به شکل رسمی عنوان می‌کنیم.

۱.۶ گرامر زبان

شاید این قسمت مهم‌ترین بخش این مستند باشد؛ قواعد تولید زبان به شکل BNF آورده شده‌اند. البته به تمام قواعد BNF پایبند نبودیم. برای سادگی درک گرامر، به نوعی از عبارات منظم هم استفاده شده است. به طور مشخص A^* یعنی صفر یا بیشتر تکرار متوالی A، A^+ یعنی یک یا بیشتر تکرار متوالی A. همه علائمی که به داخل [...] قرار داشته باشند، اختیاری هستند. جهت گروه‌بندی و خوانایی بیشتر در گرامر از علامت ⟦...⟧ استفاده شده که برای مشخص نمودن نمادهای مرتبط است. همچنین علامت + به معنای جمع در عبارات منظم است. هرجا منظور جمع بوده به شکل '+' آمده است. در مورد bracket هم این موضوع صادق است.

⟨ PROGRAM ⟩	::=	⟦ ⟨ FUNC_DEF ⟩ + ⟨ STR_DEF ⟩ + ⟨ ENV_DEF ⟩ + ⟦ ⟨ VAR_DCL ⟩; ⟧ ⟧*
⟨ VAR_DCL ⟩	::=	⟨ SIMPLE_VAR ⟩
	|	⟨ ARRAY_VAR ⟩
⟨ SIMPLE_VAR ⟩	::=	⟨ TYPE ⟩ *id* [⟨ ASSIGNMENT ⟩]
⟨ ARRAY_VAR ⟩	::=	*array* ⟨ TYPE ⟩ *id* [⟨ ASSIGNMENT ⟩]
⟨ ASSIGNMENT ⟩	::=	:= ⟨ EXPR ⟩
	|	:= *assign* < ⟨ EXPR ⟩ [, ⟨ EXPR ⟩]* >
	|	:= '[' ⟨ EXPR ⟩ [, ⟨ EXPR ⟩]* ']' *assign*
⟨ ENV_DEF ⟩	::=	*environment id* { ⟦ ⟨ VAR_DCL ⟩; ⟧* }
⟨ STR_DEF ⟩	::=	*structure id* { ⟦ ⟨ VAR_DCL ⟩; ⟧* }
⟨ FUNC_DEF ⟩	::=	*function* (⟨ TYPE ⟩ ⟦ , ⟨ TYPE ⟩ ⟧*) *id* ([⟨ ARGUMENT ⟩]*) ⟨ BLOCK ⟩
⟨ TYPE ⟩	::=	See Section ۴.۴
⟨ ARGUMENT ⟩	::=	[⟨ SPECIFIER ⟩] ⟨ VAR_DCL ⟩
⟨ SPECIFIER ⟩	::=	*out*
	|	*late*
⟨ BLOCK ⟩	::=	{ ⟦ ⟨ ENV_DEF ⟩; + ⟨ STMT ⟩; ⟧* }
⟨ STMT ⟩	::=	⟨ EXPR ⟩
	|	(⟨ ID ⟩ ⟦ , ⟨ ID ⟩ ⟧+) ⟨ ASSIGNMENT ⟩
	|	⟨ ID ⟩ ⟨ ASSIGNMENT ⟩
	|	⟨ VAR_DCL ⟩
	|	⟨ LOOP ⟩
	|	*break*
	|	*continue*

	|	*return* [(*id* ⟦ , *id* ⟧*)]
	|	⟨ CONDTIONAL ⟩
	|	⟨ CASE-OF ⟩
	|	*isvoid* (⟨ ID ⟩)
	|	*goto id*
	|	*label id*
	|	*relese id*
⟨ EXPR ⟩	::=	(⟨ EXPR ⟩)
	|	⟨ EXPR ⟩ ⟨ BINARY_OP ⟩ ⟨ EXPR ⟩
	|	⟨ UNARY_OP ⟩ ⟨ EXPR ⟩
	|	⟨ ID ⟩
	|	⟨ CONSTANT ⟩
	|	⟨ FUNCTION_CALL ⟩
	|	(⟨ EXPR ⟩ ⟦ , ⟨ EXPR ⟩ ⟧+)
⟨ FUNCTION_CALL ⟩	::=	*id*([⟨ EXPR ⟩ ⟦ , ⟨ EXPR ⟩ ⟧*])
⟨ ID ⟩	::=	[::*id.*] *id* [⟦ *.id* ⟧* + ⟦ '[' ⟨ EXPR ⟩ ⟦ , ⟨ EXPR ⟩ ⟧*']' ⟧]
⟨ UNARY_OP ⟩	::=	See Section ۱۲.۶.۴
⟨ BINARY_OP ⟩	::=	See Section ۱۲.۶.۴
⟨ CONSTANT ⟩	::=	See Section ۱.۵
⟨ LOOP ⟩	::=	*while* (⟨ EXPR ⟩) ⟨ BLOCK ⟩
	|	*do* ⟨ BLOCK ⟩ *while* (⟨ EXPR ⟩)
⟨ CONDITIONAL ⟩	::=	*if* (⟨ EXPR ⟩) ⟨ BLOCK ⟩ [*else* ⟨ BLOCK ⟩]
⟨ CASE-OF ⟩	::=	*case* ⟨ EXPR ⟩ *of* ⟦ ⟨ CONSTANT ⟩ : ⟨ BLOCK ⟩ ⟧+ *endcase*

۲.۶ تقدم عملگرها

تقدم علمگرهای بخش ۱۲.۶.۴ مشابه زبان C می‌باشد. برای اطلاعات بیشتر می‌توانید به اینجا مراجعه کنید.

۷ محیط زمان اجرا

برنامه باید تحت یک ماشین مجازی که در اختیار شما قرار داده می‌شود اجرا شود. دستورات این ماشین بسیار شبیه به آنچه در درس در بخش کدسازی خواندید می‌باشد. توجه کنید که این ماشین قابلیت‌های محدودی دارد، در صورتی که لازم است کاری انجام شود که ماشین قادر به انجام آن نیست، باید به کمک دستورات ماشین آن کار را انجام دهید.

۸ پیاده‌سازی، تحویل پروژه و قوانین آن

در این بخش به قوانین پیاده‌سازی و تحویل پروژه می‌پردازیم. لطفاً **این بخش را دقیق بخوانید!** دقیق نخواندن و دقیق اجرا نکردن این بخش **خیلی خطرناک** است!

۱.۸ ابزارها، پیاده‌سازی

به صورت خلاصه در بخش مقدمه اشاره شد که باید از گراف نحو برای پیاده‌سازی ساختاریاب خود استفاده کنید. همچنین در انتخاب ابزار پیاده‌سازی واژه‌یاب آزاد هستید. فقط LLVM و ابزارهای مشابه آن در این پروژه غیر مجاز هستند. (با توجه به اینکه ساختاریاب باید به کمک گراف نحو پیاده‌سازی شود، ابزارهایی مثل Bison نیز بدون کاربرد هستند.) دو ابزار برای ساختاریابی به کمک گراف نحو در اختیار شما قرار داده می‌شود که می‌توانید از آن‌ها استفاده کنید. همچنین می‌توانید خودتان نیز آن‌ها را پیاده‌سازی کنید ولی باید ورودی و خروجی آن مشخص باشد؛ یعنی گراف ورودی مشخص بوده و جدول پارس خروجی نیز مشخص باشد. همچنین کد آن بخش نیز جزء پروژه شما محسوب می‌شود. تاکید می‌شود که باید از روش پارس مبتنی بر گراف نحو استفاده کنید.

لازم به ذکر است که استفاده از چهارچوب‌هایی مثل QtCreator کاملاً آزاد و مجاز است!

۲.۸ برنامه تولیدی و خروجی‌ها

این بخش را دقیق‌تر بخوانید! در صورتی که موارد زیر رعایت نشوند پروژه شما تصحیح نمی‌شود!

شما باید یک برنامه کامل تحویل دهید. برنامه شما باید از طریق خط فرمان فراخوانی شده و به عنوان آرگومان ورودی فایل ورودی را دریافت کرده و دو فایل خروجی به همان نام تولید کند. فایل خروجی اول، کد قابل اجرای با زبان ماشین مجازی داده شده و فایل دوم، فقط عنوان می‌کند که آیا برنامه گرفته شده معتبر بوده یا نه. نحوه استفاده به شکل زیر است:

```
bcompiler sampleProgram.blend
```

فایل‌های تولید شده به ترتیب باید به نام‌های `sampleProgram.out` برای اجرای برروی ماشین مجازی و `sampleProgram.res` که فقط حاوی ۰ یا ۱ می‌باشد. ۱ برای مواقعی که کد دریافتی صحیح بوده و ۰ برای زمانی که کد دریافتی به هر دلیل غلط بوده است. توجه کنید که برنامه شما باید بدون خطا تمام شود!

۳.۸ توزیع نمره و ارزیابی پروژه

باید بتوانید هر کد درستی را ساختاریابی کنید. پس باید پارسر را برای کل گرامر پیاده‌سازی کنید. اما نیازی نیست برای همه بخش‌های زبان کد تولید شود. توابع و ساختارها اختیاری هستند. (البته باید تابع main و توابع تعبیه شده بخش ۵.۴ را داشته باشید! منظور تعریف و فراخوانی و... است) .

بخش ساختاریابی و تشخیص کدهای صحیح و غلط ۳۰ نمره و بخش تولید کد ۸۰ نمره دارد. پروژه‌ها به صورت اتوماتیک تصحیح می‌شوند. در بخش ساختاریابی و تشخیص کدهای صحیح و غلط، فقط به خروجی فایل `res`. توجه می‌شود. همچنین بخش تولید کد و سایر بخش‌های امتیازی بر اساس خروجی کد تولیدی مورد ارزیابی واقع می‌شوند. ارزیابی به این صورت است که موارد ریز شده و برای هر کدام یک مورد آزمون تا حد امکان مستقل، استخراج شده و کد شما با آن ارزیابی می‌شود. جزئیات نمرات هر

بخش انشاءالله متعاقباً اعلام می‌شود.

۴.۸ قوانین و تحویل پروژه

۱.۴.۸ موعد تحویل

مهلت ارسال پروژه‌ها ۱۶ بهمن ساعت ساعت ۲۳:۵۵ خواهد بود. زمان ارزیابی و تحویل حضوری ۱۹ و ۲۰ بهمن است.

۲.۴.۸ گروه‌بندی

پروژه را می‌توانید در قالب گروه‌های ۲ یا ۳ نفری انجام دهید. انجام انفرادی پروژه نیز ممکن است. البته به نمره نفراتی که به صورت انفرادی پروژه را انجام می‌دهند، ۳۰ درصد افزوده می‌شود. اعضای پروژه همگی باید در جریان پروژه باشند و به بخشی که خود مسئولیت انجام آن را داشته‌اند تسلط کامل داشته باشند و بداند در پیاده‌سازی سایر بخش‌ها از چه روش‌های و تکنینک‌هایی استفاده شده است. هر فرد باید به میزان مناسبی از پروژه را انجام داده باشد.

در صورتی که کشف شود که فردی بند بالا را رعایت نکرده، ممکن است، نمره‌اش در ضریبی که بیانگر میزان تسلطش است، ضرب شود.

هر گروه مجاز است که فقط از کدهایی که اعضایش تولید کرده‌اند، استفاده کند. در پایان کدهای شما توسط نرم‌افزارهای تقلب‌یاب بررسی خواهد شد. در صورتی که تقلبی کشف شود، نمره گروه صفر در نظر گرفته می‌شود.

همچنین گروه‌ها باید حداکثر تا ۱۵ آذر به این آدرس، دقیقاً به فرمت زیر و توسط یکی از اعضای گروه ارسال شوند. عدم اعلام گروه تا این تاریخ، به معنای عدم تمایل به انجام پروژه است:

CompilerDesign95a-GroupReg

۳.۴.۸ تحویل دادنی‌ها

شما باید کد و فایل اجرایی را با شرایط گفته شده در همین بخش، تحویل دهید. علاوه بر کد باید یک سند برای کدتان تهیه کنید؛ در این سند، باید شرح کارهای انجام شده توسط اعضای گروه آروده شود. همچنین در این سند باید مشخصات مورد نیاز برای اجرای برنامه‌تان را بیاورید و بگویید چه بخش‌هایی را انجام داده‌اید. در صورتی که سندتان بیش از حد نامناسب باشد، ممکن است از نمره پروژه شما کم شود.

کلیه موارد بند بالا را در یک فایل با پسوند zip و بانام شماره گروه به این آدرس و دقیقاً با فرمت زیر ارسال کنید:

CompilerDesign95a-GroupNo-Delivery

همچنین باید مستند و فایل اجرایی کامپایلرتان با نامی که در بخش ۲.۸ آمد، در root فایل ارسالی قرار گیرد. (در غیر اینصورت پروژه‌تان با **خطر صفر گرفتن جدی** مواجه می‌شود)

۹ نمرات قسمت‌های مختلف و تعیین بخش‌های امتیازی

در جدول زیر لیست قسمت‌های امتیازی و اجباری آمده است. نمره بخش امتیازی تقریباً به اندازه نصف بخش اجباری است.

مورد تولید کد	ضریب نسبی	امتیازی
خواندن از ورودی	۱	
نوشتن در خروجی	۱	
پیاده‌سازی نوع ساختار(ساخت، نمونه‌گیری و انتساب)	۷	*
پیاده‌سازی نوع محیط	۳	*
انتساب صحیح داده‌های هم‌نوع (به جز انتساب رشته)	۲	
انتساب صحیح داده‌های غیر هم‌نوع (تبدیل نوع غیر صریح)	۳	
محاسبات جمع و تفریق اعداد صحیح	۲	
محاسبات ضرب و تقسیم اعداد صحیح	۲	
محاسبات کلی با رعایت تقدم عملگرها (بدون نوع ممیز شناور)	۲	
محاسبات از نوع بیتی (عملگرهای بیتی)	۲	
پیاده‌سازی نوع long	۲	*
پیاده‌سازی نوع رشته	۳	
پیاده‌سازی انواع کاراکتری و بولی	۱	
پیاده‌سازی نوع اعداد حقیقی	۱	
پیاده‌سازی انواع آرایه‌ای	۶	
پیاده‌سازی انتساب رشته به آرایه کاراکتری	۲	
پیاده‌سازی توابع رشته‌ای تعبیه شده strlen و concat	۳	
محاسبات ممیز شناور	۱	
محاسبات ترکیبی انوع صحیح، کاراکتری و حقیقی	۴	
پیاده‌سازی ساختار شرطی if-else	۴	
پیاده‌سازی ساختار case	۴	
پیاده‌سازی متغیرهای سراسری	۱	*
پیاده‌سازی متغیرهای محلی	۱	*
پیاده‌سازی قطعه و تعیین درست محدوده عمر متغیرهای داخل آن	۳	
پیاده‌سازی حلقه do-while	۳	
پیاده‌سازی حلقه while	۳	
پیاده‌سازی تابع main	۱	
پیاده‌سازی توابع (فراخوانی، اجرا و توابع بازگشتی)	۱۰	*
پیاده‌سازی تعیین‌کننده‌های out و late برای آرگومان توابع	۷	*
پیاده‌سازی مقدار بازگشتی توابع (یک مقدار بازگشتی)	۳	*

پیاده‌سازی مقدار بازگشتی توابع (چند مقدار بازگشتی)	۶	*
عملکرد صحیح جدول نمادها	۳	
پیاده‌سازی دستور break	۲	
پیاده‌سازی دستور continue	۲	
تخصیص مناسب حافظه پویا	۴	
آزادسازی مناسب حافظه پویا	۴	
پیاده‌سازی تابع isvoid	۳	
پیاده‌سازی دستور goto و مدیریت صحیح برچسب‌ها	۶	