به نام خدا

پروژه‌ی درس کامپایلر

دانشگاه صنعتی اصفهان

استاد درس:
خانم دکتر مریم موزرانی

پاییز ۹۹

تعریف پروژه:

در این پروژه قصد داریم یک کامپایلر برای یک زبان ساده مد نظر خود طراحی کنیم. برای پیاده سازی این کامپایلر می توانید از ابزار های بایسون و فلکس استفاده نمایید. این کامپایلر باید بتواند یک فایل ورودی حاوی کد نوشته شده به زبان مبدا را دریافت کرده و با در نظر گرفتن semantic action و دیگر مفاهیم لازم که در درس کامپایلر خوانده اید، یک کد خروجی به زبان اسمبلی MIPS تولید کند.

کلمات کلیدی:

کلمات کلیدی کلماتی هستند که در یک زبان برنامه نویسی نمی توان از آن‌ها به عنوان مفهوم دیگری مانند نام متغیرها استفاده کرد. کلمات کلیدی مورد استفاده در این پروژه برای زبان مورد نظر ما، شامل موارد زیر می باشد:

void	int	for	while
if	else	break	continue
return	main	char	elseif

متغیر ها:

در زبان ورودی شناسه متغیر ها ترکیبی از حروف، اعداد انگلیسی و '_' هستند که حتما می بایست با یک حرف و یا '_' شروع شوند و هیچ شناسه ای با اعداد آغاز نمی شود.
زبان ورودی حساس به بزرگ و کوچک بودن حروف نیز می باشد.

کامنت ها:

کامنت ها در این زبان به صورت زیر هستند:

۱. کامنت های تک خطی که با علامت # شروع می شوند.
۲. کامنت های چند خطی که با علامت *# شروع می شوند و با علامت #* پایان می یابند.

توکن ها:

توکن ها از طریق فاصله و یا از طریق توکن های خاص از هم جدا شوند. منظور از فاصله هر نوع whitespace مانند tab و space و ... می‌باشد.
دقت شود که هر تعداد فاصله ای که بین دو توکن وارد شود بی تاثیر است و باید نادیده گرفته شود.

مقادیر ثابت:

در زبان ورودی با متغیر های int و char سر و کار داریم پس مقادیر ثابتی که داریم شامل اعداد صحیح و یک کاراکتر خواهد بود.
دقت شود که اعداد صحیح علامت دار هستند که بزرگترین مقدار آن باید $2^{32} - 1$ (متناسب با یک سیستم ۳۲ بیتی) باشد.
مقادیر کاراکتر ها نیز می بایست در داخل ' ' قرار گیرد و این علامت ها عضو مقادیر محسوب می‌شوند و توکن جداگانه ای تشکیل نمی‌دهند.

عملگر ها و توکن های خاص:

عملگر هایی که در زبان ورودی مجاز هستند شامل عملگرهای محاسباتی، منطقی و شرطی می باشد که لیست آنها در زیر آورده شده است:

```
+    -    *    /    <    <=    ==    !=    >    >=    |    &    ||
&&   ^    !
```

توکن های خاص به توکن هایی گفته می شود که نه متغیر هستند و نه کلمه کلیدی و نه عملگر که لیست آن ها در زیر آمده است:

```
(    )    {    }    $    ,    [    ]    ..
```

قوانین نحوی زبان:

زبان ورودی دارای قواعد نحوی زیر است که هر برنامه ای که به این زبان نوشته شده باشد می بایست از قواعد تبعیت کند.

۱. برنامه حتما باید دارای یک تابع main باشد که برنامه از آن شروع می شود.

۲. شرط if ممکن است در داخل کد ها وجود داشته باشد که ساختار آن به شکل زیر خواهد بود:

```
If (condition) {
        body
}
```

۳. ممکن است شرط if با else همراه باشد.

```
If (condition) {
        body
} else {
        body
}
```

۴. ممکن است شرط else با if همراه باشد.

```
If (condition) {
        body
} elseif (condition) {
        body
} else {
        body
}
```

۶. حلقه while می تواند در داخل کد موجود باشد که ساختار آن مطابق زیر خواهد بود :

```
while(condition){

}
```

۷. کد های نوشته شده می توانند حاوی تعریف توابع و فراخوانی آن ها باشند.
ضمن پیاده سازی این بخش چندین مورد را باید در نظر داشته باشید:

۱. توابع موجود در زبان تنها خروجی int یا void خواهند داشت.
۲. توابع می توانند آرگومان ورودی داشته باشند با بدون آرگومان ورودی فراخوانی شوند اما تعداد آرگومان های ورودی یک تابع در صورت نیاز حداکثر ۴ مورد خواهد بود.
۳. توابع تعریف شده prototype نداشته و تماما پیش از تابع main تعریف خواهند شد.

۸. در زبان ورودی تمام دستورات به جز دستورات حلقه و شرط با کاراکتر $ پایان می پذیرد (این کاراکتر همانند کاراکتر ; در انتهای دستورات C عمل می کند)

۹. متغیر های درون برنامه به روش زیر می توانند تعریف شوند:

1. Int var = 10$
2. Int var$

نکات پروژه:

۱. تقسیم بر صفر را کنترل کنید. در صورت احتمال وجود تقسیم بر صفر، یک هشدار چاپ کنید.
۲. تمام برنامه در قالب یک فایل ورودی به کد شما داده میشود.
۳. در صورت وجود خطا، تنها همان خطا چاپ شود و عمل کامپایل بدون تولید هیچگونه کدی پایان پذیرد.

۴. در داخل برنامه شما می بایست بلوک متغیر ها را کنترل کنید به این معنی که اگر فرضا متغیری در داخل شرط if تعریف شود، در خارج از آن قابل دسترسی نمی باشد. این شرط برای تمامی بلوک های منطقی برنامه از جمله بدنه توابع ، حلقه ، و ... می بایست برقرار باشد.

نکته ی بسیار مهم: چنانچه نوشتن بخشی از کامپایلر در توانتان نبود، لطفا پروژه را رها نکنید و بقیه ی قسمتها را انجام دهید و در یک فایل به اسم notImplemented.txt توضیح دهید که چه بخشهایی را نتوانستید بنویسید تا کد شما متناسب با آنچه که نوشتهاید تصحیح شود(تست کیس های مناسب با چیزی که تحویل دادهاید به کامپایلر شما داده خواهد شد) و تمام نمره را از دست ندهید.

گرامر زبان ورودی:

برای نمونه یک گرامر ساده در زیر آماده است که میتوانید از آن برای پیاده سازی کامپایلر خود استفاده کنید، این گرامر برخی از امکانات گفته شده را ندارد اما می توانید به راحتی آن را گسترش دهید:

PROGRAM → STMT_DECLARE PGM
PGM → TYPE ID ‘(’ ‘)’ ‘{’ STMTS ‘}’ PGM | epsilon
STMTS → STMT STMTS | epsilon
STMT → STMT_DECLARE | STMT_ASSIGN | STMT_RETURN | ‘$’
EXP → EXP ‘<’ EXP
EXP → EXP ‘<=’ EXP
EXP → EXP ‘>’ EXP
EXP → EXP ‘>=’ EXP
EXP → EXP ‘==’ EXP
EXP → EXP ‘+’ EXP
EXP → EXP ‘-’ EXP
EXP → EXP ‘*’ EXP
EXP → EXP ‘&&’ EXP
EXP → EXP ‘||’ EXP
EXP → EXP ‘|’ EXP
EXP → EXP ‘&’ EXP
EXP → EXP ‘^’ EXP
EXP → EXP ‘!=’ EXP
EXP → ‘!’ EXP
EXP → ‘-’ EXP

EXP → ‘(’ EXP ‘)’
EXP → ID
EXP → NUM
STMT_DECLARE → TYPE ID IDS
IDS → ‘$’ | ‘,’ ID IDS
STMT_ASSIGN → ID ‘=’ EXP ‘$’
STMT_RETURN → RETURN EXP ‘$’
TYPE → INT | VOID

دقت شود گرامر ارائه شده در بالا کامل نیست و همچنین ممکن است نیاز به رفع ابهام نیز داشته باشد. بنابراین تولید یک گرامر غیر مبهم بر عهده ی شما است که می توانید از قابلیت های زبان بایسون(برای اولویت بندی) نیز استفاده کنید.

تولید کد MIPS:

در این پروژه برای هر عملیات باید semantic action متناسب با آن را انجام دهید به گونهای که منجر به تولید کد اسمبلی صحیح شود. زبان اسمبلیای که برای کد خروجی در نظر گرفتهایم، زبان MIPS است.

خطا یابی:

برنامهی شما باید بتواند هر گونه خطایی را تشخیص دهد.
منظور از هر گونه خطا، تمام خطاهایی است که یک کامپایلر C آنها را در نظر میگیرد: بنابراین کامپایلر شما دقیقا مشابه یک کامپایلر C رفتار خواهد کرد مگر در مواردی که قانونی برای آنها ذکر شده باشد.

نمرهي اضافه:

۱. پیاده سازی حلقه for با ساختار زیر اختیاری بوده و شامل نمره اضافه خواهد بود.

```
for (variable definition ; condition ; step){
body
}
```

۲. پیاده سازی متغیر های global پیش از تابع main به این صورت که در سراسر برنامه قابل دسترسی و استفاده باشند.
۳. پیاده سازی دستورات break , continue در داخل حلقه ها به این صورت control flow برنامه به ابتدای حلقه بازگشته یا از حلقه خارج گردد.

۴. پیاده سازی آرایه ها و قابلیت استفاده از اعضای آرایه در تمامی دستورات شامل نمره اضافه خواهد بود. ساختار تعریف آرایه مطابق زیر می باشد:

Int arr[array_num]$

مقدار array_num همواره یک مقدار ثابت می باشد.

موارد تحویلی:

۱. همان طور که گفته شد کامپایلر نوشته شده توسط شما، باید بتواند یک فایل ورودی حاوی کد نوشته شده به زبان مبدا را دریافت کرده و با در نظر گرفتن semantic action و دیگر مفاهیم لازم که در درس کامپایلر خوانده اید، یک کد خروجی به زبان اسمبلی MIPS تولید کند.

۲. Semantic action ها، به صورت کامل و مستند به عنوان خروجی تحویل داده شود.