بسم الله الرحمن الرحیم

پروژه درس
طراحی کامپایلرها

مدرس: محسن رحمانیان
دانشگاه جهرم
دانشکده مهندسی
گروه آموزشی مهندسی کامپیوتر
نیمسال اول ۱۴۰۰-۱۴۰۱

۱ـ مقدمه و کلیات پروژه

قبل از هرچیز تاکید می‌نمایم قبل از انجام پروژه این مستند را بصورت کامل مطالعه نمایید.
همانطور که پیشتر نیز اشاره شد این درس شامل پروژهای می‌باشد که به شما در یادگیری مفاهیم اصلی کمک کند. این ترم بر روی زبان Holoo کار می‌شود. این زبان از نوع Imperative (دستوری) می‌باشد. در ادامه توضیحات بیشتری در رابطه با زبان بیان خواهد شد.
همانطور که پیشتر نیز اعلام شده بود پروژه شامل سه فاز تحلیلگر لغوی (Lexical Analyzer)، تحلیلگر نحوی (Semantic Analyzer) و کد ساز (Code Generator) می‌باشد. همچنین پروژه به صورت تکی انجام می‌شود.
در واقع شما باید یک کامپایلر بنویسید که یک کد به زبان Holoo را دریافت کرده و آن را به کد قابل اجرا به زبان اسمبلی که بعد مشخص می‌شود تبدیل کند. نکته دیگر در مورد پیاده‌سازی تحلیلگر لغوی، صحت برنامه نویسی آن است؛ یعنی برنامه نباید بی‌دلیل یا بادلیل! به طور ناگهانی و یکطرفه قطع همکاری کند! اگر هم مجبور به این کار شد باید به نحو مناسبی این کار را بکند. یعنی باید کاملاً قابل استفاده باشد.
(این پاراگراف را با تاکید بخوانید) توجه کنید که کد تولیدی، **باید اجرا شود**؛ زیرا نمره‌دهی، بر طبق خروجی برنامه تولیدی شما خواهد بود. **خروجی** شما بخش اصلی نمره را تشکیل خواهد داد، بنابراین لازم است برنامه شما بتواند از روی خط فرمان فراخوانده شده، ورودی دریافت کند و جواب را چاپ کند. مجدداً تاکید می‌شود، نمره شما، پاسخ محور است؛ در صورتی که پاسخ شما صحیح نباشد، با احتمال فراوان تنها درصد بسیار کمی از نمره را دریافت خواهید کرد. همچنین باید نتیجه تحلیلگر نحوی پروژه معلوم باشد (قابل رویت). تقلب و باهم پروژه زدن، ممنوع می‌باشد، در صورتی که پروژه شما را کپی تشخیص دهم، برای آنها، **نمره ۱۰۰ـ خواهید گرفت**. همچنین یک تحویل حضوری نیز برای بررسی صحت ادعا و همچنین توانایی شما در ارائه پروژه و اطمینان ازاینکه پروژه را خودتان کد کرده و مطالب مورد نظر را فراگرفته‌اید، نیز در نظر گرفته شده است که زمان آن بلافاصله پس از تحویل عادی است. بعلاوه باید مستندی از فعالیتهای انجام شده در طول کد کردن پروژه ارائه شود.
آزاد هستید با هر زبانی پیاده‌سازی را انجام دهید. همچنین برای استفاده از هر ابزاری برای تحلیل لغوی و نحوی آزاد هستید. بعلاوه، تاکید می‌شود مستند را به صورت کامل بخوانید، هرچند که باید فقط بخشی از زبان را پیاده‌سازی کنید، ولی تعریف کامل همه بخشهای زبان آورده شده است.

۲ـ مشخصات زبان

در این پروژه بر روی یک زبان برگرفته از زبانهای C/C++ و Java که در زیر توضیحات آن می‌آید کار می‌کنید، لطفاً هر گونه ابهامی را به سرعت بپرسید.

۲-۱ـ ملاحظات واژه‌ای

این زبان دارای ثوابت عددی، حقیقی، رشته‌ای بوده و مانند سایر زبانهای دستوری از متغیرها نیز پشتیبانی می‌کند؛ همچنین این زبان شامل تعدادی کلمه کلیدی نیز هست. بعلاوه، تعریف کامنتها و... نیز در ادامه می‌آید. توجه کنید که زبان حساس به مورد می‌باشد.

۲-۲ـ کلمات کلیدی

کلمات زیر در زبان جزء کلمات کلیدی محسوب شده و نمی‌توانند به عنوان نام متغیرها و یا اسم توابع به کار روند:

```
bool, break, case, char, const, continue, default, double, else, false, function,
float, for, if, input, int, long, output, return, sizeof, string, switch, true
```

۳-۲- اعداد صحیح، اعداد حقیقی، بولین و تعریف متغیرها

اعداد صحیح، شامل رشته‌هایی از یک یا بیشتر کاراکترهای 0-9 می‌باشند. این اعداد شامل اعداد 32 بیتی در قالب مکمل 2 و یا 64 بیتی، بازهم در سیستم مکمل 2 می‌باشند. همچنین اعداد صحیح می‌توانند به صورت Hex نیز بکار روند. مثال:
$0x23 = 35$

اعداد حقیقی به چهار صورت قابل پیاده‌سازی هستند:

۱- رشته‌ای از ارقام به صورتی که سمت چپ کاراکتر "." شامل یک یا بیشتر رقم باشد. مثال: .1

۲- رشته‌ای از ارقام به صورتی که سمت راست کاراکتر "." شامل یک یا بیشتر رقم باشد. مثال: 1.

۳- رشته‌ای از ارقام به صورتی که سمت چپ و راست کاراکتر "." شامل یک یا بیشتر رقم باشد. مثال: 1.1

۴- به صورت نماد علمی؛ رشته‌ای شامل تعدادی (بیشتر از یکی) رقم (به صورت صحیح)، کاراکتر E یا e وسپس یک رقم با علامت مثبت یا منفی. مثال: $2e+3=200$ توجه کنید که علامت + اختیاری است؛ و هر رشته که علامتی نداشته باشد، به صورت پیشفرض علامتش مثبت در نظر گرفته می‌شود. توجه کنید که اعداد حقیقی یا در بازه تک‌دقتی یا دقت مضاعف استاندارد IEEE754 قرار می‌گیرند.

توجه کنید که سمت چپ هریک از ثوابت عددی میتواند علامت منفی باشد.

همچنین دو ثابت **Boolean** (از نوع bool)، true و false نیز در زبان گنجانده شده‌اند.

متغیرهای رشته (Stringها) می‌توانند شامل همه کاراکترهای ASCII بجزEOF باشند؛ حداکثر اندازه یک رشته ۱۰۰۰ کاراکتر است. البته باید این سایز به صورت پویا تخصیص داده شود (منظور حافظه پویا نیست، بلکه منظور تخصیص به اندازه مورد نیاز است، یعنی نباید به هر رشته اندازه ثابت ۱۰۰۰ تخصیص داده شود).

متغیرها شامل رشته‌هایی از کاراکترهای حروف الفبای زبان انگلیسی، اعداد و کاراکتر _ (Underscore) می‌باشد. همچنین هیچ متغیری با عدد شروع نمیشود.

۴-۲- ثوابت رشته‌ای، کامنت‌ها و دیگر ملاحظات

ثوابت رشته‌ای، با کاراکتر ["] آغاز شده، و به همین کاراکتر نیز ختم می‌شود. همه قواعد زبان C ++ در مورد ثوابت رشته‌ای (کاراکترهای درونی آنها) برقرار می‌باشد. این مورد در رابطه با ثوابت کاراکتری نیز برقرار است، به جز آنکه کاراکتر آغازی/پایانی آن [‘] می‌باشد. مثال از رشته :

“this is a\t valid String\”…\n”

کامنتهای تک خطی با علامت @@ شروع می‌شوند. همچنین کامنتهای چندخطی با /@ شروع شده و با @/ خاتمه مییابد. مثال:

```
@@ single line comment
/@ multi line
Comment @/
```

۵-۲- نمادها و عملوندها

Equal	==
Not Equal	!=
Less or Equal	<=
Less than	<
Bigger than	>
Bigger or equal	>=
Assignment	=
Not	!
Bitwise Negation	~
Arithmetic And	&
Logical and	&&
Arithmetic Or	|
Logical Or	||
Logical/Arithmetic Xor	^
Production	*
Add	+
Increment	++
Decrement	--
Sub and unary Minus	-
Div	/
Mod	%
Opening and Closing Curly Brace	{ }
Opening and Closing Parenthesis	( )
Dot	.
Comma	,
Colon	:
Semi-Colon	;
Opening and Closing Brace	[ ]

۶-۲- گرامر زبان

این گرامر به فرم BNF نوشته شده است. این فرم برای توصیف زبانهای برنامه‌نویسی بکار می‌رود. در حقیقت، زبان برنامه‌سازی به کمک یک زبان مستقل از متن توصیف می‌شود و در این فرم با قراردادهایی این زبان را توصیف می‌کنیم.

- تمام علائمی که به فرم <a> هستند، واژه های نحوی یا Variable یا همان non-terminal های گرامر زبان هستند.
- تمامی علائمی که **ضخیم** نوشته شده‌اند مثل ***for*** ، Terminalها یا پایانه‌های زبان هستند.

ـ اگر علامتی به صورت [x] ظاهر شد، به معنای آن است که حضور x در آن عبارت اختیاری است. اگر '[' و ']' بیاید یعنی پایانه.

ـ عبارات x^* یا x^+ به ترتیب به معنای صفر یا بیشتر و یک یا بیشتر رخداد x می‌باشند. (از قواعد عبارات منظم هم استفاده شده است)

ـ همچنین به کمک علامت | سمت راست قواعد تولید با سمت چپ یکسان از یکدیگر تفکیک شده‌اند.

ـ از نماد } و { صرفاً جهت گروهبندی عبارات استفاده شده است. مگر در بین علامت '' که به معنای پایانه است.

تاکید– هر جا که از نمادهای کمکی بالا در زبان استفاده شده، از Single Quotation استفاده شده است.

```
<program>          --→ <field_dcl>* <func_dcl>+
<field_dcl>        --→ <type> <field_dcl_cnt> [,<field_dcl_cnt>]*;
<field_dcl_cnt>    --→ {<id> | <id> '['<int_const>']'};
<func_dcl>         --→ {void | <tuple>} <id> ( [ <type> <id>{, <type> <id> }* ]) <block>
<tuple>            --→ <type> | '('<type>{, <type>}* ')'
<block>            --→ '{' <var_dcl>* <statement>* '}'
<var_dcl>          --→ [const] <type> <var_dcl_cnt> [,<var_dcl_cnt>]*;
<var_dcl_cnt>      --→ <id> [= <expr>]
<type>             --→ int | bool | float | long | char | double | string
<statement>        --→ <assignment>;
                   | <func_call>;
                   | <cond_stmt>;
                   | <loop_stmt>;
                   | return;
                   | <expr>;
                   | break;
                   | continue;
                   | sizeof(<type>);
<assignment>       --→ <variable> = <expr>
                   | '(' <id>[, <id>]* ')' = <func_call>
<variable>         --→ <id> [{ '[' <expr> ']' }+]
                   | ++<variable>
                   | --<variable>
                   | <variable>++
                   | <variable>--

<func_call>        --→ <id> '(' [ <parameters> ] ')'
<parameters>       --→ <variable>
                   | <variable>, <parameters>
<cond_stmt>        --→ if '(' <expr> ')' ⟨block⟩ [ else ⟨block⟩]
                   | switch '(' <id> '){'[{case <int_const>: ⟨block⟩}*] default: ⟨block⟩ '}'
<loop_stmt>        --→ for '(' [<var_dcl>] ; <expr>; [<assignment>|⟨expr⟩] ')' ⟨block⟩
                   | while '(' <expr> ')' <block>
<expr>             --→ <expr> <binary_op> <expr>
                   | '(' <expr> ')'
                   | <func_call>
                   | <variable>
                   | <const_val>
                   | -<expr>
                   | ! <expr>
<binary_op>        --→ <arithmatic> | <conditional>
<arithmatic>       --→ + | - | * | / | % | & | '|' | ^ | '||' | &&
<conditional>      --→ ==| != | >= | <= | < | >
<const_val>        --→ <int_const>
                   | <real_const>
                   | <char_const>
                   | <bool_const>
                   | <string_const>
                   | <long_const>
```

```
<id>              --→ <alpha><alphanum>*
<alphanum>        --→ <alpha> | <dig>
<alpha>           --→ a | b | … | z | A | B | … | Z | _
<dig>             --→ 0 | 1 | 2 | … | 9
<int_const>       --→ <dec_const>
                  | <hex_const>
                  | <oct_const>
<real_const>      --→ تکلیف کلاسی
<char_const>      --→ تکلیف کلاسی
<bool_const>      --→ تکلیف کلاسی
<string_const>    --→ تکلیف کلاسی
<long_const>      --→ تکلیف کلاسی
<dec_const>       --→ تکلیف کلاسی
<hex_const>       --→ تکلیف کلاسی
<oct_const>       --→ تکلیف کلاسی
```

۷-۲- قوانین زبان

همانطور که احتمالاً تاکنون متوجه شدید، این زبان بسیار شبیه به زبانهای C و C++ است. به همین ترتیب، قواعد مشابهی نیز دارد که در ذیل می‌آیند.

مشابه زبان C++ روند اجرای برنامه از تابع here() بدون آرگومان ورودی و با نوع خروجی int آغاز می‌شود. عدم وجود این تابع باعث بروز خطای معنایی می‌شود.

ساختار برنامه ها

هر برنامه دارای تعدادی نوع، تابع و متغیر سراسری است.

- توابع

هر برنامه، حداقل باید شامل یک متد here() باشد که نقطه شروع برنامه است (البته هر برنامه دقیقاً یک تابع here() دارد). همچنین هر برنامه می‌تواند، تعدادی تابع داشته باشد. هر تابع تعدادی ورودی و تعدادی خروجی دارد. همه توابع پارامترهایشان را به شیوه call-by-reference دریافت می‌کنند. بنابراین تغییرات روی محتوای متغیرها در خارج از تابع نیز اثر می‌گذارد. همچنین خروجی توابع برای همه انواع دادهای از نوع value است.

همچنین توابع نیز می‌توانند declare یا define شوند. همچنین در این زبان، overloading نیز مجاز می‌باشد؛ به این معنا که توابع با اسامی و نوع خروجی یکسان، پارامترهای متفاوتی داشته باشند.

توابع شامل تعدادی متغیر محلی هستند، این متغیرها میتوانند با متغیرهای سراسری همنام باشند ولی نباید با اسم توابع دیگر یا نوعها همنام باشند.

همچنین هر تابع هر گاه به گزاره return برسد کار را تمام می‌کند و خروجی‌ها را باز می‌گرداند. در این برنامه توابع می‌توانند بیش از یک مقدار نیز برگرداند به شرط آنکه مقادیر در جلوی return در بین پرانتز قرار گیرند. توجه کنید که ورودی و خروجی تابع در ابتدای آن می‌آیند.

دو تابع overload یکدیگرند اگر و تنها اگر، نام یکسان داشته باشند و امضای آنها که شامل ورودیها و خروجیهای آنها است، باهم متفاوت باشد.

- متغیرها

متغیرها، به دو بخش پایه‌ای (Primitive) و غیرپایه‌ای (User-defined) تقسیم می‌شوند، همچنین از هریک از این انواع می‌توان آرایه نیز داشت. همچنین متغیرها از نظر حوزه (Scope) تعریف، یا سراسری هستند و یا محلی. متغیرهای سراسری از هر بخشی (Block) قابل دسترسی هستند. اما متغیرهای محلی فقط در داخل Block ی که در آن تعریف شده‌اند، قابل شناسایی و استفاده هستند؛ همچنین متغیرهای محلی پس از اتمام Block دیگر قابل دسترسی نیستند. متغیرهای سراسری مقداردهی اولیه ندارند.

همه آرایه‌ها باید با استفاده از حافظه پویا مقدار دهی شوند. به این منظور کافی است برای کل آرایه یک بار حافظه تخصیص دهیم. (این نکته در تعارض با زبان C است).

متغیرهای ثابت(const) باید هنگام تعریف مقداردهی شده و این مقدار نیز تا پایان برنامه ثابت می‌ماند. بدیهی است متغیر سراسری ثابت نداریم.
همچنین استفاده از ++ یا -- برای تغییر مقدار متغیرها ساده می‌باشد، یعنی ترکیبی یا چند اپراتور به پیشوندی و پسوندی نیست؛ هر چند گرامر چنین امکانی را می‌دهد. یعنی مثلاً a+(c++)*d نداریم.

- سایر ساختارها

- در هر نقطه از کد امکان تعریف متغیر وجود دارد.
- اپراتور انتساب، مانند سایر زبانهای برنامه سازی کار می‌کند؛ در صورتی که مقدار یک متغیر را برابر عبارتی قرار دهیم، مقدارش تغییر می‌کند، اما نمی‌توان این کار را با یک آرایه انجام داد. برای همه انواع پایه امکان cast وجود دارد؛ به این صورت که مقدار اعداد اعشاری یا صحیح به نزدیکترین مقدار ممکن به مقدار واقعی ابتدایی تبدیل می‌شوند. در تبدیل int به char بایت کم ارزش منتقل می‌شود. به این ترتیب تبدیل‌های (int, float), (int, double), (float, double), (int, char) بصورت ضمنی انجام می‌شوند (این توضیحات در مورد عکس این تبدیل‌ها نیز صادق است). بقیه موارد به صورت خطا هستند. تبدیلات بین انواع bool با سایر انواع پایه‌ای از قانون زیر تبعیت می‌کند. همچنین هر رشته می‌تواند به یک آرایه از کاراکترها انتساب داده شود.
- هر جا که نیاز به عبارت شرطی است، مقدار صفر به معنای false و مقدار غیر صفر true فرض می‌شود.
- عملگرهای بیتی، فقط بر روی اعداد صحیح قابل اجرا هستند.
- عبارات break و continue فقط داخل بدنه حلقه‌ها قابل استفاده هستند.
- در ساختار switch-case دقیقاً بدنه یکی از caseها اجرا می‌شود، همچنین وجود default ضروری است.
- عملگرهای ++ و -- فقط برای انواع صحیح و کاراکترها قابل استفاده هستند. نوع پیشوندی یا پسوندی هر دو پس از اتمام آن statement مقدار را تغییر می‌دهند.
- کنترل اجرا بین توابع دست به دست می‌شود، توجه کنید که این کنترل اجرا در نهایت باید به caller بازگردد. مثلاً اگر f تابع g را صدا زده پس از اتمام g باید کنترل به f بازگردد. همچنین، فراخوانی تو در تو و بازگشتی نیز مجاز است.
- در این زبان تابع ()len یک رشته گرفته و طول آن را باز می‌گرداند.
- در این زبان دو تابع input و output برای اعمال ورودی/خروجی در نظر گرفته شده اند. هر یک از توابع یک متغیر را دریافت کرده و بر اساس نوعش کار متناسب را انجام می‌دهند. ورودی/خروجی این توابع stdin/stdout می‌باشد. تست و نمرهدهی بر اساس عملکرد این دو تابع می‌باشد.
- پیاده‌سازی توابع دو بند بالا اجباری است.
- این زبان خاصیت Boolean short circuit دارد، به این معنا که در صورتی که جواب یک عبارت شرطی در میان ارزیابی آن مشخص شود، مابقی عبارت بررسی نمی‌گردد.
- فضای همه متغیرهای محلی باید در پشته باشد. (در صورت پیاده‌سازی بخش تابع)
- (خلاصه کار با آرایه‌ها)! میدانید که آرایه، ساختمان داده‌ای از عناصر همنوع مربوط به هم است. در آرایه این تاکید وجود دارد که خانه‌های آرایه پشت سرهم باشند. هر متغیری که پس از نامش حاوی تعدادی اپراتور [] باشد یک آرایه نام دارد. داخل این اپراتور تعداد خانه‌های مورد نیاز هر بعد از آرایه می‌آید. در این زبان هر فضایی به صورت پیشفرض به صورت محلی است، ولی فضای تخصیص داده شده به آرایه به صورت پویا تخصیص داده می‌شود. تا زمانی که به انتهای حوزه خود نرسیده نابود نمی‌شود. دقت نمایید سایزی که باید به آرایه تخصیص داده شود در زمان اجرا محاسبه شده و دقیقاً به همان میزان فضا تخصیص داده می‌شود. لازم به ذکر است نحوه دسترسی به هر عضو از آرایه مشابه زبان C می‌باشد. همچنین توجه کنید که آرایه‌ها عملاً یک نوع هستند و با id شان شناسایی میشوند که این id خود محلی است. فقط idیی آرایه است که در زمان تعریف به شکل آرایه تعریف شده باشد.

- قوانین اجرای برنامه

کامپایلر این زبان بایدکد داده شده به زبان سطح بالا را دریافت کرده و در نهایت کد قابل اجرا بر روی ماشین (لینوکس یا ویندوز) را تحویل دهد. این خروجی به زبان اسمبلی (x86، MIPS و یا موارد دیگر که درکلاس صحبت خواهد شد)

می‌باشد. برای تبدیل کد به یک برنامه اجرایی، فقط اجازه استفاده از یک اسمبلر (gas یا nasm) و یک linker را دارید. استفاده از همه کتابخانه‌های C++/C نیز مجاز می‌باشد. همچنین ابزار شما باید از طریق خط فرمان قابل فراخوانی باشد. این برنامه باید دو پارامتر ورودی را دریافت کند (نیازی به پیشبینی حالات خاص نیست، حتماً دو آرگومان داده می شود) پارامتر اول یک فایل به زبان holoo است (مثلاً a.holoo) پارامتر دوم هم یک فایلی است که باید خروجی در آن ریخته شود. نوع این فایل از دو حالت خارج نیست؛ یک فایل با پسوند s. که باید خروجی اسمبلی داخل آن نوشته شود یا فایلی بدون پسوند که فایل اجرایی برنامه است. به عنوان مثال اگر نام برنامه compiler باشد و قرار باشد فایل a.holoo کامپایل شود، دو حالت وجود دارد:

```
compiler a.holoo out1.s
compiler a.holoo out1
```

که اولی کد اسمبلی تولید شده را در مسیر برنامه قرار می‌دهد و دومی همین کار را با فایل اجرایی انجام می‌دهد.
همچنین به ازای هر دستور یک فایل با نام کد منبع که به دنبال آن عبارت ParseReport.txt می‌آید، مثلاً در دو دستور بالا فایل aParseReport.txt تولید می‌شود.

۳- تحویل پروژه

مهلت تحویل: در سایت cms.jahromu.ac.ir مشخص خواهد شد.
تحویل دادنیها: کدهای توسعه داده شده و مستند پروژه
جریمه تاخیر: به ازای هر ساعت 5درصد، کسر خواهد شد. (حداکثر تاخیر 8روز)
نام برنامه: compiler
به تذکرات توجه کنید! عواقب هر گونه بیتوجهی بر عهده خود شماست.
همه فایلها را باید قبل از ارسال به قالب zip. در آورید، نامگذاری فایلها باید به صورت زیر باشد:

[Std No]_[Compiler1400Project]_[Part No]
[Std No]_[Compiler1400Project]_[Final]

فایل ورودی شامل تعدادی نویسه از کاراکترهاست که لزوماً در قالب زبان برنامه سازی داده شده نیست. همچنین وظیفه شما ساختاریابی کد داده شده به زبان مبدا است.
باید خودتان یا به کمک یکی از ابزارهای داده شده، برنامه ای بنویسید، که هر کد زبان داده شده (منطبق بر گرامر و تعریف) را ساختاریابی کرده و نتیجه را گزارش کند. توجه کنید که گرامر داده شده برای Parse کردن مناسب نیست؛ باید آن را تغییر دهید؛ البته این تغییر باید به صورتی باشد که زبان آن عوض نشود و بتواند هر رشته‌ای را که منطبق بر توضیحات ارائه شده است را بپذیرد. (اگر تغییر زبان، بگونه‌ای باشد که با این قوانین منافات نداشته باشد؛ ایرادی ندارد).
بنابراین یک فایل دلخواه به کامپایلر شما داده می‌شود که لزوماً از نظر Syntax و Semantic صحیح نیست. برنامه شما باید برای هر برنامه‌ای که صحت هر دو جنبه را داشت، یک فایل اجرایی و کد اسمبلی آن فایل اجرایی را ایجاد نماید. این کد اسمبلی باید به گونهای باشد که بعد از اسمبل شدن برنامه اجرایی را تولید نماید. همچنین استفاده از هر ابزاری که تولید کد را انجام دهد، مجاز نمی‌باشد؛ کد باید توسط خود شما تولید شود.
برنامه اجرایی و کد اسمبلی باید با پلتفرم Linux و یا windows سازگار باشد. همانطور که از پیشتر نیز اشاره شد میتوانید از یکی از اسمبلرهای nasm یا gas استفاده کنید (اولی به شیوه Intel و دومی به شیوه AT&T می‌باشد) در صورتی که می‌خواهید از اسمبلرهای دیگر استفاده کنید، از قبل هماهنگ کنید. همچنین کد باید بر روی پردازندههای 32 بیتی قابل اجرا باشد.
در این پروژه، نمره به صحت عملکرد کد تولیدی شما تعلق میگیرد با این حال، اگر در مورد هر تست، کاشف به عمل آید که برنامه شما اتفاقی جواب درست را تولید کرده یا اتفاقی کدی تولید شده که جواب درست را محاسبه میکند، نمره از دست می‌دهید.

همانطور که اشاره شد، برای هر فراخوانی باید یک فایل ParseReport تولید کنید که در آن نتیجه Parse را نوشته‌اید (yes یا no) اگر نتیجه no بود، فایل درخواستی خروجی باید خالی باشد.

ورودی و خروجی برنامه تولیدی

لازم است برنامه تولیدی با stdin و stdout کار کند. همچنین فقط انواع Primitive توسط تابع input یا output پذیرفته می‌شوند.

تنها نکته در مورد خروجی اعداد اعشاری است؛ بخش صحیح باید کامل چاپ شود و بخش اعشاری تا حداکثر 6 رقم چاپ شود. (در صورتی که نیازی به 2 رقم باشد، بقیه ارقام نباید چاپ شوند مثلا: 1.25 نه 1.250000

توزیع نمرات

پروژه دو بخش دارد، یکی ساختاریابی و دیگری تولید کد. جزئیات نمرات در جدول زیر آمده است. همچنین پروژه یک بخش بهینه‌سازی هم دارد که البته اختیاری است.

توجه کنید که موارد امتیازی در بخش تولید کد تعریف شده‌اند. بنابراین هر کد صحیحی را باید بتوانید ساختار یابی کنید. همچنین لازم است اگر قصد دارید بخش امتیازی را انجام دهید اطلاع دهید. بخشهای امتیازی مشخص شده‌اند و میتوانید حدود 82 نمره از آنها بدست آورید.

هر سوالی داشتید در اسرع وقت بپرسید.

فاز	نمره
تجزیه کننده	٤٠
تولید کد	٨٠
جمع	١٢٠

مورد تولید کد	امتیاز نسبی	امتیازی
خواندن از ورودی	۴	
نوشتن در خروجی	۴	
انتساب صحیح انواع داده‌ای همگن(غیر تابع)	۱	
محاسبات ساده صحیح (جمع و تفریق)	۴	
محاسبات ساده صحیح (ضرب و تقسیم)	۵	
محاسبات صحیح (همه محاسبات + تقدم عملگرها)	۶	
محاسبات صحیح با حضور نوع long	۴	*
محاسبات صحیح بیتی	۲	
تبدیل انواع در محاسبات و انتساب (cast)	۱	
متغیر Boolean و کاراکتر	۱	
رشته، تابع len، انتساب به آرایه کاراکتری و کارهای مرتبط	۳	
محاسبات ممیز شناور	۷	*
محاسبات ترکیبی	۵	*
ساختار شرطی	۷	
تعریف صحیح متغیر سراسری	۲	
تعریف و پیاده‌سازی صحیح متغیر محلی	۴	
متغیرهای ثابت (پیش نیاز: متغیر محلی و سراسری)	۲	*
ساختار صحیح Block و طول عمر متغیرها	۳	
Inc & Dec	۲	
حلقه for	۵	

پیاده‌سازی تابع (فراخوانی و اجرا + تابع بازگشتی)	۸	*
پیاده‌سازی تابع (بازگشت مقادیر خروجی)	۵	*
پیاده‌سازی اعلان تابع(پیش نیاز: پیاده‌سازی تابع(هر دو))	۴	*
پیاده‌سازی کامل function overloading (پیش نیاز: اعلان تابع)	۴	*

نیاز به تابع here	۱	
عملکرد صحیح جدول نمادها (متغیر هم نام، کلمات کلیدی و...)	۳	
تعریف و استفاده از آرایه یک بعدی + تخصیص حافظه	۴	
تعریف و استفاده آرایه چند بعدی + تخصیص حافظه	۵	*
تخصیص مناسب حافظه پویا	۲	
آزاد سازی صحیح حافظه پویا	۲	
دستور break (پیش نیاز: پیاده‌سازی حلقه)	۱	
دستور continue (پیش نیاز: پیاده‌سازی حلقه)	۱	
دستور switch-case	۴	*
دستور switch-case با جدول پرش	۶	*
حذف کدهای غیر قابل دسترس(Dead-Code Elimination)	۸	*