


به نام خداوند بخشنده مهربان

| | |
|---|---|
|  | <p>فاز ۳ پروژه - درس اصول و طراحی کامپایلرها</p> <p>دکتر ممتازی</p> <p>ترم زمستان ۱۳۹۸-۱۳۹۹ - دانشکده کامپیوتر، دانشگاه صنعتی امیرکبیر</p> <p>زمان تحویل: ۱۵ مرداد ماه ۹۹</p> |
|---|---|

لطفا قبل از شروع به حل کردن تمرین به نکات زیر توجه فرمایید:

(۱) هدف از انجام تمرین‌ها، یادگیری عمیق‌تر مطالب درسی است. در نتیجه هرگونه کپی‌برداری موجب کسر نمره خواهد شد.

(۲) تا ساعت ۲۳:۵۵ روز ۱۵ مرداد ماه فرصت دارید تا تمرین را در مودل بارگذاری کنید. تمام فایل‌های خواسته شده را در یک فایل فشرده قرار دهید. نام فایل نهایی را شماره دانشجویی خود قرار دهید. (برای مثال phase3_Family_9631747.zip)

(۳) در کنار این فایل ۱ ویدئو برای راهنمایی شما وجود دارد که توصیه می‌کنیم آن‌ها را مشاهده بکنید. محتوای ارائه شده در فیلم‌ها در آدرس زیر قابل دسترسی است:

https://drive.google.com/drive/folders/121W_A0glhLA7_nPf9FjQJiexcVzq8t2v?usp=sharing

(۴) در صورت وجود هرگونه سوال می‌توانید از طریق ایمیل با تدریس‌یار در ارتباط باشید.

moh.robati@aut.ac.ir

sheykh@aut.ac.ir

شما در مرحله قبل یک تجزیه گر^۱ طراحی کردید که از نشانه های برگردانده شده استفاده کرده و تشخیص داد آیا برنامه با قوانین گرامر داده شده مطابق هست یا خیر. همچنین شما گرامر داده شده را رفع ابهام کردید. حال از شما می خواهیم که کد های گرامر به جز توابع و اعداد غیر صحیح را پیاده سازی کنید. (بر روی بخش هایی که نیاز به پیاده سازی ندارند خط کشیده شده و به رنگ زرد در آمده اند).

نکته: گرامر داده شده را می توانید با استفاده از اولویت^۲ ها و تغییر ناپایانه^۳ ها ، بدون اینکه زبان پذیرنده زبان عوض شود، جهت راحت تر شدن کار و ممکن شدن پیاده سازی (مانند استفاده از وصله زنی^۴) تغییر دهید .

شکل برنامه خروجی:

برنامه C شما باید به شکل زیر باشد:

```
#include <stdio.h>

int array[(int)1e5];

int Id1, Id2, ..., Idn;

int main()
{
    statement_1;
    L1: statement_2;
    .
    .
    L2: statment_k ;
    .
    .
    statement_n;
}
```

¹ Parser

² Precedence

³ Nonterminal

⁴ Backpatching

دستورات قابل استفاده در main :

| | | |
|--|---|---------------------------|
| 1) $x = y \text{ op } z$ | $op \in \{+, -, *, /, \%, , \&\&\}$ | $x, y, z \in Z \cup ID$ |
| 2) $x = op \ y$ | $op \in \{-, !\}$ | $x, y \in Z \cup ID$ |
| 3) $x = y$ | | $x, y \in Z \cup ID$ |
| 4) $\text{goto } L$ | | $L \in \{L1, \dots, Ln\}$ |
| 5) $\text{if } (x \text{ relop } y) \text{ goto } L$ | $\text{relop} \in \{<, >, <=, >=, ==, !=\}$ | $x, y \in Z \cup ID$ |
| 6) $\text{array}[x] = y$ | | $x, y \in Z \cup ID$ |
| 7) $x = \text{array}[y]$ | | $x, y \in Z \cup ID$ |
| 8) $\text{printf}(\text{"\%d"}, x)$ | | $x \in ID$ |

نکته اول: قالب خاصی برای اسم ID ها نیاز نیست. صرفاً مهم است که با قواعد زبان C سازگار باشد.

نکته دوم: شما در اینجا باید استفاده اعداد صحیح در جایگاه عبارات بولی و همچنین عبارات بولی در محاسبات اعداد را مانند زبان C پیاده سازی کنید. همچنین می توانید از این ویژگی زبان C در کد پیاده سازی شده استفاده کنید. به طور مثال:

$$x, y \in Z \rightarrow \text{if}(x \&\&y) \text{ goto } L \begin{cases} \text{if } x! = 0 \text{ And } y! = 0 \text{ goto } L \\ \text{otherwise do nothing} \end{cases}$$

$$x \in Z, y = \text{true} \rightarrow T = y + y + x = 2 + x$$

نکته سوم: استفاده از 2 برچسب⁵ پشت هم برای اشاره به یک خط مجاز نیست. هر خط حداکثر یک برچسب می تواند در پشت خود داشته باشد.

نکته چهارم: نیازی به type checking در این فاز نیست و ما فقط تعریف اعداد صحیح را داریم.

نکته پنجم: اجازه تعریف آرایه دیگری غیر از array اصلی که در شکل برنامه خروجی توضیح داده شده است را ندارید.

نکته ششم: لطفاً از دستورات و عملگرها خارج از قالب چیزهایی که در صفحه قبل توضیح داده شده اند استفاده نکنید زیرا نمره ای تعلق نخواهد گرفت.

نکته هفتم: کامپایل شدن و دادن خروجی درست برنامه نمره دارد.

⁵ Label

Grammar for phase 3:

program \rightarrow declist main () block

declist \rightarrow dec | declist dec | ϵ

dec \rightarrow vardec | funcdec

type \rightarrow int | float | bool

iddec \rightarrow id | id [exp] | id=exp

idlist \rightarrow iddec | idlist , iddec

vardec \rightarrow type idlist ;

funcdec \rightarrow type id (paramdecs) block | void id (paramdecs) block

paramdecs \rightarrow paramdecslist | ϵ

paramdecslist \rightarrow paramdec | paramdecslist , paramdec

paramdec \rightarrow type id | type id []

varlist \rightarrow vardec | varlist vardec | ϵ

block \rightarrow { varlist stmtlist }

stmtlist \rightarrow stmt | stmtlist stmt | ϵ

lvalue \rightarrow id | id [exp]

stmt \rightarrow return exp ; | exp ; | block |

while (exp) stmt |

for(exp ; exp ; exp) stmt |

if (exp) stmt elseiflist | if (exp) stmt elseiflist else stmt |

`print (id) ;`

`elseiflist` \rightarrow `elif (exp) stmt` | `elseiflist elif (exp) stmt` | ϵ

`exp` \rightarrow `lvalue=exp` | `exp operator exp` | `exp relop exp` |

`const` | `lvalue` | `id(explist)` | `(exp)` | `id()` | `- exp` | `! exp`

`operator` \rightarrow `"|"` | `&&` | `+` | `-` | `*` | `/` | `%`

`const` \rightarrow `intnumber` | `floatnumber` | `true` | `false`

`relop` \rightarrow `>` | `<` | `!=` | `==` | `<=` | `>=`

`explist` \rightarrow `exp` | `explist,exp`