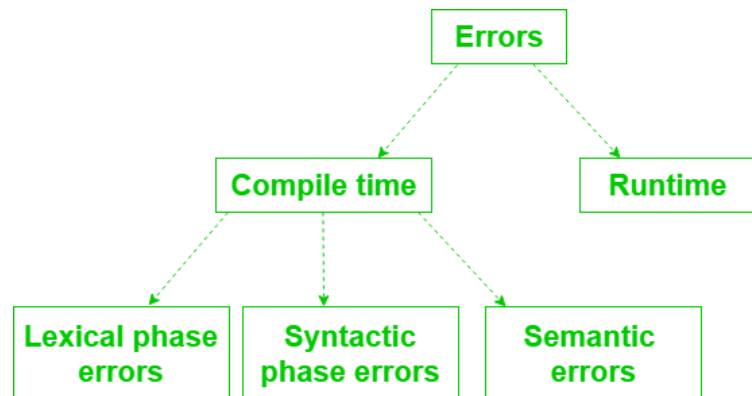


به نام خدا



تکلیف اول کامپایلر
حدیث غفوری (9825413)

سوال ۱



خطای لغوی دنباله ای از کاراکترها است که با الگوی هیچ token مطابقت ندارد. خطای فاز لغوی در حین اجرای برنامه پیدا می شود.

خطاهای لغوی چندان رایج نیستند، اما باید توسط یک اسکنر مدیریت شوند.

غلط املایی شناسه ها، عملگرها، کلیدواژه ها از جمله اشتباهات واژگانی محسوب می شوند.

به طور کلی، یک خطای لغوی به دلیل ظاهر شدن برخی از کاراکترهای غیرمجاز، بیشتر در ابتدای یک token ایجاد می شود.

خطای فاز لغوی می تواند یکی از موارد زیر باشد:

غلط املایی.

بیش از طول شناسه یا ثابت های عددی (constant) .

ظاهر شدن کاراکترهای غیرمجاز.

برای حذف کاراکتری که باید وجود داشته باشد.

برای جایگزینی یک کاراکتر با یک کاراکتر نادرست.

جابجایی دو کاراکتر

Example 1 : `printf("test-test");$`

این یک خطای لغوی است زیرا یک کاراکتر غیرمجاز \$ در انتهای عبارت ظاهر می شود.

Example 2 : `This is a comment */`

این یک خطای واژگانی است زیرا عملگر پایان کامنت وجود دارد، اما ابتدای آن نه.

```
#include <stdio.h>

void main(){
    int a,b;
    scanf("%d %d",&a,&b);
    int sum=a+b;
    printf("%d",sum);**@**
}
```

کد بالا به دلیل وجود یک کاراکتر غیرمجاز بعد از عبارت printf یک خطای لغوی ایجاد می کند.

1. Void main()
2. {
3. int x=10, y=20;
4. char * a;
5. a= &x;
6. x= 1xab;
7. }

در این کد xab1 نه عدد است و نه شناسه. بنابراین این کد خطای لغوی را نشان می دهد.

Error recovery methods

Different Error recovery methods in Compiler Design are:

1. Panic mode recovery
2. Statement mode recovery
3. Error productions
4. Global correction
5. Using Symbol table

Panic mode recovery

این ابتدایی ترین و ساده ترین راه بازیابی خطا است. در این حالت به محض اینکه تجزیه کننده در هر نقطه ای از برنامه خطایی را تشخیص داد، بلافاصله بقیه قسمت باقیمانده کد را از آن مکان خاص که خطا در آن یافت می شود با پردازش نکردن آن حذف می کند.

در این روش، کاراکترهای متوالی از ورودی یکی یکی حذف می شوند تا زمانی که مجموعه مشخصی از نشانه های همگام سازی پیدا شود. توکن های همگام سازی جداکننده هایی مانند؛ یا { هستند.

مزیت :

1. این اساسی ترین روش است و اجرای آن بسیار آسان است.

2. هرگز در یک حلقه بی نهایت نمی رود.

عیب:

1. حجم عظیمی از ورودی بدون پردازش دور ریخته می شود و خطای دیگری را شناسایی نمی کند زیرا بیشتر آن را پردازش نمی کند.

خطاهای فاز لغوی و خطاهای فاز نحوی با این روش بازیابی می شوند.

سوال ۲

Single Pass Compiler

یک کامپایلر single pass تنها یک بار از متن منبع عبور می کند، تجزیه، تجزیه و تحلیل، و کد تولید می کند. به عبارت دیگر، این اجازه می دهد تا کد منبع فقط یک بار از هر واحد کامپایل عبور کند. بلافاصله هر بخش کد را به کد ماشین نهایی خود ترجمه می کند.

مراحل اصلی کامپایلر single pass عبارتند از تجزیه و تحلیل لغوی، تجزیه و تحلیل نحوی و تولید کننده کد. ابتدا، تحلیل واژگانی کد منبع را اسکن می کند و آن را به token تقسیم می کند. هر زبان برنامه نویسی گرامر دارد. این بیان کننده نحو و بیانیه های قانونی زبان است. سپس، تحلیل نحوی ساختارهای زبانی را که توسط گرامر توصیف شده است، تعیین می کند. در نهایت، مولد کد هدف را تولید می کند. به طور کلی، کامپایلر single pass کد را بهینه نمی کند. علاوه بر این، تولید کد میانی وجود ندارد.

Multipass Compiler

یک کامپایلر multipass باعث می شود که کد منبع چندین بار در حین تولید کد میانی پس از هر مرحله، تجزیه و تحلیل، تولید و غیره را طی کند. این برنامه را در مراحل بین کد منبع و کد ماشین به یک یا چند نمایش میانی تبدیل می کند. کل واحد کامپایل را در هر گذر متوالی دوباره پردازش می کند.

تفاوت های Single Pass و Multipass

کامپایلر تک گذر نوعی کامپایلر است که تنها یک بار از قسمت های هر واحد کامپایل عبور می کند و بلافاصله هر بخش کد را به کد ماشین نهایی خود تبدیل می کند. کامپایلر چند پاسی نوعی کامپایلر است که کد منبع یا درخت نحو انتزاعی یک برنامه را چندین بار پردازش می کند. از این رو، این تعاریف تفاوت اصلی بین کامپایلر تک پاسی و کامپایلر چندگذری را توضیح می دهد.

سرعت

سرعت یک تفاوت عمده بین کامپایلر تک پاس و چند پاس است. یک کامپایلر چند گذری کندتر از کامپایلر تک پاسی است زیرا هر پاس یک فایل میانی را می خواند و می نویسد.

مترادف ها

کامپایلر تک پاسی را کامپایلر باریک narrow compiler می نامند در حالی که کامپایلر چند گذری کامپایلر گسترده نامیده می شود.

محدوده

علاوه بر این، یک کامپایلر تک پاسی دارای دامنه محدود است در حالی که یک کامپایلر چند گذری دامنه بیشتری دارد.

بهینه سازی کد

تفاوت دیگر بین کامپایلر تک پاسی و کامپایلر چند گذری این است که در کامپایلر تک پاسی بهینه سازی کد وجود ندارد، بر خلاف کامپایلر چند پاسی که دارای بهینه سازی کد است.

کدهای میانی

کدهای میانی نیز بین کامپایلر تک پاسی و کامپایلر چند گذری تفاوت ایجاد می کنند. تولید کد میانی در کامپایلرهای تک پاس وجود ندارد. با این حال، تولید کد میانی در کامپایلرهای چندگذری وجود دارد.

زمان تدوین

علاوه بر این، یک کامپایلر تک پاس در مقایسه با یک کامپایلر چند گذر، حداقل زمان لازم را برای کامپایل نیاز دارد.

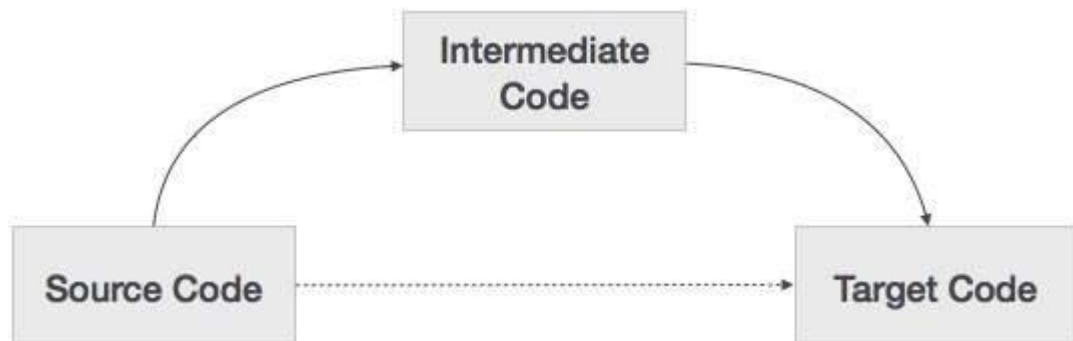
مصرف حافظه

همچنین مصرف حافظه در یک کامپایلر چند گذری بیشتر از یک کامپایلر تک پاس است.

زبانهای برنامه نویسی

زبان های برنامه نویسی مانند پاسکال را می توان با استفاده از یک کامپایلر واحد پیاده سازی کرد در حالی که زبان های برنامه نویسی مانند جاوا را می توان با استفاده از کامپایلر چند پاسی پیاده سازی کرد.

SINGLE PASS COMPILER	MULTIPASS COMPILER
A type of compiler that passes through the parts of each compilation unit only once, immediately translating each code section into its final machine code	A type of compiler that processes the source code or abstract syntax tree of a program several times
Faster than multipass compiler	Slower as each pass reads and writes an intermediate file
Called a narrow compiler	Called a wide compiler
Has a limited scope	Has a great scope
There is no code optimization	There is code optimization
There is no intermediate code generation	There is intermediate code generation
Takes a minimum time to compile	Takes some time to compile
Memory consumption is lower	Memory consumption is higher
Used to implement programming languages such as Pascal	Used to implement programming languages such as Java



ما اساساً دو فاز کامپایلر داریم، یعنی فاز تحلیل و فاز سنتز. مرحله تجزیه و تحلیل یک نمایش میانی از کد منبع داده شده ایجاد می کند. مرحله سنتز یک برنامه هدف معادل از نمایش میانی ایجاد می کند.

قسمت جلویی یک کامپایلر (front end) یک برنامه منبع را به یک کد میانی مستقل ترجمه می کند، سپس انتهای کامپایلر (back end) از این کد میانی برای تولید کد هدف (که توسط ماشین قابل درک است) استفاده می کند.

کد متوسط بین زبان سطح بالا و سطح ماشین است. این کد میانی باید به گونه ای تولید شود که ترجمه آن به کد ماشین مورد نظر آسان شود.

به دلیل اینکه کد میانی مستقل از ماشین است، قابلیت portability افزایش می یابد.

اگر یک کامپایلر بدون داشتن گزینه ای برای تولید کد میانی، زبان مبدأ را به زبان ماشین مقصد خود ترجمه کند، برای هر ماشین جدید، یک کامپایلر native کامل مورد نیاز است.

کد میانی نیاز به یک کامپایلر کاملاً جدید برای هر ماشین منحصر به فرد را با حفظ بخش تجزیه و تحلیل برای همه کامپایلرها برطرف می کند.

قسمت دوم کامپایلر، سنتز، با توجه به ماشین هدف تغییر می کند.

اعمال اصلاحات کد منبع برای بهبود عملکرد کد با استفاده از تکنیک های بهینه سازی کد روی کد میانی آسان تر می شود.

هدف گذاری مجدد تسهیل می شود.

```

int main() {
    // 2 variables
    int a=10, b=20;
    printf("sum is %d", a+b);
    printf("i=%d, &i=%x", i, &i);
    return 0;
}

```

UWY

valid token are

'int', 'main', '(', ')', '{', '}', 'int', 'a', '=',
 '10', ';', 'b', '=', '20', '}', 'printf',
 '(', "sum is %d", '}', 'a', '+', 'b', ')',
 ';',
 'printf', '(', "i=%d, &i=%x",
 '}', 'i', '}', '&', 'i', ')', ';', 'return', '0',
 '}'

'int' , 'main' , 'printf' , 'return' B token (توکن)
B keyword (کلیدواژه)
 '(', ')', ',', ';', '{', '}' B separator (فصل‌دهنده)
 'a', 'b', 'c', 'i' B identifier (شناسگر)
 '=', '+', '&' B operator (عملگر)
 '10', '20', '0' B integer (عدد صحیح)
 "sum is %d" , B string literal (سریال رشته)
 "i = %d , &i = %x" *

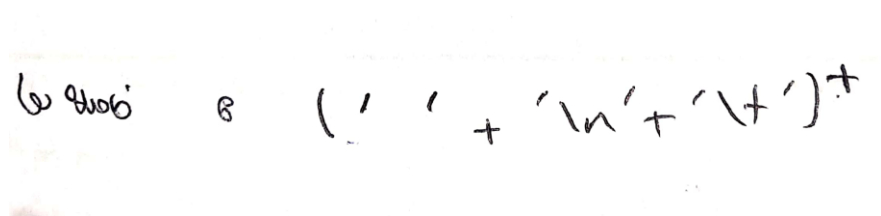
سوال ۵

(الف)

(650)-723-3232

digit = '0' + '1' + '2' + '3' + '4' + '5' + '6' + '7' + '8' + '9'
 Σ = digits U { - , (,) } exchange = digit⁺ (digits = digit⁺)
 phone = digit⁺ area = digit⁺
 phone_number = '(' area ')' - exchange - phone ->

(د)



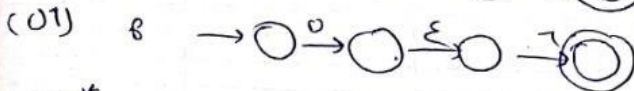
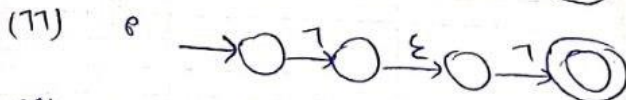
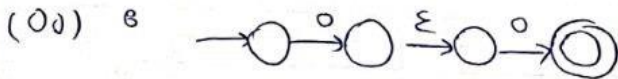
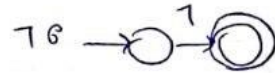
A photograph of a piece of paper with a handwritten mathematical expression in black ink. The expression is $((00)^*(11) + 01)^*$. The handwriting is somewhat cursive and the paper has some texture and slight discoloration.

سوال ۶

$$(((00)^*(11)) + 01)^*$$

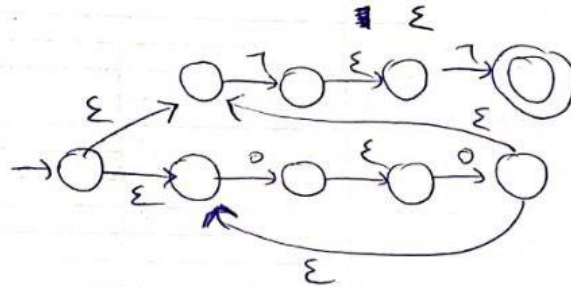
F.A به معنای

$$(((00)^*(11)) + 01)^*$$



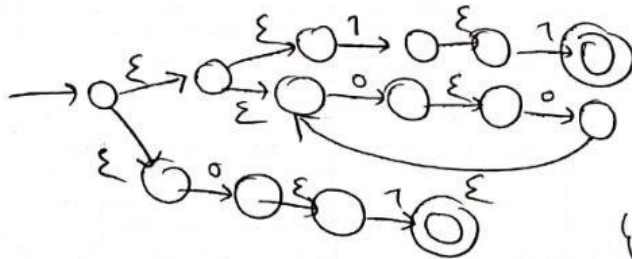
$$(00)^*(11)$$

Final state
در $(00)^*$ حذف
می کنند و به ϵ یا
از آن به ابتدای (11)
وصل می کنند



رابطه ای که برای $(00)^*(11)$ را به دست آوریم را
یا NFA - (01) or می کنند

$$(((00)^*(11)) + 01)$$



فصل مربوط به
 $(00)^*(11)$

(01) - NFA

به state می رسد که فایناال تمام هست به ابتدای
NFA اضافه می کنند و چون روی عبارت قبلی که * داریم
از final state های عبارت قبلی به ابتدای
یا از می کنند

تابع انتقال در nfa

state	0	1	epsilon
Q0			Q1
Q1			Q2,Q11
Q2			Q3,Q7
Q3		Q4,Q5	
Q4			Q5
Q5		Q6,Q1,Q2,Q3,Q7,Q11	
Q6			Q1
Q7	Q8,Q9		
Q8			Q9
Q9	Q10,Q3,Q7		
Q10			Q3,Q7
Q11	Q12,Q13		
Q12			Q13
Q13		Q14,Q1,Q2,Q3,Q7,Q11	
Q14			Q1

سوال ۷

Syntax error is found during the execution of the program.

Some syntax error can be:

- Error in structure
- Missing operators
- Unbalanced parenthesis

For example 1: Using "=" when "==" is needed

semantic error can be:

this type of error are detected at compile time.

- Incompatible types of operands
- Undeclared variable

Not matching of actual argument with formal argument

باتوجه به مطالب بالا:

Syntactic phase (الف)

Semantic errors (ب)

Syntactic phase (ج)

Syntactic phase (د)

Semantic errors (ه)

سوال ۸

$\text{Token}_A \quad cd^*a^*$
 $\text{Token}_B \quad c^*a^*cd$
 $\text{Token}_C \quad c^*b$

$\rightarrow \{ \text{Token}_A, \text{Token}_B \} \quad \in \text{cdcccd}$ (برای تکیه)

$\underbrace{cd}_{\text{Token}_A} \underbrace{cccd}_{\text{Token}_B}$

$\in \text{cacccbd}$ (برای تکیه)

این رشته با توکن A در ۲ حرف تفاوت دارد؛ توکن B در ۲ حرف و
 با توکن C در ۱ حرف ← ~~با توکن C در ۱ حرف~~

$\underbrace{cac}_{\text{Token}_B} \underbrace{cb}_{\text{Token}_C}$

پس اول با Token_B و سپس با Token_C مطابقت دارد.
 $\rightarrow \{ \text{Token}_B, \text{Token}_C \}$