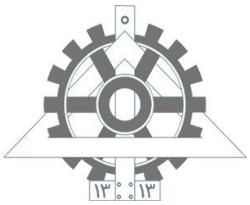
تمرین سوم طراحی کامپایلر و
زبان‌های برنامه‌نویسی



دانشکده مهندسی برق و
کامپیوتر

بهار ۱۴۰۴
مهلت تحویل: ۱۴۰۴/۰۳/۰۶

طراح ارشد: گلبو رشیدی، طراحان تمرین: فرشته باقری، دریا انصاری‌پور، امیر فریدی

---

1. با توجه به گرامر زیر به سوالات پاسخ دهید.

$P \rightarrow E$

$E \rightarrow E + E \mid E * E \mid T$

$T \rightarrow (E) \mid num$

الف)قواعد معنایی (Semantic Rules) مربوط به محاسبه حداکثر تعداد علامت‌های + پشت سر هم را برای گرامر داده شده بنویسید.

ب) parse tree این زبان را برای رشته num + num + num * num + num + num + num رسم کنید. سپس نحوه محاسبه مقادیر attribute-ها را برای هر غیرپایانه در گراف وابستگی نشان دهید.

2. قطعه‌کد زیر را در نظر بگیرید:

```
#include <iostream>
using namespace std;



int x = 10;
int y = 20;
int z = 30;

void func1() {
    int x = 5;
    if (x > y) {
        z = x + y;
        cout << "Inside func1, x = " << x << ", y = " << y << ", z = " << z
<< endl;
```

```
    } else {
        y = z - x;
        cout << "Inside func1, x = " << x << ", y = " << y << ", z = " << z
<< endl;
    }
    func2(x);
}

void func2(int b) {
    if (z < b) {
        x = y + z;
        cout << "Inside func2, x = " << x << ", y = " << y << ", z = " << z
<< endl;
    } else {
        z = b + x;
        cout << "Inside func2, x = " << x << ", y = " << y << ", z = " << z
<< endl;
    }
    func3(b);
}

void func3(int c) {
    if (y > c) {
        x = c + y;
        cout << "Inside func3, x = " << x << ", y = " << y << ", z = " << z
<< endl;
    } else {
        y = x + c;
        cout << "Inside func3, x = " << x << ", y = " << y << ", z = " << z
<< endl;
    }
}

int main() {
    cout << "Inside main, x = " << x << ", y = " << y << ", z = " << z <<
endl;
    func1();
    return 0;
}
```

الف)با فرض اینکه این کد در زبانی با dynamic scoping نوشته شده است، خروجی آن را توضیح دهید.
ب) با فرض اینکه این کد در زبانی با static scoping نوشته شده است، خروجی آن را توضیح دهید.

3. گرامر زیر را در نظر بگیرید.

$P \rightarrow S$

$S \rightarrow n\ S \mid p\ S \mid q\ S \mid n \mid p \mid q$

قواعد معنایی را برای این زبان به گونه‌ای بنویسید که به ازای هر رشته‌ی ورودی، مقدار `P.count` برابر با تعداد زیررشته‌هایی باشد که مطابق با عبارت منظم $n\ (p\ +\ q)^*\ n$ باشند. مثلا در رشته‌ی npqnqqpn، دو زیررشته‌ی npqn و nqqpn مطابق با این عبارت منظم هستند.

4. الف) فرض کنید قطعه‌کد زیر در زبانی با static scoping نوشته شده است. آیا در این قطعه‌کد division by zero رخ می دهد؟ توضیح دهید.
ب) اگر فرض کنیم قطعه‌کد زیر در زبانی با dynamic scoping اجرا شود، آیا استدلال قسمت (الف) همچنان برقرار است؟ توضیح دهید.

```
#include <iostream>
using namespace std;

int x = -3;
int f (int y){
    int res;
    if (y%2 == 0){
        y += 1 ;
    }
    else{
        res = 1/(1+x+y);
    }
    return x;
}

int g(){
    int x = -8;
    int y;
    cin >> y;
    return f(y);
```

```
}

int main(){
    cout << g() << endl;
    return 0;
}
```

5. تعریف کلاس زیر را در نظر بگیرید.

```
class X {
    int j;
    bool x[10];
    void f(int a, float b[10]) {
        for (int i = 0; i < a; i++) {
            x[i] = b[i] * a < z[b[i] + j]
        }
    }
    string z;
};
```

الف) مقدار environment را در خطوط مختلف این قطعه‌کد نشان دهید.

ب) قانون تحلیل نام را برای دسترسی به آرایه بنویسید. به عبارت دیگر یک inference rule بنویسید که نشان دهد $\Gamma \vdash x[e]$ برقرار است.

ج) درستی خط داخل حلقه‌ی for را از لحاظ تحلیل نام‌ها ارزیابی کنید و درخت استنتاج آن را رسم کنید.

6. قوانین type checking را برای داده ساختار map طراحی کنید و با استفاده از قوانین طراحی شده، خط مشخص شده در قطعه‌کد داده زیر را از نظر type checking بررسی کنید.

```
class TestChecker {

    private:

        int i;

        int j;

        map<int, int> intMap;
```

```
    map<string, int> strMap;
  public:
    TestChecker() {
        i = 0;
        j = 1;
        intMap[0] = 2;
        intMap[1] = 3;
        strMap["key"] = 5;
    }
    int compute(int x) {
        return x * x;
    }
    void access(map<int, int>& data, int k) {
        data[j] = data[data[i] + compute(k)];
        strMap["result"] = data[intMap[j]] + strMap["key"]; // Here
    }
    void print() {
        cout << "intMap[j]: " << intMap[j] << "\n";
        cout << "strMap[result]: " << strMap["result"] << "\n";
    }
};
```

7. حساب لامبدای دارای type!

فرض کنید زبانی از type ها و term ها به شکل زیر داریم:

$T ::= \ Bool \mid Nat \mid T \to T$

$t ::= \ x \mid true \mid false \mid if\ t\ then\ t\ else\ t \mid 0 \mid succ\ t \mid pred\ t \mid iszero\ t \mid \lambda x : T.\, t \mid tt$

در این زبان، قوانین تعیین type نیز به شکل زیر می‌باشند:

- $(T-Var)$: $If\ x : T\ is\ in\ the\ context\ \Gamma,\ then \qquad \Gamma \vdash x : T$
- $(T-Abs)$: $If\ \Gamma,\ x : T1 \vdash t2 : T2,\ then \qquad \Gamma \vdash \lambda x : T1.\, t2 : T1 \to T2$
- $(T-App)$: $If\ \Gamma \vdash t1 : T1 \to T2\ and\ \Gamma \vdash t2 : T1,\ then \qquad \Gamma \vdash t1\ t2 : T2$
- $(T-If)$: $If\ \Gamma \vdash t1 : Bool,\ \Gamma \vdash t2 : T,\ and\ \Gamma \vdash t3 : T,\ then \quad \Gamma \vdash if\ t1\ then\ t2\ else\ t3 : T$
- $(T-Succ)$: $If\ \Gamma \vdash t : Nat,\ then \qquad \Gamma \vdash succ\ t : Nat$
- $(T-Pred)$: $If\ \Gamma \vdash t : Nat,\ then \qquad \Gamma \vdash pred\ t : Nat$
- $(T-IsZero)$: $If\ \Gamma \vdash t : Nat,\ then \qquad \Gamma \vdash iszero\ t : Bool$

عبارت زیر را در این زبان در نظر بگیرید:

$(\lambda f : Nat \to Nat.\, \lambda x : Nat.\, f(f(x)))(\lambda y : Nat.\, if\ iszero\ y\ then\ succ\ y\ else\ pred\ y)$

نشان دهید این عبارت در این زبان well-typed است و inference tree آن را بنویسید.

8. کامپایلرها گاهی اوقات expression ها را ساده‌سازی می‌کنند تا بررسی نوع داده‌ها را آسان‌تر کرده یا کد بهینه‌تری تولید کنند. در ادامه دو ساده‌سازی احتمالی زبان Slang آورده شده است:

expression	simplified expression
if true then e_1 else e_2	e_1
(fun (x:t) -> e_1) e_2	(e_2/x}e_1 (substitution}

ساده‌سازی دوم، معمولاً در مواقع زیر استفاده می‌شود:

۱. **در زمان اجرا:** هنگام اعمال تابع به آرگومان، بدنه تابع با جایگزینی پارامتر x به آرگومان اجرا می‌شود.

۲. **در بهینه‌سازی‌های کامپایلر:** برای کاهش هزینه فراخوانی تابع و افزایش بهینه‌سازی‌ها، تابع به صورت inline و با جایگزینی آرگومان ساده‌سازی می‌شود.

۳. **در interpreter ها:** برای ارزیابی مرحله به مرحله عبارات و رسیدن به مقدار نهایی استفاده می‌شود.

۴. **در اثبات‌های رسمی:** برای تحلیل معادل بودن برنامه‌ها و درستی تبدیل‌ها به کار می‌رود.

یک ساده‌سازی $e_1 \rightarrow e_2$ برای type checking درست است اگر هر دو عبارت e_1 و e_2 تایپ یکسان داشته باشند یا اینکه هر دو ill-typed باشند.

یک ساده‌سازی $e_1 \rightarrow e_2$ برای بهینه سازی درست است اگر با این جایگزینی رفتار برنامه تغییری نکند.

در نظر داشته باشید که زبان slang یک functional language می‌باشد.

a. برای هر ساده‌سازی توضیح دهید تحت چه شرایطی برای بررسی نوع داده‌ها در static type checking صحیح است.

b. برای هر ساده‌سازی توضیح دهید تحت چه شرایطی برای بهینه‌سازی صحیح است. یک مثال از شرایطی که این ساده‌سازی ها باعث تغییر رفتار برنامه می‌شود بزنید.

c. به نظر شما، مرحله ساده‌سازی کد، باید قبل از type checking انجام شود یا بعد از آن؟ استدلال خود را بنویسید.

توضیحات:

- یک فایل به نام HW3-SID.pdf را آپلود کنید که SID شماره دانشجویی شما می‌باشد.
- در صورت تشخیص شباهت و تقلب میان حل تمرین شما و دیگران، برای هر دو دانشجو نمره صفر در نظر گرفته خواهد شد.

موفق باشید.