1- نصب کلاس‌های اصلی:

- اول method table ، class table ، object table ها را تعریف میکنیم.

- در cool پنج کلاس اولیه داریم: Object , IO , Int , Bool , String
این 5 کلاس را تعریف میکنیم:

Object class دارای 3 method است:

abort(): Object	aborts the program
type_name(): Str	returns a string representation of class name
copy(): SELF_TYPE	returns a copy of the object

IO_class دارای 4 method است:

out_string(Str): SELF_TYPE	writes a string to output
out_int(Int): SELF_TYPE	writes an int to output
in_string(_:Str	reads a string from input
in_int():Int	reads an int from input

Int_class هیچ method ای ندارد و فقط یک attribute دارد: مقدار int

Bool_class هم هیچ method ای ندارد ولی یک مقدار دارد (true/false)

Str_class دارای چند attribute و method دارد:

val	value of the string
str_field	the string itself
length(): Int	returns length of the string
concat(arg:Str): Str	a method to perform string concatenation
substr(arg:Int, arg2:Int): Str	selects substring

2- تعریف کلاسها:

چند قانون را در تعریف کلاس‌ها باید چک کنیم:
کلاس نمیتواند SELF_TYPE باشد.
اگر کلاس را قبلاً تعریف کرده ایم، نمیتوان دوباره تعریف کرد.
کلاس parent نمیتواند Int یا String با Bool یا SELF_TYPE باشد.
اگر این مشکلات پیش نیامد، میتوان کلاس را تعریف کرد.

3- وراثت:

در تابع check_inheritance اول چک کنیم که کلاس مد نظر از کلاسی تعریف نشده ارث نبرده باشد.
سپس چک میکنیم که در درخت وراثت حلقه ایجاد نشده باشد.

4- چک کردن main:

در check_main چک کنیم کلاس main تعریف شده باشد.

5- تعریف method:

هر کلاس را نگاه میکنیم و method هایش را چک میکنیم. اگر method از قبل موجود بود، error میدهیم که قبلاً تعریف شده. وگرنه method جدید را تعریف میکنم.

توابع کمکی:

دو تابع مینویسیم که درخت وراثت را برای class و symbol تعریف کند.
یک تابع conform برای بررسی تطابق تایپ دو کلاس مینویسیم
تابع LCA برای بدست آوردن least common ancestor دو کلاس است به این صورت که درخت وراثت این دو را میگیریم و بالا می‌رویم تا به یک ancestor مشابه برسیم.
تابع getMethod برای درآوردن یک method در کلاسِ داده شده است.
در تابع check_inheritance اول چک کنیم که کلاس مد نظر از کلاسی تعریف نشده ارث نبرده باشد.
سپس چک میکنیم که در درخت وراثت حلقه ایجاد نشده باشد.

6- check_methods :

چند چیز را وقتی از یک method استفاده میکنیم باید چک کرد که بکمک این تابع انجام میدهیم:
یک method که به ارث برده شده، نباید متفاوت از method در کلاسِ پدر باشد.
در method تایپِ پارامتر ها باید با تایپ در کلاس پدر برابر باشد.
تعداد پارامتر های method باید درست باشد.
یک attribute کلاسش تعریف شده باشد.
اگر کلاس attribute تعریف شده باشد ولی تایپِ داده شده درست نباشد، باید ارور داد.

7- type checking:

type checking rule هایی که در بخش 12.2 فایل cool-manual نوشته شده را پیاده‌سازی میکنیم

- برای کلاس‌های int و bool و string و not_type تایپ های مربوطه بازگردانده میشوند.

- Assignment: باید Id object در object environment فعلی معتبر باشد. اگر پیدا نشد ارور مناسب را چاپ میکنیم.
  سپسب باید چک کرد که Exp type سمت راست یک تایپ به ارث رسیده از سمت چپ باشد (بکمک تابع conform این موضوع را چک میکنیم)
  در صورت valid بودن، return type تایپ عبارت سمت راست است.

- Static Dispatch : کلاسی که متد آن صدا زده می‌شود باید قبلاً تعریف شده باشد.
  متد مورد نظر باید یک متد valid در کلاس فوق باشد.
  مقدار formal های متد صدا زده شده باید برابر با متد اصلی باشد.
  تایپِ exp باید یک زیرمجموعه از کلاس type_name باشد.
  درنهایت تایپ برگشتی برابر با تایپ exp است.

- Dispatch : اول چک کنیم که در کلاس درست dispatch را انجام داده ایم.
  سپس چک میکنیم که متدِ صدا شده وجود داشته باشد.
  اگر مشکلی نبود چک کنیم تایپ های داده شده به متد زیرمجموعه ای از تایپ هایی باشند که متد میتواند بگیرد و همچنین چک کنیم تعداد argument های داده شده به متد درست باشد.
  تایپ برگشتی بربر با تایپ برگشتی متد است و اگر self_type باشد برابر با تایپ e0.

- Condition : بصورت if pred then <then_exp> else <else_exp> fi
اول چک کنیم که if predicate از تایپ bool باشد.
تایپ بازگشتی کوچکترین جد مشترک then_exp و else_exp است.

- Loop : تایپ predicate باید Bool باشد.
سپس تایپ بازگشتی را ز نوع object قرار میدهیم و با loop exp کاری نداریم.

- Case : چک میکنیم که اولین exp در کلاس فعلی تعریف شده باشد.
هر یک از شاخه‌های دیگر باید تایپ معتبر داشته باشد و بعد از هر تایپ چکنیگ object environment را برای تایپ چکنیگ بعدی تغییر میدهیم.
استیتمنت دو شاخه نمیتواند تایپ یکسان داشته باشد.
در هر شاخه return type باید تایپی کوچکتر از declared type شاخه باشد.
و return type نهایی کوچکترین جد مشترک return type شاخه ها است.

- Block : بصورت <e0 , … , en> است.
هر یک از exp ها در body موجود ارزیابی میشود.
تایپ بازگشتی برابر با تایت آخرین exp است.

- Let : باید exp اولیه ارزیابی شود و به scope جدید وارد شود.
سپس type decleration و assigned exp باید یک زیر کلاس valid از تایپ مشخص شده باشند.
تایپ برگشتی عبارت برابرbody است.

- Operator : عملیات + , - , * , / : شامل دو exp است: e1 , e2 و باید چک کنیم که تایپ این دو برابر با int باشد. اگر نبود ارور برمیگردانیم. سپس تایپ برگشتی برابر int است
برای > و >= و = دو طرف باید تایپ int داشته باشند و تایپ بازگشتی برابر با Bool است.
برای ~ باید چک کنیم که exp (فقط یک exp داریم) تایپ int دارد و تایپ بازگشتی برابر int میشود.

- Equality : عملیات = , < , <= , not شامل دو exp است و تایپ هر دو باید برابر باشد.
این تایپ ها میتوانند bool یا string یا int باشدو
در < و <= فقط میتوان تایپ int داشت.
اگر ارور نداشته باشیم تایپ بازگشتی برابر با bool است وگرنه برابر با object است.

- Complement : باید تایپ exp برابر با bool باشد و تایپ بازگشتی هم bool است.

- New : چک مکنیم که تایپی که به کلاس جدید میدهیم تعریف شده باشد (یا self_type باشد). اگر نبود ارور میدهیم.
تایپ بازگشتی برابر با تایپی است که تخصیص داده ایم(یک کلاس یا self_type)

- isvoid : باید چک کرد که exp در کلاس فعلی تایپ تعریف شده داشته باشد.
سپس تایپ بازگشتی برابر با bool است.

چند مثال بد:

چرخه:

bad1.cl

File Edit Search Options Help

```
1 (*   bad example for inheritance cycle  *)
2 class A inherits B {};
3 class B inherits C {};
4 class C inherits A {};
5
6
7 class Main inherits IO {
8    main(): SELF_TYPE {
9         out_string("Hello, World.\n")
10   };
11 };
12
```

compilers@compilers-vm: ~/cool/cool-compiler-master/assignments/PA4

File Edit Tabs Help

```
compilers@compilers-vm:~/cool/cool-compiler-master/assignments/PA4$ ./mysemant
bad1.cl
bad1.cl:2: Class A, or an ancestor of A, is involved in an inheritance cycle.
bad1.cl:3: Class B, or an ancestor of B, is involved in an inheritance cycle.
bad1.cl:4: Class C, or an ancestor of C, is involved in an inheritance cycle.
Compilation halted due to static semantic errors.
compilers@compilers-vm:~/cool/cool-compiler-master/assignments/PA4$
```

کلاس تعریف نشده:

bad2.cl

File Edit Search Options Help

```
1 class A inherits B {};
2
3 class Main inherits IO {
4    main(): SELF_TYPE {
5         out_string("Hello, World.\n")
6    };
7 };
8
```

compilers@compilers-vm: ~/cool/cool-compiler-master/assignments/PA4

File Edit Tabs Help

```
compilers@compilers-vm:~/cool/cool-compiler-master/assignments/PA4$ ./mysemant
bad2.cl
bad2.cl:1: Class A inherits from an undefined class B.
Compilation halted due to static semantic errors.
compilers@compilers-vm:~/cool/cool-compiler-master/assignments/PA4$
```

کلاس اشتباه:

bad3.cl

File Edit Search Options Help

```
1 class A {
2         var : Int <- 0;
3         var : Str <- "hi";
4 };
5
6
7 class Main inherits IO {
8    main(): SELF_TYPE {
9                 out_string("Hello, Worldn")
10   };
11 };
```

compilers@compilers-vm: ~/cool/cool-compiler-master/assignments/PA4

File Edit Tabs Help

```
compilers@compilers-vm:~/cool/cool-compiler-master/assignments/PA4$ ./mysemant
bad3.cl
bad3.cl:3: Class Str of attribute var is undefined.
Compilation halted due to static semantic errors.
compilers@compilers-vm:~/cool/cool-compiler-master/assignments/PA4$
```