۲۹ مهر

source code → **lexi analyzer →** lexi tokens → **syntax analyzer** → parse three → **semantic analyzer →** intermediate code → optimizec code → **code generation →** object code → **linking** → final result

مراحل کامپایل

Bootstrrapping (Bootstrapped compiler): کامپایلرهایی‌اند که وقتی زبان جدید تعریف می‌کنند کامپایلرش را هم با آن می‌نویسند. مثل پاسکال

روش‌های راه‌اندازی این نوع کامپایلرها: *روش دستی: خود کدنویس‌ها، دستی اجرا می‌کنند.

*استفاده از زبان‌های دیگر برای اجرای کامپایلر .

اولیه‌ی زبان جدید

فصل پنجم: انواع داده‌ی اولیه (primitive data types):

داده‌ها به دو دسته تقسیم می‌شوند:

۱ – نوع داده‌ی اولیه: یک value یک کاراکتر دارد مثل int و float که به صورت پیش‌فرض تعریف شده‌اند. یک واحد دارد

۲– ساختمان (struct): چند فیلد مختلف دارد که باید مقداردهی شوند. مثل آرایه‌ها، کلاس‌ها.

هر نوع داده‌ای می‌تواند bindingهای مختلفی داشته باشد.

*اولین صفت برای نوع داده، نام آن است. * هر نوع داده مقداری دارد که یک واحد است. * به هر متغیری یک مقدار مشخص اختصاص داده می‌شود. * هر متغیر اسم دارد. البته ممکن است متغیری اسم نداشته باشد. * متغیرهای با مقدار ثابت.

انواع ثابت‌ها:

۱ – literal: رشته، عدد

۲– تعریف شده توسط برنامه‌نویس (و اختصاص یک اسم به آن‌ها)

#define a ۱۲ (ماکرو)

ماکروها الزاماً متغیر نیستند. Define یک دستور است که هر جا a می‌بیند به جای آن ۱۲ می‌گذارد.

constant int a = ۱۲

برای a صفت است و اجازه نمی‌دهد مقدار آن تغییر کند.

طول عمر متغیر:

* متغیر که در زیربرنامه تعریف می‌شود، به محض اتمام زیربرنامه عمرش تمام می‌شود.

* اگر global باشد در تمام طول اجرای برنامه عمر دارد. البته بعد از exit شدن آن هم عمرش تمام می‌شود.

* اگر داده بخواهد ماندگار باشد باید موجودیتی خارجی مثل فایل یا DB ثبت شود.

انواع داده‌ی اولیه مثل int و float و .. اگر نوع آن‌ها در زبان معین نباشد هنگام برنامه‌نویسی، در هنگام runtime انقیادها انجام می‌شود.

اما اگر در زمان برنامه‌نویسی نوع داده‌ها معلوم باشد، انقیاد نوع متغیر در زمان compile انجام می‌شود.

ممکن است زبان/کامپایلر برای مثال فقط int را به طور پیش‌فرض داشته باشد و bool و float و ... را خودمان باید تعریف کنیم.

Float = int.int (قسمت اعشاری.قسمت صحیح)

اگر کامپایلر ما بیشتر bindingها را زمان اجرا انجام دهد، یعنی تعریف نوع‌ها در آن اجباری نباشد، باید جدولی داشته باشد تا هر لحظه مقادیر و نوع آن‌ها را نگه دارد. > بردار خصیصه

در این مدل، نوع را برنامه‌نویس مستقیماً تعریف نمی‌کند. در runtime مشخص می‌شود.

اگر زبان ما bindingهایش را از قبل انجام داده باشد، نوع مشخص بوده و در symbol table هنگام ایجاد زبان ذخیره شده است.

اعلان عملیات

float Area (float W, float L)

نوع خروجی ، اسم تابع، نوع و ترتیب و اسم alias ورودی‌های یک تابع را اعلان عملیات آن می‌گویند

اهداف اعلان:

۱ – انتخاب نمایش حافظه: میزان حافظه‌ی مورد نیاز فیزیکی برای هر حافظه: نوع و نمایش آن

۲–مدیریت حافظه:

اولاً چون متغیرها برای زیربرنامه هستند عمرشان به اندازه عمر زیربرنامه است. مدیریت فیزیکی بهتر حافظه، کنار هم قرار دادن حافظه‌های مورد نیاز در –یک بلوک حافظه، هنگام آزاد کردن آن‌ها راحت‌تر آزاد می‌کند یا عوض می‌کند. چون کنار هم هستند.

–هرم یا heap: وقتی که یک اشاره‌گر تعریف می‌شود روی هیپ قرار می‌گیرد، وقتی که کار زیربرنامه تمام شود پاک نمی‌شود. خود برنامه‌نویس باید حواسش باشد. بعضی زبان‌ها مثل java از gc برای این کار استفاده می‌کنند.

```
function (…., …) {
int a,b;
float c;
.
.
.
}
```

۳– عملیات چندریختی: مانند یک جمع که دو عدد int را با هم جمع می‌کند و مانند یک جمع که دو float را با هم جمع می‌کند.

```
Int add(int, int);
float (float, float);
```

کنترل نوع:

گفتیم که اگر زبان تعریف نوع اجباری نباشد هنکام runtime نوع‌ها را می‌فهمد. اگر باشد هنگام compile می‌فهمد.

باید کنترل نوع توسط زبان انجام شود، تا خطاهایی مانند جمع اشتباه int و float به عنوان int و int انجام نگیرد.

کنترل نوع دو نوع است:

۱ – ایستا: در هنگام کامپایل کنترل نوع انجام می‌شود و در هنگام برنامه‌نویسی توسط برنامه‌نویس نوع متغیرها اعلام شده است. چه در تعریف تابع‌ها، چه در تعریف متغیرها و چه در جاهای دیگر مانند:

```
int fun (int a, int b);
int a = ۵;
float b = ۶.۵;
func(a, b)
```

در مثال بالا کنترل نوع ایستا در هنگام کامپایل خطا می‌دهد.

۲– کنترل نوع پویا: در هنگام اجرا کنترل نوع انجام می شود.

```
 fun (int a, int b);
 a = ۵;
 b = ۶.۵;
func(a, b)
```

در این مثال هنگام اجرا باید کنترل نوع انجام شود.
ایستا سریع‌تر از پویا است. ایستا قابلیت خطاگیری راحت‌تر نسبت به پویا دارد. پویا انعطاف بیشتری از ایستا دارد.
در ایستا چون نوع معلوم هست، نیازی به نگه‌داری نوع‌ها نیست. اما در پویا جدولی جدا برای نگه‌داری نوع متغیرها نیاز است که در نتیجه حافظه‌ی بیشتری از ایستا مصرف می‌کند.
اشکال‌زدایی ایستا راحت‌تر است. منعطف بودن کنترل نوع پویا باعث به وجود خطاهایی نیز می‌شود. مانند مثال زیر:

```
int index = ۵;
indx = ۶;
no error in dynamic type checking, compile error in static type checking
```

در ضمن اگر قسمتی از کد قابل دسترسی نباشد، در کنترل نوع پویا، کنترل نمی‌شود. درواقع پوشش‌دهی کد پایین‌تر است. اما در ایستا همه‌ی کد چک می‌شود.

Strongly typed:

تمام خطاهای مربوط به نوع در زمان کامپایل کشف شوند، آن‌گاه زبان از نظر نوع قوی است. اگر زبانی دارای کنترل نوع ایستای محض باشد، آن‌گاه حتماً از نظر نوع قوی است.

امنیت نوع:

```
int add(int, int);
```

زبانی دارای امنیت نوع هست که وقتی بر اساس دامنه‌ی ورودی مقدار می‌گیرد خروجی آن نیز اساس برد خروجی تعریف شده باشد. در دستور بالا اگر ورودی‌ها هر دو max_int باشند و خروجی امنیت نوع نداشته باشد، long–int نتیجه می‌شود. زبان دارای امنیت نوع از این جلوگیری می‌کند.

استنتاج نوع:

مانند زبان ML

```
fun Area (int Length, int Width) int = Length*width
    fun Area (int Length, Width) = Length*width
    fun Area (Length, int Width) = Length*width
    fun Area (Length, Width) int = Length*width
```

در زبان ML تمامی موارد بالا با هم برابرند. یعنی خود زبان استنتاج می‌کند بر اساس ورودی‌ها یا خروجی‌ها بقیه‌ی متغیرهای موجود در signature باید چه نوعی باشند.
اما این مثال مجاز نیست:

fun Area (Length, Width) = Length*width

چون هیچ نوعی داده نشده تا از آن استنتاج شده و بقیه‌ی نوع‌ها به دست آید.

تبدیل نوع:

وقتی کنترل نوع انجام می‌شود ولی نتیجه می‌گیرد که نوع آرگومان‌های تابع با پارامترهای فراخوانی شده هماهنگی ندارد:

۱ – بعضی از زبان‌ها خطا می‌دهند.

۲ – بعضی دیگر تبدیل نوع انجام می‌دهند و به اجرا ادامه می‌دهند.

انواع تبدیل نوع:

۱ – صریح: واضح نوع تبدیل بیان شده است: int round(float) در اینجا صراحتاً int به float تبدیل شده است. یا int b= (int)a

۲ – ضمنی: برنامه‌نویس مستقیماً آن را فراخوانی نمی‌کند. برای مثال:

```
float a = ۵.۵;
int b = ۶;
float c= a + b;
```

در مثال بالا که زبان c است، به صورت ضمنی ابتدا b به float تبدیل می‌شود و سپس با a جمع می‌شود.

بسته به نوع زبان ممکن است بتوان هر نوعی را به نوعی دیگر تبدیل کرد، همچنین ممکن است که نتوان.

:انواع تبدیل نوع ضمنی

۱ – گسترش و ارتقاء: مثل تبدیل int به float و short به long و bool به int. در این‌ها هیچ داده‌ای از دست نمی‌رود.

۲ – محدودکننده: (narrowing) مثل تبدیل float به int وقتی که اعشار داریم.

انتساب:

assignment(=): integer۱*integer۲ → integer۳

assignment(=): integer۱*integer۲ → void

در بعضی زبان‌ها بعد از عمل انتساب، خروجی برمی‌گردد. یعنی خود عمل انتساب، یک نتیجه برمی‌گرداند که می‌توان آن را به چیز دیگر منتسب کرد، مثل زبان c

```
c = a = b
```

در بعضی دیگر این‌چنین نیست:

```
X c = a = b
A = B
```

rvalue(B) در lvalue(A) قرار می‌گیرد. به آدرس حافظه lvalue و به مقدار متغیر rvalue می‌گوییم.

در اشاره‌گر مقدار rvalue خودش از جنس lvalue است. یعنی خود مقدار آن‌ها، آدرس حافظه است.

تساوی و هم‌ارزی:

x = ۲ + ۳

۱) x = ۵

۲) x = c(۲ + ۳)

اگر کنترل نوع ایستا باشد، بر اساس نوع x می‌فهمد که باید نتیجه را در x ریخت یا خود دستور را، ولی اگر کنترل نوع پویا باشد زبان دچار ابهام می‌شود.

برای مثال در prolog عملگر is ابتدا نتیجه را حساب می‌کند بعد انتساب را انجام می‌دهد. اما = به معنای انتساب دستور است:

۱) ; x is ۲ + ۳; x =۵ → true

۲) x = ۲ + ۳; x = ۵ → false