به نام خدا

اصول طراحی کامپایلر - شرح پروژه
دانشگاه اصفهان

مقدمه

در این پروژه قصد داریم برای یک زبان برنامه‌نویسی به نام Trust یک کامپایلر طراحی و پیاده‌سازی کنیم. زبان Trust یک زبان دستوری شبیه به زبان Rust است. یک کامپایلر از تحلیل‌گر لغوی، تحلیل‌گر نحوی، تحلیل‌گر معنایی، تولیدکننده و بهینه‌ساز کد میانی، و تولیدکننده و بهینه‌ساز کد اسمبلی تشکیل شده است. یک کامپایلر کامل، یک برنامه نوشته شده در یک زبان برنامه‌نویسی را از یک فایل متنی دریافت کرده، معادل کد اسمبلی آن را تولید می‌کند. کامپایلر طراحی شده در این پروژه یک کامپایلر کامل نیست، بدین معنا که این کامپایلر کد اسمبلی تولید نمی‌کند. کامپایلر طراحی و پیاده‌سازی شده در این پروژه، یک برنامه به زبان برنامه نویسی Trust را دریافت کرده، توسط تحلیل‌گر لغوی توکن‌های آن را استخراج می‌کند، و توکن‌های استخراج شده را به تحلیل‌گر نحوی ارسال می‌کند. تحلیل‌گر نحوی، بر اساس گرامر زبان، و یک یا دو الگوریتم تجزیه (مانند تجزیه بالا به پایین و پایین به بالا)، یک درخت نحوی تولید کرده و در صورت وجود خطا، پیام خطا صادر می‌کند. در صورتی که در برنامه ورودی خطایی وجود نداشت، کامپایلر برنامه را پس از تحلیل معنایی اجرا می‌کند.

پیاده‌سازی کامپایلر توسط هر زبانی (مانند سی++، جاوا، و پایتون) می‌تواند انجام شود. تهیه یک گزارش جامع برای توضیح نحوهٔ طراحی هر یک از قسمت‌های کامپایلر الزامی است. در طراحی و پیاده‌سازی کامپایلر دانشجویان می‌توانند از هر منبعی استفاده کنند، اما باید منبع را حتما ذکر کنند. در صورت استفاده از موتورهای هوش مصنوعی، لازم است منبعی که موتور هوش مصنوعی استفاده کرده است ذکر شود. برای مثال بینگ، منابعی را که از آن اطلاعات استخراج می‌کند ذکر می‌کند.

۱ تحلیل‌گر لغوی

در این قسمت از پروژه می‌خواهیم یک تحلیل‌گر لغوی برای زبان Trust طراحی و پیاده‌سازی کنیم. تحلیل‌گر لغوی یک فایل حاوی یک برنامه را دریافت می‌کند. در صورتی که برنامه حاوی توکن‌های معتبر در زبان Trust باشد، دنباله‌ای از توکن‌های تشخیص داده شده چاپ می‌شوند، و در غیر اینصورت پیام خطا چاپ می‌شود. واژه‌های زبان حساس به حروف کوچک و بزرگ [1] هستند، بنابراین X و x دو کلمه متفات‌اند.

۱.۱ کلمات کلیدی

کلمات کلیدی زبان با حروف کوچک نوشته می‌شوند که در زیر ذکر شده‌اند.

```
bool   break   continue   else   false     fn      i32
if     let     loop       mut    println!  return  true
```

۲.۱ شناسه‌ها

یک شناسه نامی برای یک موجودیت در یک زبان برنامه‌نویسی است. دو موجودیت در این زبان برنامه‌نویسی عبارتند از متغیر و تابع. یک متغیر موجودیتی است که یک یا دنباله‌ای از خانه‌های حافظه را اشغال می‌کند و دارای یک نام است. یک تابع موجودیتی است که تعدادی ورودی دریافت می‌کند، محاسباتی را انجام می‌دهد و در برخی موارد یک مقدار بازمی‌گرداند. یک تابع با یک نام مشخص می‌شود.

شناسه‌ها با یک حرف یا علامت زیرخط [2] _ آغاز می‌شوند و می‌توانند حاوی ارقام، حروف و علامت‌های زیرخط باشند. شناسه‌ها نمی‌توانند برابر با هیچ‌یک از کلمات کلیدی باشند.

۳.۱ علامت‌های نشانه‌گذاری

علامت‌های نشانه‌گذاری در این زبان به شرح زیر هستند.

[1] case-sensitive
[2] underline

- علامت‌های آکولاد باز } و آکولاد بسته { در تعریف بلوک‌ها استفاده می‌شوند.
- علامت‌های پرانتز باز) و پرانتز بسته (در تعریف توابع، فراخوانی توابع، و عبارت‌های محاسباتی استفاده می‌شوند.
- علامت‌های کروشه باز] و کروشه بسته [برای تعریف آرایه‌های استفاده می‌شوند.
- علامت ویرگول , برای جدا کردن ورودی‌های تابع از یکدیگر در تعریف و فراخوانی تابع استفاده می‌شود.
- علامت نقطه‌ویرگول ; در پایان تعریف متغیرها، دستورات محاسبه‌ای و فراخوانی توابع، و همچنین در تعریف حلقه‌ها استفاده می‌شوند.

۴.۱ توضیحات

توضیحات [3] با دو علامت اسلش [4] یا خط اریب // آغاز می‌شوند و با کاراکتر پایان خط معادل کد اسکی ۱۰ یا \n پایان می‌یابند. تحلیل‌گر لغوی توضیحات را به تحلیل‌گر نحوی ارسال نمی‌کند، اما پس از تحلیل لغات لیست همهٔ توکن‌ها را چاپ می‌کند.

۵.۱ مقادیر عددی

مقادیر عددی می‌توانند در مبنای ده (دهدهی یا دسیمال) [5] یا در مبنای شانزده (شانزده‌شانزدهی یا هگزادسیمال) [6] باشند. یک عدد دهدهی می‌تواند مثبت یا منفی باشد که در صورت منفی بودن با علامت - آغاز می‌شوند. اعداد هگزادسیمال با دو کاراکتر 0x آغاز می‌شوند.

۶.۱ رشته‌های ثابت

یک رشتهٔ ثابت را با دنباله‌ای از کاراکترها که در بین دو علامت نقل قول " " قرار گرفته‌اند، نشان می‌دهیم. برای نشان دادن علامت نقل قول در یک رشته ثابت از "\ استفاده می‌شود.

۷.۱ عملگرها

در یک عبارت می‌توان از عملگرهای حسابی [7] مانند + ، - ، * ، / برای جمع، تفریق، ضرب، و تقسیم، استفاده کرد. برای به دست آوردن باقیمانده از عملگر % استفاده می‌شود. عملگرهای یگانی + و - برای تعیین مثبت و منفی بودن اعداد به کار می‌روند.

عملگرهای یگانی + و - بالاترین اولویت را دارند و پس از آنها * ، / ، % هم اولویت بوده و در درجهٔ دوم اولویت قرار دارند و در نهایت + و - اولویت سوم قرار می‌گیرند.

عملگرهای رابطه‌ای [8] < ، =< ، > ، => ، == و =! برای مقایسه دو مقدار به کار می‌روند. عملگر < مقدار درست را بازمی‌گرداند اگر عملوند اول از عملوند دوم بزرگتر باشد. به همین ترتیب عملگرهای بعدی مقدار درست را بازمی‌گردانند اگر عملوند اول آنها از عملوند دوم بزرگتر یا مساوی، کوچکتر، کوچکتر یا مساوی، مساوی، نامساوی باشد. عملگرهای رابطه‌ای نسبت به عملگرهای حسابی اولویت کمتری دارند.

عملگرهای منطقی [9] عطف && و فصل || و نقیض ! نیز در عبارات منطقی به کار می‌روند. یک عبارت منطقی عبارتی است که از متغیرهای منطقی و عملگرهای منطقی تشکیل شده و مقدار آن درست یا نادرست است. اولویت عملگرهای منطقی از عملگرهای حسابی و رابطه‌ای کمتر است.

عملگر انتساب = برای مقداردهی یک متغیر به کار می‌رود و اولویت آن از عملگرهای حسابی، مقایسه‌ای، و منطقی کمتر است.

عملگر : برای تعیین نوع یک متغیر در تعریف آن به کار می‌رود.

عملگر <- برای تعیین نوع مقدار بازگردانده شده توسط یک تابع به کار می‌رود.

۸.۱ فاصله‌های خالی

توکن‌ها توسط یک فاصلهٔ خالی [10] و یا ترکیبی از فاصله‌های خالی از یکدیگر جدا می‌شوند. یک فاصله خالی شامل کاراکتر فاصله با کد اسکی ۳۲ ، کاراکتر خط جدید با کد اسکی ۱۰ ، و کاراکتر ستون جدید با کد اسکی ۹ می‌شود.

[3] comments
[4] slash
[5] decimal
[6] hexadecimal
[7] arithmetic operators
[8] relational operators
[9] logical operator
[10] whitespace

۹.۱ لیست توکن‌ها

در جدول زیر توکن‌های زبان Trust به همراه نام توکن‌ها ذکر شده اند. خروجی تحلیل‌گر لغوی دنباله‌ای از نام توکن‌های یک برنامهٔ ورودی است.

	Lexeme	Token		Lexeme	Token
۱	`bool`	T_Bool	۲۴	`!=`	T_ROp_NE
۲	`break`	T_Break	۲۵	`==`	T_ROp_E
۳	`continue`	T_Continue	۲۶	`&&`	T_LOp_AND
۴	`else`	T_Else	۲۷	`||`	T_LOp_OR
۵	`false`	T_False	۲۸	`!`	T_LOp_NOT
۶	`fn`	T_Fn	۲۹	`=`	T_Assign
۷	`i32`	T_Int	۳۰	`(`	T_LP
۸	`if`	T_If	۳۱	`)`	T_RP
۹	`let`	T_Let	۳۲	`{`	T_LC
۱۰	`loop`	T_Loop	۳۳	`}`	T_RC
۱۱	`mut`	T_Mut	۳۴	`[`	T_LB
۱۲	`println!`	T_Print	۳۵	`]`	T_RB
۱۳	`return`	T_Return	۳۶	`;`	T_Semicolon
۱۴	`true`	T_True	۳۷	`,`	T_Comma
۱۵	`+`	T_AOp_Trust	۳۸	`:`	T_Colon
۱۶	`-`	T_AOp_MN	۳۹	`->`	T_Arrow
۱۷	`*`	T_AOp_ML	۴۰	variable or function names	T_Id
۱۸	`/`	T_AOp_DV	۴۱	decimal integers	T_Decimal
۱۹	`%`	T_AOp_RM	۴۲	hexadecimal integers	T_Hexadecimal
۲۰	`<`	T_ROp_L	۴۳	constant strings `"[string]"`	T_String
۲۱	`>`	T_ROp_G	۴۴	`//[string]\n`	T_Comment
۲۲	`<=`	T_ROp_LE	۴۵	whitespace (newline، tab، and space characters)	T_Whitespace
۲۳	`>=`	T_ROp_GE			

۲ تحلیل‌گر نحوی

در این قسمت از پروژه می‌خواهیم یک تحلیل‌گر نحوی برای زبان Trust طراحی و پیاده‌سازی کنیم. تحلیل‌گر نحوی دنباله‌ای از توکن‌ها را، که توسط تحلیل‌گر لغوی از یک فایل حاوی برنامه‌ای به زبان Trust استخراج شده‌اند، دریافت می‌کند. در صورتی که برنامه فاقد خطای نحوی باشد، درخت نحوی را در خروجی چاپ می‌کند، در غیر اینصورت خطاهای برنامه را گزارش می‌کند. ساده‌ترین روش برای پیاده‌سازی تحلیل‌گر نحوی استفاده از تجزیه‌کنندهٔ پیش‌بینی کننده است. در صورتی که ورودی با خطا روبرو شد، کامپایلر باید با استفاده از روش‌هایی بازیابی خطا انجام دهد.

۱.۲ متغیرها و نوع‌های داده‌ای

دو نوع دادهٔ اصلی در این زبان وجود دارند. نوع داده‌های صحیح و منطقی که با i32 و bool نمایش داده می‌شوند. برای تعریف یک متغیر غیرقابل تغییر از کلمهٔ کلیدی let استفاده می‌شود. برای تعریف یک متغیر قابل تغییر از کلمهٔ کلیدی mul let استفاده می‌شود. همچنین برای تعریف چند متغیر به صورت همزمان از علامت پرانتز باز و بسته استفاده می‌شود. نوع متغیر را با استفاده از علامت دو نقطه می‌توان مشخص کرد. همچنین می‌توان توسط عملگر براکت باز و بسته آرایه تعریف کرد. یک آرایه متغیری است که دنباله‌ای محدود از خانه‌های حافظه را توسط یک نام معین توصیف می‌کند. اندازهٔ آرایه با یک عدد صحیح ثابت تعیین می‌شود. برای تعریف یک متغیر، نوع متغیر و نام آن مشخص می‌شوند. همچنین متغیرها می‌توانند در هنگام تعریف با استفاده از عملگر انتساب مقداردهی شوند. در زیر چند نمونه تعریف متغیر نشان داده شده است.

```
let x1;
x1 = 10;
let x2: i32;
let x3 = 12;
let x4: i32 = 5;
let b: bool = true;

let (x, b) : (i32, bool);
let (x, y) = (1, 2);
let (x, y, z) : (i32, bool, bool) = (1, true, false);

let array1 = [1, 2, 3, 4, 5];
let mut array2 : [i32; 2] = [10, 20];
array2[0] = 100;
let mut booleans : [bool; 4] = [true, true, true, false];
```

۲.۲ دستورات شرطی

یک دستور شرطی برای بررسی یک شرط و اجرای پاره‌ای از دستورات در صورت برقراری شرط استفاده می‌شود. در یک دستور شرطی از کلمات کلیدی if و else استفاده می‌شود. در صورتی که شرط متعلق به if برقرار باشد، دستورات متعلق به آن که در بلوک متعلق به if با استفاده از علامت‌های آکولاد باز و بسته مشخص شده‌اند، اجرا می‌شوند. در صورتی که شرط برقرار نباشد، دستورات متعلق به بلوک if اجرا نمی‌شوند. در صورتی که بخواهیم هنگامی که شرط برقرار نیست، دستوراتی را اجرا کنیم می‌توانیم از کلیدواژهٔ else و بلوک متعلق به آن استفاده کنیم. امکان تعریف if و else های تودرتو نیز وجود دارد، بدین معنی که شرطی بررسی شده، در صورت برقراری شرط دستوراتی اجرا می‌شوند و در صورتی که شرط برقرار نباشد و شرط دیگری برقرار باشد، پاره‌ای دیگر از دستورات اجرا می‌شوند. در زیر چند نمونه دستورات شرطی تعریف شده‌اند.

```
if x == 1 {
    y = 6;
    z = 7;
}
if y <= 2 && z > 7 || w != 9 {
   x = 4 + y;
} else {
```

```
    x = 3 * z;
}
if x >= 5 || z < 7 && !b {
    b = true;
} else if x == 3 {
    array[4] = 43;
} else {
    c = true;
}
```

۳.۲ حلقه‌ها

یک حلقه برای تکرار تعدادی دستورات استفاده می‌شود. در یک حلقه از کلمهٔ کلیدی `loop` استفاده می‌شود. درون حلقه می‌توان از دستورات `break` و `continue` استفاده کرد. اگر از دستور `break` در یک حلقه استفاده شود، تکرار حلقه متوقف می‌شود و کنترل برنامه از حلقه خارج می‌شود و از اگر دستور `continue` استفاده شود، دستورات بعد از آن اجرا نمی‌شوند و کنترل برنامه به ابتدای حلقه باز می‌گردد. در زیر چند نمونه حلقه تعریف شده اند.

```
loop {
    x = x + 1;
    if x == 10 {
        break;
    }
}
loop {
    if x % 7 == 0 {
        continue;
    }
    s = s + x;
    x = x + 1;
}
```

۴.۲ توابع

یک تابع با استفاده از کلمهٔ کلیدی `fn` تعریف می‌شود و پس از آن نام تابع و پارامترهای ورودی تابع و نوع آنها مشخص می‌شوند. یک تابع شامل دسته‌ای از دستورات است که در بلوک تابع بین دو علامت آکولاد باز و بسته نوشته می‌شوند. یک تابع می‌تواند یک آرایه بازگرداند. دستور `return` یک مقدار را از تابع باز می‌گرداند. در زیر چند نمونه تابع تعریف شده‌اند.

```
fn multiply(x: i32, y: i32) -> i32 {
    let z = x*y;
    return z;
}
fn add(x, y) {
    return x + y;
}
fn create_array() -> [i32; 5] {
    let array = [1, 2, 3, 4, 5];
    return array;
}
fn main() {
```

```
    let r = multiply(3, 4);
}
```

۵.۲ دستور چاپ

از دستور چاپ برای چاپ مقادیر عددی و رشته‌ها استفاده می‌شود. این دستور یک رشته، و صفر یا تعدادی متغیر را دریافت کرده، مقادیر متغیرها را در درون رشته چاپ می‌کند. رشتهٔ دریاف شده، رشته قالب‌بندی نامیده می‌شود. در درون رشته قالب‌بندی تعدادی تعیین کننده استفاده شده‌اند که برای چاپ اعداد و رشته‌ها و متغیرها استفاده می‌شوند. تعیین کننده‌ها باید با علامت آکولاد باز و بسته شده و داخل آن‌ها شماره متغیرها یا نام متغیرها نوشته شود. در زیر چند نمونه دستور چاپ نشان داده شده است.

```
println!("Trust is good")
println!("x is {} and twelve is {}", x, 12);
println!("x is {1} and twelve is {0}", 3*4, x);
println!("x is {x} and twelve is {twelve} and i am {w}.", w="happy", x=x, twelve=12);
```

۳ تحلیل‌گر معنایی

در این قسمت از پروژه می‌خواهیم یک تحلیل‌گر معنایی برای زبان Trust طراحی و پیاده‌سازی کنیم. تحلیل‌گر معنایی درخت تجزیه را که توسط تحلیل‌گر نحوی تولید شده دریافت می‌کند و با پیمایش درخت تجزیه خطاهای معنایی را گزارش می‌کند. یک تحلیل‌گر معنایی همچنین می‌تواند از درخت تجزیه [11] درخت نحوی [12] تولید کند.

قوانین زیر باید در تحلیل‌گر معنایی رعایت شوند.

- یک شناسه باید قبل از استفاده تعریف شده باشد. یک شناسه می‌تواند نام یک متغیر یا یک تابع باشد.
- هر شناسه دارای یک حوزه تعریف [13] است. یک حوزه تعریف توسط یک بلوک مشخص می‌شود و یک بلوک با آکولاد باز آغاز می‌شود و با آکولاد بسته به اتمام می‌رسد. حوزه‌های تعریف می‌توانند تو در تو باشند و در یک حوزه تعریف می‌توان یک حوزه تعریف دیگر داشت. بنابراین حوزه‌های تعریف دارای یک ساختار سلسله‌مراتبی [14] هستند. دو متغیر همنام می‌توانند در دو حوزه تعریف وجود داشته باشند، اما در یک حوزه تعریف نمی‌توان دو متغیر همنام تعریف کرد. همچنین دو تابع نمی‌توانند همنام باشند.
- نوع عملوندهای عملگرهای محاسباتی `i32` و نوع عملوندهای عملگرهای منطقی باید `bool` باشد.
- اندیس یک آرایه باید مقداری بزرگتر از صفر داشته باشند و نوع آن الزاما `i32` باشد.
- نوع عبارتی که در شرط دستور `if` قرار می‌گیرد باید `bool` باشد.
- برنامه دارای یک تابع به نام `main` است.
- در فراخوانی توابع باید نوع و تعداد آرگومان‌ها با نوع و تعداد پارامترها در تعریف تابع همخوانی داشته باشند.
- اگر نوع متغیرها مشخص نشده باشد در اولین مقداردهی نوع آنها مشخص می‌شود. اگر نوع آرگومان‌های یک تابع مشخص نشده باشد در اولین فراخوانی مشخص می‌شود و قابل تغییر نیز نیستند.
- متغیرهایی که `mut` نیستند را نمی‌توان مقدار آنها را تغییر داد.
- در عبارت‌های انتساب مقدار، نوع سمت چپ و نوع سمت راست عملگر انتساب باید یکسان باشند. در صورتی که در سمت راست عملگر انتساب فراخوانی تابع وجود داشته باشد، نوع تابع و نوع متغیر سمت چپ عملگر انتساب باید یکسان باشند.
- نوع تابع و نوع عبارتی که بعد از کلمه کلیدی `return` در تابع قرار گرفته باید یکسان باشند.

---

[11] parse tree
[12] syntax tree
[13] scope
[14] hierarchical structure

۴ تولید کد

در این قسمت از پروژه می‌خواهیم یک تولیدکننده کد برای زبان Trust طراحی و پیاده‌سازی کنیم. تولیدکننده کد درخت نحوی را که توسط تحلیل‌گر معنایی تولید شده دریافت می‌کند و با پیمایش درخت نحوی کد معادل زبان سی را تولید می‌کند.

تولیدکننده کد باید کد معادل زبان سی را به صورت یک رشته تولید کند و در انتها آن را در فایل `output.c` ذخیره کند. کد تولید شده باید شامل هدرهای لازم باشد و در انتها تابع `main` را نیز داشته باشد.

پس از تولید کد می‌توان آن را با استفاده از کامپایلر زبان سی کامپایل کرد و در صورت عدم وجود خطا، برنامه را اجرا کرد.

کد تولید شده باید به صورت زیر باشد:

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    // generated code
    return 0;
}
```