

プログラミング言語の設計とコンパイラの実装

1. 研究テーマ概要

- プログラミング言語を設計
- 言語仕様に沿ってソースコードをコンパイルすることのできるコンパイラを実装

2. 使用ソフトウェア、言語、ライブラリ

- Visual Studio 2022
- C++ 23 (MSVC /std:c++latest)
- LLVM15.0.1
- Python
- CMake
- 7Zip

3. 研究課程

< 1 学期 >

まず初めに以下の要件を満たす環境の構築を行いました。

- LLVM API を使用可能
- C++ 23 をビルドできる
- C++ 23 のコードの内容をインテリセンスが正しく解釈できる

次にコンパイラをどのように実装するか、その実装を楽にするクラス的设计を行いました。コンパイラは主に Tokenize→Parse→Transform→Code Generate という流れに沿ってコードを生成します。プロトタイプとして作成したコンパイラは Tokenize を正規表現で行い、Parse は各パース処理を関数としてコーディングし、Transform に当たる AST の作成もその関数内で行っており、Code Generate は独自の命令語を用いて自作の Virtual Machine に対しての自作アセンブリを出力するという形をとっていました。プロトタイプを作成し、実装のイメージを持ったことで、以下の課題が浮き彫りになりました。

- Tokenize, Parse, AST の作成において、共通部分が多く規模が増えるとミスが増えやすく変更が行いにくいこと

- 自作 Virtual Machine の実装までをやっていたらアセンブリ言語の設計まで行わなければならないこと
開発コストが非常に高くなること

※趣味ならば問題はないかもしれませんが、1年間という限られた期間なので他の作業に使用できる時間が大きく減ってしまうため課題としました

前者の問題を解決するために C++ の演算子オーバーロードを用いて直感的に Tokenize から AST の作成までを行えるクラスを設計しました。後者については、コード生成以降をコンパイラ基盤である LLVM に頼る選択をしました。

<2 学期>

実際に言語仕様を考え、それに応じて1学期に書いたコードを活用しながらコンパイラの実装を行いました。プログラミング言語の設計にあたっては以下のような点に重点を置いて考えました。

- ・プログラマの無駄をなくす
- ・シンプルかつ表現力のある構文
- ・読みやすいコード書ける

命名規則一つでも人や言語、組織によって違っており統一が取れないことや、読み手に意図した意味が伝わらないことがあります。そのような細かな問題にも目を向けて、かつインターネットなどで意見を参考にしながら一つ一つ言語仕様を決めました。

4. 研究成果

以下のような言語仕様を持ったプログラムを設計し、実行できるコンパイラを実装しました。

- ・コメント/範囲コメント
- ・文字列/整数/浮動小数点リテラル
- ・関数定義/呼び出し/再起関数(型推論有)
- ・Block/Named Block
- ・指定されたブロックに値を返す(return)
- ・変数/定数(命名規則によって使い分け)
- ・シャドーイング
- ・extern 宣言
- ・四則/比較演算子
- ・if 文/if 式
- ・for 文/for 式
- ・配列
- ・クラス(構造体)
- ・Uniform Function Call Syntax

図1(サンプルコード) 図2(サンプルコード)

```
//コメント
/*範囲コメント*/
extern putchar(i8)->i8;
class Class_Name{
    pub fn new(){
        var this.x=0i32;
        return this.ref();
    }
    pub fn method(){
        return this.x;
    }
}
fn recursive(i32 arg){
    return if(arg==0i32){
        return if 777i32;
    }else{
        return reucrsive(arg-1i32);
    };
}

fn main(){
    var A=recursive(10i32);
    var A=10i32.recursive();
    var a=[[0.0f,1.0f],[[0.0,1.0f]][0i32][0i32];
    var a=Class_Name::new();
    var a=namedblock{
        return namedblock 1i32+2i32*3i32;
    };
    if(0i32){
        return 0i32;
    }else if (0i32){
        return 0i32;
    }else{
        var str="text";
    }
    for(;;){
        for(void:;){
            return for;
        }
    }
    var a=for(var i=0i32;i<=10i32;i=i+1i32){
        return for 0i32;
    };
    return 0i32;
}
```

※サンプルプログラムに特に意味はなく現在の言語仕様とは異なる可能性があります。

5. まとめ、感想

各言語仕様単体で見ると、どの仕様も何かしらのプログラミング言語で使われていることが多かったです。しかし言語仕様全体で見ると、今回設計したプログラミング言語と全く同じ言語仕様のプログラミング言語は現時点では存在していないと認識しています。現在普及しているプログラミング言語やその他ソフトウェアもこのようにして試行錯誤されて編み出されていくという過程や、低レイヤについて知れて良かったと感じました。

6. 今後の課題

Tokenize から AST の作成に作成したクラスは汎用的ではありますが、LL 法によってソースコードを解析しているため、ネストした表現のコンパイルが目に見えて遅くなります。他にも、メモリや速度面でコンパイル時間を最適化できる部分があるのが課題です。言語仕様自体の課題はテンプレートやライブラリ等、現代のプログラミング言語の使用としてメジャーな機能を追加していきたいです。