Rem

もっとC言語を書きやすく

KADOKAWAドワンゴ情報工科学院 高等部1年
らずるしぇーにゃ

概要

専らLinuxに最適化されたプログラミング言語です！

- C言語の制御力はそのままに、手軽に書きやすくする
- ラップや抽象化では無く、**単純化(Simplify)**こそが目的

構文解析器からコード生成器までコンパイラは完全自作！

特徴

- System V ABI 完全準拠で GNU C Library と互換性有り
- 単純化された型システムでボイラープレートコードを削減
- 全ては**評価可能な式**で値を返し関数型っぽい表現が出来る

```
fn is_prime(n) {
    if n < 2 then return false
    if n < 4 then return true

    let i = 2
    while i < n do {
        if n % i == 0 then return false
        let i = i + 1
    }
    true
}
```

```
fn inc(n) n + 1
fn double(n) n * 2
fn compose(f, g, x) f(g(x))

fn main() {
    let x = compose(double, inc, 3)
    printf("x = %d\n", x)
}
```

関数ポインタも使える

直接 x86_64 アセンブラを出力する独自バックエンド

```
for Define(name, args, body) in &defines {
    let mut addr = 8;
    let mut prologue = String::new();
    for (idx, _arg) in args.iter().enumerate() {
        if let Some(reg) = ABI.get(idx) {
            prologue += &format!("\tmov [rbp-{addr}], {reg}\n");
        } else {
            prologue += &format!(
                "\tmov rax, [rbp+{}]\n\tmov [rbp-{addr}], rax\n",
                (idx - 4) * 8
            );
        }
        addr += 8;
    }

    ctx.local = Function::default();
    ctx.local.var = args.clone();
    let body = body.emit(ctx)?;

    output += &format!(
        "{name}:\n\tpush rbp\n\tmov rbp, rsp\n\tsub rsp, {}\n{prologue}{body}\tleave\n\tret\n\n",
        {
            let bytes = ctx.local.var.len() * 8;
            if bytes % 16 == 0 { bytes } else { bytes + 8 }
        }
    );
}
```

```
Expr::Call(callee, args) => {
    let mut pusher = String::new();
    let mut argset = String::new();
    for (idx, arg) in args.iter().rev().enumerate() {
        pusher += &format!("{}\tpush rax\n", arg.emit(ctx)?);
        if let Some(reg) = ABI.get(idx) {
            argset += &format!("\tpop {reg}\n");
        }
    }

    let pre = pusher + &argset + &callee.emit(ctx)?;
    Ok(format!("{pre}\tmov r10, rax\n\txor rax, rax\n\tcall r10\n"))
}
Expr::Variable(name) => {
    if let Some(i) = ctx.local.var.get_index_of(name) {
        Ok(format!("\tmov rax, [rbp-{}]\n", (i + 1) * 8))
    } else {
        ctx.global.func.insert(name.clone());
        Ok(format!("\tlea rax, [{name}]\n"))
    }
}
Expr::Pointer(var) => {
    if let Some(i) = ctx.local.var.get_index_of(var) {
        Ok(format!("\tlea rax, [rbp-{}]\n", (i + 1) * 8))
    } else {
        Err(format!("undefined variable: {var}"))
    }
}
```

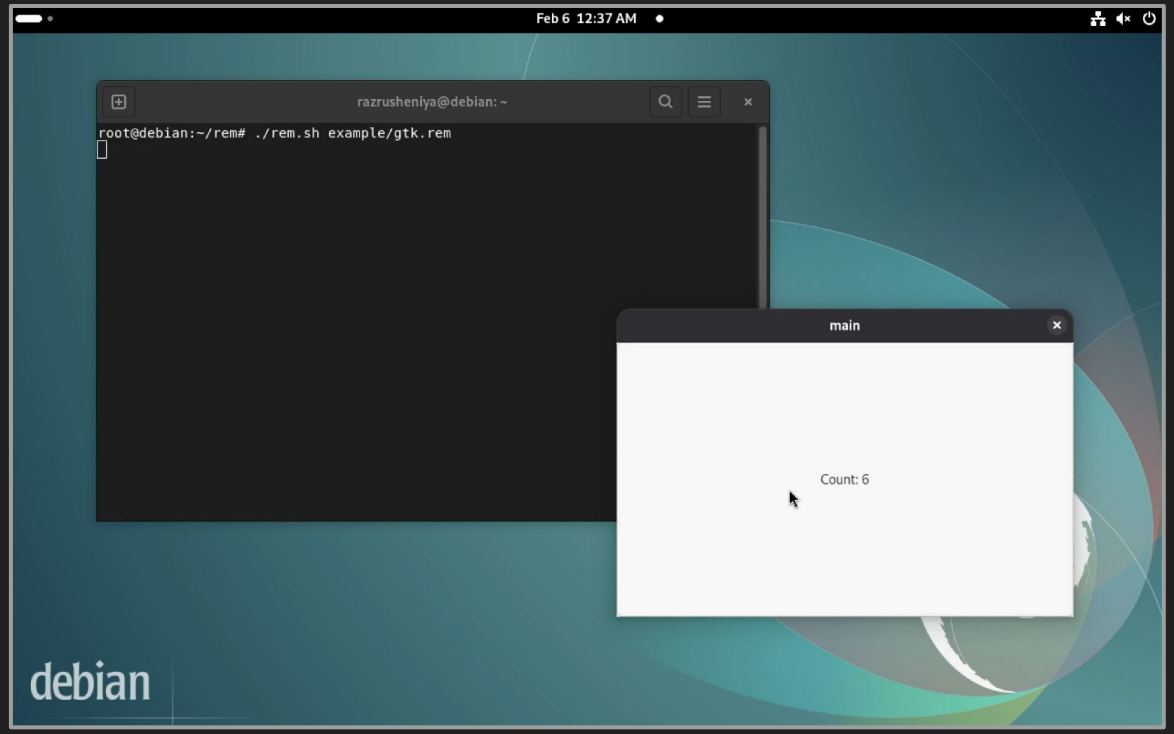
抽象構文木を構築する再帰下降パーサー

```rust
fn parse_op(source: &str) -> Result<Expr, String> {
    let tokens: Vec<String> = tokenize(source, SPACE)?;
    let n = ok!(tokens.len().checked_sub(2))?;
    let operator = ok!(tokens.get(n))?;
    let lhs = &ok!(tokens.get(..n))?.join(SPACE);
    let rhs = &ok!(tokens.get(n + 1..))?.join(SPACE);
    Ok(match operator.as_str() {
        "+" => Expr::Add(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        "-" => Expr::Sub(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        "*" => Expr::Mul(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        "/" => Expr::Div(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        "%" => Expr::Mod(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        "==" => Expr::Eql(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        "≠" => Expr::NotEq(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        ">" => Expr::Gt(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        "<" => Expr::Lt(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        "≥" => Expr::GtEq(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        "≤" => Expr::LtEq(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        "&" => Expr::And(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        "|" => Expr::Or(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        "^" => Expr::Xor(Box::new(Expr::parse(lhs)?), Box::new(Expr::parse(rhs)?)),
        op => return Err(format!("unknown operator: {op}")),
    })
}
```

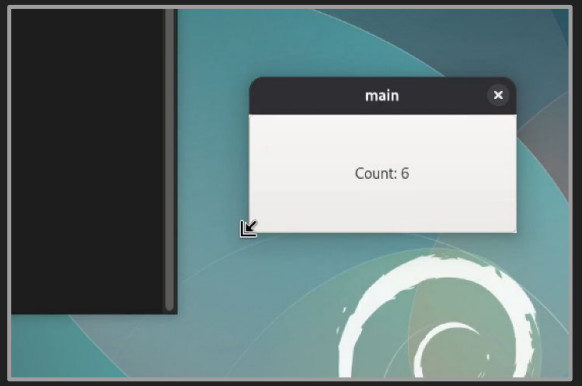
一切パーサーコンビネーターを使わない

```rust
impl Define {
    pub fn parse(source: &str) -> Result<Vec<Define>, String> {
        let mut result = Vec::new();
        for line in tokenize(source, "\n")? {
            if let Some(func) = line.strip_prefix("fn ") {
                let (head, body) = ok!(func.split_once(")"))?;
                let (name, args) = ok!(head.split_once("("))?;
                let args = tokenize(args, ",")?
                    .iter()
                    .map(|x| Name::new(x.trim()))
                    .collect::<Result<IndexSet<Name>, String>>()?;
                let body = Expr::parse(body)?;
                result.push(Define(Name::new(name)?, args, body));
            }
        }
        Ok(result)
    }
}
```

GTK+3.0を使ってRemで書いたGUIアプリ



GNOME環境のDebianで
実際に動作確認済み！！



C言語との比較: Remはどれほど単純化されたか？

```c
#include <gtk/gtk.h>
#include <stdlib.h>

static void destroy(GtkWidget *widget, gpointer data) {
    gtk_main_quit();
}

static void onclick(GtkWidget *widget, gpointer data) {
    long *count = (long *)data;
    (*count)++;

    char *label = g_strdup_printf("Count: %ld", *count);
    gtk_button_set_label(GTK_BUTTON(widget), label);
    g_free(label);
}

int main(int argc, char **argv) {
    long *count = malloc(sizeof(long));
    *count = 0;

    gtk_init(&argc, &argv);
    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_default_size(GTK_WINDOW(window), 500, 300);

    GtkWidget *button = gtk_button_new_with_label("Counter App");
    gtk_container_add(GTK_CONTAINER(window), button);

    g_signal_connect_data(button, "clicked", G_CALLBACK(onclick), count);
    g_signal_connect_object(window, "destroy", G_CALLBACK(destroy), NULL);

    gtk_widget_show_all(window);
    gtk_main();

    free(count);
    return 0;
}
```

```rem
fn destroy(widget, data) gtk_main_quit()
fn onclick(widget, data) {
    let *data = *data + 1
    let label = g_strdup_printf("Count: %d", *data)
    gtk_button_set_label(widget, label)
    g_free(label)
}

fn main(argc, argv) {
    let count = malloc(8)
    let *count = 0

    gtk_init(&argc, &argv)
    let window = gtk_window_new(0)
    gtk_window_set_default_size(window, 500, 300)

    let button = gtk_button_new_with_label("Counter App")
    gtk_container_add(window, button)

    g_signal_connect_data(button, "clicked", onclick, count)
    g_signal_connect_object(window, "destroy", destroy, 0)

    gtk_widget_show_all(window)
    gtk_main()

    free(count)
}
```

同じ意味論のコードでも
Remの方が楽に書ける！

型注釈	不要
マクロ	不要
ヘッダ読み込み	不要
ライブラリ関数	同じ
呼び出し規約	同じ

ご覧いただき、ありがとうございました！

Follow Me on GitHub!

https://github.com/razrusheniya/rem

https://deepwiki.com/razrusheniya/rem

GitHub

DeepWiki