

# 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  
らずるしえーにや

# 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

```
1 fn inc(n) n + 1
2 fn double(n) n * 2
3 fn compose(f, g, x) f(g(x))
4
5 fn main() printf("%ld\n", compose(double, inc, 3))
6
```

関数ポインタも使える

# 直接 x86\_64 アセンブリを出力する独自コンパイラ

```
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!("{}", &format!("{}tmov qword [rbp-{}addr], {}reg", "\n"));
        } else {
            prologue += &format!(
                "{}tmov rax, [rbp+{}]\n{}tmov qword [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{}\n{}body{}\n{}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!("{}", &format!("{}tpush rax", "\n"), arg.emit(ctx));
        if let Some(reg) = ABI.get(idx) {
            argset += &format!("{}", &format!("{}tpop {}reg", "\n"), reg);
        }
    }

    let pre = pusher + &argset + &callee.emit(ctx);
    Ok(&format!("{}", &format!("{}pre{}tmov r10, rax\n{}txor rax, rax\n{}tcall r10", "\n"), pre))
}

Expr::Variable(name) => {
    if let Some(i) = ctx.local.var.get_index_of(name) {
        Ok(&format!("{}", &format!("{}tmov rax, [rbp-{}]", "\n"), (i + 1) * 8))
    } else {
        ctx.global.func.insert(name.clone());
        Ok(&format!("{}", &format!("{}tlea rax, [{}name]", "\n"), name))
    }
}

Expr::Pointer(var) => {
    if let Some(i) = ctx.local.var.get_index_of(var) {
        Ok(&format!("{}", &format!("{}tlea rax, [rbp-{}]", "\n"), (i + 1) * 8))
    } else {
        Err(&format!("{}", &format!("undefined variable: {}var", "\n"), var))
    }
}
```

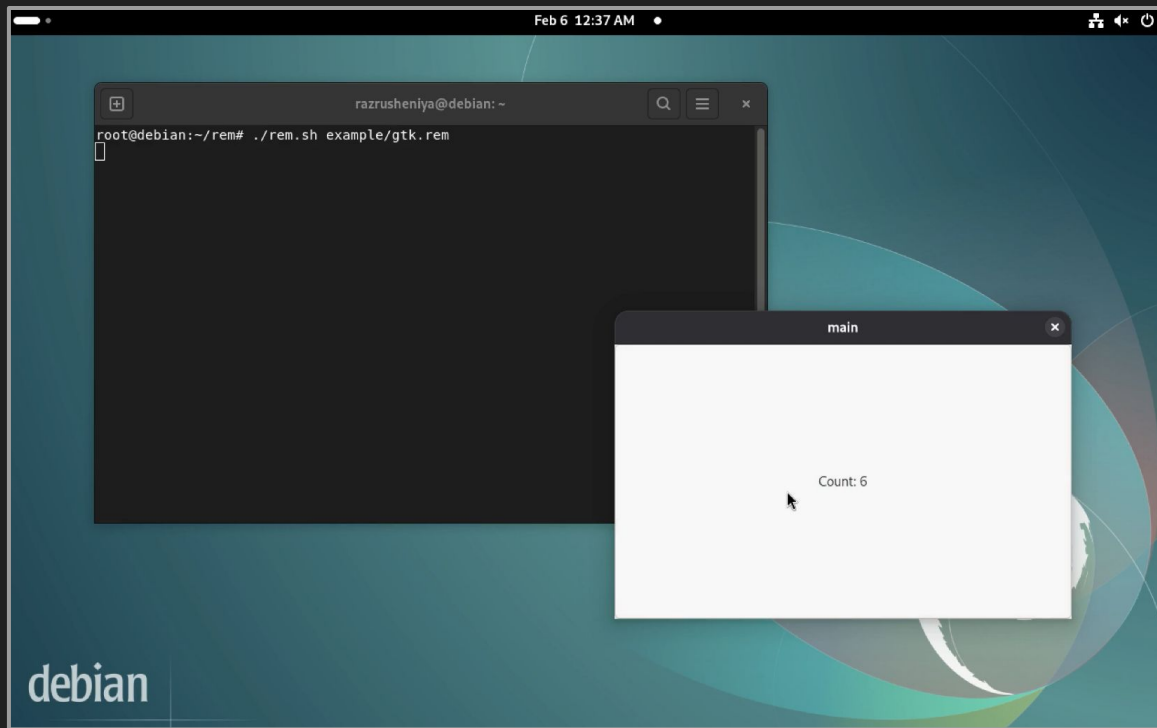
# 抽象構文木を構築する再帰下降パーサー

```
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::Eq(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)?)),  
        _ ⇒ return Err(format!("unknown operator: {op}")),  
    })  
}
```

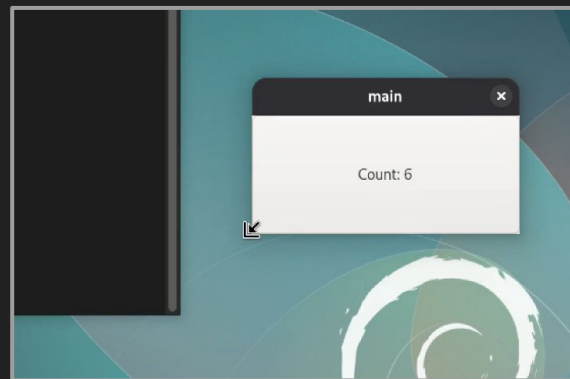
一切パーサーコンビネーターを使わない

```
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？

```
1 #include <gtk/gtk.h>
2 #include <stdlib.h>
3
4 static void destroy(GtkWidget *widget, gpointer data) {
5     gtk_main_quit();
6 }
7
8 static void onclick(GtkWidget *widget, gpointer data) {
9     long *count = (long *)data;
10    (*count)++;
11
12    char *label = g_strdup_printf("Count: %ld", *count);
13    gtk_button_set_label(GTK_BUTTON(widget), label);
14    g_free(label);
15 }
16
17 int main(int argc, char **argv) {
18     long *count = malloc(sizeof(long));
19     *count = 0;
20
21     gtk_init(&argc, &argv);
22     GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
23     gtk_window_set_default_size(GTK_WINDOW(window), 500, 300);
24
25     GtkWidget *button = gtk_button_new_with_label("Counter App");
26     gtk_container_add(GTK_CONTAINER(window), button);
27
28     g_signal_connect_data(button, "clicked", G_CALLBACK(onclick), count);
29     g_signal_connect_object(window, "destroy", G_CALLBACK(destroy), NULL);
30
31     gtk_widget_show_all(window);
32     gtk_main();
33
34     free(count);
35     return 0;
36 }
37
```

```
1 fn destroy(widget, data) gtk_main_quit()
2 fn onclick(widget, data) {
3     let *data = *data + 1
4     let label = g_strdup_printf("Count: %d", *data)
5     gtk_button_set_label(widget, label)
6     g_free(label)
7 }
8
9 fn main(argc, argv) {
10    let count = malloc(8)
11    let *count = 0
12
13    gtk_init(&argc, &argv)
14    let window = gtk_window_new(0)
15    gtk_window_set_default_size(window, 500, 300)
16
17    let button = gtk_button_new_with_label("Counter App")
18    gtk_container_add(window, button)
19
20    g_signal_connect_data(button, "clicked", onclick, count)
21    g_signal_connect_object(window, "destroy", destroy, 0)
22
23    gtk_widget_show_all(window)
24    gtk_main()
25
26    free(count)
27 }
28
```

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！

<https://github.com/razrusheniya/rem>

<https://deepwiki.com/razrusheniya/rem>



**らずるしえーにや**  
razrusheniya · разрушения

低レイヤ技術／コンパイラ開発

👤 7 followers · 0 following

📖 KADOKAWAドワンゴ情報工科学院

🦋 @razrusheniya.bsky.social

Follow Me on GitHub!

