A Linguagem de Programação DAGG

Introdução

A linguagem de programação DAGG é uma linguagem de programação estruturada, estaticamente tipada e de médio nível. A DAGG foi criada para servir de ferramenta de treino de pensamento computacional para programadores iniciantes, e tomou como inspiração linguagens como C, Golang, Python e Swift.

A DAGG procura livrar o programador do gerenciamento de estruturas dinâmicas e memória, e tem uma sintaxe leve e flexível.

O compilador da linguagem DAGG foi desenvolvida com as ferramentas Yacc e Lex.

Design da implementação

Analisador léxico

Diversos simbolos e palavras chaves são definidos no arquivo Lex, incluindo operadores aritméticos, booleanos, relacionais. Estes são auto-explicativos, e se encontram no `dagglexer.l`.

Nomes (IDs) definidos pelo programador devem seguir a seguinte regra:

- 1ª caractere: letra maiúscula/minúscula
- 2ª caractere em diante: letras maiúscula/minúscula, digitos e underscores

Literais de numeros reais (ponto flutuantes), exigem pelo menos um dígito após o ponto, mas não exige nenhum antes.

Tabela de símbolos

A tabela de símbolos foi implementada nos arquivos `table.h` e `table.c`. A tabela é composta por três colunas: nome, tipo e escopo. A estrutura de dados usada pela tabela é a lista ligada.

No mesmo arquivo, se encontra o controle de escopo, implementado através de um contador e uma pilha. A pilha guarda o valor dos escopos, e o contador guarda o valor do próximo escopo a ser empurrado na pilha.

Seguem a descrição das funções do header:

- `up()`: entra em um novo escopo filho, interno ao atual
- `down()`: sai do escopo atual e retorna ao escopo pai
- `insert()`: insere um símbolo na tabela, retorna sucesso se não há um símbolo no mesmo escopo
- `lookup()`: busca um símbolo na tabela, retorna sucesso se o símbolo foi encontrado no escopo
- `print_table()`: imprime a tabela no terminal
- `free_table()`: libera a memória da tabela

Estruturas condicionais e de repetição

As estruturas `se`, `senao` e `entao`, `para` e `enquanto` são traduzidas para C com a utilização de `goto`s e labels `_ELIF`, `_ELSE`, `_JUMP`, e `_SKIP`.

- se, senao e entao

```
a: int <- 0
se a > 0 {
    escreva "a > 0"
} senao se a < 0 {
    escreva "a < 0"
} senao {
    escreva "a = 0"
}
```

```
int a=0;
if (!(a>0)) goto _ELIF0;
{
printf("%s","a > 0");
;}
goto _SKIP0;
_ELIF0:
if (!(a<0)) goto _ELSE0;
{
printf("%s","a < 0");
;}
goto _SKIP0;
_ELSE0:
{
printf("%s","a = 0");
;}
goto _SKIP0;
_SKIP0:
;
```

- para

```
b: [[int]] <- [ [9, 8, 7], [6, 5, 4], [3, 2, 1] ]
para v em b {
    para i em v {
        escreva i
    }
}
```

```
 _Matrix* b=_mof(3,_vof(3,9,8,7),_vof(3,6,5,4),_vof(3,3,2,1));
{
int _v=0;
_LOOP1:
if (_v>=b->size) goto _JUMP1;
_Vector* v=b->data[_v];
{
{
int _i=0;
_LOOP0:
if (_i>=v->size) goto _JUMP0;
int i=v->data[_i];
{
printf("%d",i);
;}
_i++;
goto _LOOP0;
_JUMP0:
;}
;}
_v++;
goto _LOOP1;
_JUMP1:
```

- enquanto

```
 a: int <- 0
enquanto a <= 10 {
    a = a + 1
}
```

```
 int a=0;
_LOOP0:
if (!(a<=10)) goto _JUMP0;
{
a=a+1;
;}
goto _LOOP0;
_JUMP0:
```

Subprogramas

Os subprogramas contém checagens de tipo, parametros e do retorno da função. Outra funcionalidade é que não precisam necessariamente de um tipo de retorno podendo ter funções void.

```
soma(a:int, b:int): int {
    retorne a+b
}

main() {
    i: int <- soma(1, 2)
    escreva "hello world"
}
```

```
int soma(int a, int b)
{
return a+b;
;}

void main()
{
int i=soma(1,2);
printf("%s","hello world");
;}
```

Semântica estática

Sempre que possível, o código emite um erro quanto encontra um erro semântico, que vai de erros de declarações e checagens de tipo, até verificação de acesso válido a vetores e checagem de passagem de argumentos em subprogramas. Todos os erros se encontram no arquivo `errors.h`.

Temos como exemplo de erros:

```
- Variavel não Declarada
- Váriavel Já Declarada
- Não é um ID
- Erro de type
- Tipo Incompatível
- Erro Fatal
- Erro de Referencia
- Erro de Falta de Suporte
```

Instruções de uso

Gerando o compilador

Com as ferramentas `bison`, `flex` and `gcc` (ou compilador C de preferência) já instaladas no sistema Linux, execute no terminal:

```
lex dagglexer.l
yacc daggparser.y -d
gcc lex.yy.c y.tab.c table.c -o dagg
```

Executando o compilador

Para compilar um arquivo `.dagg` em um arquivo `.c` simplificado, execute no terminal:

```
./dagg SOURCE.dagg DESTINATION.c
```

Se o código em DAGG estiver, nada será impresso no terminal. Se o código não estiver correto, um erro adequado será impresso no terminal.

Também é possível executar o compilador apenas redirecionando a entrada padrão. O código DAGG compilado será impresso na tela.

```
./dagg < SOURCE.dagg
```