



**Instituto Politécnico Nacional**

**Escuela Superior de Cómputo**

**Compiladores**

**Práctica #5**

**“HOC 5”**

**3CM7**

Alumno: Zepeda Flores Alejandro de Jesús

Profesor: Tecla Parra Roberto

**INTRODUCCIÓN**

HOC es un acrónimo para **High Order Calculator**, es un lenguaje de programación interpretado que fue usado en 1984 en el libro *“El Entorno de Programación de UNIX”* para demostrar como construir interpretes usando una herramienta llamda YACC y lenguaje C.

HOC fue desarrollado por Brian Kernighan y Rob Pike como una grandiosa calculadora interactiva. Su función básica es evaluar expresiones numéricas de puntos flotantes e.g. “1+2\*sin(0.7)”. Después variables fueron agregadas, expresiones condicionales, ciclos, funciones definidas por el usuario, simple entrada/salida y más, todo esto usando una sintaxis parecida a lenguaje C.

Hasta ahora, las 6 etapas de HOC son:

* HOC1: Calculadora Básica
* HOC2: Calculadora con 26 variables
* HOC3: Calculadora Científica
* HOC4: Máquina Virtual de Pila
* **HOC5: Ciclos / Decisiones**
* HOC6: Funciones y Procedimientos

**OBJETIVO**

Con HOC4, que ya usa la Máquina Virtual de Pila, agregar ciclos while y decisiones, esto haciendo unos pequeños cambios al programa de la práctica anterior.

**DESARROLLO**

En esta práctica si modificamos el código de inicialización, le agregamos palabras clave, como: **for, while, if:**

**static** **struct** */\*Palabras clave\*/*

{

char \*name;

int kval;

}

keywords[] =

{

"\_if", IF,

"else", ELSE,

"while", WHILE,

"print", PRINT,

0, 0

};

En la máquina virtual de pila se agregaron las funciones para las condiciones, y para el ciclo while:

void whilecode()

{

Datum d;

Inst \*savepc = pc; */\*cuerpo de la iteración\*/*

execute(savepc + 2); */\*condición\*/*

d = pop();

**while** (d.val->real)

{

execute(\*((Inst \*\*)(savepc))); */\*cuerpo\*/*

execute(savepc + 2);

d = pop();

}

pc = \*((Inst \*\*)(savepc+1)); */\*siguiente proposición\*/*

}

void ifcode()

{

Datum d;

Inst \*savepc = pc; */\*parte then\*/*

execute(savepc + 3); */\* condición \*/*

d = pop();

**if**(d.val->real) execute(\*((Inst \*\*)(savepc)));

**else** **if** (\*((Inst \*\*)(savepc + 1))) */\*¿parte else?\*/*

execute(\*(( Inst \*\*) (savepc + 1)));

pc = \*((Inst \*\*)(savepc + 2)); */\*siguiente proposición\*/*

}

void gt()

{

Datum d1, d2;

d2 = pop();

d1 = pop();

d1.val->real = (double)(d1.val->real > d2.val->real);

push(d1);

}

void lt()

{

Datum d1, d2;

d2 = pop();

d1 = pop();

d1.val->real = (double)(d1.val->real < d2.val->real);

push(d1);

}

void ge()

{

Datum d1, d2;

d2 = pop();

d1 = pop();

d1.val->real = (double)(d1.val->real >= d2.val->real);

push(d1) ;

}

void le()

{

Datum d1, d2;

d2 = pop();

d1 = pop();

d1.val->real = (double)(d1.val->real <= d2.val->real);

push(d1);

}

void eq()

{

Datum d1, d2;

d2 = pop();

d1 = pop();

d1.val->real = (double)(d1.val->real == d2.val->real);

push(d1);

}

void ne()

{

Datum d1, d2;

d2 = pop();

d1 = pop();

d1.val->real = (double)(d1.val->real != d2.val->real);

push(d1);

}

void and()

{

Datum d1, d2;

d2 = pop();

d1 = pop();

d1.val->real = (double)(d1.val->real != 0.0 && d2.val->real != 0.0);

push(d1);

}

void or()

{

Datum d1, d2;

d2 = pop();

d1 = pop();

d1.val->real = (double)(d1.val->real != 0.0 || d2.val->real != 0.0);

push(d1);

}

void not()

{

Datum d;

d = pop();

d.val->real = (double)(d.val->real == 0.0);

push(d);

}

Y en la sección de reglas se agregó unas cuantas producciones más:

stmt:

expr {code(pop);}

| PRINT expr {code(prexpr); $$ = $2;}

| **while** cond stmt end

{

($1)[1] = (Inst)$3; */\* cuerpo de la iteración \*/*

($1)[2] = (Inst)$4; */\* terminar si la condición no se cumple \*/*

}

| **if** cond stmt end */\* proposición if que no emplea else \*/*

{

($1)[1] = (Inst)$3; */\* parte then \*/*

($1)[3] = (Inst)$4; */\* terminar si la condición no se cumple \*/*

}

| **if** cond stmt end ELSE stmt end */\* proposición if con parte else \*/*

{

($1)[1] = (Inst)$3; */\*parte then \*/*

($1)[2] = (Inst)$6; */\*parte else \*/*

($1)[3] = (Inst)$7; */\*terminar si la condición no se cumple \*/*

}

| '{' stmtlist '}' {$$ = $2;}

| '\n' '{' stmtlist '}' {$$ = $3;}

;

cond:

'(' expr ')' '\n' {code(STOP); $$ = $2;}

| expr {code(STOP); $$ = $1;}

;

while:

WHILE {$$ = code3(whilecode, STOP, STOP);}

;

if:

IF {$$ = code(ifcode); code3(STOP, STOP, STOP);}

;

end:

*/\* nada \*/* {code(STOP); $$ = progp;}

;

stmtlist:

*/\* nada \*/* {$$ = progp;}

| stmtlist '\n'

| stmtlist stmt

;

Pruebas de funcionamiento:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**CONCLUSIONES**

Esta práctica fue relativamente sencilla, ya que el código de HOC5 que nos pasó el profesor esta prácticamente todo, solo fue adaptar como en la practica anterior.