INSTITUTO POLITÉCNICO NACIONAL
Escuela Superior de Cómputo

Compiladores

Pacheco Castillo Isaías

Manual técnico LOGO

3CM7

Contenido

Gramática

Para poder desarrollar el proyecto se utilizó la gramática implementada en HOC y se agregaron los siguientes tokes e instrucciones.

```
%token  <sym>   NUMBER STRING PRINT VAR BLTIN UNDEF WHILE FOR IF ELSE
%token  <sym>    FD BK RT LTA CT CS REPEATA PENUP PENDOWN END SETHA PENC//Tokens para LOGO
%token  <sym>   FUNCTION PROCEDURE RETURN FUNC PROC READ
%token  <narg>  ARG
%type   <inst>  expr stmt asgn prlist stmtlist
%type   <inst>  cond while for if begin end
%type   <inst>  repeat  logocomand//LOGO
%type   <sym>   procname
%type   <narg>  arglist
%right  '=' ADDEQ SUBEQ MULEQ DIVEQ MODEQ
%left   OR
%left   AND
%left   GT GE LT LE EQ NE
%left   '+' '-'
%left   '*' '/' '%'
%left   UNARYMINUS NOT INC DEC
%right  '^'
```

Una expresión de tipo **expr** puede derivar en un comando de LOGO por medio de **logocomand**.

```
expr: VAR     { $$ = code3(varpush, (Inst)$1, eval); }
    | NUMBER {       code2(constpush, (Inst)$1);      }
    ...
    ...
    | logocomand
    ;
```

Las derivaciones y acciones gramaticales de **logocomand** permitirán agregar a la máquina las acciones a realizar.

```
logocomand:FD   expr             { code(fd);   }
        |BK     expr             { code(bk);   }
        |RT     expr             { code(rt);   }
        |LTA    expr             { code(lta);  }
        |CT                      { code(ct);   }
        |CS                      { code(cs);   }
        |PENUP                   { $$ = code(penup);    }
        |PENDOWN                 { $$ = code(pendown);  }
        |SETHA expr              { $$ = code(sethang);  }
        |PENC  expr expr expr    { $$ = code(pcolor);   }
    ;
```

El código máquina generado por estas instrucciones es el siguiente.

Instrucción	Código máquina	Operación
fd n	execute constpush fd	El número **n** es ingresado en la pila mediante **constpush** y este es usado para ingresar un nodo en la estructura encargada de almacenar los puntos para dibujar.
bk n	execute constpush bk	El número ingresado **n** es ingresado en la pila mediante **constpush** y este es usado para ingresar un nodo en la estructura encargada de almacenar los puntos para dibujar.

rt n	execute constpush rt	El número **n** es ingresado en la pila mediante **constpush** y este es usado para modificar la variable global ángulo restando **n.**
lt n	execute constpush bk	El número **n** es ingresado en la pila mediante **constpush** y este es usado para modificar la variable global ángulo sumando **n.**
ct	execute ct	El comando toma la variable global ángulo y la establece a 90º e ingresa un punto al centro de la pantalla en la estructura encargada de almacenar los puntos para dibujar.
cs	execute cs	El comando toma la variable global ángulo y la establece a 90º y borra todos los puntos almacenados en la estructura que almacena los puntos para dibujar.
penup	execute penup	El comando establece la pluma a un color transparente.
pendown	execute pendown	El comando establece la pluma a el color anterior.
seth n	execute constpush sethang	El número **n** es ingresado a la pila por medio de **constpush** y este es usado para establecer la variable global ángulo a **n**.
setpencolor r g b	execute constpush constpush constpush pcolor	Los números ingresados **r g b** son ingresados a la pila por medio de **constpush** y estos son usados para cambiar el color de la pluma.

Las palabras utilizadas en los comandos fueron definidas como palabras reservadas en el archivo **init.c** y estas se cargan al iniciar el programa.

```
. . .
static struct {
    char *name;
    int kval;
} keywords[] = {
    "if",        IF,
    "else",      ELSE,
    "while",     WHILE,
    "print",     PRINT,
    "for",       FOR,
    "proc", PROC,
    "func", FUNC,
    "return", RETURN,
    "end", END,
    "fd", FD,
    "bk", BK,
    "rt", RT,
    "lt", LT,
    "ct", CT,
    "cs", CS,
    "pu", PENUP,
    "pd", PENDOWN,
    "repeat", REPEAT,
    "setpencolor", PENC,
    "seth", SETHA,
    0,  0
};
. . .
```

Procedimientos y ciclos

Además de los bucles **for** y **while** se implementó la instrucción **repeat**.

```
stmt:       expr  {  }
  ...
  | while '(' cond ')' stmt end {
          ($1)[1] = (Inst)$5;     /* stmr */
          ($1)[2] = (Inst)$6; }  /* final del ciclo  */
  | for '(' cond ';' cond ';' cond ')' stmt end {
          ($1)[1] = (Inst)$5;     /* condicion */
          ($1)[2] = (Inst)$7;     /* actualizacion */
          ($1)[3] = (Inst)$9;     /* stmt */
          ($1)[4] = (Inst)$10; } /* final*/
  | repeat ':' cond stmt end {
          ($1)[1] = (Inst)$4;    /* stmt */
          ($1)[2] = (Inst)$5;    /* final*/
                              }
  ...
  ;
```

Para las funciones y procedimientos se utilizaron las mismas producciones que HOC.

```
defn : FUNC procname { $2->type=FUNCTION; indef = 1; }
        '(' ')' stmt { code(procret); define($2); indef = 0; }
    |  PROC procname { $2->type=PROCEDURE; indef=1; }
        '(' ')' stmt { code(procret); define($2); indef=0; g_print("entro");  }
```

Interfaz gráfica

Para poder implementar la interfaz gráfica se utilizó la biblioteca GTK y Cairo, para ello se agregó lo siguiente.

```
#include <cairo.h>
#include <gtk/gtk.h>
```

Para la ventana principal se creó un objeto GtkWidget *window, este objeto fue el contenedor para los demás elementos de la interfaz.

```
//Se crea la ventana
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
//Agregar título a la ventana
gtk_window_set_title (GTK_WINDOW (window), "LOGO");
//Asignar un ancho y alto a la ventana
gtk_window_set_default_size (GTK_WINDOW (window), 1000, altowindow);
//Agregar un borde
gtk_container_set_border_width( GTK_CONTAINER(window), 10 );
//Señal que llama al evento para cerrar ventana
g_signal_connect( window, "delete-event", gtk_main_quit, NULL );
```

La ventana principal se dividio en 3 secciones, cada una de ellas con un frame, para ello fue necesario utilizar el objeto GtkWidget de nuevo para crear 3 frames.

```
//Frames
GtkWidget *frameComandos;
GtkWidget *framePintar;
GtkWidget *frameLogs;
```

```
//Crear frames
frameComandos = gtk_frame_new("Comandos");
framePintar   = gtk_frame_new("Pintar");
frameLogs = gtk_frame_new("Log");
```

Un frame es una ventana interna en la cual podemos agregar nuevos elementos. El frame **frameComandos** muestra una lista de los comandos implementados, esto se hace mediante un TextView el cuál necesita de un GtkTextBuffer para poder cargar el texto. Para mayor facilidad el buffer se llenó con el archivo **comandos.txt**.

```
//Buffer para texto
GtkTextBuffer *buffer;
//Iterador
GtkTextIter start, end;
//Apuntador a archivo
FILE *listaComados;

//Abrir archivo de comandos
listaComados = fopen("comandos.txt", "r");
if(!listaComados )
    g_print("Error con archivo comandos");
//Ir al final del archivo
if( fseek(listaComados, 0L, SEEK_END) == 0){
    long tambuff = ftell(listaComados);     //obtener tamaño para el buffer
    if( tambuff == -1 )
        g_print("Error con archivo comandos");
    //Reservar memoria
    comandosbuffer = malloc(sizeof(char) * (tambuff + 1));
    //Ir al inicio del archivo
    if(fseek(listaComados, 0L, SEEK_SET) != 0)
        g_print("Error con archivo comandos");


    size_t newLen = fread( comandosbuffer, sizeof(char), tambuff, listaComados);
    if( ferror( listaComados ) != 0 )
        g_print("Error con archivo comandos");
    else
        comandosbuffer[newLen++] = '\0';
    fclose(listaComados);
   }

//Agregar el buffer al textview
buffer = gtk_text_view_get_buffer( GTK_TEXT_VIEW(tvComandos));
gtk_text_buffer_set_text( buffer, comandosbuffer, -1);
```

Para el frame **framePintar** se implementó el widget **GtkDrawingArea**, un **TextView** para poder ingresar los comandos y un botón para poder ejecutar los comandos.

```
//Area para pintar
GtkWidget *drawingArea;
//Inicializar área
drawingArea = gtk_drawing_area_new();
//Asignar tamaño del área
gtk_widget_set_size_request ( drawingArea, ancho, alto);


//Asignar buffer a tvCódigo
bufferCodigo = gtk_text_view_get_buffer( GTK_TEXT_VIEW(tvCodigo));
//Comando inicial para prueba
gtk_text_buffer_set_text( bufferCodigo, "fd 40", -1);
```

Aunado a GTK se utilizó la librería Cairo la cual permite dibujar en un **GtkDrawingArea**. Para poder dibujar las líneas se necesitan 4 puntos los cuales corresponden al punto (x,y) inicial y el pundo (x,y) final, estos puntos fueron almacenados en una estructura de datos FIFO la cual tiene la siguiente estructura.

```
    //Estructura de datos para los puntos
    struct nodo {
        float x; //punto x inicial
        float y; //punto y inicial
        float dx; //punto en x final
        float dy; //punto en y final
        char penup_pendown;
        float r,g,b;
        struct nodo *sig;
    };

    //variable global que apunta al primer elemento de la cola
    struct nodo *raiz=NULL;

    //Variable global que apunta al fondo de la cola
    struct nodo *fondo=NULL;
```

También se implementaron los métodos para insertar, saber si hay elementos y eliminar los valores de la estructura.

```
//insertar un nodo en la lista
void insertar(float x, float y, float dx, float dy, char penup_pendown, float r, float g,
float b )
{
    struct nodo *nuevo;
    nuevo=malloc(sizeof(struct nodo));
    nuevo->x  = x;
    nuevo->y  = y;
    nuevo->dx = x + dx;
    nuevo->dy = y + dy;
    nuevo->r = r;
    nuevo->g = g;
    nuevo->b = b;
    nuevo->penup_pendown = penup_pendown;
    nuevo->sig=NULL;
    if (vacia())
    {
        raiz = nuevo;
        fondo = nuevo;
        inicioPuntos = nuevo;
    }
    else
    {
        fondo->sig = nuevo;
        fondo = nuevo;
    }
}


//Saber si la estructura está vacía
int vacia()
{
    if (raiz == NULL)
        return 1;
    else
        return 0;
}
```

```
//Método para saber si la estructura está vacía
void eliminar()
{
    struct nodo *reco = raiz;
    struct nodo *bor;
    while (reco != NULL)
    {
        bor = reco;
        reco = reco->sig;
        free(bor);
    }
    raiz   = NULL;
    fondo  = NULL;

    /* Ya que no hay elementos en la lista se agrega
       Uno nuevo en el centro de la pantalla para
       Comenzar a pintar de nuevo, el ángulo por
       defecto se establece a 90°
    */
    angulo = 90;
    insertar(0,0,0,0,1, 0.20, 0.53, 0.16);
}
```

Mientras se está ejecutando la ventana el método **do_drawing** es llamado constantemente, en este método se recorre la estructura obteniendo los puntos guardados y se pintan en la ventana. Para cada línea pintada se tomará el punto final como referencia para comenzar a dibujar.

```
//Método llamado por window para actualizar y pintar
static void do_drawing(cairo_t *cr)
{
    //Cambio de escala
    cairo_scale( cr, escalaX, escalaX );
    //Cálculo del centro de la ventana
    double xsize  = (1/escalaX)*ancho;
    double ysize  = (1/escalaX)*alto;
    //Mover el cursor al centro del área para pintar
    cairo_translate(cr, (xsize/2) + dxWindow , (ysize/2) + dyWindow );
    //Agregar la imagen de la tortuga en el centro
    gdk_cairo_set_source_pixbuf(cr, imagenTortuga, -27+fondo->dx,fondo->dy);
    //La imagen será cargada en un pequeño rectángulo
    cairo_rectangle (cr,
                -27+fondo->dx,
                fondo->dy,
                55,
                55);
    //Rellenar rectángulo con la imagen
    cairo_fill(cr);

    //Apuntador al inicio de la cola
    struct nodo *reco = raiz;
    //Mentra haya más elementos en la cola...
    while (reco != NULL)
    {
        //Se establece el color de la pluma
        cairo_set_source_rgba(cr, reco->r, reco->g, reco->b, reco->penup_pendown );
        //Cambiar el tamaño de la pluma
        cairo_set_line_width(cr, 7);
        //Desplazamiento al punto de inicio de la línea
        cairo_move_to(cr, reco->x, reco->y);
        //Desplazamiento al punto final de la línea
        cairo_line_to(cr, reco->dx, reco->dy);
        //El apuntador apunta al siguiente elemento
```

```
        reco = reco->sig;
        //Unir los puntos
        cairo_stroke(cr);
    }

   //Colocar mensaje de la acción que se realizó
   gtk_label_set_text(lblLog, msj);

}
```

Para poder ejecutar los comandos se implementó un botón el cual al ser pulsado llama a la función **callback()** la cual escribe en un archivo los comandos los cuales son leídos mediante **moreinput()** de HOC**.**

```
    //boton
    GtkWidget *btnGo;


    //Se inicializa el botón
    btnGo = gtk_button_new_with_label("PINTAR");
    gtk_widget_set_size_request ( btnGo, 70, 50);


    //Señal para conectar el botón con la función callback()
    g_signal_connect (btnGo, "clicked", G_CALLBACK (callback),
                                    (gpointer) "cool button");

    //Evento del boton
    static void callback( GtkWidget *widget,
                      gpointer   data )
    {

    //Si el archivo existe se elimina
    remove("test.txt");

    //Apuntador a archivo
    FILE *file;
    //Abrir archivo en modo de escritura
    file = fopen("test.txt", "w");
    //Se obtiene el texto de textview
    char *cad = get_text_of_textview( tvCodigo );

    //Se escribe el texto en el archivo
    fprintf(file, "%s\n", cad);
    //Se cierra el archivo
    fclose(file);

    //Se ejecuta el código del archivo
    while( moreinput() )
        run();

    gargc = 1;

}
```

Instrucciones de la máquina

Se definieron las siguientes instrucciones en **hoc.h** para poder ser utilizadas en **code.c**

```
IsaiasP$ cat hoc.h

. . .

//LOGO
extern fd(), bk(), rt(), lt(), ct(), cs(), penup(), pendown(), repeat(), sethang(),
pcolor();
```

```
//Controla el angulo
extern float angulo;

//Controla la pluma
extern char pup_pdown;

//Fondo de la estructura de puntos
extern struct nodo *fondo;

//Agregar un punto a la lista
extern void insertar(float x, float y, float dx, float dy, char penup_pendown, float r,
float g, float b);

//Eliminar los puntos
extern void eliminar();
```

IsaiasP$ cat code.c

```
//Avanzar n
fd(){
    //Escribe la instrucción en el archivo log.txt
    fputs("fd \n", archivo);
    //El número de unidades para desplazarse esta en el tope de la pila
    Datum fd_pos;
    fd_pos = pop();
    //Calcular la dirección
    float x = fd_pos.val*cos(angulo * 3.14159265f / 180.0f);
    float y = fd_pos.val*sin(angulo * 3.14159265f / 180.0f);
    //Insertar el punto para ser dibujado
    insertar( fondo->dx, fondo->dy, x, -y , pup_pdown, fondo->r, fondo->g, fondo->b);
    //Se escribe el mensaje para el textview log
    snprintf(msj, sizeof msj, "Avanzar %0.2f unidades, angulo de %0.2f°", fd_pos.val,
angulo );
    }


//Retroceder n
bk(){
    //Escribe la instrucción en el archivo log.txt
    fputs("bk \n", archivo);
    //El número de unidades para desplazarse esta en el tope de la pila
    Datum bk_pos;
    bk_pos = pop();
    //Calcular la dirección
    float x = bk_pos.val*cos(angulo * 3.14159265f / 180.0f);
    float y = bk_pos.val*sin(angulo * 3.14159265f / 180.0f);
    //Insertar el punto para ser dibujado
    insertar( fondo->dx, fondo->dy, x, y , pup_pdown, fondo->r, fondo->g, fondo->b);
    //Se escribe el mensaje para el textview log
    snprintf(msj, sizeof msj, "Retorceder %0.2f unidades, angulo de %0.2f°", bk_pos.val,
angulo );
}

//Rota el angulo en direccion de las manecillas del reloj
rt(){
    //Escribe la instrucción en el archivo log.txt
    fputs("rt \n", archivo);
    //El ángulo para rotar estará en el tope de la pila
    Datum ang;
    ang = pop();
```

```
    //El ángulo debera estar entre 0 - 360
    angulo = fmod( (angulo - ang.val), 360 );
    //Se escribe el mensaje para el textview log
    snprintf(msj, sizeof msj, "Rotar %0.2f, angulo de %0.2f° establecido", ang, angulo );
}

//Rota el angulo en direccion contraria a las mancillas del reloj
lt(){
    //Escribe la instrucción en el archivo log.txt
    fputs("lt \n", archivo);
    //El ángulo para rotar estará en el tope de la pila
    Datum ang;
    ang = pop();
    //El ángulo debera estar entre 0 - 360
    angulo = fmod( (angulo + ang.val), 360 );
    //Se escribe el mensaje para el textview log
    snprintf(msj, sizeof msj, "Rotar %0.2f, angulo de %0.2f° establecido", ang, angulo );
}
//Centrar
ct(){
    //Escribe la instrucción en el archivo log.txt
    fputs("ct \n", archivo);
    //El ángulo de inicio es de 90°
    angulo = 90;
    //Se inserta un punto al centro del área para pintar
    insertar(0,0,0,0, pup_pdown, fondo->r, fondo->g, fondo->b);
    //Se escribe el mensaje para el textview log
    snprintf(msj, sizeof msj, "Centrar, ángulo de %0.2f° establecido", angulo );
}

//Limpiar pantalla
cs(){
    //Escribe la instrucción en el archivo log.txt
    fputs("cs \n", archivo);
    //Llamar a la función eliminar
    eliminar();
    //Se escribe el mensaje para el textview log
    snprintf(msj, sizeof msj, "Ventana limpiada", 0 );
}

//Levantar pluma
penup(){
    //Escribe la instrucción en el archivo log.txt
    fputs("penup \n", archivo);
    //Se escribe el mensaje para el textview log
    snprintf(msj, sizeof msj, "Lapiz levantado", 0 );
    /*Se modifica la variable global pup_pdown la cual
    Establece en 0 el valor alfa del color lo que hace
    que el color no se vea */
    pup_pdown = 0;
}

//Colocar pluma
pendown(){
    //Escribe la instrucción en el archivo log.txt
    fputs("pendown \n", archivo);
    //Se escribe el mensaje para el textview log
    snprintf(msj, sizeof msj, "Lapiz colocado", 0 );
    /*Se modifica la variable global pup_pdown la cual
    establece en 1 el valor alfa del color lo que hace
    que el color se vea */
    pup_pdown = 1;
}
```

```
//Permite establecer el angulo
sethang(){
    //Escribe la instrucción en el archivo log.txt
    fputs("sethang \n", archivo);
    //En número para el ángulo está en el tope de la pila
    Datum d;
    d = pop();
    //El ángulo debe estar entre 0 - 360
    angulo = fmod( d.val, 360 );
    //Se escribe el mensaje para el textview log
    snprintf(msj, sizeof msj, "Angulo de %0.2f establecido", angulo );
}




//Permite cambiar el color de la pluma
pcolor(){
    //Escribe la instrucción en el archivo log.txt
    fputs("pcolor \n", archivo);
    //Los 3 valores para el color estarán en la pila
    Datum col;
    float r,g,b;

    col = pop();
    b = col.val/255;

    col = pop();
    g = col.val/255;

    col = pop();
    r = col.val/255;
    //Se inserta un punto en la última posición pero con e nuevo color
    insertar(fondo->dx,fondo->dy,0,0, fondo->penup_pendown, r, g, b);
    //Se escribe el mensaje para el textview log
    snprintf(msj, 0, "Nuevo color establecido", 0 );
}

//Permite la implementacion de ciclos
repeat(){
    //Escribe la instrucción en el archivo log.txt
    fputs("repeat \n", archivo);

    Datum d;
    Inst *savepc = pc; /* cuerpo */
    int i = 0;

    execute( savepc+2 );
    d = pop(); //Se saca el numero de repeticiones de la pila

    for( int i = 0; i<d.val; i++ ){
        execute( *((Inst **)(savepc)) ); //Se ejecuta el cuerpo
        if( returning )
            break;
    }

    if(!returning)
        pc = *((Inst **)(savepc + 1)); //pc apunta a STOP
}
```