1 Introducción

Como práctica de encontrar una herramienta informática para la realización de programas de software, he estado en la busqueda del intercambio de la sintaxis de los lenguajes de programación: *C*, *PHP*, *Ruby*, *Python*, *ICON*.

La programación de software debe ser algo claro, sencillo y concisa para el escribiente que pueda implementar sus ideas partiendo de su experiencia cotidiana en este caso el idioma cotidiano. Con el tiempo y la especialización en el desarrollo de software, no dejamos como seres humanos de tomar sentimiento a lo que se hace y este sentimiento ya no es solo por el código del programa sino por su importancia y utilidad que puede tener para con la sociedada, la programación se torna entre ciencia y arte.

A continuación presento un sencillo pero claro ejemplo de la posibilidad que existe de escribir software en nuestro idioma ("español"), en lenguaje de programación *C*[1].

Aclaro que este texto no pretende enseñar acerca del lenguaje de programación *C* o de programación estructura, lo único que pretende ser una base en la posibilidad de la escritura en *español* utilizando un lenguaje de medio/bajo nivel que se y ser poder visualizar un poco su facilidad de compresión al tener su léxico en espñol. Este lenguaje le he denominado *CSPA*
La cabezera "*cspa.h*" que se presenta a continuación traduce no literalmente las palabras claves del lenguaje *C*.

⟨*Cabezera CSPA*⟩≡

```
#ifndef CSPA_H
#define CSPA_H

/*Control de flujo*/
#define si if

#define pero else
#define pero else
#define cualquier default

#define cuando switch
#define es case
#define para break
#define siguiente continue

/*Ciclo*/
#define mientras while
#define hacer do
#define cada for

/*Tipos nombres largos*/
```

[1] Para esto recomiendo la utilización del compilador de software libre *GCC*

```
#define estructura struct
#define entero int
#define largo long
#define caracter char
#define real float
#define cadena char *
#define punteroA *

#define no(X) !(X)

/*tipo booleano, logico*/
typedef enum {VERDAD, FALSO} binario;


#define ciclo while(1)
#define retorna return
#endif
```

2 CSPA

Cspa un macro lenguaje en *C* y *C* para escribir software en español, claro aunque esto no se puede realizar completamente ya que la mayoria y muchas de las librerias estan principalmente en inglés, pero podemos empezar a crear interfaces conceptuales para la construcción de los programas de software es decir continuar trabajando con estas librerias pero a la vez empezar a usar las nuestras en nuestro idioma, y en caso de querer exponer esto en otro idioma se pueden propagar sus traducciones a través de encabezados. El único criterio que hay voy a tener presente es el siguiente: *No crear alias o reemplazos a los nombres de las funciones de las librerias ya existente que se hagan uso* el motivo principal es:

- la documentación, guias, tutorials, howtos, FAQs, API estan escritas con el nombre de la función esto puede llevar a un desgate innecesario para comprender el funcionamiento de las funciones.

Que mejor forma de mostrar la prácticidad a la hora de comprender y desarrollar software que escribiendo.

2.1 1 Ejemplo

Empezemos con un ejercicio sencillo, la impresión[2] desde el programa:

⟨*1 Ejemplo*⟩≡
```
⟨Cabezera CSPA⟩
#include <stdio.h>

    entero main(entero argc, caracter ** argv){
    /*GCC soporta en char UTF-8, podemos usar acentos*/
    printf("CSPA es C con macros para la programacin en Espaol");
    retorna 0;
  }
```

[2]Por fuera del inglés es necesario codificar los textos mirése :http://www.evanjones.ca/unicode-in-c.html

2.2 2 Ejemplo

Ahora para observar ciclos y condicionales

⟨*2 Ejemplo*⟩≡

```
⟨Cabezera CSPA⟩
#include <stdio.h>
  /* cuenta e imprime en letras si es numero o par o impart del 1 al 100 */
  entero main(entero argc, caracter ** argv){
  entero c = 0;

  cada(c = 0; c < 100; c++)
  {
    si(c % 2 == 0)
      printf("par\n");
    pero si(c % 1 == 0)
      printf("impar\n");
  }

 }
```

2.3 3 Ejemplo

Más de ciclos y condicionales

⟨*3 Ejemplo*⟩≡

```
⟨Cabezera CSPA⟩
#include <stdio.h>

  entero main(entero argc, caracter ** argv){
  entero estado = 2;
  mientras(estado > -1){
    cuando(estado)
    {
      es 0:
        printf("Estado es cero\n");
      para;
        es 1:
        printf("Estado es uno\n");
      para;
      cualquier:
        printf("Bueno cualquier estado\n");
    }

    estado -= 1;
  }
 }
```

2.4 Adivina el número

Ahora un ejemplo algo más divertido, una implementación del juego "Adivina el número" este pide un número e indica si esta "despés" o "antes" del número a adivinar

⟨*Adivina el número*⟩≡

```
⟨Cabezera CSPA⟩
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/*genera numero aleatoria a adivinar*/
entero adivina_el_numero(){
  srand(time(NULL));
  retorna (rand() % 100) + 1; /* entre 1 y 100*/
}

entero main(entero argc, caracter ** argv)
{
  entero entrada = 0;
  entero numero_a_adivinar = adivina_el_numero();
  printf("Holas bienvenidxs a \"Adivina el numero\" versin 0.0\n");
  printf("Ahora intenta adivinar el numero. Presinar Ctrl+C para salir. :)\n");

  ciclo{ /*esto reemplaza un while(1)*/
    printf("Ingrese el numero y presione la tecla \"Intro\":");
    scanf("%d", &entrada);
    si(entrada == numero_a_adivinar)
    {
      printf("Muy bien has adivinado el numero %d..\n", numero_a_adivinar);
      para; /*termina el ciclo*/
    }
    pero si(entrada > numero_a_adivinar)
    {
      printf("Estas despus del numero\n");
    }
    pero si(entrada < numero_a_adivinar)
    {
      printf("Estas antes del numero\n");
    }
  }
}
```

2.5 Puja y Empuja

La siguiente implementación es un pequeño video juego de lógica, en la cual hay que organizar de forma ascendiente el número indicado. Ejemplo si el numero mostrados es *4231* se debe organizar a *1234* pero esta organización se realiza utilizando las siguiente clave *<+[0-9]+*[3] donde por cada *<* indica cuantas veces se debe desplazar a la izquierda su número de la derecha(*[0-9]+*) pudiendose desplazar. Este juego le he llamado "Puja y empuja". Lo más importante de este ejemplo es poder observar que a medida que vamos abstrayendo las funcionalidad y utilizando nombres en nuestro idioma nos facilitamos la escritura y la lectura.

grupos de números ejemplos:

<<2 desplazar dos veces a la izquierda el numero *2*

<42 desplazar una vez a la izquiera el numero *42*

⟨*Puja y Empuja*⟩≡
  ⟨*Cabezera CSPA*⟩
  ⟨*Puja y Empuja: Cabezera base*⟩
  ⟨*Puja y Empuja: Funciones*⟩
  ⟨*Puja y Empuja: Pruebas Funcionales*⟩

```
entero main(entero argc, caracter ** argv){

  pruebas_funcionales();

  caracter entrada[200];
  caracter numero_a_organizar[200];
  time_t tiempo_inicio = time(NULL);

  printf("Bienvenidxs a \"Puja y Empuja\" un sencillo videojuego de lgica\n");
  printf("Para salir presiona Ctrl+C o escribir la palabra salir.\n");
  generar_numero_a_organizar(numero_a_organizar);
  printf("Su numero a organizar es: %s\n", numero_a_organizar);
  ciclo{
    printf("Ingrese secuencia de orden:");
    scanf("%s", entrada);

    si(strcmp(entrada, "salir") == 0)
    {
      printf("Gracias por jugar..\n");
      para;
    }

    si(valida_entrada_con_patron(entrada) == FALSO)
```

[3] Expresión regular para indicar el patrón de la clave

```
    {
      printf("Entrada invalida intentlo de nuevo.\n");
      siguiente;
    }

    desplazar_numero(numero_a_organizar, obtener_numero(entrada), obtener_veces(entrada
    printf("==> %s\n", numero_a_organizar);
    si(comprobar_numero_ordenado(numero_a_organizar) == VERDAD)
    {
      printf("Muy bien has ordenado el numero en %d segundos..:)\n", (int)(time(NULL) -
      return 0;
    }

  }
  return 0;
}
```

Esta cabezera es báse con librerias estandars de C

⟨*Puja y Empuja: Cabezera base*⟩≡

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <ctype.h>
#include <assert.h>
```

Una dicho muy común antes aplicado a la guerra y ahora al desarrollo del software es: *divide y venceras*. Se divide las rutinas en lo más necesario para poder realizar prue de funcionalidad.

⟨*Puja y Empuja: Funciones*⟩≡

```
binario comprobar_numero_ordenado(cadena Cnumero){
  cada(entero pdigito = 0, sdigito = 0; pdigito < strlen(Cnumero);  pdigito += 1, sdigi
    si(isdigit(Cnumero[pdigito]) && isdigit(Cnumero[sdigito]) && Cnumero[pdigito] > Cnu
    {
      retorna FALSO;
    }
  }
  retorna VERDAD;
}

void generar_numero_a_organizar(cadena numero_en_cadena){
  srand(time(NULL));
  hacer{
    sprintf(numero_en_cadena, "%d", (rand() % 9999) + 1);
  }mientras(comprobar_numero_ordenado(numero_en_cadena) == VERDAD);
}

entero encontrar_posicion_de_numero(cadena Cnumero, cadena Ca_numero){
  cadena p_a_numero = strstr(Cnumero, Ca_numero);
  si(p_a_numero == NULL)
    retorna 0;
  retorna (p_a_numero - Cnumero) + 1;
}

cadena obtener_subnumero_de_numero(cadena Cnumero, cadena Ca_numero){
 return strstr(Cnumero, Ca_numero);
}

binario desplazar_numero(cadena Cnumero, cadena Ca_numero, entero veces){
  cadena tmp_numero[5];
  entero pos_numero = encontrar_posicion_de_numero(Cnumero, Ca_numero);
  memset(tmp_numero, 0, sizeof(tmp_numero));

  si(pos_numero == 0 || pos_numero - veces < 0)
    retorna FALSO;

  pos_numero -= 1; /*arreglos empiezan desde Cero*/

  strncpy(tmp_numero, &Cnumero[pos_numero - veces], strlen(Ca_numero));
  strncpy(&Cnumero[pos_numero - veces], Ca_numero, strlen(Ca_numero));
  strncpy(&Cnumero[pos_numero], tmp_numero, strlen(tmp_numero));
```

```
}

entero obtener_veces(cadena entrada){
  entero veces = 0;
  cada(entero p = 0; p < strlen(entrada); p++)
    si(entrada[p] == '<')
    veces += 1;
  retorna veces;
}

 binario valida_entrada_con_patron(const cadena entrada){

   cada(entero p = 0; p < strlen(entrada); p++){
     si(entrada[p] == '<' && isdigit(entrada[p - 1]))
       retorna FALSO;
   }
   retorna VERDAD;
 }

cadena obtener_numero(cadena entrada){
  cadena p = entrada;
  while(!isdigit(*p++));
  p--;
  retorna p;
 }
```

Aqui retomo un poco el concepto de pruebas de “Unidad” ya que ayudan mucho a garantizar que las funciones realizen lo que esperamos antes de realmente utilizarlas.

⟨*Puja y Empuja: Pruebas Funcionales*⟩≡

```
/**
 *Estas sencillas pruebas ayudan a asegura
 *el correcto procedimiento de las funciones
 */
void pruebas_funcionales(){
  assert(valida_entrada_con_patron("<<<2") == VERDAD);
  assert(valida_entrada_con_patron("<2<") == FALSO);
  assert(valida_entrada_con_patron("2<<") == FALSO);
  assert(strcmp(obtener_numero("<<123"), "123") == 0);
  assert(obtener_veces("<<") == 2);
  assert(obtener_veces("<<<") == 3);
  assert(encontrar_posicion_de_numero("1432", "3") == 3);
  assert(encontrar_posicion_de_numero("1432", "4") == 2);
  assert(encontrar_posicion_de_numero("1753", "53") == 3);
  assert(encontrar_posicion_de_numero("2737", "737") == 2);
  assert(encontrar_posicion_de_numero("3731", "66") == 0);
  assert(comprobar_numero_ordenado("1234") == VERDAD);
  assert(comprobar_numero_ordenado("134") == VERDAD);
  assert(comprobar_numero_ordenado("3725") == FALSO);
  assert(comprobar_numero_ordenado("2465") == FALSO);


  {
    cadena numero_desplazado[200];
    strcpy(numero_desplazado, "1432");
    desplazar_numero(numero_desplazado, "2", 1);
    assert(strcmp(numero_desplazado,"1423") == 0);
    desplazar_numero(numero_desplazado, "2", 1);
    assert(strcmp(numero_desplazado,"1243") == 0);
    desplazar_numero(numero_desplazado, "3", 1);
    assert(strcmp(numero_desplazado, "1234") == 0);
  }

  /*se comprueba que no se genere numero ordenado*/
  {
    entero c = 0;
    caracter Cnumero[200];
    cada(c = 0; c < 99999; c++){
      generar_numero_a_organizar(Cnumero);
      assert(comprobar_numero_ordenado(Cnumero) == FALSO);
    }
```

```
  }
}
```

3 Ratón Alzado

Ahora para terminar implementaremos un programa de software de dibujo a "ratón alzado" el cual permitira dibujar en la ventana con el botón presionado el color que se haya seleccionado, además tendremos una pequeña paleta de colores al lado izquierdo de la ventana y un borrador y será implementada utilizando la libreria *SDL* por su sencilles y portabilidad. Para empezar la estructura general del aplicativo será la siguiente:

⟨*raton_alzado.c*⟩≡

```
⟨Cabezera CSPA⟩
⟨Ratón Alzado: Cabezera Base y Definiciones⟩
⟨Ratón Alzado: Estructuras⟩
⟨Ratón Alzado: Globales⟩
⟨Ratón Alzado: Funciones⟩

entero main(entero argc, caracter **argv){
  ⟨Ratón Alzado: Inicialización de sistema gráfico y otros⟩
  mientras(salir_p == FALSO){
    ⟨Ratón Alzado: Leer entradas del usuarix⟩
    ⟨Ratón Alzado: Actualizar pantalla⟩
  }
  ⟨Ratón Alzado: Finalización del sistema gráfico y otros⟩
}
```

La cabezera de inicialización de las librerias dependientes.

⟨*Ratón Alzado: Cabezera Base y Definiciones*⟩≡

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

#include <SDL/SDL.h>

#define PANTALLA_ANCHO 640
#define PANTALLA_ALTO 480
#define PANTALLA_BPP 32
#define PANTALLA_FLAGS SDL_SWSURFACE
```

El programa de software consta de un tablero, un pincel, una paleta y un borrador (el cual es un pincel con el color del area de dibujo) entonces reflejare estas ideas en las estructuras, lo más importante es asignar lo mejor posible las responsabilidades y relaciones entre las estructuras.

⟨*Ratón Alzado: Estructuras*⟩≡

```
typedef struct {
  SDL_Color *color;
  entero cantidad;
} Paleta;

typedef struct{
  SDL_Color fondo; /*color del fondo blanco*/
  SDL_Surface *s; /*superficie de dibujo*/
  SDL_Color *pincel; /*el pincel de dibujo*/
  Paleta paleta; /*paleta de colores*/
}Tablero;
```

Definimos y declaramos las variables que van ha ser utilizadas durante todo el programa, en los aplicativos utilizando SDL por lo general se crea una superficie SDL_Surface para el dibujado en pantalla, pero en nuestro caso utilizaremos la superficie del tablero como la superficie de la pantalla.

⟨*Ratón Alzado: Globales*⟩≡

```
binario salir_p = FALSO;
SDL_Event evento;
Tablero tablero;
```

Necesitaremos de varias funciones para los diferentes componentes (entrada, lógica e impresión): inicialización, actualización, procesamiento, dibujado, limpieza. La inicialización y finalización es fundamental para el ya que una incorrecta carga o descarga de los datos en memoria puede un fallo de segmentación y terminación inesperada del programa.

⟨*Ratón Alzado: Funciones*⟩≡

```
void inicializar_tablero(){
  si(SDL_Init(SDL_INIT_VIDEO) < 0){
    fprintf(stderr, "SDL_Init error:%s", SDL_GetError());
    exit(EXIT_FAILURE);
  }

  atexit(SDL_Quit);

  si((tablero.s = SDL_SetVideoMode(PANTALLA_ANCHO,
                                   PANTALLA_ALTO,
                                   PANTALLA_BPP,
                                   PANTALLA_FLAGS)) == NULL){
    fprintf(stderr, "SDL_SetVideoMode error:%s", SDL_GetError());
    exit(EXIT_FAILURE);
  }


  /**
   *se puede inicializar el color de fondo
   */
  /*tablero.fondo = {...}*/
  tablero.fondo = (SDL_Color){255, 255, 255, 0};

  {
    entero cantidad_colores = 20;
    SDL_Color *colores = (SDL_Color*) malloc(sizeof(SDL_Color)*cantidad_colores);
    colores[0] = tablero.fondo;
    srand(time(NULL));
    cada(entero i = 1; i < cantidad_colores; i++){
      colores[i].r = (rand() % 255) + 1;
      colores[i].g = (rand() % 255) + 1;
      colores[i].b = (rand() % 255) + 1;
    }
    tablero.paleta.color = colores;
    tablero.paleta.cantidad = cantidad_colores;
  }
  tablero.pincel = &tablero.paleta.color[1];
}
```

```
void finalizar_tablero(){
  si(tablero.s != NULL){
    SDL_FreeSurface(tablero.s);
  }
  si(tablero.paleta.color != NULL){
    free(tablero.paleta.color);
  }
}
```

Al dibujar o imprimir el tablero en pantalla, primero pi

⟨*Ratón Alzado: Funciones*⟩+≡

```
void dibujar_fondo_completo_tablero(){
 SDL_FillRect(tablero.s, NULL, SDL_MapRGB(tablero.s->format,
                                          tablero.fondo.r,
                                          tablero.fondo.g,
                                          tablero.fondo.b));
}
 void dibujar_paleta(){
   entero icolor = 0;
   SDL_Rect pos_color;
   pos_color.x = 0;
   pos_color.y = 0;
   pos_color.h = PANTALLA_ANCHO;
   pos_color.w = PALETA_ANCHO+5;

   /*se dibuja borde de paleta*/
   SDL_FillRect(tablero.s, &pos_color, 0);
   pos_color.h = (PANTALLA_ALTO / tablero.paleta.cantidad);
   pos_color.w = PALETA_ANCHO;

   cada(icolor = 0; icolor < tablero.paleta.cantidad; icolor++){
     pos_color.y = icolor * (PANTALLA_ALTO / tablero.paleta.cantidad);
     SDL_FillRect(tablero.s, &pos_color, SDL_MapRGB(tablero.s->format,
                                                    tablero.paleta.color[icolor].r,
                                                    tablero.paleta.color[icolor].g,
                                                    tablero.paleta.color[icolor].b));
   }
 }

 void dibujar_pincel_actual(){
   SDL_Rect pos_pincel = {PANTALLA_ANCHO-PALETA_ANCHO,0, PALETA_ANCHO, PALETA_ANCHO};
   SDL_FillRect(tablero.s, &pos_pincel, SDL_MapRGB(tablero.s->format,
                                                   tablero.pincel->r,
                                                   tablero.pincel->g,
                                                   tablero.pincel->b));
 }

 void dibujar_tablero(){
   dibujar_paleta();
   dibujar_pincel_actual();
 }
```

Definimos el ancho de la paleta.

⟨*Ratón Alzado: Cabezera Base y Definiciones*⟩+≡
```
#define PALETA_ANCHO 40
```

Se inicializa el tablero, se carga paleta, pincel.

⟨*Ratón Alzado: Inicialización de sistema gráfico y otros*⟩≡
```
    inicializar_tablero();
```

El tablero solo se imprime en pantalla cuando se inicializa el aplicativo y posteriormente se actualiza solo el area de dibujo.

⟨*Ratón Alzado: Inicialización de sistema gráfico y otros*⟩+≡
```
    dibujar_fondo_completo_tablero();
    dibujar_tablero();
```

En cada ciclo del aplicativo hay que verificar y actualizar las entradas del usuarios.

⟨*Ratón Alzado: Leer entradas del usuarix*⟩≡
```
   mientras(SDL_PollEvent(&evento)){
    ⟨Ratón Alzado: Procesar eventos de entrada⟩
   }
```

Cada vez que se precione cerrar ventana finalizamos el aplicativo, cortando el ciclo principal a través de la variable —evento—.

⟨*Ratón Alzado: Procesar eventos de entrada*⟩≡
```
    si(evento.type == SDL_QUIT){
    salir_p = VERDAD;
  }
```

Y se notifica que hay que redibujar lo visto en pantalla cada ciclo. Hay mejores algoritmos de realizar este redibujado sin consumir tanto procesamiento, pero en nuestro caso no hay problema.

⟨*Ratón Alzado: Actualizar pantalla*⟩≡
```
    SDL_Flip(tablero.s);
```

Y cuando se termina el ciclo principal hay que liberar el espacio utilizado en memoria.

⟨*Ratón Alzado: Finalización del sistema gráfico y otros*⟩≡
```
    finalizar_tablero();
```

Hasta aqui el aplicativo ya imprime en pantalla el tablero, la paleta se dibuja desde y 0 hasta —PANTALLA_ALTO— donde cada color ocupa —PANTALL_ALTO/tablero.paleta.cantida estas medidas las necesitamos para la selección de los colores de las paletas, la rutina se ejecutara con el evento —SDL_MOUSEBUTTONDOWN— el cual se propaga cuando se presiona cualquier botón del ratón.

⟨*Ratón Alzado: Funciones*⟩+≡

```
void seleccionar_color_de_paleta(){
  entero icolor = 0;
  si(evento.type == SDL_MOUSEBUTTONDOWN){
    /*esta dentro de paleta*/
    si(evento.button.x > 0 && evento.button.x < PALETA_ANCHO && evento.button.y > 0
       && evento.button.y < PANTALLA_ANCHO){
      icolor = (entero)floor(evento.button.y /
                             (PANTALLA_ALTO / tablero.paleta.cantidad));
      tablero.pincel = &tablero.paleta.color[icolor];
      dibujar_pincel_actual();
    }
  }
 }
```

Agregamos procesamiento en el ciclo de eventos.

⟨*Ratón Alzado: Procesar eventos de entrada*⟩+≡

```
  seleccionar_color_de_paleta();
```

Y para finalizar, ya con la posibilidad de seleccionar el color del pincel desde la paleta dibujamos un pequeño cuadro simulando la punta del pincel.

⟨*Ratón Alzado: Funciones*⟩+≡

```
void dibujar_en_tablero_con_pincel(){
 SDL_Rect pos_pincel;
 pos_pincel.w = 5;
 pos_pincel.h = 5;

 si(evento.type == SDL_MOUSEMOTION && evento.motion.state & SDL_BUTTON(1)){
   /*se esta en el area de dibujo*/
   si(evento.motion.x > PALETA_ANCHO && evento.motion.x < PANTALLA_ANCHO
      && evento.motion.y > 0 && evento.motion.y < PANTALLA_ALTO){
     pos_pincel.x = evento.motion.x;
     pos_pincel.y = evento.motion.y;
     SDL_FillRect(tablero.s, &pos_pincel,
                  SDL_MapRGB(tablero.s->format,
                             tablero.pincel->r,
                             tablero.pincel->g,
                             tablero.pincel->b));
   }
 }
}
```

Ahora cada vez que se presione cualquier botón del ratón se dibujo un pequeño recuadro en el area de dibujo.

⟨*Ratón Alzado: Procesar eventos de entrada*⟩+≡

```
dibujar_en_tablero_con_pincel();
```

4 Conclución

Como se pudo observar en todos los ejemplos anteriores con las macros se puede traducir el lexíco del lenguaje de programación y aun continuar utilizando todas las librerias que actualmente estan disponibles para este lenguaje siendo este inherente del preprocesador de *C*. *CSPA* puede ser utilizado no solo para el aprendizaje de la programación de software sino para el desarrollo de aplicativos de software de gran embergadura llevadas por personas de un mismo idioma o bien de varios idiomas propagando módulos en los idiomas respectivos.

No todo programa de software es universal y no todo es aplicable a cada unas de las situaciones alrededor del mundo donde la diversidad humana se refleja en todo su que hacer, las computadoras tienen un lenguaje, la humanidad tiene multitudes de idiomas, dialectos, porque minimizar la escritura a un determinado idioma (inglés)? mi respuesta es; no hay necesidad podemos escribir el software en el idioma que nos plazca, pero desde hace siglos todo circunda alrededor del comercio y el capital el cual a su vez busca un idioma globalizado para facilitar todo su desenvolmiento. Actualmente el desarrollo de software es una industria para el beneficio monopolistico de algunos y algunas de sus compañias, esto se puede por ejemplo con la cantidad de video juegos (la vida es un video juego, centenares mueren y ya) para las diferentes consolas que año trás año salen al mercado para que aquel o aquella que sea lo suficientemnte "competente" acceda a esta como un bien de suma satisfacción personal y engalorne su miserable vida, ya que no encuentra por fuera de esto algo por que vivir. O bien la supuesta necesidad de un programa de software para cada actividad de la vida, no niego que muchas veces este han podido tener una utilidad social (áreas como: la medicina, la astronomia, la biologia, las matemáticas..) pero es bien sabido que estas herramientas en manos de la burguesia es otro de sus medios para mantenerse, prolongarse, oprimir, vigilar y controlar.

Para terminar la invitación es a unirse al proyecto *LENGUA*(http://www.bit4bit.co) el cual es un meta-lenguaje para la creación de lenguajes traducibles tanto para el idioma del escribiente como para el lenguaje programación preferido, es decir poder escribir nuestro lenguaje de programación que posteriormente pueda ser traducido a uno de los tantos lenguajes de programación que existe actualmente.

Durante la construcción de este árticulo se crearón programas de software auxiliares:

- Un "Modo mayor para el coloreo de sintaxis" para EMACS del lenguaje *CSPA*, mirar el archivo *doc/cspa-mode.el*.
- Filtro para *noweb* y uso del paquete *listing* de *LaTex* pero tuve inconvenientes con el escapado de las llaves.
- Una clase para "mmm-mode" de *EMACS* para *CSPA*. *mmm-mode* tiene inconvenientes genera "string nil" en algunos bloques de *CSPA*.

5 Un Futuro

Sería de gran ayuda tener:

- Un traductor de los mensajes de errores de *GCC*.
- Un formateador de estilo (*Pretzel* puede servir).