|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Decorativo | | | | |
|  | 21/03 |  | |  |
| MANUAL TÉCNICO CH MÁQUINA  version 2021 | | |
| santiago aristizábal correa | | saaristizabalco@unal.edu.c |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Decorativo | | | | |
|  |  |  | |  | |
|  | ch máquina  El ChMaquina cuenta con diversas clases que se conectan para llevar a cabo funciones deseadas, y que a su vez estas clases contiene métodos que realizan tareas específicas para dar como resultado procesos sobresalientes para el ChMaquina.  El contenido de las clases y sus métodos se encuentra explícito de manera más a fondo en el código fuente realizado en este manual se nombrará y explicará el funcionamiento de cada una de las clases y sus métodos. | |  | |

|  |  |
| --- | --- |
|  |  |
| MANUAL TÉCNICOPRIMERA FASE El ChMaquina es un programa diseñado en el entorno java bajo del concepto de programación estructurada y programación por objetos, además de la parte gráfica, ChMaquina es una herramienta para el desarrollo de programas, funciona básicamente con clases y métodos que permiten llevar a cabo tareas específicas las cuales pueden ser:  • Interpretación y creación de archivos con extencion.ch.  • Compilación y ejecución de archivos .ch.  • Administración de memoria de un computador ficticio.  • Editar chprogramas creados.  • Identificar la finalización de ejecución de varios chprogramas en la RAM.  • Separar espacio de memoria sin sobrepasar límites de sobrecarga.  • Conexión interfaz gráfica de usuarios con el entorno de programación.  • Interpretación y corrección de sintaxis de cada una de las instrucciones para ser ejecutas.  • Interpretación de entradas de usuario generando salidas esperadas.  • Interpretación de los eventos generados por el usuario.  El ChMaquina cuenta con diversas funciones deseadas, que a su vez ejecutan instrucciones las cuales crean tareas específicas para dar como resultado procesos sobresalientes para el ChMaquina.  El contenido de las funciones y sus instrucciones se encuentra explícito de manera más a fondo en el código fuente realizado en este manual se nombrará y explicará el funcionamiento de cada una.  **CREACIÓN DE VARIABLES:**  Se pretenden crear las variables de alcance local y global, que se guarden y ejecuten instrucciones. | |
| let memoria = []; 🡪 CREA EN UNA VARIABLE DE ALCANCE LOCAL, UN ARRAY VACÍO  let btnMemoria = document.getElementById('memoria-btn'); 🡪 CREA EN UN VARIABLE DE ALCANCE LOCAL, UN LLAMADO AL ID MEMORIA-BTN, PROVENIENTE DEL HTML  let kernel = document.getElementById('kernel');🡪 CREA EN UN VARIABLE DE ALCANCE LOCAL, UN LLAMADO AL ID MEMORIA-BTN, PROVENIENTE DEL HTML  let memoriaInput = document.getElementById('memoria-input');🡪 CREA EN UN VARIABLE DE ALCANCE LOCAL, UN LLAMADO AL ID “KERNEL”, PROVENIENTE DEL HTML  let btnEncender = document.getElementById('encender');🡪 CREA EN UN VARIABLE DE ALCANCE LOCAL, UN LLAMADO AL ID “ENCENDER”, PROVENIENTE DEL HTML  let btnApagar = document.getElementById('apagar');🡪 CREA EN UN VARIABLE DE ALCANCE LOCAL, UN LLAMADO AL ID “APAGAR”, PROVENIENTE DEL HTML  let modo = document.getElementById('modo');🡪 CREA EN UN VARIABLE DE ALCANCE LOCAL, UN LLAMADO AL ID “MODO”, PROVENIENTE DEL HTML  var file = document.getElementById('files');🡪 CREA EN UN VARIABLE DE ALCANCE GLOBAL UN LLAMADO AL ID ‘FILES’, PROVENIENTE DEL HTML  let divMemoria = document.getElementById('memoria');🡪 CREA EN UN VARIABLE DE ALCANCE LOCAL, UN LLAMADO AL ID ‘MEMORIA’, PROVENIENTE DEL HTML  let cerrar = document.getElementById('cerrar');🡪 CREA EN UN VARIABLE DE ALCANCE LOCAL, UN LLAMADO AL ID ‘CERRAR’, PROVENIENTE DEL HTML  var lFinal = [];🡪 CREA EN UNA VARIABLE DE ALCANCE LOCAL LLAMADA ‘LFINAL’, UN ARRAY VACÍO | |  |
|  | |  |
| FUNCION ON() La función encender permite condicionar que antes de ejecutar los programas, se pueda demostrar que el valor que hay en memoria, no exceda de 9999 posiciones. Si se cumple esta condición se crea la lista, con el acumulador, el espacio de memoria (kernel) y las posiciones de memoria restantes. Permitiendo dar entrada a la selección de archivos.  Por otro lado, permite cambiar del botón  al  y desactiva la edición de los contenedores de los valores de, KERNEL, MEMORIA Y ACOMULADOR. Abre la celda de memoria, dónde están las instrucciones en la interfaz y crea espacio por instrucción en el contenedor.  En caso de que de que exceda el espacio de memoria (9999), lanza un error en contenedor de alerta y apaga la maquina.  function ON() {      if (Number(memoriaInput.value) > 9999) {          alert('EL ESPACIO DE MEMORIA HA EXCEDIDO');          OFF();      }      kernel.disabled=true;      memoriaInput.disabled=true;      btnEncender.style.display = 'none';      btnApagar.style.display = 'inline-block';      modo.innerHTML = 'Modo usuario';      cerrar.style.display='block';      files.disabled = false;      let memoriaMostrar = [];      let acumulador = [0,'Acumulador'];      memoriaMostrar.push(acumulador);      console.log(memoriaMostrar);      for(let i = 1; i<Number(memoriaInput.value)+1; i++) {          if(i <= Number(kernel.value)) {              let so = [i];              console.log(so);              so.push('CHSO\_V2021');              memoriaMostrar.push(so);          } else{              memoriaMostrar.push(i)          }      }      console.log(memoriaMostrar);      document.getElementById('memoria').style.display = 'block';      document.getElementById('memoria').innerHTML = memoriaMostrar.join('<br></br>');  } Función OFF(): Siendo el antónimo de encender, las funcionalidades que tiene es lo contrario, sin embargo, cuando entra a esta función, recarga la página, haciendo que la máquina empiece de nuevo.  function off() {      location.reload()      kernel.disabled = false;      memoriaInput.disabled = false;      btnApagar.style.display = 'none';      btnEncender.style.display = 'inline-block';      modo.innerHTML = 'Modo kernel';      files.disabled = true;      btnMemoria.disabled = false;  }   Función LeerArchivo() : Está función con el archivo ya seleccionado tipado con ‘.ch’, así permitiendo el procesamiento y la verificación de sintaxis, con lo cual, se puede agregar la lista con los archivos permitidos dentro del espacio de memoria, en el contenedor de memoria. Por otro lado, se hace la correcta lectura de las instrucciones que crean variables y entradas, para agregarlas en sus correspondientes contenedores.  function leerArchivo(*evento*) {  *//console.log(evento.target.files.length);*      for(let i=0; i<*evento*.target.files.length; i++) {      document.getElementById('instrucciones').innerHTML = "";      document.getElementById('variables').innerHTML = "";      document.getElementById('etiquetas').innerHTML = "";      let archivo = *evento*.target.files[i];      procesarArchivo(archivo, function(*result*) {      let lArchivo = [];      lArchivo = *result*.split('\n');      for(let i=0; i<lArchivo.length; i++) {          if(lArchivo[i] == "") {              lArchivo.splice(i, 2);          }      }      let listaPrueba = [];  *//console.log(lArchivo);*      for(instruccion of lArchivo) {          listaPrueba.push(instruccion.split(' '))      }      function verificarSintaxis() {      for(instruccion of listaPrueba) {          if(instruccion[0]=='nueva') {              return true          }else if(instruccion[0] =='lea') {              return true          }else if(instruccion[0] =='cargue') {              return true          }else if(instruccion[0] =='almacene') {              return true          }else if(instruccion[0] =='vaya') {              return true          }else if(instruccion[0] =='vayasi') {              return true          }else if(instruccion[0] =='etiqueta') {              return true          }else if(instruccion[0] =='sume') {              return true          }else if(instruccion[0] =='reste') {              return true          }else if(instruccion[0] =='multiplique') {              return true          }else if(instruccion[0] =='divida') {              return true          }else if(instruccion[0] =='potencia') {              return true          }else if(instruccion[0] =='modulo') {              return true          }else if(instruccion[0] =='concatene') {              return true          }else if(instruccion[0] =='elimine') {              return true          }else if(instruccion[0] =='Y') {              return true          }else if(instruccion[0] =='O') {              return true          }else if(instruccion[0] =='NO') {              return true          }else if(instruccion[0] =='muestre') {              return true          }else if(instruccion[0] =='imprima') {              return true          }else if(instruccion[0] =='retorne') {              return true          }else if(instruccion[0] =='porcentaje') {              return true          } else {              return false;          }      }    }        let bool = verificarSintaxis();      let sum = +kernel.value + +listaPrueba.length;      console.log(sum);      console.log(sum <= Number(memoriaInput.value));      if(bool === true && sum <= Number(memoriaInput.value)) {          console.log(listaPrueba);          for(instruccion of listaPrueba) {              lFinal.push(instruccion)          }          let cont = Number(kernel.value) + 1;          for(recorrer of lFinal) {              if(recorrer[0] == cont) {                  recorrer.shift();              }              cont++          }          let w = Number(kernel.value)+1;          for(k of lFinal) {              k.unshift(w);              w++;          }          let contador = 0;          let arrayMemoria= [];          let suma = +kernel.value + +lFinal.length;          arrayMemoria.push('0 Acumulador');          for(let s=1; s<=memoriaInput.value; s++) {              if(s<=kernel.value) {                  arrayMemoria.push(`${s} CHSO\_V2021`);              } else if(s>kernel.value && s<=suma){                  arrayMemoria.push(lFinal[contador]);                  contador++;              } else {                  arrayMemoria.push(s)              }          }          console.log(arrayMemoria);          document.getElementById('memoria').innerHTML = arrayMemoria.join('<br></br>');          let instrucciones = arrayMemoria.slice(+kernel.value+1, +suma+1);          console.log(instrucciones);          let listaVariables = [];          let listaEtiquetas = []; *//Lista para variables y etiquetas para Mostrar en el div*          for(linea of instrucciones) {              if(linea[1] == 'nueva') {                  listaVariables.push(linea)              } else if(linea[1] == 'etiqueta') {                  listaEtiquetas.push(linea);              }          }          document.getElementById('instrucciones').innerHTML= instrucciones.join('<br></br>');          document.getElementById('variables').innerHTML= listaVariables.join('<br></br>');          document.getElementById('etiquetas').innerHTML= listaEtiquetas.join('<br></br>');          } else {          document.getElementById('instrucciones').innerHTML = 'Error de sintaxis';          alert('Error de sintaxis');          location.reload();      }      });      }      } Función ProcesarArchivo() : Función que entra como parámetro de otra función se llama callback, en este caso, cumple la función de parámetro de una función anónima que trae el resultado, con el cual va a ir modificando las instrucciones del archivo, e ir añadiéndolo en un arreglo con los elementos depurados de espacios.  Además de esto permite revisar ir comparando si el tamaño de las líneas de instrucciones con los archivos seleccionados, exceden el espacio de memoria.  function procesarArchivo(*ch*, *callback*) {      var reader = new FileReader();      reader.readAsText(*ch*);      reader.onload = function () {          callback(reader.result);      }  }  procesarArchivo(archivo, function(*result*) {      let lArchivo = [];      lArchivo = *result*.split('\n');      for(let i=0; i<lArchivo.length; i++) {          if(lArchivo[i] == "") {              lArchivo.splice(i, 2);          }      }      let listaPrueba = [];  *//console.log(lArchivo);*      for(instruccion of lArchivo) {          listaPrueba.push(instruccion.split(' '))      }      function verificarSintaxis() {      for(instruccion of listaPrueba) {          if(instruccion[0]=='nueva') {              return true          }else if(instruccion[0] =='lea') {              return true          }else if(instruccion[0] =='cargue') {              return true          }else if(instruccion[0] =='almacene') {              return true          }else if(instruccion[0] =='vaya') {              return true          }else if(instruccion[0] =='vayasi') {              return true          }else if(instruccion[0] =='etiqueta') {              return true          }else if(instruccion[0] =='sume') {              return true          }else if(instruccion[0] =='reste') {              return true          }else if(instruccion[0] =='multiplique') {              return true          }else if(instruccion[0] =='divida') {              return true          }else if(instruccion[0] =='potencia') {              return true          }else if(instruccion[0] =='modulo') {              return true          }else if(instruccion[0] =='concatene') {              return true          }else if(instruccion[0] =='elimine') {              return true          }else if(instruccion[0] =='Y') {              return true          }else if(instruccion[0] =='O') {              return true          }else if(instruccion[0] =='NO') {              return true          }else if(instruccion[0] =='muestre') {              return true          }else if(instruccion[0] =='imprima') {              return true          }else if(instruccion[0] =='retorne') {              return true          }else if(instruccion[0] =='porcentaje') {              return true          } else {              return false;          }      }    }        let bool = verificarSintaxis();      let sum = +kernel.value + +listaPrueba.length;      console.log(sum);      console.log(sum <= Number(memoriaInput.value));      if(bool === true && sum <= Number(memoriaInput.value)) {          console.log(listaPrueba);          for(instruccion of listaPrueba) {              lFinal.push(instruccion)          }          let cont = Number(kernel.value) + 1;          for(recorrer of lFinal) {              if(recorrer[0] == cont) {                  recorrer.shift();              }              cont++          }          let w = Number(kernel.value)+1;          for(k of lFinal) {              k.unshift(w);              w++;          }          let contador = 0;          let arrayMemoria= [];          let suma = +kernel.value + +lFinal.length;          arrayMemoria.push('0 Acumulador');          for(let s=1; s<=memoriaInput.value; s++) {              if(s<=kernel.value) {                  arrayMemoria.push(`${s} CHSO\_V2021`);              } else if(s>kernel.value && s<=suma){                  arrayMemoria.push(lFinal[contador]);                  contador++;              } else {                  arrayMemoria.push(s)              }          }          console.log(arrayMemoria);          document.getElementById('memoria').innerHTML = arrayMemoria.join('<br></br>');          let instrucciones = arrayMemoria.slice(+kernel.value+1, +suma+1);          console.log(instrucciones);          let listaVariables = [];          let listaEtiquetas = []; *//Lista para variables y etiquetas para Mostrar en el div*          for(linea of instrucciones) {              if(linea[1] == 'nueva') {                  listaVariables.push(linea)              } else if(linea[1] == 'etiqueta') {                  listaEtiquetas.push(linea);              }          }          document.getElementById('instrucciones').innerHTML= instrucciones.join('<br></br>');          document.getElementById('variables').innerHTML= listaVariables.join('<br></br>');          document.getElementById('etiquetas').innerHTML= listaEtiquetas.join('<br></br>');          } else {          document.getElementById('instrucciones').innerHTML = 'Error de sintaxis';          alert('Error de sintaxis');          location.reload();      }      }); FUNCIÓN VerificarSintaxis(): Está función permite identificar las líneas de texto en el archivo seleccionado, y identificar si son instrucciones permitidas dentro de la lectura del ChMaquina. Y si no lo es, devuelve un ‘Error de sintaxis’.  function verificarSintaxis() {      for(instruccion of listaPrueba) {          if(instruccion[0]=='nueva') {              return true          }else if(instruccion[0] =='lea') {              return true          }else if(instruccion[0] =='cargue') {              return true          }else if(instruccion[0] =='almacene') {              return true          }else if(instruccion[0] =='vaya') {              return true          }else if(instruccion[0] =='vayasi') {              return true          }else if(instruccion[0] =='etiqueta') {              return true          }else if(instruccion[0] =='sume') {              return true          }else if(instruccion[0] =='reste') {              return true          }else if(instruccion[0] =='multiplique') {              return true          }else if(instruccion[0] =='divida') {              return true          }else if(instruccion[0] =='potencia') {              return true          }else if(instruccion[0] =='modulo') {              return true          }else if(instruccion[0] =='concatene') {              return true          }else if(instruccion[0] =='elimine') {              return true          }else if(instruccion[0] =='Y') {              return true          }else if(instruccion[0] =='O') {              return true          }else if(instruccion[0] =='NO') {              return true          }else if(instruccion[0] =='muestre') {              return true          }else if(instruccion[0] =='imprima') {              return true          }else if(instruccion[0] =='retorne') {              return true          }else if(instruccion[0] =='porcentaje') {              return true          } else {              return false;          }      }    } | |  |

## FUNCIÓN mostrarMemoria():

Es una función que se activa con el estado ‘onclick()’, es decir, que cuando se ejecuta un click sobre este, procede a crear el contenedor de memoria y muestra el botón de cerrar

function mostrarMemoria() {

    divMemoria.style.display = 'block';

    cerrar.style.display = 'inline-block';

}

}

## FUNCIÓN cerrarContenedor():

Es una función que se activa con el estado ‘onclick()’, es decir, que cuando se ejecuta un click sobre este, procede esconder el contenedor de memoria, y se esconde así mismo.

function cerrarContenedor() {

    divMemoria.style.display = 'none';

    cerrar.style.display = 'none'

}