## Especificación de Código

|  |  |
| --- | --- |
| **Función de Código** | **Plantillas de Código** |
| run[[Programa]] | run⟦Programa  →  definiciones:Definicion\*⟧ =  #SOURCE {file}  CALL main  HALT  For (definición in definiciones)  define⟦definición⟧ |
|  |  |
| define[[Definicion]] | define⟦DefinicionFuncion  →  nombre:String  params:DefinicionVariable\*   retorno:Tipo  locales:DefinicionVariable\* sentencias:Sentencia\*⟧ =  {nombre:}  #FUNC {nombre}  For(definición in params)  #PARAM definición.nombre : ejecutar[[definición.tipo]]  #RET ejecutar[[retorno]]  For (definición in locales)  #LOCAL definición.nombre : ejecutar[[definición.tipo]]  Int tamañoLocales  Si locales.isEmpty()  tamañoLocales = 0  Else  tamañoLocales = -locales.get(locales.size()-1).direccion  ENTER {tamañoLocales}  Int tamañoParams = params.stream()  .mapToInt(param -> param.tipo.tamaño)  .sum()  Int tamañoReturn = tipo.tamaño  For (sentencia in sentencias)  Ejecuta[[sentencia]] (tamañoReturn, tamañoLocales, tamañoParams)  Si node.tipo == TipoVoid:  RET tamañoReturn, Abs(tamañoLocales), tamañoParams |
|  | define⟦DefinicionVariable →  nombre:String tipo:Tipo⟧ =  #GLOBAL {nombre} : ejecutar[[tipo]] |
|  | define⟦DefinicionStruct →  nombre:String campos:Campo\*⟧ =  #TYPE {nombre} : {  For (campo in campos)  define[[campo]]  } |
|  | define⟦**Campo** → nombre:String tipo:Tipo⟧ =  {\t nombre} : ejecutar[[tipo]] |
| ejecuta[[**Tipo**]] | ejecuta[[**TipoEntero**]] =  {int} |
|  | ejecuta[[**TipoReal**]] =  {float} |
|  | ejecuta[[**TipoChar**]] =  {char} |
|  | ejecuta[[**TipoArray** → *longitud*:int *tipo*:Tipo]] =  {longitud \* } ejecutar[[tipo]] |
|  | ejecuta[[**TipoStruct** → *nombre*:String *campos*:Campo\*]] =  {nombre} |
|  | ejecuta[[**TipoVoid**]] =  {void} |
|  |  |
| ejecuta[[Sentencia]] | ejecuta⟦Print  →  expr:Expresion\* *tipo\_print*:String⟧ =  #LINE {linea}  Si expresión.size() > 0  valor[[expr.get(0)]]  OUT <expr.get(0).tipo>  si tipo\_print == “printsp”  PUSHB 32 (espacio)  OUTB  si tipo\_print == “println”  PUSHB 10 (salto de línea)  OUTB |
|  | ejecuta[[**Read** → expresion:Expresion]] =  #LINE {linea}  Dirección[[expresion]]  IN<expresion.tipo>  STORE<expresion.tipo> |
|  | ejecuta⟦Asignacion  →  izquierda:Expresion  derecha:Expresion⟧ =  #LINE {linea}  dirección[[izquierda]]  valor[[derecha]]  STORE <izquierda.tipo> |
|  | ejecuta[[**If** → *condicion*:Expresion *verdadero*:Sentencia\* *falso*:Sentencia\*]] =  valor[[condicion]]  JNZ etiqueta1  For (sentencia in falso)  Ejecutar[[sentencia]]  JMP etiqueta2  Etiqueta 1:  For(sentencia in verdadero)  Ejecutar[[sentencia]]  Etiqueta2: |
|  | ejecuta[[**While** → *condicion*:Expresion *sentencias*:Sentencia\*]] =  etiqueta1:  valor[[condición]]  JZ etiqueta2  For (sentencia in sentencias)  Ejecutar[[sentencia]]  JMP etiqueta1  Etiqueta2 |
|  | ejecuta[[**Invocacion** → *nombre*:String *params*:Expresion\*]] =  #LINE {linea}  For (param in params)  valor[[param]]  CALL {nombre}  Si invocacion.definicion.tipo != TipoVoid  POP <invocacion.definicion.tipo> |
|  | ejecuta[[**Return**→ *expresiones*:Expresion\*]]  (tamañoReturn, tamañoLocales, tamañoParams) =  #LINE {linea}  Si expresiones.size > 0  value[[expresiones.get(0)]]  RET {tamañoReturn, Abs(tamañoLocales), tamañoParams} |
|  |  |
| valor[[Expresion]] | valor[[**ConstanteEntero**→  *valor*:int ]] =  PUSH {valor} |
|  | valor[[**ConstanteReal**→  *valor*:double ]] =  PUSHF {valor} |
|  | valor[[**ConstanteChar**→  *valor*:String ]] =  String valorMod = valor.replace(“ ’ ”. “”)  si valor.length == 1:  PUSHB {valorMod.charAt(0)}  Else:  PUSHB 10 (salto de línea) |
|  | valor[[**Variable** →  *nombre*:String ]] =  direccion[[variable]]  LOAD<variable.tipo> |
|  | valor[[**ExpresionAritmetica**→  *izq*:Expresion  *operador*:String  *der*:Expresion ]] =  value[[izq]]  value[[der]]  si operador == “+”  ADD<arithmeticExpression.tipo>  si operador == “-”  SUB<arithmeticExpression.tipo>  si operador == “\*”  MUL<arithmeticExpression.tipo>  si operador == “/”  DIV<arithmeticExpression.tipo>  si operador == “%”  MOD |
|  | valor[[**ExpresionLogica**→  *izq*:Expresion  *operador*:String  *der*:Expresion ]] =  value[[izq]]  value[[der]]  si operador == “&&”  AND  si operador == “||”  OR |
|  | valor[[**Comparacion**→  *izq*:Expresion  *operador*:String  *der*:Expresion ]] =  value[[izq]]  value[[der]]  si operador == “<”  LT<izq.tipo>  si operador == “>”  GT<izq.tipo>  si operador == “≥”  GT<izq.tipo>  value[[izq]]  value[[der]]  EQ<izq.tipo>  OR  si operador == “≤”  LT<izq.tipo>  value[[izq]]  value[[der]]  EQ<izq.tipo>  OR  si operador == “==”  EQ<izq.tipo>  si operador == “!=”  NEQ<izq.tipo> |
|  | valor[[**ExpresionUnaria**→  *expresion*:Expresion  *operador*:String ]] =  value[[expresion]]  NOT |
|  | valor[[**Conversion**→  *nuevoTipo*:Tipo  *expresion*:Expresion ]] =  valor[[expresion]]  si expresion.tipo == TipoEntero  si nuevoTipo == TipoReal  I2F  si nuevoTipo == TipoChar  I2B  si expresion.tipo == TipoReal  si nuevoTipo == TipoEntero  F2I  si nuevoTipo == TipoChar  F2I  I2B  si expresion.tipo == TipoChar  si nuevoTipo == TipoEntero  B2I  si nuevoTipo == TipoReal  B2I  I2F |
|  | valor[[**InvocacionExpresion**→  *nombre*:String  *params*:Expresion\* ]] =  for(param in params)  valor[[param]]  CALL {nombre} |
|  | valor[[**AccesoArray**→  *array*:Expresion  *indice*:Expresion ]] =  dirección[[accesoArray]]  LOAD<accesoArray.tipo> |
|  | valor[[**AccesoCampo**→  *struct*:Expresion  *campo*:Expresion ]] =  dirección[[accesoCampo]]  LOAD<accesoCampo.tipo> |
| Dirección[[**expresion**]] | direccion[[**Variable** →  *nombre*:String ]] =  si Variable.ambito ≠ global  PUSHA BP  PUSH {Variable.definicion.direccion}  ADD  Else  PUSH {Variable.definicion.direccion} |
|  | direccion[[**AccesoArray**→  *array*:Expresion  *indice*:Expresion ]] =  dirección[[array]]  valor[[indice]]  PUSH accesoArray.tipo.tamaño  MUL  ADD |
|  | direccion[[**AccesoCampo**→  *struct*:Expresion  *campo*:Expresion ]] =  dirección[[struct]]  PUSH accesoCampo.tipo.campos.get(campo).direccion  ADD |