## Especificación de Código

|  |  |
| --- | --- |
| **Función de Código** | **Plantillas de Código** |
| run : Program → Instruccion\* | run [[ Program  →  classnode:classNode methodcallsentence:methodCallSentence]] =  run[[classnode]]  ejecuta[[methodcallsentence]] |
|  |  |
| Run : **ClassNode** → Instrucción\* | run [[ **ClassNode** → *name*:String *definition*:definition\* *createMethod*:String\* *method*:method\* ]] =  define[[definitioni]]  run[[method]] |
|  |  |
| Run : **Method** → Instrucción\* | Run [[ **Method** → *name*:String *parameter*:parameter\* *retorno*:type *definition*:definition\* *sentence*:sentence\* ]] =  {name}:  ENTER { Σ definitioni.type.size }  ejecuta[[sentencei]]  si retorno == VoidType  RET 0, {Σ definitioni.type.size}, {Σ parameteri.definition.type.size} |
|  |  |
| define : Definition → Instruccion\* | define[[ TupleDefinition  → *name*:String *vardefinition*:varDefinition\*]] =  define [[**VarDefinition**:definition → *name*:String\* *type*:type ]] = |
|  |  |
| ejecuta : Sentence → Instruccion\* |  |
|  | ejecuta [[ Print  →  *string*:String *expr*:expr\*]] =  para cada exp en expr  valor[[exp]]  OUT<exp.tipo>  ejecuta [[ Read  →  expr:expr\* ]] =  para cada exp en expr  dirección[[exp]]  IN<exp.type>  STORE<exp.type>  ejecuta[[Assignment  →  *left*:expr *right*:expr]] =  direccion[[left]]  valor[[right]]  STORE<left.type>  ejecuta[[Conditional  →  *condition*:expr *iftrue*:sentence\* *iffalse*:sentence\*]] =  valor[[condition]]  jz else{n}  ejecuta[[iftruei]]  jmp finIf{n}  else{n}:  ejecuta[[iffalsei]]  finif{n}:  Ejecuta[[Loop  → *init*:sentence\* *condition*:expr *sentence*:sentence\*]] =  Ejecuta[[initi]]  inicioLoop{n}:  valor[[condition]]  jnz finLoop{n}  ejecuta[[sentencei]]  jmp inicioLoop{n}  finLoop{n}:  Ejecuta[[ReturnNode  →  expr:expr ]] =  Si expr != null  RET {expr.type.size}, { Σ expr.method.definition.type.size}, { Σ expr.method.parameter.definition.type.size}  Ejecuta[[MethodCallSentence  →  *name*:String *args*:expr\*]] =  CALL {name}  POP<methodCallSentence.method.retorno> |
|  |  |
| valor: Expr → Instruccion\* | Valor[[LitEnt  →  string:String ]] =  PUSH {string}  Valor[[LitReal   →  string:String ]] =  PUSHF {string}  Valor[[LitChar   →  string:String ]] =  PUSHB {string}  Valor[[Variable  →  string:String ]] =  Dirección[[var]]  LOAD<variable.type>  Valor[[ExprBinariaAritmetica  →  *left*:expr *op*:String *right*:expr]] =  Valor[[left]]  Valor[[right]]  Si *op* ==’+’  ADD<left.type>  Sino si *op* ==’-’  SUB<left.type>  Sino si *op* ==’\*’  MUL<left.type>  Sino si *op* ==’/’  DIV<left.type>  Valor[[ExprBinariaLogica  →  *left*:expr *op*:String *right*:expr ]=  Valor[[left]]  Valor[[right]]  Si *op* == ’>’  GT<left.type>  Sino si *op* == ’<’  LT<left.type>  Sino si *op* == ’>=’  GE<left.type>  Sino si *op* == ’<=’  LE<left.type>  Sino si *op* == ’and’  AND  Sino si *op* == ’or’  OR  Valor[[**ExprUnariaAritmetica**  →  *op*:String  *expr*:expr]]=  Si *op == ‘-‘*  Valor[[expr]]  PUSH<expr.type> -1  MUL<expr.type>  Valor[[**ExprUnariaLogica**  →  *op*:String  *expr*:expr]]=  Si op == ‘not’  Valor[[expr]]  NOT  Valor[[**Cast**  →  *typeToConvert*:tipo  *expr*:expr]] =  Valor[[expr]]  Si typeToConvert == intType && expr.type == charType  B2I  Sino si typeToConvert == intType && expr.type == realType  F2I  Sino si typeToConvert == charType && expr.type == intType  I2B  Sino si typeToConvert == realType && expr.type == intType  I2F  valor[[**ArrayAccess** →  *left*:expr  *right*:expr ]]=  direccion[[left]]  valor[[right]]  PUSH {left.type.type.size}  MUL  ADD  LOAD<left.type.type>  valor[[**Access**  →  *left*:expr *op*:String *right*:String]]=  direccion[[left]]  PUSH {left.type.definition.vardefinition[string].address}  ADD  LOAD<left.type.definition.vardefinition[string].type}    Valor[[**MethodCallExpr** →  *name*:String *args*:expr\*]] =  Valor[argsi]]  CALL {name} |
| dirección: **Expr** -> Instrucción\* | dirección[[Variable  →  string:String ]] =  Si si def.ambito != GLOBAL  PUSHA BP  PUSH {def.address }  ADD  Sino  PUSHA {def.address}  direccion[[**Access**  →  *left*:expr *op*:String *right*:String]]=  dirección[[left]]  PUSH {left.type.definition.vardefinition[string].address}  ADD  direccion[[**ArrayAccess** →  *left*:expr  *right*:expr]]=  dirección[[left]]  PUSH {left.type.type.size}  Valor[[right]]  MUL  ADD |
|  |  |