|  |  |
| --- | --- |
| **Derivation rule**  Syntax translation definision  שמות הסטודנטים : א. נדב שלו 204233456  ב. בן מגריסו 308548569 | **Semantic action** |
| PROGRAM -> Block | stack = createStack(); |
| BLOCK -> **block**  DEFINITIONS ;  **begin** COMMANDS;  **end** | hashTableToPushToStack =createHashTable();  push(stack, hashTableToPushToStack);  DEFINIONS  COMMANDS  pop(stack); |
| DEFINITIONS -> DEFINITION | DEFINITIONS' |  |
| DEFINITIONS' ->; DEFINITION | DEFINITIONS' |  |
| DEFINITIONS' -> ᵋ |  |
| DEFINITION -> VAR\_DEFINITION | TYPE\_DEFINITIONS |  |
| VAR\_DEFINITIONS -> id: VAR\_DEFINITION' |  |
| VAR\_DEFINITION' -> BASIC\_TYPE | currentDataItem = searchInsideHashTableAndReturnItem( top(stack), key);  if (currentDataItem != NULL)  {  duplicateIdInTheSameScope();  return;  }  insert(top(stack), tempDataItem); |
| VAR\_DEFINITION' -> type\_name | currentDataItem = searchInsideHashTableAndReturnItem( top(stack), key);  if (currentDataItem != NULL)  {  duplicateIdInTheSameScope();  return;  }  dataItemThatReturnFromTheHashtable = searchInsideHashTableIfTheSubTypeExist(KeyOfVariableType));  if (dataItemThatReturnFromTheHashtable == NULL ){  printTypeDefined();}  else if (dataItemThatReturnFromTheHashtable->m\_Data->role == Variable){  Error();}  else  {  insert(top(stack), tempDataItem);  } |
| **type** type\_name **is** TYPE\_INIDICATOR | DataItem\* currentDataItem = searchInsideHashTableAndReturnItem(top(stack), type\_name\_key);  if (currentDataItem != NULL)  {  duplicateIdInTheSameScope()  } |
| TYPE\_INDICATOR ->  BASIC\_TYPE |  ARRAY\_TYPE |  POINTER\_TYPE |  |
| BASIC\_TYPE -> **integer | real**  POINTER\_TYPE'-> BASIC\_TYPE  ARRAY\_TYPE -> **array** [SIZE] **of** BASIC\_TYPE | insert(top(stack), tempDataItem); |
| POINTER\_TYPE'-> type\_name | dataItemThatReturnFromTheHashtable = searchInsideHashTableIfTheSubTypeExist(type\_name\_key));  if (dataItemThatReturnFromTheHashtable == NULL)  {  printTypeDefined();  }  else if (dataItemThatReturnFromTheHashtable->m\_Data->role == Variable)  {  Unexpected Variable();  }  else  {  insert(top(stack), tempDataItem);  } |
| COMMANDS -> COMMANDS | COMMANDS'  COMMANDS'-> COMMAND | COMMANDS'  COMMANDS'-> ᵋ |  |
| Command -> RECEIVER = EXPRESSION |  **when**(EXPRESSION rel\_op EXPRESSION) **do** commands; **default**  COMMANDS; end**\_**when |  **for**(id=EXPRESSION;id rel\_op EXPRESSION;id++) COMMANDS; **end\_for** |  id= **malloc**(**size\_of**(type\_name)) |  **free**(id) |  BLOCK |  |
| **free**(id) | currentItem = searchInsideHashTableIfTheSubTypeExist(id.key));  if (currentItem == NULL)  {  printVarNotDefined;  return;  }  else if (id.role== UserDefinedType)  {  printExpectedTypeOrVar();  return;  }  if (currentItem->m\_Data-> id.Catergory!= POINTER)  {  printError();  } |
| id= **malloc**(**size\_of**(type\_name)) | id.Category=pointer && id.type=type\_name |
| RECEIVER = EXPRESSION |  |
| RECEIVER 🡪 id RECEIVER' | checkIfIdExistInTheStack(id); |
| RECEIVER'🡪 [EXPRESSION] | forAllExpressionInsideBracketCheckIfInteger(EXPRESSION) |
| RECEIVER'🡪 ^ | id.Category=ptr |
| RECEIVER'🡪 ԑ |  |
| EXPRESSION 🡪  int\_num | real\_num |  id EXPRESSION' | &id | **size\_of**(type\_name) |  |
| int\_num | id.Type=integer |
| real\_num | id.Type=real |
| &id | id.Category=ptr && without '^'  id.Type=id.Type |
| **size\_of**(type\_name) | isTypeDefinition(type\_name);  id.Type= type\_name.Type |
| id EXPRESSION'🡪 ԑ | id==Variable && isIdInTheStack(id) && d.Type=id.Type |
| id EXPRESSION'🡪 [EXPRESSION] | id==Variable && isIdInTheStack(id) && d.Type=id.Type  forAllExpressionInsideBracketCheckIfInteger(EXPRESSION) |
| id EXPRESSION'🡪 ar\_op EXPRESSION | id==Variable && isIdInTheStack(id) && id.Type=id.Type |
| id EXPRESSION'🡪 ^ | id==Variable && isIdInTheStack(id) && id.Type=id.Type |
| **for** (id = EXPRESSION;  id rel\_op EXPRESSION  id++)  COMMANDS;  **end\_for** | id==Variable && isIdInTheStack(id) && id.Type=Expression.Type  id==Variable && isIdInTheStack(id) && id.Type=id.Type  id==Variable && isIdInTheStack(id) && id.Type=integer  \*see Commands\* |
| **when** (EXPRESSION rel\_op EXPRESSION)  **do** COMMANDS;  **default** COMMANDS;  **end\_when** | \*see expression REL\_OP \*see Expression  \*see Commands\* |