Universidad Rafael Landívar
Tradición Jesuita en Guatemala

Facultad de Ingeniería
Compiladores
Segundo Ciclo 2018
Mgtr. Diana Gutiérrez

Tercer Examen Parcial

Nombre: Mynor Ottoniel Xico Tzian **Carné:** 1051916

No	Producciones	Reglas semánticas
1	Program → ListDecl	
2	ListDecl → Stmt	
3	ListDecl → $ListDecl_1$ Stmt	
4	Stmt → VarDecl	
5	Stmt → Asign	
6	Stmt → IfStmt	

7	VarDecl → Type ListID Units **;**	{ListID.dType = Type.dType;
		ListID.uType = Units.uType;
		ListID.Factor= Units.Factor;
		`if(CheckIdList(ListID)){ // Ya se definió un valor`
		`Error('Ya se encuentra definido un identificador con ese nombre');`
		`}else{`
		`for(int i = 0; i < ListID.length; i++){`
		`Identificador ident = new Identificador();`
		`ident.dType = Type;`
		`ident.uType = Units.uType;`
		`ident.Factor = Units.Factor;`
		`AddId(ident);`
		`}`
		`}`
		}

8	Type → **entero**	{Type.dType = int; }
9	Type → **real**	{Type.dType = real; }
10	ListID → **ID**	{ID.dType = ListID.dType; ID.uType = ListID.uType; ID.Factor = ListID.Factor }
11	ListID → **ID ,** $ListID_1$	{ID.dType = ListID.dType; ID.uType = ListID.uType ; ID.Factor = ListID.Factor $ListID_1$.dType = ListID.dType; $ListID_1$.uType = ListID.uType ; $ListID_1$.Factor = ListID.Factor; }
12	Units → **unidades** UnitType	{Units.Factor = UnitType.Factor; Units.uType = UnitType.uType}
13	Units → ε	{Units.Factor = NULL; Units.uType = NULL}
14	UnitType → **mt**	{UnitType.Factor = 1; UnitType.uType = size}
15	UnitType → **dm**	{UnitType.Factor = 0.1; UnitType.uType = size}
16	UnitType → **cm**	{UnitType.Factor = 0.01; UnitType.uType = size}
17	UnitType → **mm**	{UnitType.Factor = 0.001; UnitType.uType = size}

18	UnitType → **rad**	{UnitType.Factor = 1; UnitType.uType = angule}
19	UnitType → **grad**	{UnitType.Factor = π/180; UnitType.uType = angule}

20	Asign → **ID =** Exp **;**	{
		`if (!CurrentScopesContainsId(ID)){ Error('El identificador ' + ID.name + 'no se encuentra en el contexto actual'); return; }`
		if (ID.uType == Exp.uType) {
		if (ID.Factor == Exp.Factor) {
		if (ID.dType == Exp.dType) {
		ID.value = Exp.value;
		Asign.Factor = Exp.Factor;
		Asign.uType = Exp.uType;
		Asign.dType = Exp.dType;
		Asign.value = Exp.value}
		else { Error("No se puede hacer asignación entre diferentes tipos de dato") }
		}
		else { Error("No se puede hacer asignación entre diferentes medidas, ¿tal vez necesite realizar una conversión antes?") }
		}
		else { Error("No se puede hacer asignación entre diferentes unidades de medida") }
		
		}

21.	Exp → Exp_1 **+** Exp_2	{
		if (Exp_1.uType == Exp_2.uType) {
		if (Exp_1.Factor == Exp_2.Factor) {
		if (Exp_1.dType == Exp_2.dType) {
		Exp.Value = Exp_1.Value + Exp_2.Value;
		Exp.Factor = Exp_1.Factor;
		Exp.uType = Exp_1.uType;
		Exp.dType = Exp_1.dType; }
		else { Error("No se puede hacer sumatoria entre diferentes tipos de dato") }
		}
		else { Error("No se puede hacer sumatoria entre diferentes medidas, ¿tal vez necesite realizar una conversión antes?") }
		}
		else { Error("No se puede hacer sumatoria entre diferentes unidades de medida") }
		}

22	Exp → Exp_1 **-** Exp_2	{
		if (Exp_1.uType == Exp_2.uType) {
		if (Exp_1.Factor == Exp_2.Factor) {
		if (Exp_1.dType == Exp_2.dType) {
		Exp.Value = Exp_1.Value - Exp_2.Value;
		Exp.Factor = Exp_1.Factor;
		Exp.uType = Exp_1.uType;
		Exp.dType = Exp_1.dType; }
		else { Error("No se puede hacer sustracción entre diferentes tipos de dato") }
		}
		else { Error("No se puede hacer sustracción entre diferentes medidas, ¿tal vez necesite realizar una conversión antes?") }
		}
		else { Error("No se puede hacer sustracción entre diferentes unidades de medida") }

			}
23	Exp →	Exp_1 **/** Exp_2	{ if (Exp_1.uType == Exp_2.uType) { if (Exp_1.Factor == Exp_2.Factor) { if (Exp_1.dType == Exp_2.dType) { Exp.Value = Exp_1.Value / Exp_2.Value; Exp.Factor = NULL; Exp.uType = NULL; Exp.dType = Exp_1.dType; } else { Error("No se puede hacer división entre diferentes tipos de dato") } } else { Error("No se puede hacer división entre diferentes medidas, ¿tal vez necesite realizar una conversión antes?") } } else { Error("No se puede hacer división entre diferentes unidades de medida") } }
24	Exp →	Cast	{ Exp.value = Cast.value; Exp.dType = Cast.dType Exp.uType = Cast.uType; Exp.Factor = Cast.Factor }
25	Exp →	**(** Exp_1 **)**	{ Exp.value = Exp_1.value; Exp.dType = Exp_1.dType Exp.uType = Exp_1.uType; Exp.Factor = Exp_1.Factor }
26	Exp →	**ID**	{ `if (!CurrentScopesContainsId(ID)){ Error('El identificador ' + ID.name + 'no se encuentra en el contexto actual'); return; }` Exp.value = ID.value; Exp.dType = ID.dType Exp.uType = ID.uType; Exp.Factor = ID.Factor }
27	Exp →	**num_int**	{ Exp.value = lexema(num_int); Exp.dType = int; Exp.uType = NULL; Exp.Factor = NULL; }
28	Exp →	**num_real**	{ Exp.value = lexema(num_real); Exp.dType = real Exp.uType = NULL; Exp.Factor = NULL; }
29	Cast →	UnitType **(**Exp**)**	{ if (UnitType.uType == Exp.uType OR Exp.uType == NULL) { if (Exp.uType != NULL) { Exp.value = (Exp.value * Exp.Factor) / UnitType.Factor; } Exp.Factor = UnitType.Factor; Cast.uType = UnitType.uType; Cast.dType = UnitType.dType; Cast.value = Exp.value Cast.Factor = UnitType.Factor; } else { Error("No se puede hacer conversión de tipos entre diferentes unidades de medida") } }
30	Comp →	Exp_1 **>** Exp_2	{

		if (Exp_1.uType == Exp_2.uType) {
		if (Exp_1.Factor == Exp_2.Factor) {
		if (Exp_1.dType == Exp_2.dType) {
		If(Exp_1.Value > Exp_2.Value)
		{ Comp.Value = 1 }
		else{ Comp.Value = 0 }
		}
		Comp.Factor = Exp_1.Factor;
		Comp.uType = Exp_1.uType;
		Comp.dType = Exp_1.dType; }
		else { Error("No se puede comparar entre diferentes tipos de dato") }
		}
		else { Error("No se puede comparar entre diferentes medidas, ¿tal vez necesite realizar una conversión antes?") }
		}
		else { Error("No se puede comparar entre diferentes unidades de medida") }
		}

31	Comp → Exp_1 **==** Exp_2	{
		if (Exp_1.uType == Exp_2.uType) {
		if (Exp_1.Factor == Exp_2.Factor) {
		if (Exp_1.dType == Exp_2.dType) {
		If(Exp_1.Value == Exp_2.Value)
		{ Comp.Value = 1 }
		else{ Comp.Value = 0 }
		}
		Comp.Factor = Exp_1.Factor;
		Comp.uType = Exp_1.uType;
		Comp.dType = Exp_1.dType; }
		else { Error("No se puede comparar entre diferentes tipos de dato") }
		}
		else { Error("No se puede comparar entre diferentes medidas, ¿tal vez necesite realizar una conversión antes?") }
		}
		else { Error("No se puede comparar entre diferentes unidades de medida") }
		}

32	IfStmt →
	si Comp **entonces** ListDecl Else **fin_si**
		{
		if (Comp.Value == 1) { GOTO: ListDecl}
		else {GOTO Else}
		}

33	Else → **si_no** ListDecl	endCurrentScope();|
34	Else → ε	

```
struct Scope{
    ArrayList<String> IdList = new ArrayList<String>();
}

ArrayList<Scope> ScopeStack = new ArrayList<Scope>();

public void SetCurrentScope(int i){
    ScopeStack.Add(i);  // Se mantienen los scopes anteriores
}
public void endCurrentScop(){
    ScopeStack.RemoveAt(ScopeStack.size()-1);   // Se elimina el último scope al que entró
}

public bool ScopeContainsID(String Id, Scope s){
    for(int i = 0; i < s.IdList.size(); i++){
        if(s.IdList[i] == IdList){
            return true;
        }
    }
    return false;
}
public bool CurrentScopesContainsId(String id){
    foreach(Scope s in ScopeStack){
        if(ScopeContainsID(id, s)){
            return true;
        }
    }
    return false;
}

public bool CheckIdList(String[] IdList){
    foreach(int i = 0;  i < IdList.length; i++){
        if(!(ScopeContainsID(s))
           return false;
    }
    return true;
}

public bool AddId(String id){
    ScopeStack[ScopeStack.size()-1].IdList.Add(id); // Se añade un nuevo símbolo
}
```