6.14

Al introducirse en la pila de valores cada uno de los elementos que se reducen a var-list la posición de la pila nunca será estable. A menos que se le retiren las reglas de copiado en la pila la posición de *dtype* en la pila no será fija.

6.20

suma () es la función que suma dos elementos
resta () es la función que resta dos elementos
división () es la división para números flotantes
mult () es la función para multiplicar dos numeros

```
exp -> exp + exp {
        if($1.type == float || $3.type == float){
                FLOAT($1);
                FLOAT($3);
        }
        $$.type = $1.type;
        suma($1, $3);
}
exp -> exp - exp {
        if($1.type == float || $3.type == float){
                FLOAT($1);
                FLOAT($3);
        }
        $$.type = $1.type;
        resta($1, $3);
}
exp -> exp * exp {
        if($1.type == float || $3.type == float){
                FLOAT($1);
                FLOAT($3);
        }
        $$.type = $1.type;
        mult($1, $3);
}
exp -> exp / exp {
        if($1.type == float || $3.type == float){
                FLOAT($1);
                FLOAT($3);
                division($1, $3);
        }
        else{
                div($1, $3);
        }
        $$.type = $1.type;
}
```

```
exp -> ( exp ) {
	$$.type = $2.type;
}
exp -> num {
	$$.type = $1.type;
}
exp -> num.num {
	if($1.type == float || $3.type == float)    {
		FLOAT($1);
		FLOAT($3);
	}
	$$.type = $1.type;
	mult($1, $3);

}
```

6.21

a)

La estructura sería un árbol compuesto de 2 nodos, uno que contiene todos los tipos de todos los parámetros de la función y el otro nodo seria una hoja que contiene el valor que retornara la función.

```
struct funType
{
	vector<Type> vars;
	string name;
	Type ret_val;
}

bool typeEqual(funType f1, funType f2)
{
	bool retVal = false, cicleVer = true;
	if(f1.vars.size() == f2.vars.size() && f1.ret_val == f2.ret_val)
	{
		for(int x = 0; x < f1.vars.size(); x++)
		{
			if(f1.vars[x] != f2.vars[x])
			{
				cicleVer = false;
				break;
			}
		}
		retVal = cicleVer;
	}
	return retVal;
}
```

b)

```
Fun-decls -> fun id (var-delcs):type-exp;cuerpo{
        $2.ret_val = $7.ret_val;
        $2.vars = $4.vars;
        $2.vars.push_back($4.ret_val);
        Insert($2.name, Type::function;
}
Exp -> id(exps){
        If(checkType($1.name) == Type::function && validateFunction($1, $3))
        {
                $$.ret_val = $1.ret_val;
        }
        Else {
                typeError();
        }
}
```

6.22

a)

Para eliminar esta ambigüedad en sintaxis se buscaría en la tabla de símbolos cual seria el tipo de A, si A es una variable que se encuentra en la tabla de símbolos se procede con la operación de resta, mientras que si esta fue definida como un tipo se procederá con un casting, esta modificación seria hecha en la regla gramática para operaciones, donde la acción semántica dependería de este chequeo.

b)

No creo que el lexer sea capaz de corregir esta ambiguedad ya que el lexer desconoce cuales son los tipos de las variables o incluso que operación se esta realizando, y por lo tanto sale del alcance de los problemas que soluciona el lexer.