Comprobaciones

Diego Díaz
Compilador Lenguaje C

Contenido

Introducción

En la primera parte de este documento se revisarán las comprobaciones semánticas dirigidas por la sintaxis, después la verificación de que la tabla de símbolos y de tipos están implementadas correctamente en todos los ámbitos; por último se comprueba el funcionamiento de los test de los programas propuestos por el equipo docente y de otros programas que verifican el buen funcionamiento de otras especificaciones del lenguaje.

1. Comprobaciones semánticas.

Se revisan los errores semánticos en tiempo de compilación, se debe obtener por la salida un mensaje que indique el fallo ocurrido. Se han ampliado los mensajes con la clase semanticErrors dentro del paquete compiler.zextra y un método wraper incluido en el fichero Parser.cup en la zona del *"action code".* Se ha añadido una nueva batería de programas *"semanticoX.p"* en la librería *"cuned>doc>test>Ctrl-Semantico",* que se utilizan con este propósito.

Se pueden repetir los símbolos de cualquier índole siempre y cuando **no** estén en el mismo ámbito, es por ello que los errores deben de aparecer en declaraciones hechas en un ámbito común.

1.1. Declaración Constantes

Esta comprobación solamente se hace en el ámbito global que es donde se pueden declarar las constantes.

Detecta duplicidad de simbolo (identificador de cte)

Código de "semantico1.cuned"	Acción del analizador semántico
#define nombre 9; *#define nombre 9;* *void main() {* *}*	**Salida:** "Constante ya declarada con ese identificador"

1.2. Declaración Tipo Registro

Se hacen las mismas comprobaciones de las declaraciones de Tipo Registro en todos los ámbitos posibles (función main, funciones y bloques de sentencias). Se pone el código de ejemplo del ámbito global.

Detecta duplicidad de símbolo del identificador del tipo estructurado con una constante ya declarada.

Código de "semantico2.cuned"	Acción del analizador semántico
#define nombre 9; *struct nombre {* *int edad;* *}* *void main() {* *}*	**Salida:** " El identificador ya ha sido utilizado al declarar una constante "

Global	OK	Función Principal	No procede	Otras funciones	No procede	Bloque	No procede

Detecta si el tipo ya ha sido declarado.

Código de "semantico2.cuned"	Acción del analizador semántico
struct nombre { *int edad;* *}*	**Salida:** "Tipo duplicado"

struct nombre { *int dni;* *}* *void main() {* *}*	

Global	OK	Función Principal	OK	Otras funciones	OK	Bloque	OK

Detecta si el identificador de campo del registro ya ha sido declarado

Código de "semantico2.cuned"	Acción del analizador semántico
struct nombre { *int dni, dni;* *}* *void main() {* *}*	**Salida:** " El identificador para el campo ya ha sido utilizado "

Global	OK	Función Principal	OK	Otras funciones	OK	Bloque	OK

Detecta si el identificador de campo del registro ya ha sido declarado (versión 2)

Código de "semantico2.cuned"	Acción del analizador semántico
struct nombre { *int dni;* *int dni;* *}* *void main() {* *}*	**Salida:** " El identificador para el campo ya ha sido utilizado "

Global	OK	Función Principal	OK	Otras funciones	OK	Bloque	OK

Más de dos anidamientos no está permitido en las declaraciones anidadas de registros.

Código de "semantico10.cuned"	Acción del analizador semántico
struct Tpersona {	

int dni; *int edad;* *}* *struct Tfecha {* *int dia, mes, anyo;* *}* *struct Tcita {* *Tpersona empleado;* *Tfecha fecha;* *}* *struct TCitaDoble {* *Tcita cita;* *}* *void main() {* *}*	**Salida:** " No se permite más de 2 anidamientos en la declaración de registros "

Global	OK	Función Principal	OK	Otras funciones	OK	Bloque	OK

El identificador de tipo del campo no ha sido definido previamente en ese mismo ámbito.

Código de "semantico11.cuned"	Acción del analizador semántico
struct Tcita { *Tfecha fecha;* *}* *void main() {* *}*	**Salida:** " Tipo no declarado"

Global	OK	Función Principal	OK	Otras funciones	OK	Bloque	OK

Asignación de los campos se verifica que el campo no pertenece al tipo

Código de "semantico11.cuned"	Acción del analizador semántico
void main (){ *struct Tfecha {* *int dia, mes, anyo;* *}*	**Salida:** " Símbolo no declarado"

Tfecha f; *f.hola=13;* *f.mes=4;* *f.anyo=2017;* *}*	

Global	OK	Función Principal	OK	Otras funciones	OK	Bloque	OK

Asignación de los campos se verifica que el campo no pertenece al tipo (Versión 2)

Código de "semantico11.cuned"	Acción del analizador semántico
void main (){ *struct Tfecha {* *int dia, mes, anyo;* *}* *Struct Cumple{* *Tfecha nacimiento;* *}* *Cumple nace;* *Tfecha f;* *nace.nacimiento.dia = 22;* *nace.incorrecto.dia = 23;* *f.hola=13;* *f.mes=4;* *f.anyo=2017;* *}*	**Salida:** " Símbolo no declarado"

Global	OK	Función Principal	OK	Otras funciones	OK	Bloque	OK

1.3. Declaración Variables

Se hacen las mismas comprobaciones de las declaraciones de las variables en todos los ámbitos posibles. Se pone el código de ejemplo del ámbito global.

Detecta que el identificador que se quiere utilizar para declarar la variable ya ha sido previamente utilizado para declarar una constante.

Código de "semantico2.cuned"	Acción del analizador semántico
#define nombre 8; *int nombre;* *void main() {* *}*	**Salida:** " Símbolo duplicado "

Global	OK	Función Principal	No procede	Otras funciones	No procede	Bloque	No procede

Detecta que el identificador ya se ha utilizado para declarar un tipo

Código de "semantico2.cuned"	Acción del analizador semántico
struct nombre{ *int diego;* *}* *int nombre;* *void main() {* *}*	**Salida:** " Identificador ya ha sido utilizado para declarar un tipo"

Global	OK	Función Principal	OK	Otras funciones	OK	Bloque	OK

Detecta que el identificador que se quiere utilizar para declarar la variable ya ha sido previamente utilizado para declarar otra variable.

Código de "semantico2.cuned"	Acción del analizador semántico
int nombre; *int nombre;* *void main() {* *}*	**Salida:** " Símbolo duplicado "

Global	OK	Función Principal	OK	Otras funciones	OK	Bloque	OK

Detecta que el identificador que se quiere utilizar para declarar la variable ya ha sido previamente utilizado para declarar otra variable (versión 2).

Código de "semantico2.cuned"	Acción del analizador semántico
int nombre, nombre; *void main() {* *}*	**Salida:** " Símbolo duplicado "

Global	OK	Función Principal	OK	Otras funciones	OK	Bloque	OK

Detecta que el tipo no ha sido declarado en ese mismo ámbito.

Código de "semantico8.cuned"	Acción del analizador semántico
tpersona a; *void main() {* *}*	**Salida:** "Tipo no declarado"

Global	OK	Función Principal	OK	Otras funciones	OK	Bloque	OK

Inicialización de variable incorrecta. Detecta que la inicialización no corresponde con una constante (o que dicha constante no se ha declarado anteriormente)

Código de "semantico8.cuned"	Acción del analizador semántico
struct Tpersona { *int dni;* *int edad;* *}* *int a = Tpersona;* *void main() {* *}*	**Salida:** "Valor no válido para asignación en la declaración de la variable"

Global	OK	Función Principal	OK	Otras funciones	OK	Bloque	OK

1.4. Declaración Funciones

Detecta duplicidad de símbolo del identificador de la función con una constante ya declarada.

Código de "semantico2.cuned"	Acción del analizador semántico
#define ocho 8; *int ocho(){* *return 8;* *}* *void main() {* *}*	**Salida:** " El identificador de función/procedimiento ya está siendo utilizado "

Detecta duplicidad de símbolo del identificador de la función con un tipo ya declarado.

Código de "semantico2.cuned"	Acción del analizador semántico
struct ocho{ *int a;* *}* *int ocho(){* *return 8;* *}* *void main() {}*	**Salida:** " El identificador de función/procedimiento ha sido declarado anteriormente por un tipo "

Detecta duplicidad de símbolo del identificador de la función con otra función ya declarada.

Código de "semantico2.cuned"	Acción del analizador semántico

int ocho(){
 	
return 8;
 	
}
 	
int ocho(){
return 7;
}
 	
void main() {}
	Salida: " El identificador de función/procedimiento ya está siendo utilizado "

Detecta duplicidad de símbolo del identificador del parámetro de entrada con la declaración de un tipo.

Código de "semantico2.cuned"	Acción del analizador semántico
int ocho (int nombre){ struct nombre{	**Salida:** " El identificador ya ha sido utilizado al declarar una constante/parámetro "

int a;
}
return 8;
}
 	
void main() {
 	
}	

Detecta duplicidad de símbolo del identificador del parámetro de entrada con la declaración de una variable.

Código de "semantico2.cuned"	Acción del analizador semántico

int ocho (int nombre){
int nombre;
 	
return 8;
}
 	
void main() {
 	
}
	Salida: " Símbolo duplicado"

No se encuentra la instrucción de retorno en función

Código de "semantico13.cuned"	Acción del analizador semántico
int devuelve2 (){ *}* *void main() {* *}*	**Salida:** "No existe instrucción de retorno"

La instrucción de retorno

Código de "semantico13.cuned"	Acción del analizador semántico
int amigo2(){ *return hola;* *}* *void main() {* *}*	**Salida:** " EL operador/retorno no ha sido declarado previamente "

Tipos incompatibles en la instrucción de retorno. Detecta que el tipo de la instrucción de retorno no corresponde con el declarado en la función (será siempre entero o void)

Código de "semantico14.cuned"	Acción del analizador semántico
int devuelve2 (){ *struct Tpersona {* *int dni;* *int edad;* *}* *Tpersona persona;* *return persona;* *}* *void main() {* *}*	**Salida:** " La instrucción de retorno no devuelve un tipo entero (cte, var entera o expresión resta)"

Si el RETURN devuelve una expresión, solamente será válida la Expresión RESTA.

Código de "semantico14.cuned"	Acción del analizador semántico
int mayorDeEdad(int a){ *return a>17;* *}* *void main() {* *}*	**Salida:** " La instrucción de retorno no devuelve un tipo entero (cte, var entera o expresión resta)"

Tipos incompatibles en la instrucción de retorno (Versión 2). Detecta que el tipo de la instrucción de retorno no corresponde con el declarado en la función (será siempre entero o void)

Código de "semantico14.cuned"	Acción del analizador semántico
void devuelveNada (){ *int persona;* *return persona;* *}* *void main() {* *}*	**Salida:** " La instrucción de retorno debe ser vacía en procedimientos/funciones VOID "

1.5. Expresiones y sentencias

1.5.1. Invocación de funciones.

Detecta que no hay una función con dicho nombre declarada

Código de "semantico3.cuned"	Acción del analizador semántico
void main() { *int a;* *int b;* *x = resta(a,b);* *}*	**Salida:** "La función no ha sido declarada previamente"

Detecta que algún parámetro no ha sido declarado (b)

Código de "semantico3.cuned"	Acción del analizador semántico
void main() { *int a;* *x = resta(a,b);* *}*	**Salida:** " EL operador/identificador/retorno no ha sido declarado previamente "

Numero de parámetros incorrecto. Detecta que el número de parámetros en la llamada al procedimiento no corresponde con su declaración.

Código de "semantico15.cuned"	Acción del analizador semántico
int resta (int a, int b){ *return a-b;* *}* *void main() {* *int resultado, uno, dos, tres;* *resultado = resta(uno,dos,tres);* *}*	**Salida:** " Numero incorrecto de parametros "

Tipo de parámetros no coincide con la declaración. Detecta que el tipo de los parámetros en la llamada al procedimiento no corresponde con su declaración.

Código de "semantico16.cuned"	Acción del analizador semántico
int resta (int a, int b){ *return a-b;* *}*	**Salida:** " Tipo del parámetro incorrecto "

void main() { *struct Tpersona {* *int dni;* *int edad;* *}* *int resultado, uno;* *Tpersona persona;* *resultado = resta(uno,persona);* *}*	

Detecta que la función invocada no devuelve un tipo entero (en una expresión siempre tiene que devolver tipo entero)

Código de "semantico16.cuned"	Acción del analizador semántico
void resta (int a, int b){ *return;* *}* *void main() {* *int resultado, uno, dos;* *resultado = resta(uno,dos);* *}*	**Salida:** " La función tiene que devolver un tipo entero "

Detecta que la función invocada en la sentencia no devuelve un tipo correcto (en sentencias siempre la función ha de devolver tipo Void)

Código de "semantico16.cuned"	Acción del analizador semántico
int resultado; *int x;* *int menosUno(int x){* *printi(x);*	**Salida:** " No se puede devolver un tipo entero en una sentencia "

return x-1; *}* *void main() {* *x=9;* *menosUno(x);* *}*	

1.5.2. Operadores

Tipos incompatibles en una expresión aritmética (resta)

Código de "semantico9.cuned"	Acción del analizador semántico
void main() { *struct Tpersona {* *int dni;* *int edad;* *}* *int a;* *x = a - Tpersona;* *}*	**Salida:** " Tipos incompatibles "

Tipos incompatibles en la expresión relacional (mayor que)

Código de "semantico16.cuned"	Acción del analizador semántico
void main() { *struct hola* *{* *int m;* *}* *int x;* *hola amigo;* *if (x > amigo)* *}*	**Salida:** " Tipos incompatibles "

Tipos incompatibles en la expresión relacional (igual)

Código de "semantico16.cuned"	Acción del analizador semántico
void main() { *struct hola* *{*	**Salida:** " Tipos incompatibles "

int m; *}* *int x;* *hola amigo;* *if (x == amigo)* *}*	

Tipos incompatibles en las expresiones lógicas (OR).

Código de "semantico16.cuned"	Acción del analizador semántico
void main() { *struct hola* *{* *int m;* *}* *int x;* *hola amigo;* *if (x || amigo)* *}*	**Salida:** " Tipos incompatibles "

Alguno de los operadores (operación aritmética, operación relacional u operación lógica) no ha sido declarado previamente

Código de "semantico16.cuned"	Acción del analizador semántico
void main() { *int x;* *if (x || amigo)* *}*	**Salida:** " EL operador no ha sido declarado previamente "

1.5.2. Asignaciones

Cualquier campo de la asignación no ha sido declarado

Código de "semantico16.cuned"	Acción del analizador semántico
int amigo2(){ *struct persona{* *int campo;* *}* *int amigo;* *int hola;*	**Salida:** " EL operador no ha sido declarado previamente "

a= hola; *return hola;* *}* *void main() {* *}*	

No se permite ese tipo de asignación

Código de "semantico16.cuned"	Acción del analizador semántico
void main (){ *struct Tfecha {* *int dia, mes, anyo;* *}* *Tfecha f;* *f = 5;* *}*	**Salida:** " Esta asignacion no esta permitida"

Se verifica que el identificador de la variable es de tipo Registro

void main (){
 	
struct Tfecha {
int dia, mes, anyo;
}
 	
int a;
 	
Tfecha f;
 	
nor.dia=6;
 	
}
	Salida: " Símbolo no declarado "

Se verifica que el identificador de la variable es de tipo Registro

void main (){
 	
struct Tfecha {
int dia, mes, anyo;
}
 	
int a;
 	
Tfecha f;
 	
a.dia=6;
 	
}
	Salida: " Tipo incorrecto "

Se verifica que el campo al que se va a hacer la asignación está contenido en el registro.

void main (){
 	
struct Tfecha {
int dia, mes, anyo;
}
 	
int a;
 	
Tfecha f;
 	
f.decenio=6;
 	
}
	Salida: " Símbolo no declarado "

// Asignación en registros ligados

Se verifica que la variable ha sido declarada

void main (){
 	
struct Tfecha {
int dia, mes, anyo;
}
 	
struct Tcita {
 	
Tfecha fecha;
int urgente;
}
 	
Tfecha f;
 	
Tcita c1, c2;
 	
nor.fecha.dia=6;
 	
}
	Salida: " Símbolo no declarado "

Se verifica que el identificador de la variable declarada es de tipo Registro

int a;
 	
void main (){
 	
struct Tfecha {
int dia, mes, anyo;
}
 	
struct Tcita {
 	
Tfecha fecha;
int urgente;
}
 	
Tfecha f;
 	
Tcita c1, c2;
 	
a.fecha.dia=6;
 	
}
	Salida: " Tipo incorrecto "

Se verifica que el identificador de en medio está contenido en el tipo Registro al que pertenece la variable declarada de la izquierda.

void main (){
 	
struct Tfecha {
int dia, mes, anyo;
}
 	
struct Tcita {
 	
Tfecha fecha;
int urgente;
}
 	
Tfecha f;
Tcita c1, c2;
 	
c1.nope.dia=6;
 	
}
	Salida: " El campo del registro no tiene declarado este campo "

Se verifica que el identificador de en medio es de tipo Registro

void main (){
 	
struct Tfecha {
int dia, mes, anyo;
}
 	
struct Tcita {
 	
Tfecha fecha;
int urgente;
}
 	
Tfecha f;
Tcita c1, c2;
 	
c1.urgente.dia=6;
 	
}
	Salida: " Tipo Incorrecto "

Se verifica que el campo está dentro del tipo Registro al que corresponde el identificador de en medio.

void main (){
 	
struct Tfecha {
int dia, mes, anyo;
}
 	
struct Tcita {
	Salida: " El campo del registro no tiene declarado este campo "

Tfecha fecha;
int urgente;
}
 	
Tfecha f;
Tcita c1, c2;
 	
c1.fecha.decenio=6;
 	
}	

1.5.3. Bucles (FOR)

El indice se debe declarar previamente

Código de "semantico16.cuned"	Acción del analizador semántico

/* Bucles FOR */
 	
void main() {
int a;
int c;
 	
printc("Vamos con los bucles");
 	
c=10;
 	
for (z=10 ; a>5; a=a-1){
 	
printc("Valor temporal indice");
printi(a);
 	
}
 	
printi(c);
}
	Salida: " Indice de control del bucle FOR no ha sido declarado "

El identificador se inicializa con un valor constante.

Código de "semantico16.cuned"	Acción del analizador semántico
/* Bucles FOR */ void main() { int a;	**Salida:** " Solamente se permite iniciar la variable con un valor entero valido "

int c;
 	
printc("Vamos con los bucles");
 	
c=10;
 	
for (a=c ; a>5; a=a-1){
 	
printc("Valor temporal indice");
printi(a);
 	
}
 	
printi(c);
}	

El control de salida se hace con una operación relacional (>)

Código de "semantico16.cuned"	Acción del analizador semántico

/* Bucles FOR */
 	
void main() {
int a;
int c;
 	
printc("Vamos con los bucles");
 	
c=10;
 	
for (a=10 ; a==5; a=a-1){
 	
printc("Valor temporal indice");
printi(a);
 	
}
 	
printi(c);
}
	Salida: " Solamente se permite expresion relacional (a>NUM) "

El índice se ha de decrementar (no hecho).

Código de "semantico16.cuned"	Acción del analizador semántico

El índice que se decrementa es el índice declarado. (no hecho).

Código de "semantico16.cuned"	Acción del analizador semántico

2. Tablas de símbolos y de tipos

En este apartado se comprueba el buen funcionamiento de la tabla de tipos y de símbolos. Se comprueba todo con un programa "*tablas.cuned*" que se encuentra en la carpeta *"cuned>doc>test> ".*[1]

2.1. Comprobaciones

En la parte izquierda de la tabla se puede ver el código de los programas utilizados y los resultados obtenidos.

2.2.1. Declaraciones en los diferentes ámbitos.

Se muestra como ejemplo el ámbito Global pero también se han hecho las verificaciones para los otros ámbitos (funciones, bloque sentencias, Función principal-Main)

Fragmento de código	Acciones en Tabla de Tipos / Tabla de símbolos
CONSTANTES	
#define nombre 9; void main() {}	** SYMBOL TABLES ** SYMBOL TABLE [global] - Symbol - SymbolConstantInteger [scope = global, name = nombre, type = Integer] - {entera=9}
TIPOS DEFINIDOS (REGISTRO)	
struct Tpersona { *int dni;* *int edad;* *}* void main(){}	** TYPE TABLES ** Type - TypeRecord [scope = global, name = Tpersona] - {camposTabla={edad=Symbol - SymbolVariable [scope = global, name = edad, type = Integer] - { desplazamiento=0, _direccion=null}, dni=Symbol - SymbolVariable [scope = global, name = dni, type = Integer] - { desplazamiento=1, _direccion=null}}, totales=2}

struct Tfecha {
int dia, mes, anyo;
}
 	
void main() {}
	** TYPE TABLES **
	Type - TypeRecord [scope = global, name = Tfecha] - {camposTabla={mes=Symbol - SymbolVariable [scope = global, name = mes, type = Integer] - { desplazamiento=1, _direccion=null}, dia=Symbol - SymbolVariable [scope = global, name = dia, type = Integer] - { desplazamiento=0, _direccion=null}, anyo=Symbol - SymbolVariable [scope = global, name = anyo, type = Integer] - { desplazamiento=2, _direccion=null}}, totales=3}

struct Tpersona {
int nombre;
}
	** TYPE TABLES **
	Type - TypeRecord [scope = global, name = Tcita] - {camposTabla={patron=Symbol - SymbolVariable [scope =

[1] En este archivo se va introduciendo el código mostrado en la tabla, es decir cada celda es un programa independiente.

struct Tcita {
Tpersona empleado, patron;
}
void main(){}
	global, name = patron, type = Tpersona] - { desplazamiento=1, _direccion=null}, empleado=Symbol - SymbolVariable [scope = global, name = empleado, type = Tpersona] - { desplazamiento=0, _direccion=null}}, totales=2}
	
	Type - TypeRecord [scope = global, name = Tpersona] - {camposTabla={nombre=Symbol - SymbolVariable [scope = global, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}

struct Tpersona { *int nombre;* *}* *struct Tcita {* *Tpersona empleado;* *}* void main(){}	** TYPE TABLES ** Type - TypeRecord [scope = global, name = Tcita] - {camposTabla={empleado=Symbol - SymbolVariable [scope = global, name = empleado, type = Tpersona] - { desplazamiento=0, _direccion=null}}, totales=1} Type - TypeRecord [scope = global, name = Tpersona] - {camposTabla={nombre=Symbol - SymbolVariable [scope = global, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}
VARIABLES	
int nombre; int dni; void main() {}	** SYMBOL TABLES ** Symbol - SymbolVariable [scope = global, name = nombre, type = Integer] - { desplazamiento=0, _direccion=2} Symbol - SymbolVariable [scope = global, name = dni, type = Integer] - {desplazamiento=0, _direccion=3}
int nombre, dni; void main() {}	** SYMBOL TABLES ** Symbol - SymbolVariable [scope = global, name = nombre, type = Integer] - {desplazamiento=0, _direccion=2} Symbol - SymbolVariable [scope = global, name = dni, type = Integer] - {desplazamiento=0, _direccion=3}

struct Tpersona {
int dni;
}
 	
Tpersona diegodiaz, jose;
 	
void main() {}
	** TYPE TABLES **
	Type - TypeRecord [scope = global, name = Tpersona] - {camposTabla={nombre=Symbol - SymbolVariable [scope = global, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}
	
	** SYMBOL TABLES **
	SYMBOL TABLE [global]
	
	Symbol - SymbolVariable [scope = global, name = jose, type = Tpersona] - {desplazamiento=0, _direccion=null}
	
	Symbol - SymbolVariable [scope = global, name = diego, type = Tpersona] - {desplazamiento=0, _direccion=2}

#define numeroClase 9;
 	
int edad = 2;
int clase = numeroClase;
	** SYMBOL TABLES **

void main() {}	
#define numeroClase 9; int edad = 2, clase = numeroClase; void main() {}	

2.2.2. Ocultación de símbolos en ámbitos diferentes

Solamente se muestran las líneas que muestran los símbolos repetidos que no deben generar error ya que son ámbitos diferentes.

Fragmento de código	Acciones en Tabla de Tipos / Tabla de símbolos
CONSTANTE - TIPO REGISTRO	

#define nombre 9;
 	
int resta(){
 	
struct nombre {
int dni;
}
 	
{
struct nombre {
int dni;
}
 	
}
 	
return 8;
 	
}
 	
void main() {
 	
struct nombre {
int edad;
}
 	
}
	** TYPE TABLES **
	
	Type - TypeRecord [scope = resta, name = nombre] - {camposTabla={dni=Symbol - SymbolVariable [scope = resta, name = dni, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}
	
	TypeRecord [scope = anonymous - 2, name = nombre] - {camposTabla={dni=Symbol - SymbolVariable [scope = anonymous - 2, name = dni, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}
	
	Type - TypeRecord [scope = main, name = nombre] - {camposTabla={edad=Symbol - SymbolVariable [scope = main, name = edad, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}
	
	** SYMBOL TABLES **
	SYMBOL TABLE [global]
	Symbol - SymbolConstantInteger [scope = global, name = nombre, type = Integer] - {entera=9}

CONSTANTE - VARIABLE	
#define nombre 9; int resta(){	** SYMBOL TABLES ** SYMBOL TABLE [global] Symbol - SymbolConstantInteger [scope = global, name =

int nombre;
 	
{
int nombre;
 	
}
 	
return 8;
 	
}
 	
void main() {
int nombre;
}
	nombre, type = Integer] - {entera=9}
	
	SYMBOL TABLE [resta]
	Symbol - SymbolVariable [scope = resta, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}
	
	SYMBOL TABLE [anonymous - 2]
	Symbol - SymbolVariable [scope = anonymous - 2, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}
	
	SYMBOL TABLE [main]
	Symbol - SymbolVariable [scope = main, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}

TIPO REGISTRO - TIPO REGISTRO	

struct nombre {
 	
int edad;
}
 	
int resta(){
 	
struct nombre {
int dni;
}
 	
{
struct nombre {
int dni;
}
 	
}
 	
return 8;
 	
}
 	
void main() {
 	
struct nombre {
int edad;
}
 	
}
	** TYPE TABLES **
	
	Type - TypeRecord [scope = global, name = nombre] - {camposTabla={edad=Symbol - SymbolVariable [scope = global, name = edad, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}
	
	Type - TypeRecord [scope = resta, name = nombre] - {camposTabla={dni=Symbol - SymbolVariable [scope = resta, name = dni, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}
	
	TypeRecord [scope = anonymous - 2, name = nombre] - {camposTabla={dni=Symbol - SymbolVariable [scope = anonymous - 2, name = dni, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}
	
	Type - TypeRecord [scope = main, name = nombre] - {camposTabla={edad=Symbol - SymbolVariable [scope = main, name = edad, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}

TIPO REGISTRO - VARIABLE	
struct nombre { int edad;	** TYPE TABLES ** Type - TypeRecord [scope = global, name = nombre] -

}
 	
int resta(){
 	
int nombre;
 	
{
int nombre;
}
 	
return 8;
 	
}
 	
void main() {
 	
int nombre;
 	
}
	{camposTabla={edad=Symbol - SymbolVariable [scope = global, name = edad, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}
	
	** SYMBOL TABLES **
	SYMBOL TABLE [resta]
	Symbol - SymbolVariable [scope = resta, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}
	
	SYMBOL TABLE [anonymous - 2]
	Symbol - SymbolVariable [scope = anonymous - 2, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}
	
	SYMBOL TABLE [main]
	Symbol - SymbolVariable [scope = main, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}

VARIALBLE - VARIABLE	

int nombre;
 	
int resta(){
 	
int nombre;
 	
{
int nombre;
 	
}
 	
return 8;
 	
}
 	
void main() {
 	
int nombre;
 	
}
	** SYMBOL TABLES **
	
	SYMBOL TABLE [global]
	Symbol - SymbolVariable [scope = global, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}
	
	SYMBOL TABLE [resta]
	Symbol - SymbolVariable [scope = resta, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}
	
	SYMBOL TABLE [anonymous - 2]
	Symbol - SymbolVariable [scope = anonymous - 2, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}
	
	SYMBOL TABLE [main]
	Symbol - SymbolVariable [scope = main, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}

VARIABLE - TIPO REGISTRO	

int nombre;
 	
int resta(){
 	
struct nombre {
int edad;
	** TYPE TABLES **
	
	Type - TypeRecord [scope = resta, name = nombre] - {camposTabla={dni=Symbol - SymbolVariable [scope = resta, name = dni, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}

}
 	
{
struct nombre {
int edad;
}
 	
}
 	
return 8;
 	
}
 	
void main() {
 	
struct nombre {
int edad;
}
 	
}
 	
}
	TypeRecord [scope = anonymous - 2, name = nombre] - {camposTabla={dni=Symbol - SymbolVariable [scope = anonymous - 2, name = dni, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}
	
	Type - TypeRecord [scope = main, name = nombre] - {camposTabla={edad=Symbol - SymbolVariable [scope = main, name = edad, type = Integer] - {desplazamiento=0, _direccion=null}}, totales=1}
	
	** SYMBOL TABLES **
	SYMBOL TABLE [global]
	Symbol - SymbolVariable [scope = global, name = nombre, type = Integer] - {desplazamiento=0, _direccion=null}

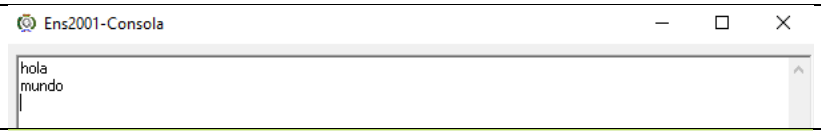
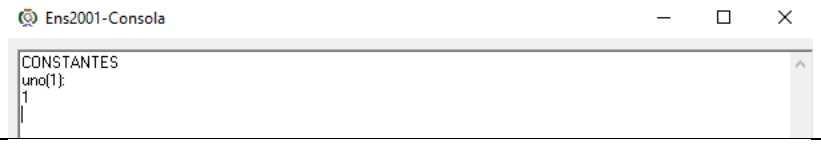
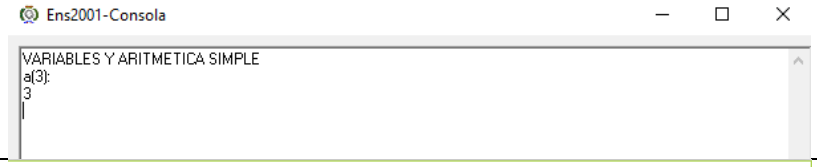
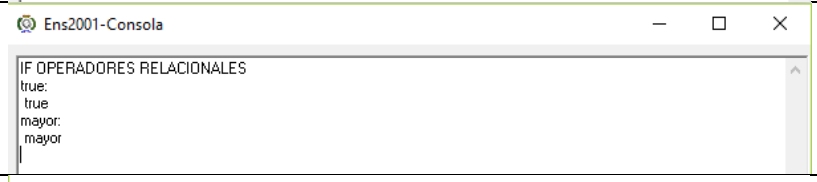
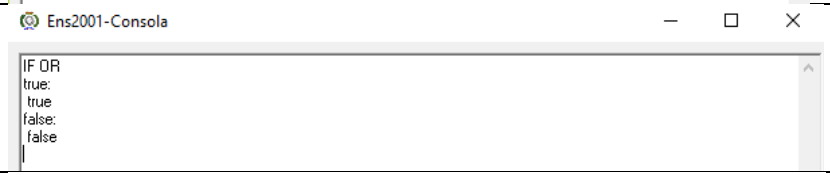
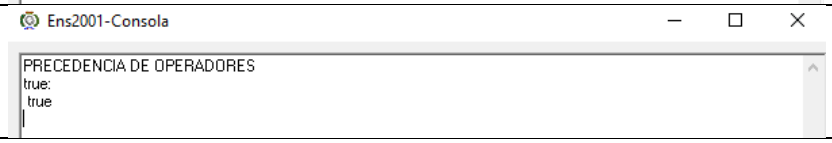
3. Programas completos

En este apartado se comprueba el buen funcionamiento de programas completos que incluyen expresiones y sentencias.

Al ejecutar la tarea Ant "finalTest" se obtienen los archivos con el código ensamblador. Se utiliza el emulador Ens2001 para interpretar dicho código.

3.1. Test propuestos por el equipo docente

El equipo docente "testCase0X.cuned" que se encuentran dentro de la carpeta *"cuned > doc > test".*

Código ensamblador	*Salida obtenida con emulador Ens2001*
testCase01.ens	*C:\Users\Diego\workspace\cuned\doc\test\testCase01.ens ensamblado correctamente*
testCase02.ens	
testCase03.ens	
testCase04.ens	
testCase05.ens	
testCase06.ens	
testCase07.ens	
testCase08.ens	

3.2. Otros programas

10. Inicialización de variables
11. Invocación de Funciones sin parámetros
12. Invocación de Funciones con parámetros
13. Invocación de Funciones con parámetros (2)
14. Invocación de Funciones con parámetros (3)
15. Sentencia que invoca función con parámetros, constantes y sentencias de selección.
16. Sentencia que invoca función con parámetros, la cual invoca otra función void
17. Asignación de campos de registro (con y sin anidamiento)
18. Asignación del valor de un campo registro a una variable entera
19. Asignación del valor de una variable inicializada a un campo registro
20. Asignación del valor de una variable a un campo registro
21. Bucles FOR
22. Recursividad