

Clase #24 de 27

Smalltalk vs C y Punteros

Octubre 31, Lunes

Agenda para esta clase

- Análisis comparativo de Smalltalk y C
- Síntesis de Arreglos, Punteros, y Funciones

Smalltalk vs C

Smalltalk vs C

Concepto	C	Smalltalk
Léxico		
Literales	1 1.0 'a' "abc"	1 1.0 \$a 'abc' #symbol \$()
Identificadores	abc	abc
Puntuación	Decenas...	() [] : . := ^
Sintaxis		
Expresiones	Complejas, 45 operadores	Cuatro variantes
Declaraciones	Sigue a las expresiones	var [:var ...
Sentencias	selección iteración expresión ...	No hay
Semántica		
Tipos	Básicos y derivados	Entendimiento de mensajes
Flujo de ejecución	Sentencias	Mensajes
Precendencia	Dada por la gramática	PUBKAR

Análisis comparativo: Sintaxis de Smalltalk en una postcard, por Ralph Johnson

```
exampleWithNumber: x
```

"A method that illustrates every part of Smalltalk method syntax except primitives. It has unary, binary, and key word messages, declares arguments and temporaries (but not block temporaries), accesses a global variable (but not and instance variable), uses literals (array, character, symbol, string, integer, float), uses the pseudo variable true false, nil, self, and super, and has sequence, assignment, return and cascade. It has both zero argument and one argument blocks. It doesn't do anything useful, though"

```
|y|
```

```
true & false not & (nil isNil) ifFalse: [self halt].
```

```
y := self size + super size.
```

```
#($a #a 'a' 1 1.0)
```

```
do: [:each | Transcript
```

```
    show: (each class name);
```

```
    show: (each printString);
```

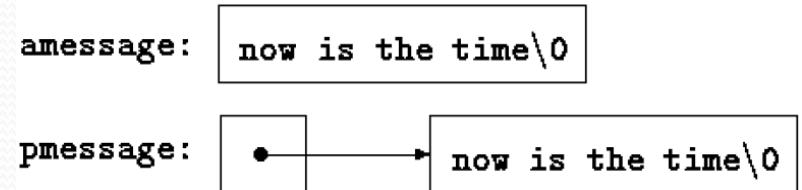
```
    show: ' '].
```

```
^ x < y
```

Síntesis de Arreglos, Punteros, y Funciones

Puntero a Caracteres y Funciones

```
char amessage[] = "now is the time"; /*an array*/  
char *pmessage = "now is the time"; /*a pointer*/
```



```
char a[4+1], b[]="holá", *c="chau";  
a=b      a=c      b=c      c=b    // ¿Cuáles son válidas?
```

```
// strcpy: copy t to s; array subscript version  
void strcpy(char *s, char *t){  
    int i;  
    i = 0;  
    while ((s[i] = t[i]) != '\0')  
        i++;  
}
```

```
/. strcpy: copy t to s; pointer version  
void strcpy(char *s, char *t)  
{  
    int i;  
    i = 0;  
    while ((*s = *t) != '\0') {  
        s++;  
        t++;  
    }  
}
```

```
// strcpy: copy t to s; pointer version 2  
void strcpy(char *s, char *t){  
    while ((*s++ = *t++) != '\0')  
        ;  
}
```

```
/* strcpy: copy t to s; pointer version 3,  
the Idiom */  
void strcpy(char *s, char *t){  
    while (*s++ = *t++)  
        ;  
  
<string.h>  
char *strcpy(char *s, const char *t);
```

Arreglos en Expresiones: La Regla y las Excepciones

- La regla: Una **expresión de tipo "arreglo de tipo"** se **convierte** en una **expresión con tipo "puntero a tipo"**,
- con **valor dirección del primer elemento** del arreglo y
- **no es un valor-l**
- Aplicaciones de la regla en **azul**
 1. int a[]={1,2,3}, *pa, i=2;
 2. char s[]{"abcd", *ps;
 3. a // &a[0]
 4. s // &s[0]
 5. a[i] ≈ *(a+i) ≈ *(i+a) ≈ i[a] // 3
 6. s[i] ≈ *(s+i) ≈ *(i+a) ≈ i[a] // 'c'
 7. pa=a, ps=s
 8. pa[i] ≈ *(pa+i) ≈ *(i+pa) ≈ i[pa]// 3
 9. ps[i] ≈ *(ps+i) ≈ *(i+ps) ≈ i[ps]// 'c'
 10. void f(int*); // recomendada
 11. // void f(int[]); // equivalente
 12. // void f(int[N]); // equivalente
 13. void g(char*); // recomendada
 14. // void g(char[]); // equivalente
 15. // void g(char[N]); // equivalente
 16. f(a), f(pa) , g(s) , g(ps)
 17. ps="YXZ"
 18. ps[i] ≈ *(ps+i) ≈ *(i+ps) ≈ i[ps] // 'z'
 19. "YXZ"[i] ≈ *(("YXZ"+i) ≈ *(i+"YXZ") ≈ i["YXZ"] // 'z'
- Las excepciones a la regla: En las siguientes situaciones **no se aplica la regla de conversión de tipo de la expresión**
 - (1) cuando es la **cadena literal** que **inicializa** un **arreglo**
 - ó (2) Cuando es el operando del **operador sizeof**
 - ó (3) del **operador unario &**
- Aplicaciones de la regla en **azul**, excepciones en **rojo**
 18. int a[]={1,2,3,4};
 19. char *ps="wxyz", s[]={abcd";
 20. sizeof ps //sizeof(char*)
 21. sizeof a //sizeof(int[4]) ≈ sizeof(int)*4
 22. sizeof s //sizeof(char[5])≈ sizeof(char)*5 ≈ 1*5 ≈5
 23. sizeof "1234" //sizeof(char[5])≈ sizeof(char)*5 ≈ 1*5 ≈5
 24. //Expr Type valores ejemplo de direcciones
 25. a // int* 100
 26. s // char* 132
 27. "1234" // char* 137
 28. &a // int(*)[4] 100
 29. &s // char(*)[5] 132
 30. &"1234" // char(*)[5] 137.



¿Consultas?



Fin de la clase

Clase #25 de 27

Sintaxis & Semántica y Expresiones

Nov 7, Lunes

Agenda para esta clase

- Sintaxis & Semántica
- Sintaxis de Expresiones de C

Ejemplo de Sintaxis y Semántica

Conectores Lógicos

Desde el Usuario Programador vs. Desde el Compilador

- MROC
 - Cada “constructo”
 - Sintaxis (Lenguaje Independiente del Contexto)
 - Restricciones (semánticas, Lenguaje Sensible al Contexto)
 - Semántica (comportamiento)
- Sintácticamente Correcto
 - Usuario programador
 - "Está bien escrito"
 - "No da errores"
 - Compilador
 - Es derivable según las reglas gramaticales de una Gramática Independiente del Conecto)
- Semánticamente Correcto
 - Compiladores
 - Cumple con las restricciones semánticas (Lenguaje Sensible al Contexto)

Ejemplo de S&S: Operador OR lógico

6.5.14 Logical OR operator

Syntax

logical-OR-expression:

logical-AND-expression

logical-OR-expression || *logical-AND-expression*

- Identificar:
 - Valor
 - Tipo
 - Efecto de lado
 - Asociatividad
 - Precedencia
 - Orden de Evaluación

Constraints

Each of the operands shall have scalar type.

Semantics

The || operator shall yield 1 if either of its operands compare unequal to 0; otherwise, it yields 0. The result has type `int`.

Unlike the bitwise | operator, the || operator guarantees left-to-right evaluation; there is a sequence point after the evaluation of the first operand. If the first operand compares unequal to 0, the second operand is not evaluated.

Ejemplo de S&S: Operador AND lógico

6.5.13 Logical AND operator

Syntax

logical-AND-expression:

inclusive-OR-expression

logical-AND-expression **&&** *inclusive-OR-expression*

Constraints

Each of the operands shall have scalar type.

Semantics

The **&&** operator shall yield 1 if both of its operands compare unequal to 0; otherwise, it yields 0. The result has type **int**.

Unlike the bitwise binary **&** operator, the **&&** operator guarantees left-to-right evaluation; there is a sequence point after the evaluation of the first operand. If the first operand compares equal to 0, the second operand is not evaluated.

- Identificar:
 - Valor
 - Tipo
 - Efecto de lado
 - Asociatividad
 - Precedencia
 - Orden de Evaluación

Sintaxis de Expresiones de C

Expresión (1/10)

Coma, Asignación y Condicional

expresión

expresión-de-asignación

expresión , expresión-de-asignación

expresión-de-asignación

expresión-condicional

expresión-unaria operador-de-asignación expresión-de-asignación

operador-de-asignación uno de

= *= /= %= += -= <<= >>= &= ^= |=

expresión-condicional

expresión-O-lógico

expresión-O-lógico ? expresión : expresión-condicional

Expresión (2/10)

Conectores lógicos

expresión-O-lógico

expresión-Y-lógico

expresión-O-lógico || *expresión-Y-lógico*

expresión-Y-lógico

expresión-O-inclusivo

expresión-Y-lógico && *expresión-O-inclusivo*

Expresión (3/10)

Operadores binarios a nivel bits

expresión-O-inclusivo

expresión-O-excluyente

expresión-O-inclusivo | *expresión-O-excluyente*

expresión-O-excluyente

expresión-Y

expresión-O-excluyente ^ *expresión-Y*

expresión-Y

expresión-de-igualdad

expresión-Y & *expresión-de-igualdad*

Expresión (4/10)

Igualdad y relación

expresión-de-igualdad

expresión-relacional

expresión-de-igualdad == *expresión-relacional*

expresión-de-igualdad != *expresión-relacional*

expresión-relacional

expresión-de-corrimiento

expresión-relacional < *expresión-de-corrimiento*

expresión-relacional > *expresión-de-corrimiento*

expresión-relacional <= *expresión-de-corrimiento*

expresión-relacional >= *expresión-de-corrimiento*

Expresión (5/10)

Corrimiento de bits

expresión-de-corrimiento

expresión-aditiva

expresión-de-corrimiento << *expresión-aditiva*

expresión-de-corrimiento >> *expresión-aditiva*

Expresión (6/10)

Adiciones y Multiplicaciones

expresión-aditiva

expresión-multiplicativa

expresión-aditiva + *expresión-multiplicativa*

expresión-aditiva - *expresión-multiplicativa*

expresión-multiplicativa

expresión-de-conversión

expresión-multiplicativa * *expresión-de-conversión*

expresión-multiplicativa / *expresión-de-conversión*

expresión-multiplicativa % *expresión-de-conversión*

Expresión (7/10)

Casteo

expresión-de-conversión

expresión-unaria

(*nombre-de-tipo*) *expresión-de-conversión*

Expresión (8/10) Unarios, pre-incremento y decremento, y sizeof

expresión-unaria

expresión-sufijo

++ *expresión-unaria*

-- *expresión-unaria*

operador-unario *expresión-de-conversión*

sizeof *expresión-unaria*

sizeof (*nombre-de-tipo*)

operador-unario uno de

& * + - ~ !

Expresión (9/10)

Operadores sufijos

expresión-sufijo

expresión-primaria

expresión-sufijo [*expresión*]

Subindicación

expresión-sufijo (*lista-de-argumentos?*)

Invocación

expresión-sufijo . *identificador*

Estructura

expresión-sufijo -> *identificador*

Puntero a estructura

expresión-sufijo ++

Posincremento

expresión-sufijo --

Posdecremento

lista-de-argumentos

expresión-de-asignación

lista-de-argumentos , *expresión-de-asignación*

Expresión (10/10)

Expresiones primarias

expresión-primaria

identificador

constante

constante-cadena

(expresión)



¿Consultas?



Fin de la clase

Clase #26 de 27

Declaraciones y

Parser Descendente Recursivo

Nov 14, Lunes

Agenda para esta clase

- Categorías Léxicas y Sintácticas
- Sintaxis de las Declaraciones C
- Parser

Categorías Léxicas y Sintácticas

Categorías

- Categorías Léxicas ó Tokens
 - Identificadores
 - Palabras reservadas
 - Literales
 - Enteros
 - Reales
 - Caracteres
 - Cadenas
 - Operadores y puntuación
 - Otros
- ¿Cuántos hay?
- Categorías Sintácticas ó Gramaticales ó Frases
 - Expresiones
 - Declaraciones
 - Sentencias
- ¿Cuántos hay?
- Frase principal
 - Unidad de Traducción

Sintaxis de las Declaraciones C

Declaraciones

Sintaxis y un poco de semántica

Conceptos principales

Forma general

Especificadores de declaración

struct

union

enum

typedef

Declaradores

Inicializadores

Bibliografía

MUCH

K&R

Declaraciones (1/11) –

Axioma

declaración

especificadores-de-declaración *lista-de-declaradores-inic?* ;

especificadores-de-declaración

calificador-de-tipo *especificadores-de-declaración?*

especificador-de-clase-de-almacenamiento *especificadores-de-declaración?*

especificador-de-tipo *especificadores-de-declaración?*

lista-de-declaradores-inic

declarador-inic

lista-de-declaradores-inic , *declarador-inic*

declarador-inic

declarador

declarador = *inicializador*

Declaraciones (2/11) – Almacenamiento y Tipo

especificador-de-clase-de almacenamiento

static
auto
register
extern
typedef

especificador-de-tipo

void
char
short
int
long
float
double
signed
unsigned
especificador-de-struct-o-union
especificador-de-enum
nombre-de-typedef

Declaraciones (3/11) – Estructuras y Uniones

especificador-de-struct-o-union

struct-o-union *identificador?* { *lista-de-declaraciones-struct* }

struct-o-union *identificador*

struct-o-union

struct

union

lista-de-declaraciones-struct

declaración-struct

lista-de-declaraciones-struct *declaración-struct*

Declaraciones (4/11) – Estructuras y Uniones (cont.)

declaración-struct

lista-de-especificadores-calificadores lista-de-declaradores-struct ;

lista-de-especificadores-calificadores

especificador-de-tipo lista-de-especificadores-calificadores?

calificador-de-tipo lista-de-especificadores-calificadores?

lista-de-declaradores-struct

declarador-struct

lista-de-declaradores-struct , declarador-struct

declarador-struct

declarador

declarador? : expresión-constante

Declaraciones (5/11) –

Enumeraciones

especificador-de-enum

enum *identificador?* { *lista-de-enumeradores* }

enum *identificador*

lista-de-enumeradores

enumerador

lista-de-enumeradores , *enumerador*

enumerador

constante-de-enumeración

constante-de-enumeración = *expresión-constante*

Declaraciones (6/11) – Calificadores de Tipo

calificador-de-tipo

const

volatile

```
const struct s { int mem; } cs = { 1 };
struct s ncs; /* the object ncs is modifiable */
typedef int A[2][3];
const A a = {{4,5,6},{7,8,9}}; /* array of array of const int */
int *pi;
const int *pci;

ncs = cs; /* valid */
cs = ncs; /* violates modifiable lvalue constraint for = */
pi = &ncs.mem; /* valid */
pi = &cs.mem; /* violates type constraints for = */
pci = &cs.mem; /* valid */
pi = a[0]; /* invalid: a[0] has type 'const int *' */
```

Declaraciones (7/11) – Declaradores

declarador

puntero? *declarador-directo*

declarador-directo

identificador

(*declarador*)

declarador-directo [*expresión-constante?*]

declarador-directo (*lista-tipos-parámetros*)

declarador-directo (*lista-de-identificadores?*)

puntero

* *lista-calificadores-tipos?*

* *lista-calificadores-tipos?* *puntero*

lista-calificadores-tipos

calificador-de-tipo

lista-calificadores-tipos *calificador-de-tipo*

Declaraciones (8/11) –

Declaradores: Parámetros Funciones

lista-tipos-parámetros

lista-de-parámetros

lista-de-parámetros , ...

lista-de-parámetros

declaración-de-parámetro

lista-de-parámetros , *declaración-de-parámetro*

declaración-de-parámetro

especificadores-de-declaración declarador

especificadores-de-declaración declarador-abstracto?

lista-de-identificadores

identificador

lista-de-identificadores , *identificador*

Declaraciones (9/11) – Nombre de tipo y declarador abstracto

nombre-de-tipo

lista-de-calificadores declarador-abstracto?

declarador-abstracto

puntero

puntero? declarador-abstracto-directo

declarador-abstracto-directo

(declarador-abstracto)

declarador-abstracto-directo? [expresión-constante?] ,
declarador-abstracto-directo? (lista-tipos-parámetros?)

Declaraciones (10/11) –

Typedef

nombre-de-typedef

identificador

- Ejemplos typedef
 - `typedef int *Avpi[20];`
 - `Avpi a;`
 - `sizeof (Avpi)`
 - `typedef struct Punto{double x, y} Punto;`
 - ó
 - `typedef struct {double x, y} Punto;`
 - `struct Punto p;`
 - .
- Es diferente a *nombre-de-tipo*
- Ejemplos de nombre-de-tipo
 - `struct Punto {double x, y};`
 - `struct Punto p;`
 - `int *a[20];`
 - `sizeof (int *[20])`

Declaraciones (11/11) –Inicialización

inicializador

expresión-de-asignación

{ *lista-de-inicializadores* }

{ *lista-de-inicializadores* , }

lista-de-inicializadores

inicializador

lista-de-inicializadores , *inicializador*

constante-de-enumeración

identificador

Glosario

- Especificadores de declaración
- Especificadores de clase de almacenamiento
- Especificadores de tipo
- Calificadores de tipo
- Declarador
- Inicializador
- **struct**
- **union**
- **enum**
- **typedef**

Ejemplo de Parser

K&R1988 5.12 Declaraciones Complicadas
MUCH2012 v2s3.2.5 UN PARSER PARA MICRO

Declaraciones complicadas

`int *f();`

f: function returning pointer int (*daytab)[13]

to int

daytab: pointer to array[13]

of int

`int (*pf)();`

pf: pointer to function
returning int

`int *daytab[13]`

daytab: array[13] of pointer
to int

`void *comp()`

comp: function returning
pointer to void

`char (*(*x())[])()`

x: function returning pointer
to array[] of pointer to
function returning char

`void (*comp)()`

comp: pointer to function
returning void

`char (*(*x[3])())[5]`

x: array[3] of pointer to
function returning pointer to
array[5] of char.

`char **argv ;`

argv: pointer to pointer to
char



¿Consultas?



Fin de la clase

Clase #27 de 27

Evaluación y Cierre de Cursada

Nov 21, Lunes

Agenda para esta clase

- Examen y resolución
- Cierre de curso

Apellido _____ Legajo _____ Equipo _____

- No se responden consultas durante la evaluación; por lo tanto, de ser necesario, escriba hipótesis de trabajo, las cuales también serán evaluadas.
- Resuelva en esta misma hoja; no se aceptan otras.

1 Escriba una **expresión ANSI C** que sea **sintácticamente correcta** pero **semánticamente incorrecta**, justifique.

2 Indique si el lexema '\' es léxicamente correcto, justifique.

3 Indique la salida del programa que remueve comentarios (TP #1, ejercicio #23) si su entrada es: **a/*/*b*/"c**

Salida:

4 Escriba el contenido de la pila resultante del programa analizador sintáctico simple (TP #2, ejercicio #24) si su entrada es: '['/*(*)/"]'

Recuerde que la cima de la pila se escribe a la izquierda. Pila inicial: \$

Pila resultante:

5 Complete las líneas indicadas para que cumpla con el TP #1:

```
enum {FueraDeTodo, PosibleSalidaDeComentario, DentroDeComentario /*otros más*/} s=FueraDeTodo;
int c;
while((c=getchar()) ..... ) // Completar
    switch(s){
        case DentroDeComentario:
            switch(c){
                case '*': s = ..... ; ..... // Completar
                case '/':
                default : s = ..... ; ..... // Completar
```

6 Complete las líneas indicadas para que cumpla con el TP #2:

```
void FueraDeTodo(void){
    switch(c=getchar()){
        case ..... : // Completar
            Push("]\"");
            FueraDeTodo();
            return;
        case ']':
            switch( ..... ){ // Completar
                case ..... : // Completar
                    FueraDeTodo();
                    return;
                default:
                    puts(.....); // Completar con mensaje descriptivo.
                    exit(1); // Finaliza la ejecución del programa
```

- No se responden consultas durante la evaluación; por lo tanto, de ser necesario, escriba hipótesis de trabajo, las cuales también serán evaluadas.
- Resuelva en esta misma hoja; no se aceptan otras.

1 Indique la salida del programa que remueve comentarios (TP #1, ejercicio #23) si su entrada es: **x"/*/*y*/"z**
Salida:

2 Escriba una **sentencia ANSI C** que sea **sintácticamente correcta** pero **semánticamente incorrecta**, justifique.

3 Indique si el lexema "****" es léxicamente correcto, justifique.

4 Escriba el contenido de la pila resultante del programa analizador sintáctico simple (TP #2, ejercicio #24) si su entrada es: '**{'/*[*]"**'
Recuerde que la cima de la pila se escribe a la izquierda. Pila inicial: \$
Pila resultante:

5 Complete las líneas indicadas para que cumpla con el TP #2:

```
void FueraDeTodo(void){\n    switch(c=getchar()){\n        case ..... : // Completar\n            Push("]");\n            FueraDeTodo();\n            return;\n        case ']':\n            switch( ..... ){ // Completar\n                case ..... : // Completar\n                    FueraDeTodo();\n                    return;\n                default:\n                    puts(.....); // Completar con mensaje descriptivo.\n                    exit(1); // Finaliza la ejecución del programa\n    }\n}
```

6 Complete las líneas indicadas para que cumpla con el TP #1:

```
enum {FueraDeTodo, PosibleSalidaDeComentario, DentroDeComentario /*otros más*/} s=FueraDeTodo;\nint c;\nwhile((c=getchar()) ..... ) // Completar\n    switch(s){\n        case DentroDeComentario:\n            switch(c){\n                case '*': s = ..... ; ..... // Completar\n                case '/':\n                default : s = ..... ; ..... // Completar\n            }\n    }
```



¿Consultas?



Fin de la clase



Fin de la clase y
de la cursada
Gracias