Introducción

- Escrito con la intención de tener un mejor herramienta para reescribir es sistema operativo unix
- Historia
  - En 1969 se reescribe unix en lenguaje B derivado de BCPL, que no tiene tipos de datos.
  - En 1971 crean el “nuevo B” que si tiene tipos de datos
  - En 1972 cambia el nombre a lenguaje C y en 1973 se reescribe unix en C.
  - En 1978 sale el libro de Brian Kernighan y Dennis Ritchie que actuó de “estándar de facto” hasta el primer estándar verdadero

Estándares

- ANSI C
  - ANSI X3.159-1989 conocido como C89
  - ISO/IEC 9899:1990 conocido como C90 (es prácticamente idéntico a C89)
- C99
  - ISO/IEC 9899:1999
- C11 (por 2011)
  - ISO/IEC 9899:2011
- C17
  - ISO/IEC 9899:2018
- Más datos en:
  - https://en.cppreference.com/w/c/language/history

Diseño

- Con la intención de simplificar el compilador para hacerlo fácilmente portable se lo dividió en:
  - Preprocesador: basado en directivas que modifican en forma automática el fuente antes de que intervenga el compilador
  - El compilador propiamente dicho
  - La biblioteca estándar que es un conjunto de funciones usadas habitualmente, varias de ellas típicamente integradas en otros lenguajes, como la entrada salida, simplificando el lenguaje y por tanto el compilador

Partes del Estándar

- El estándar trata cada parte del lenguaje con la siguiente organización
  - Preliminares: temas organizativos, referencias, glosario. Capítulos 1 al 4.
  - Entorno: de la traducción y posibles ambientes de ejecución. Capítulo 5.
  - El lenguaje: sintaxis, restricciones y semántica. Cubre el lenguaje y el preprocesador. Capítulo 6.
  - Biblioteca estándar: Capítulo 7.
  - Anexos: Resúmenes de sintaxis y biblioteca estándar y otros temas como comportamiento, consideraciones de portabilidad y otros.

Entorno

- Entorno de traducción
  - Compilación por partes + vinculador (linker)
  - Unidad de traducción (translation unit): cada fuente luego del preprocesador
- Entorno de ejecución
  - Entorno independiente (Freestanding)
    - Sin sistema operativo, típicamente un embebido.
    - El nombre de la función donde inicia el programa puede ser cualquiera (definido por la implementación)
    - Solo está obligado a dar soporte a un subconjunto de la biblioteca estándar
  - Entorno alojado (Hosted)
    - Bajo un sistema operativo

Entorno

- Entorno alojado (Hosted)
  - El programa comienza en la función main que devolverá 0 para indicar éxito y distinto de cero para indicar que terminó con error. Tendrá una de dos formas
    - `int main(void)`
    - `int main(int argc, char *argv[])`
  - argc deb ser positivo, incluye el nombre del programa en argv[0] (si está disponible), el resto son los argumentos de la función.
  - argv[argc] es un puntero nulo
  - argc y argv viven durante toda la ejecución del programa y pueden ser modificados
  - Si no hay sentencia return, devuelve cero

Ejemplo de Código

ORIGINAL

```
 1  /* Basado en ejercicio 30 */
 2  #include <stdio.h>
 3  #include <stdlib.h>
 4  #define MAX 10
 5  int main (int argc, char *argv[])
 6  {
 7      /* argv lo puedo ver como char **argv */
 8      char vec [MAX+1];
 9      int i;
10      if (argc > MAX) {
11          printf ("No se puede\n");
12      } else {
13          for (i=0; i < argc; i++)
14              vec[i] = argv[i][0];
15          vec[i] = '\0';
16          printf ("La cadena creada es %s\n", vec);
17      }
18      return EXIT_SUCCESS;
19  }
```

DESPUES DEL PREPROCESADOR

```
 492
 493  extern int printf (__const char *__restrict
__format, ...);
1853
1854  int main (int argc, char *argv[])
1855  {
1856
1857   char vec [10 +1];
1858   int i;
1859   if (argc > 10) {
1860    printf ("No se puede\n");
1861   } else {
1862    for (i=0; i < argc; i++)
1863     vec[i] = argv[i][0];
1864    vec[i] = '\0';
1865    printf ("La cadena creada es %s\n", vec);
1866   }
1867   return 0;
1868  }
```

Categorías Léxicas

- Lenguaje C usa varios LR a los cuales llama categorías léxicas o tokens
- Cada palabra de alguno de estos lenguajes se conoce como lexema
- En lenguaje C se reconocen los siguientes tokens
  - PalabraReservada (keyword)
  - Identificador
  - constante
  - LiteralCadena (string-literal)
  - CaracterPuntuación (punctuator)
    - Incluye a los operadores, que en ANSI C se los consideraba una categoría separada

Palabras Reservadas (C 17)

Nuevo en estándar			
Ansi C	C99	C11	
auto	extern	short	while
break	float	signed	_Alignas
case	for	sizeof	_Alignof
char	goto	static	_Atomic
const	if	struct	_Bool
continue	inline	switch	_Complex
default	int	typedef	_Generic
do	long	union	_Imaginary
double	register	unsigned	_Noreturn
else	restrict	void	_Static_assert
enum	return	volatile	_Thread_local

Identificadores - Conceptos relacionados

- **Objeto**: zona contigua de memoria que puede contener un valor de un tipo dado.
- **Tipo**: define un conjunto de valores posibles y un conjunto de operaciones.
- **Valor**: es un conjunto de bits en la memoria interpretados según un tipo.
- Un **identificador** sirve para designar
  - Objetos: variables
  - Funciones
  - El nombre o los miembros de: estructuras, uniones y enumeraciones
  - "Alias" de tipos de datos (typedef)
  - Una etiqueta (para saltos)
  - Una macro o sus parámetros

Identificadores – Declaración y Definición

- Declaración: especifica los atributos y la interpretación del identificador.
  - Sirve también para que una unidad de traducción referencie a objetos o funciones definidas en otra unidad de traducción.
- Definición: es una declaración que además:
  - Si es un objeto causa que se reserve memoria para alojarlo.
  - Si es una función incluye el código de la misma.
- Para un objeto o función se puede tener varias declaraciones pero solo una definición

Ámbito y Vinculación

- Ámbito (Scope): partes de la unidad de traducción donde un identificador puede ser usado para referirse al objeto que designa.
  - Un mismo identificador puede designar diferentes objetos en distintos ámbitos, por ejemplo dos funciones pueden tener una variable que se llame igual.
- Vinculación (Linkage): Se refiere a que el mismo identificador pueda designar el mismo objeto (o no) en distintas unidades de traducción o solo dentro de una
  - Externa: Desde distintas unidades de traducción
  - Interna: Desde una unidad de traducción
  - Sin vinculación: solo en su bloque (variables locales y argumentos de la función)

Definiciones tentativas - Predefinido

- En C se admite una “mala práctica” por cuestiones históricas. Se puede definir varias veces una variable y tomar una sola como definición, el resto son “tentativas” que terminan actuando como declaración
  - En distintas unidades de traducción
  - En la misma pero a nivel archivo
- Identificador predefinido: en cada función está disponible sin necesidad de definirlo el identificador `__func__` que es arreglo de caracteres con el nombre de la función

Identificadores - Ejemplos

```c
extern int var; //declaro var, definida en otro fuente
int varex; //ámbito el archivo, vinculación externa
static int varin; //ámbito el archivo, vinculación interna

int f1(void) //por defecto las funciones tiene vinculación externa
{
    int a; //defino a
    ...
}
static int f2(int arg) //vinculación interna
{
    int a; //diferente a la de f1, otro ámbito, sin vinculación
    static int b; //ámbito en f2 pero "vive" durante toda la
                  //ejecución
    int c = 1;
    int d, e = 2, *pi, arr[3]; //mal estilo en general, pero
                               //permitido
        ...
 }
```

Constantes Numéricas Enteras

- Decimales: dígitos del 0 al 9 pero **NO puede** comenzar con 0
- Octales: comienza con 0 y solo utilizan los dígitos del 0 al 7
- Hexadecimales: comienzan con 0x o 0X y además de los dígitos agregan las letras de la A a la F, pueden ser mayúsculas o minúsculas, incluso entremezcladas.
- Sin sufijo son de tipo `int` (en principio)
- Sufijos: `l L u U ll LL` pueden combinarse en cualquier orden, salvo `L` con `LL`, pueden ser mayúsculas o minúsculas
- Ejemplos
  - `12` //decimal, tipo int
  - `012u` //octal, tipo unsigned int
  - `0x2fLL` //hexadecimal, tipo long long int

Constantes Numéricas Reales

- Decimales
  - Debo poner la parte fraccionaria o el exponente, uno de los dos es obligatorio. El exponente comienza con e o E y su signo es optativo.
- Hexadecimales
  - Comienzan con 0x y luego la parte fraccionaria en hexadecimal
  - El exponente es obligatorio y comienza con p o P y luego en decimal el exponente cuya base es 2.
- Ambos
  - El . de la parte fraccionaria puede no tener dígitos adelante o atrás
  - Sin sufijo son de tipo `double`
  - Sufijos: `l L f F` (`F` para `float` y `L` para `long double`)

Constantes Reales - Ejemplos

- `5e4` //double con valor $5x10^4$
- `.2f` //float con valor 0,2
- `3.L` //long double con valor 3,0
- `7.2E-3` //double con valor 0.0072
- `0xBp3` //double con valor $11x2^3$
- `0X12.cP-5`
  //valor: (16+2+12/16)/32 = 0,585938

Constantes de Carácter

- De carácter
  - Delimitadas entre comillas simples `''`
  - Si bien las variables las declaramos de tipo `char` las constantes son de tipo `int`
  - Secuencias de escape
    - `\', \", \?, \\, \dig-oct, \xdig-hex`
    - `\a , \b , \f , \n , \r , \t  \v`
- Caracteres “anchos” (wide)
  - `wchar_t` definido en `stddef.h` usa prefijo `L`
  - `char16_t` definido en `uchar.h` usa prefijo `u`
  - `char32_t` definido en `uchar.h` usa prefijo `U`

Constantes de enumeración

- De enumeración
  - Las que se definen con enum
    - `enum colores {ROJO, AMARILLO = 3, VERDE};`
    - `enum colores col = VERDE; //asigna 4`
  - Son de tipo `int`
  - En C son simples constantes, no definen un nuevo tipo de datos como si puede hacer C++
    - `col = 300;`
      - válido en C
      - error: invalid conversion from ‘int’ en C++

Literales de cadena

- Se encierran entre comillas dobles
- El final está delimitado por `'\0'`
- Se puede aplicar \ para poder incluir una comillas doble y cualquier secuencia de escape válida para constantes de carácter.
- El tipo de dato es `char[]` (generalmente usado como `char*` que es a lo que degrada)
- Cadenas adyacentes se concatenan
- Se pueden agregar prefijos: u8 para utf8, L, u y U con el mismo significado que para caracteres

Puntuación (punctuator - C17)

Del estándar C17 6.4.6: Un signo de puntuación es un símbolo que tiene un significado sintáctico y semántico independiente. Dependiendo del contexto, puede especificar una operación a realizar (que a su vez puede generar un valor o el designante de una función, producir un efecto lateral o alguna combinación de los mismos) en cuyo caso se conoce como operador (otras formas de operador también existir en algunos contextos).

```
punctuator: one of
[ ] ( ) { } . ->
++ -- & * + - ~ !
/ % << >> < > <= >= == != ^ | && ||
? : ; ...
= *= /= %= += -= <<= >>= &= ^= |=
, # ##
```

Operadores

Fuente: https://es.cppreference.com/w/c/language/operator_precedence

Precedence	Operator	Description	Associativity
1	`++ --`	Suffix/postfix increment and decrement	Left-to-right
	`()`	Function call	
	`[]`	Array subscripting	
	`.`	Structure and union member access	
	`->`	Structure and union member access through pointer	
	`(`*`type`*`){`*`list`*`}`	Compound literal(C99)	
2	`++ --`	Prefix increment and decrement	Right-to-left
	`+ -`	Unary plus and minus	
	`! ~`	Logical NOT and bitwise NOT	
	`(`*`type`*`)`	Type cast	
	`*`	Indirection (dereference)	
	`&`	Address-of	
	`sizeof`	Size-of	
	`_Alignof`	Alignment requirement(C11)	
3	`* / %`	Multiplication, division, and remainder	Left-to-right

Operadores

Fuente: https://es.cppreference.com/w/c/language/operator_precedence

Precedence	Operator	Description	Associativity
4	+ -	Addition and subtraction	Left-to-right
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional	Right-to-Left
14	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Puntuación – Otros casos

- { } delimitan bloques
- ; transforma una expresión en sentencia o separan expresiones dentro de for
- , se utiliza para separar variables en su declaración o definición y para separar argumentos de una función.
- : se usa en etiquetas, ya sea para goto o para los distintos casos dentro de un switch
- ... se utilizan para declarar o definir funciones con cantidad variable de argumentos
- # directiva del preprocesador

Detección de lexemas

```
/* fragmento de ejemplo*/
int mifunc(int a)
{
        int r,aux=1;
        r = aux+++a;  /*solo para mostrar +++ */
        return r;
}
```

int	palabraReservada	**,**	caracterPuntuación	**++**	operador
mifunc	identificador	**aux**	identificador	**+**	operador
(	caracterPuntuación	**=**	caracterPuntuación	**a**	identificador
int	palabraReservada	**1**	constante	**;**	caracterPuntuación
a	identificador	**;**	caracterPuntuación	**return**	palabraReservada
)	caracterPuntuación	**r**	identificador	**r**	identificador
{	caracterPuntuación	**=**	operador	**;**	caracterPuntuación
int	palabraReservada	**aux**	identificador	**}**	caracterPuntuación
r	identificador				

Licencia