Elementi lessicali, operatori e il sistema C

Francesco Isgrò

- Il C è un linguaggio
  - Alfabeto e regole per formare le parole
  - Sintassi
- Il compilatore verifica che il programma è sintatticamente corretto
- Correttezza sintattica necessaria per la traduzione

Caratteri ed elementi lessicali

Character Set

- The character set of C represents alphabet, digit or any symbol used to represent information.

Types	Character Set
Uppercase Alphabets	A, B, C, ... Y, Z
Lowercase Alphabets	a, b, c, ... y, z
Digits	0, 1, 2, 3, ... 9
Special Symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /
White spaces	Single space, tab, new line.

- Un programma è una sequenza di caratteri leciti
- I caratteri sono raggruppati dal compilatore in token

```
/* Read in two integers and print their sum. */

#include <stdio.h>

int main(void)
{
   int   a, b, sum;

   printf("Input two integers:  ");
   scanf("%d%d", &a, &b);
   sum = a + b;
   printf("%d + %d = %d\n", a, b, sum);

   return 0;
}
```

```c
/* Read in two integers and print their sum. */

#include <stdio.h>

int main(void)
{
   int   a, b, sum;

   printf("Input two integers:  ");
   scanf("%d%d", &a, &b);
   sum = a + b;
   printf("%d + %d = %d\n", a, b, sum);

   return 0;
}
```

- Il compilatore raccoglie i caratteri in token d 4 tipi
  - Identificatori: main, a, b sum
  - Operatori: ( ) dicono che main è una funzione
  - Punteggiatura: { } ; ,
  - Parole chiave: int

```c
/* Read in two integers and print their sum. */

#include <stdio.h>

int main(void)
{
   int   a, b, sum;

   printf("Input two integers:  ");
   scanf("%d%d", &a, &b);
   sum = a + b;
   printf("%d + %d = %d\n", a, b, sum);

   return 0;
}
```

- Il compilatore usa gli spazi per distinguere i token
- Lo spazio bianco è necessario. Non si può scrivere

  inta, b, sum;

  Il compilatore non riesce a staccare int da a
- Non tutti gli spazi bianchi lo sono

  int a,b,sum;
- Le virgole separano già i token.
- Sono necessarie
  - int a,b,sum;

  Interpretata come una sola variabile absum di tipo int

```c
/* Read in two integers and print their sum. */

#include <stdio.h>

int main(void)
{
   int   a, b, sum;

   printf("Input two integers:  ");
   scanf("%d%d", &a, &b);
   sum = a + b;
   printf("%d + %d = %d\n", a, b, sum);

   return 0;
}
```

- printf e scanf sono identificatori.
- Le parentesi informano che sono funzioni
- Dopo la compilazione il linker crea l’eseguibile
- La “definizione” delle due funzioni viene presa dalla libreria standard
- Le parole printf e scanf sono riservate e non dovrebbero essere usate per altri scopi. Un esempio dopo.

- “Input two integers”
- Una serie di caratteri fra doppi apici è interpretata come una stringa costante
- Il compilatore la tratta come un singolo token

- &a, &b;
  - Il carattere & è l’operatore di indirizzo
  - Trattato come un token, separato dalla variabile.
  - Non è sintatticamente necessario scrivere & accanto alla variabile
  - Altre scritture lecite
    - & a, & b;
  - Devono comunque essere separate dalla ,
    - & a & b non è lecito

```
/* Read in two integers and print their sum. */

#include <stdio.h>

int main(void)
{
   int   a, b, sum;

   printf("Input two integers:  ");
   scanf("%d%d", &a, &b);
   sum = a + b;
   printf("%d + %d = %d\n", a, b, sum);

   return 0;
}
```

- I caratteri + e = sono operatori.
- Gli spazi bianchi sono ignorati
- Scritture lecite
  - sum=a+b;
  - sum = a + b ;
- Non lecito
  - s u m = a + b ;
  - Il compilatore tratta ognuna delle lettere s u m come un diverso token.

Commenti

- I commenti sono stringhe arbitrarie di testo fra i delimitatori /* e */
- I commenti non sono token
- Il compilatore trasforma i commenti in un spazio vuoto

- I commenti sono usati come documentazione del codice
- Dovrebbero essere scritti contemporaneamente al codice
- Una volta che il codice funziona si tende a scrivere commenti molto brevi

Parole chiave

- Parole riservate che non possono essere usate in altri contesti
- Alcune implementazione hanno parole chiave aggiuntive
- Il C ha, nel complesso, un numero basso di parole chiave
-

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	Volatile
const	float	short	Unsigned

Identificatori

- Un identificatore è un sequnza di caratteri composta da
  - Lettere
  - Cifre
  - Il carattere speciale _
- C è *case sensitive*: lettere maiuscole e minuscole sono considerate differenti
- Scegliere identificatori che contribuiscano alla
  - Leggibilità
  - Documentazione

- Formalmente

  $$Identifier ::= \{letter \mid underscore\}_1 \{letter \mid underscore \mid digit\}_{0+}$$

- Esempi leciti
  - K
  - _id
  - iamanidentifier2
  - so_am_i

- Formalmente
  - $Identifier ::= \{letter \mid underscore\}_1 \{letter \mid underscore \mid digit\}_{0+}$
- Esempi non leciti
  - not#me
  - 101_south
  - -plus

- In sistemi vecchi solo i primi 8 caratteri venivano considerati
  - i_am_an_identifier e i_am_an_elephant sarebbero considerati uguali
  - I primi 8 caratteri: i_am_an_
- In ANSI C sono considerati almeno i primi 31 caratteri
- Molti sistemi considerano anche più di 31 caratteri

- Scelta degli identificatori
  - Se vogliamo calcolare quanto è l’iva su un prodotto
  - float tax_iva, price , tax_iva_rate;
  - tax_iva = price * tax_rate;
- La semantica di ogni variabile è chiara

- Per convenzione identificatori di sistema iniziano con underscore, a volte 2
- Per evitare possibili conflitti meglio evitare di creare identificatori che iniziano con underscore

Operatori

- Gli operatori hanno regole di precedenza e associatività
- Determinano come le espressione sono calcolate
- Le espressioni dentro le parentesi sono eseguite per prime
- Le parentesi si possono usare per
  - Chiarire l’ordine di esecuzione delle operazioni
  - Modificare l’ordine delle operazioni

- Consideriamo l’espressione

  1 + 2 * 3
- L’operatore * ha una priorità più alta di +
- Risultato è 7
- Espressione equivalente a

  1 + (2 * 3)

- Consideriamo l’espressione

  (1 + 2) * 3
- Le parentesi hanno priorità
- Risultato 9

- Se consideriamo

  1 + 2 - 3 + 4 - 5
- Gli operatori binari + e – hanno la stessa priorità
- L’ordine viene stabilito dalla regola associativa *left to right*
- Le operazioni sono eseguite da sinistra verso destra
- Equivalente quindi a
  - (((1 + 2) - 3) + 4) - 5

Operator precedence and asociativity						
Operator						Associativity
()	++ *postfix*	– *postfix*				left to right
+ *unary*	- *unary*	++ *prefix*	– *prefix*			right to left
*	/	%				left to right
+	-					left to right
=	+=	*=	/=			right to left

Incremento e decremento

- Incremento e decremento sono operatori unari
- Possono essere usati in due modi
  - Operatore prefisso
  - Operatore postfisso
- Possono essere applicati a variabili ma non a costanti
- Operatore prefisso e postfisso non hanno lo stesso comportamento

- Usi leciti
  - ++i
  - cnt--
- Usi non leciti
  - 777++
  - ++(a*b - 1)

- Ogni espressione ha un valore
- ++i
  - La variabile i viene incrementata di 1
  - L’espressione ritorna il valore di i
- i++
  - L’espressione ritorna il valore di i
  - La variabile i viene incrementata di 1

```
int a, b c==0;

a = ++c;
b = c++;
printf(“%d %d %d\n”, a, b, ++c);
```

```
int a, b c==0;

a = ++c;
b = c++;
printf(“%d %d %d\n”, a, b, ++c);
```

Output

1 1 3

- --i
  - Il valore della variabile i viene decrementato
  - Viene ritornato il valore di i
- i--
  - Viene ritornato il valore di i
  - Il valore della variabile i viene decrementato

- ++ e – modificano la variabile su cui operano
- Nessun problema in espressioni unarie
- In espressioni complesse bisogna prestare attenzione
- Meglio evitarle e scrivere espressamente i+1

```
#include <stdio.h>

int main()
{
	int a = 1;

	printf("a*(a+1) = %d\n", a*(a+1));
	printf("a*(a+1) = %d\n",a*a++);
	a = 1;
	printf("a*(a+1) = %d\n",a*++a);

	return 0;
}
```

```c
#include <stdio.h>

int main()
{
	int a = 1;

	printf("a*(a+1) = %d\n", a*(a+1));
	printf("a*(a+1) = %d\n",a*a++);
	a = 1;
	printf("a*(a+1) = %d\n",a*++a);

	return 0;
}
```

```
a*(a+1) = 2
a*(a+1) = 2
a*(a+1) = 4
```

Libreria standard

- Cosa succede se si ridefiniscono funzioni della libreria standard
- Proviamo a vedere che succede se ridefiniamo la funzione scanf
- A tal fine non includiamo il file stdio.h per non avere conflitto sui prototipi

```
void scanf(const char *s)
{
	printf(s);
}

int main()
{
	scanf("The printing scanf\n");

	return 0;
}
```

```
void scanf(const char *s)
{
	printf(s);
}

int main()
{
	scanf("The printing scanf\n");

	return 0;
}
```

```
main.c:2:6: warning: conflicting types for built-in function ‘scanf’ [-Wbuiltin-declaration-mismatch]
 void scanf(const char *s)
      ^~~~
main.c: In function ‘scanf’:
main.c:4:2: warning: implicit declaration of function ‘printf’ [-Wimplicit-function-declaration]
  printf(s);
  ^~~~~~
main.c:4:2: warning: incompatible implicit declaration of built-in function ‘printf’
main.c:4:2: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
main.c:4:2: warning: format not a string literal and no format arguments [-Wformat-security]
```

- stdio.h non è incluso
- Il compilatore si accorge comunque che si sta usando il nome di una built-in function
- Viene dato un warning, non errore
- L’eseguibile viene creato
- Viene usata la nuova definizione

- stdio.h non è incluso
- Il compilatore si accorge comunque che si sta usando il nome di una built-in function
- Warning, non errore
- L’eseguibile viene creato
- Viene usata la nuova definizione
- Output

  The printing scanf

main.c

```
void scanf(const char *s);

int main()
{
	scanf("The printing scanf\n");

	return 0;
}
```

scanf.c

```
void scanf(const char *s)
{
	printf(s);
}
```

main.c

```
void scanf(const char *s);

int main()
{
        scanf("The printing scanf\n");

        return 0;
}
```

scanf.c

```
void scanf(const char *s)
{
        printf(s);
}
```

```
main.c:2:6: warning: conflicting types for built-in function ‘scanf’ [-Wbuiltin-declaration-mismatch]
 void scanf(const char *s);
      ^~~~~
scanf.c:1:6: warning: conflicting types for built-in function ‘scanf’ [-Wbuiltin-declaration-mismatch]
 void scanf(const char *s)
      ^~~~~
scanf.c: In function ‘scanf’:
scanf.c:3:2: warning: implicit declaration of function ‘printf’ [-Wimplicit-function-declaration]
  printf(s);
  ^~~~~~
scanf.c:3:2: warning: incompatible implicit declaration of built-in function ‘printf’
scanf.c:3:2: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
scanf.c:3:2: warning: format not a string literal and no format arguments [-Wformat-security]
```