

Lezione 4

Elementi lessicali e espressioni logiche

Linguaggi di Programmazione I

Matricole 2-3

Linguaggi di Programmazione I



Elementi lessicali

- il linguaggio C ha un suo “vocabolario di base” i cui elementi sono detti *token*
- esistono 6 tipi di token:
 - parole chiave
 - identificatori
 - costanti
 - costanti stringa
 - operatori
 - simboli di interpunzione

Linguaggi di Programmazione I



Elementi lessicali

- Il primo passo del processo di compilazione consiste nell’*analisi lessicale* del programma.
- Durante l’analisi lessicale
 - i caratteri del programma vengono raggruppati in *token*
 - vengono rimossi i commenti (ciascun commento è sostituito con uno spazio)
- La sequenza di *token* prodotta durante l’analisi lessicale viene poi sottoposta all’analisi sintattica per controllare che formi una stringa legale secondo la sintassi del linguaggio

Linguaggi di Programmazione I



Le parole chiave

- Parole *riservate* al linguaggio che non possono essere ridefinite o usate in altri contesti dal programmatore

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

Linguaggi di Programmazione I



Identificatori

- Composti da lettere, cifre e il carattere `_`
 - il primo carattere deve essere una lettera o il carattere `_`
- Usati per definire nomi unici per oggetti in un programma (variabili, funzioni etc.)
- Lunghezza degli identificatori:
 - ANSI C: 31 caratteri (almeno)
- E' buona regola usare identificatori con un significato:
 - esempio: `x= y*z;` NO!
 - `area = base *altezza ;` OK!
- Gli identificatori che iniziano con il carattere `_` dovrebbero essere utilizzati esclusivamente dai programmatori di sistema

Linguaggi di Programmazione I



Costanti

- Rappresentano valori
- Costanti intere: `50 -7`
- Costanti virgola mobile: `13.5 9.0`
- Costanti carattere: `'a' ';' '\n'`

Linguaggi di Programmazione I



Costanti stringa

- sequenze di caratteri racchiuse tra doppie virgolette
 - “Hello world!”
- per utilizzare il carattere “ all’interno di una stringa bisogna farlo precedere da `\`
 - “Antonella gli chiese : \“ come ti chiami?” ”
- per utilizzare il carattere `\` all’interno di una stringa bisogna farlo precedere da un altro `\`
 - “il carattere `\\` è chiamato backslash ”
- le sequenze di caratteri che potrebbero avere un significato al di fuori della costante stringa vengono considerate semplici sequenze di caratteri all’interno delle doppie virgolette:
 - “`a=1*2 ;` ” non esegue alcuna operazione

Linguaggi di Programmazione I



Simboli di interpunzione e operatori

- I **segni di interpunzione** permettono al compilatore di separare e distinguere gli elementi del linguaggio
 - Esempi : `() ; , { }`
 - Gli **operatori** sono caratteri a cui è associata una operazione
 - Esempi: gli operatori aritmetici: `+ - * / %`
 - Gli operatori separano gli identificatori. Comunque separare operatori con spazi migliora la leggibilità del programma
- $$x=z*(3.16+y/2); \quad x = z * (3.16 + y / 2);$$
- Le parentesi `()` dopo il nome di una funzione vengono trattate come operatori

Linguaggi di Programmazione I



Esempio di analisi lessicale

```

/* File: fattoriale.c
Esempio dell'uso di while
*/
#include <stdio.h>
main ()
{
  int i,prod;
  i=1;
  while (i<=10)
  {
    prod=prod*i;
    i=i+1;
  }
  printf ("%d\n", prod);
}

```

Lines 1-2: comment (identificatore)
 Line 3: preprocessor directive (operatore)
 Line 4: function name (operatore)
 Line 5: opening brace (simbolo di interpunzione)
 Line 6: variable declaration (identificatore)
 Line 7: assignment (operatore)
 Line 8: while loop condition (operatore, costante)
 Line 9: opening brace (simbolo di interpunzione)
 Line 10: multiplication (operatore)
 Line 11: increment (operatore)
 Line 12: closing brace (simbolo di interpunzione)
 Line 13: printf statement (operatore)
 Line 14: closing brace (simbolo di interpunzione)

Linguaggi di Programmazione I



Operatori di incremento ++ e decremento --

- incrementa/decrementa di 1 il valore di una variabile.
 - Esempio: `i--`
- Funzionano in due fasi:
 - Operazione di incremento/decremento (side-effect)
 - Valutazione per uso in espressioni
- **Operatore prefisso:** operazione prima della valutazione
 - Esempio: `int x, i = 4;`
`x = ++i;` assegna ad x valore 5
- **Operatore postfixo:** valutazione prima dell'operazione
 - Esempio: `int x, i = 4;`
`x = i++;` assegna ad x valore 4



Priorità ed associatività degli operatori

- Regole per stabilire l'ordine di applicazione degli operatori
- Operatori con priorità più alta vengono applicati prima di quelli a priorità più bassa
- A parità di priorità si segue la associatività (che può essere da sinistra o da destra)



Priorità ed associatività degli operatori

- Di seguito sono elencati alcuni operatori in ordine di priorità decrescente. Gli operatori sulla stessa riga hanno uguale priorità. Per ciascuna riga è indicata l'associatività degli operatori che si trovano su quella riga:

Operatori	Associatività
<code>() ++ (postfisso) -- (postfisso)</code>	Da sinistra
<code>+ -(unari) ++(prefisso) --(prefisso)</code>	Da destra
<code>* / %</code>	Da sinistra
<code>+ -</code>	Da sinistra
<code>= += -= *= /= %= ...</code>	Da destra



Alcuni esempi

```
int a=3, b=4, c=2;
```

espressione	espressione equivalente	valore
$a*b/c$	$(a*b)/c$	6
$a+b*c$	$a+(b*c)$	11
$b-- / c - ++a$	$((b--)/c)-(++a)$	-2
$a - - c * ++ b$	$a - ((-c)*(++b))$	13
$a +=b=3+ c$	$a+=(b=(3+c))$	8

Linguaggi di Programmazione I



Espressioni logiche

- espressioni logiche (booleane): assumono valori di *true* o di *false*
- Il C non definisce un tipo di dati booleano.
 - si usa il tipo di dati *int*
 - *false* è 0
 - *true* è qualsiasi valore diverso da 0
- nelle espressioni logiche vengono utilizzati gli *operatori relazionali, di uguaglianza e logici*

Linguaggi di Programmazione I



Operatori relazionali, di uguaglianza

- Operatori relazionali:
 - $>$ maggiore
 - $<$ minore
 - $>=$ maggiore o uguale
 - $<=$ minore o uguale
- Operatori di uguaglianza
 - $==$ uguale
 - $!=$ diverso

Linguaggi di Programmazione I



Operatori relazionali

- $>$, $<$, $>=$, $<=$
- operazioni binarie che restituiscono un risultato di tipo *int* che può essere 0 o 1
- associatività da sinistra
 - attenzione!

```
i=9;
printf("%d\n", 5 < i < 8)
```

stampa 1 in quanto $5 < i < 8$ è equivalente a $(5 < i) < 8$

- $a < b$ è equivalente a $(a-b) < 0$
 - attenzione! Il risultato dipende dalla precisione della macchina :

```
x = 9e+33;
y = 0.0001;
printf("%d\n", x < x + y );
```

può stampare 0 in quanto x e $x+y$ potrebbero essere uguali in base alla precisione della macchina

Linguaggi di Programmazione I



Operatori di uguaglianza

- **== (uguale), != (diverso)**
- operazioni binarie che restituiscono un risultato di tipo *int* che può essere 0 o 1
- associatività da sinistra
- $a != b$ è equivalente a $!(a == b)$
- $a == b$ è implementata come $(a-b) == 0$;
- alle espressioni che sono operandi di operatori di uguaglianza vengono applicate le usuali regole di conversione aritmetica
 - l'espressione $0.0 == 0$ ha valore 1

Linguaggi di Programmazione I



Operatori logici

- Operatori logici
 - **!** NOT logico (unario)
 - **||** OR logico (binario)
 - **&&** AND logico (binario)

Linguaggi di Programmazione I



NOT logico

- Il NOT logico (negazione logica) restituisce un valore *int*.
- **!expr** è uguale a:
 - **0 (false)** se **expr** è **1 (true)**
 - **1 (true)** se **expr** è **0 (false)**
- Il NOT logico è *associativo a destra*:
 - **!!expr** è equivalente a **!(!expr)**

Linguaggi di Programmazione I



OR logico

- L' OR logico restituisce un valore *int*.
- **expr1 || expr2** è uguale a:
 - **0 (false)** se **expr1** e **expr2** sono entrambe **0 (false)**
 - **1 (true)** se almeno una tra **expr1** e **expr2** è **1 (true)**
- L' OR logico è *associativo a sinistra*:
 - **expr1 || expr2 || expr3** è equivalente a **(expr1 || expr2) || expr3**

Linguaggi di Programmazione I



AND logico

- L' AND logico restituisce un valore *int*.
- `expr1 && expr2` è uguale a:
 - `0 (false)` se almeno una tra `expr1` e `expr2` è `0 (false)`
 - `1 (true)` se `expr1` e `expr2` sono entrambe `1 (true)`
- L' AND logico è associativo a sinistra:
 - `expr1 && expr2 && expr3` è equivalente a `(expr1 && expr2) && expr3`



Priorità ed associatività degli operatori relazionali, di uguaglianza e logici

Operatori

() ++ (*postfisso*) -- (*postfisso*)
 + - (*unari*) ++ (*prefisso*) -- (*prefisso*) !
 * / %
 + -
 < <= > >=
 == !=
 &&
 ||
 = += -= *= /= %= ...

Associatività

Da sinistra
 Da destra
 Da sinistra
 Da sinistra
 Da sinistra
 Da sinistra
 Da sinistra
 Da sinistra
 Da sinistra
 Da destra



Alcuni esempi

```
int a=3, b=4, c=2;
double d = 0.0;
char x = 'B';
```

espressione	espressione equivalente	valore
<code>!x</code>	<code>!x</code>	0
<code>!d</code>	<code>!d</code>	1
<code>a && c && d</code>	<code>(a && c) && d</code>	0
<code>d a && b - 4</code>	<code>d (a && (b - 4))</code>	0
<code>3 <= b - 2 a + 4 > 6</code>	<code>(3 <= (b-2)) ((a+4) > 6)</code>	1



Lo *short-circuit* della valutazione

- Efficienza del C: la valutazione di espressioni che sono operandi di `||` e `&&` la valutazione si arresta non appena risulta noto il valore del risultato.

- Esempio:

- nell' espressione `expr1 && expr2`

`expr2` viene valutata solo se `expr1` è *true*

- nell' espressione `expr1 || expr2`

`expr2` viene valutata se `expr1` è *false*



Alcuni esempi di *short-circuit*

se l'utente inserisce un valore maggiore di 5 allora la moltiplicazione 3^a non viene effettuata.

```
.  
. .  
scanf ("%d", &a);  
if ( (a<5) && ( 3*a > 20) )  
. .  
.
```

Attenzione agli effetti collaterali:

```
if ( (a<5) && ( i++ > 9) )
```

se l'utente inserisce un valore maggiore di 5 allora i non viene incrementato

```
.  
. .  
scanf ("%d", &a);  
if ( (a<5) && ( i++ > 9) )  
. .  
.
```